The Contract

Not a promise. A program.

February 2026

Not a privacy policy

Every company has a privacy policy. Every privacy policy says the company takes your privacy seriously. Every user scrolls past it. And when the company violates it, the enforcement mechanism is a lawsuit, a regulator, or public outrage — all of which happen after the damage is done, if they happen at all.

A connection contract is not a privacy policy.

A connection contract is a structured, versioned, machine-readable document that declares exactly what a service needs from you, exactly what it can do with your data, and exactly what limits apply. It is presented by your vault in plain language before you accept. And once you accept, your vault enforces it on every single request the service makes, for the entire lifetime of the connection.

The difference is not subtle. A privacy policy is a promise. A connection contract is a program. The service does not need to honor the agreement — the architecture honors it. A request that violates the contract terms never reaches your data. It is blocked by your vault, logged in your audit trail, and you are notified. The service does not get a second chance. It does not get a grace period. It gets a rejection.

> *Today's privacy policies are unenforceable wishes. A connection contract is enforced by cryptography and architecture. The service cannot violate terms it cannot bypass.*

Anatomy of a contract

A connection contract is short because it is structured. There is no boilerplate, no legal preamble, no fifty-page document written by lawyers for lawyers. It is a set of declarations that the vault can read, present, and enforce. Here is what it contains.

Identity

Every contract identifies who published it: the service's name, its verified status, its unique identifier, the contract version number, and when it was published. If the service has human-readable terms or a privacy policy, the contract includes links to those documents. But the vault does not rely on the human-readable version for enforcement. The structured declarations are what the vault enforces. The links are there for people who want the full legal context.

Field access levels

This is the heart of the contract. Every piece of data a service can request from your vault is classified into one of four tiers. Each tier has different rules, and the vault enforces those rules on every request.

Tier	What it means	Enforcement
Required	You must have these fields to connect at all. The service declares why it needs each one.	Connection cannot be established without these fields. If you don't have a required field, your vault tells you and offers to add it.
Optional	Requested but not required. You can decline any or all of these and still connect.	Service can ask for these. You can say no. The connection proceeds either way. This is the only wiggle room in the contract.
On-demand	Service can request these at any time during the connection, within rate limits.	Vault provides the data automatically if the request is within contract terms and rate limits. No per-request approval needed.
Consent	Every single access requires your explicit approval. The vault blocks the	Request arrives. Your vault shows you what is being asked, by whom,

	request until you say yes.	and why. You approve or deny. The service waits.

Every field in the contract includes a purpose statement: why the service needs this data. And every field includes a retention policy.

Retention

Retention is per-field, not per-contract. A service might need your email with “until revoked” retention for ongoing communication, but your shipping address with “session” retention that disappears after the current transaction. The contract declares this per field, and the vault enforces it.

Retention type	What happens
Session	Data is available for the current interaction only. Gone when the session ends.
Time-limited	Data is available for a declared period (30 days, 90 days, etc.) and then auto-purged by the vault.
Until revoked	Data remains available for as long as the connection is active. Revoke the connection and access ends.

This is not a suggestion. The vault tracks retention per field and enforces expiration. A service cannot request a field whose retention period has expired. It has to make a new request, and that request is subject to the same tier rules as the original.

Permissions

Beyond field access, the contract declares what the service is allowed to do through the connection. Each permission is a clear yes or no.

Can store data: Can the service add rings to your keychain? If yes, the contract also declares what categories of data and how much storage.

Can send messages: Can the service send you messages through the connection? Notifications, updates, alerts.

Can request authentication: Can the service ask your vault to authenticate you? Login requests, identity verification.

Can request payment: Can the service initiate payment requests through the connection?

Rate limits

The contract sets hard limits on how frequently the service can make requests: maximum requests per hour and maximum storage in megabytes. These are not guidelines. The vault runs a token bucket algorithm per connection. Exceed the limit and the request is rejected, an event is logged, and you are notified. A service that hammers your vault gets blocked, not throttled.

> *A contract is a set of declarations: what data, at what tier, with what retention, with what permissions, at what rate. Every declaration is enforced by the vault on every request. The service does not interpret the contract. The vault does.*

How you review a contract

When a service wants to connect, your vault fetches the service’s profile and contract. Your vault — not the service — presents the terms to you. This matters. The service does not get to design the consent experience. Your vault presents the contract in a consistent, familiar format that you see for every connection, from every service.

Here is what you see.

The service’s identity: Name, verified status, organization, contact information. If the service is verified, you see a verification badge. If it is not, your vault tells you that too.

What they are requesting: Every field, organized by tier. Required fields are listed with their purpose: “Email — for order confirmations.” Optional fields are clearly marked as optional: “Phone (optional) — for delivery updates.” You see exactly what the service wants and exactly why it says it needs it.

What they can do: The permissions, in plain language. “Can store purchase history on your keychain. Can send you messages. Can request payments. Can make up to 100 data requests per hour.”

What you are missing: If the contract requires a field you do not have in your profile, your vault tells you and offers to add it. You cannot connect without the required fields. This is not a punishment — it is clarity. The service needs your email to send order confirmations. If you do not have an email in your vault, you cannot use the service until you add one.

Links to the service’s human-readable terms and privacy policy are available if you want them. Trusted resources — the service’s official websites, verified mobile apps with signed download hashes, verified contact information — are also visible. But the structured contract is the authoritative document. The human-readable version is context. The machine-readable version is law.

You accept or decline. There is no negotiation. The service published its terms. You review them in your vault. You decide. The only flexibility is in the optional fields: you can accept the connection while declining any optional field the service requested. Everything else is take it or leave it. The power is not in negotiating better terms. The power is in enforcement and the ability to walk away cleanly.

> *Your vault presents every contract in the same format, from every service. The service does not design the consent experience. You see identity, fields, permissions, and limits — the same way, every time. Accept or walk away.*

Enforcement, not promises

This is the section that makes everything before it real. A contract is a document. Enforcement is what turns it into a guarantee.

Every request a service makes travels through the messaging layer to your vault. Before the vault manager even considers invoking a handler, it checks the request against the accepted contract. This is not an optional step. It is the architecture.

The checklist

On every incoming service request, the vault runs through a series of checks.

Is the contract current? If the service has published a new contract version and the user has not yet accepted it, the old contract is in a suspended state. The service cannot access data under the old terms while an update is pending. It must wait for acceptance or face disconnection.

Is the field in the contract? If the service requests a field that is not declared in the contract, the request is blocked. It does not matter if the field exists in the user's profile. The contract did not include it. Access denied.

What tier is the field? On-demand fields are served automatically. Consent fields create a pending request that blocks until the user explicitly approves. The vault will not silently upgrade a consent field to on-demand access.

Has retention expired? If the field's retention period has lapsed, the vault will not serve the data. The service must make a new request, which is subject to its tier rules.

Are we within rate limits? If the service has exceeded its declared maximum requests per hour, the request is rejected. The vault logs the violation and notifies the user.

If any check fails, the request is blocked. Not delayed. Not queued. Blocked. The response to the service is a rejection. The event is logged in your audit trail. And depending on the severity — a rate limit warning versus an unauthorized field request — you may be notified immediately.

This is what makes a connection contract different from every privacy policy ever written. The enforcement is not reactive. It is not complaint-driven. It is not dependent on a regulator discovering a violation. The enforcement is structural. The vault checks every request, every time, with no exceptions and no overrides. The service cannot sweet-talk the vault. The vault is a program.

What gets logged

Every request, whether approved or denied, is logged. The audit trail records: what was requested, by which connection, at what time, under which contract version, what the result was, and if denied, why. This is not optional logging that the service can turn off. It is a structural consequence of the architecture — every request passes through the vault manager, and the vault manager logs everything it processes.

The result is a complete, tamper-evident record of every interaction a service has ever had with your data. Not because you asked for auditing. Not because a regulation requires it. Because the architecture makes it impossible for an interaction to happen without being recorded.

> *The vault checks every request against the contract: field, tier, retention, rate limit, version. If any check fails, the request is blocked and logged. Enforcement is not a feature. It is the architecture.*

When the contract changes

Services evolve. A retailer adds alcohol sales and needs age verification. A healthcare provider starts offering telemedicine and needs video consent. A financial service expands to new markets and needs additional compliance fields. The contract has to be able to change.

But a contract change cannot be silent, automatic, or retroactive. The update model is explicit and binary.

The update flow

When a service publishes a new contract version, your vault receives a notification. The vault presents the changes to you in the same consistent format you saw when you first connected, but now with a clear diff: what is new, what has been removed, what has changed.

Added: New fields the service is requesting, with their tier, purpose, and retention. New permissions the service wants. You see exactly what is being asked that was not asked before.

Removed: Fields the service no longer needs. Permissions it is giving up. This is usually good news — the service is asking for less.

Changed: Fields where the tier, purpose, or retention has changed. A field that was “session” retention is now “until revoked.” A field that was on-demand is now consent. You see the before and after.

You accept or reject. There is no third option.

Accept or disconnect

If you accept the update, the new contract version takes effect immediately. The vault begins enforcing the new terms on every subsequent request.

If you reject the update, the connection is terminated. Immediately. The service loses all access to your vault. All pending requests are auto-denied. The service’s namespace on your keychain remains in your vault — you still have the data — but the service can no longer reach it. If you want to use the service again, you must reconnect from scratch and accept the current contract.

There is no grace period. There is no “keep using the old terms.” Once a service publishes a new contract version, the old version is suspended. The service cannot access data under the old contract while an update is pending. This prevents a pattern that plagues traditional privacy policies:

the company changes its terms, gives you thirty days to “opt out,” and in the meantime keeps accessing your data under the old agreement. In ZKT, the moment the update is published, the old contract is frozen until you decide.

> *A contract update is a clear diff: added, removed, changed. You accept the new terms or you disconnect. There is no gray area. There is no grace period. The old contract is frozen the moment the update is published.*

Two kinds of connections, two kinds of contracts

Not every connection uses a formal contract. The vine supports two patterns, each appropriate to the relationship.

Peer connections: mutual and lightweight

When you connect to a friend — a real person, vault to vault — there is no formal contract document. The “contract” is implicit in the connection type: both sides present profiles, both sides review, both sides accept. The consent is bidirectional and equal. Neither side publishes terms. Neither side dictates what the other can access.

What you share with a peer is controlled by your profile sharing settings: which fields are visible to this connection, what capabilities you offer. The peer can see what you chose to share. They cannot request fields you did not offer. The enforcement is the same — your vault controls what is visible — but the governance is symmetric. Two people, two vaults, mutual consent, no formal contract needed.

Service connections: formal and one-sided

When you connect to a service — a retailer, a healthcare provider, an employer — the service publishes a contract. The contract is the service's offer: this is what we need, this is what we can do, these are the limits. You accept the offer as published. The terms are not negotiated. The only flexibility is in the optional fields, which you can decline.

This sounds one-sided, and it is. But the one-sidedness is in the terms, not in the power. The service dictates what it asks for. But you dictate whether to accept. And once you accept, the vault — your vault, running in your enclave — enforces the terms. The service cannot change the terms without your explicit acceptance. The service cannot exceed the terms without being blocked. The service cannot access data outside the terms without being logged and flagged.

The asymmetry is deliberate. A retailer has legitimate reasons to ask for your shipping address. You have legitimate reasons to evaluate that request and decide. What is not legitimate is the retailer accessing your address without telling you, selling it to a third party, or keeping it after you asked them to stop. The contract makes the legitimate requests explicit and the illegitimate requests impossible.

Shared sandbox contracts

When a connection involves a shared sandbox, the contract includes additional terms: who hosts the sandbox, what each side can do in the mutual area, and what happens to the sandbox if the connection ends. These terms follow the same model — declared by the creator, accepted by the other party, enforced by the vault. The sandbox contract is an extension of the connection contract, not a separate agreement.

> *Peer connections are mutual: two profiles, two consents, no formal contract. Service connections are formal: the service publishes terms, you accept or decline, the vault enforces. Different relationships, same enforcement model, same principle: nothing happens without consent, and consent is enforced by architecture.*

Why this matters

Connection contracts solve a problem that regulation has been trying to solve for decades and failing: how do you ensure that a service actually does what it says it will do with your data?

The regulatory approach is to write laws. GDPR. CCPA. HIPAA. These laws create requirements: tell people what data you collect, get consent, let them delete it, report breaches. But the enforcement is after the fact. The regulator investigates. The company pays a fine. The data was already misused. The damage was already done. And most violations are never discovered at all because the user has no visibility into what the service is actually doing with their data.

Connection contracts do not replace regulation. But they make enforcement structural rather than reactive. The service declares its terms in a machine-readable contract. The vault enforces those terms on every request. The audit trail records every interaction. The user can see exactly what every service has accessed, when, and under what terms. If a service tries to access data outside the contract, the request is blocked before the data is exposed — not investigated after the fact.

The contract is short because it is structured: a set of fields, tiers, retention policies, permissions, and rate limits. It is readable because your vault presents it in a consistent format. It is enforceable because the vault is a program that checks every request against the terms, every time, without exception. And it is revocable because the user can sever the connection at any moment and the service loses all access immediately.

No scrolling past fifty pages. No hoping the service honors its promises. No waiting for a regulator to discover a violation. The contract is short, clear, enforced, and yours.

> *A privacy policy is a promise a company makes to itself. A connection contract is a program your vault runs on every request. That is the difference between hoping for compliance and enforcing it.*