DATA PROTECTION & SOVEREIGNTY

Sovereignty at the Level of the Person

Per-user data residency, enforced by key topology: one project, every user in-jurisdiction, unified analytics that never move PII.

July 2026 · v1.0 · 15 min read

01 The wall: residency at the project level doesn't survive contact with a second jurisdiction
02 The core model: not every table has a data subject
03 Enforced by key topology, not policy
04 Request routing: single-region by default, cross-region only when asked for
05 Unified analytics without moving PII: scatter, decrypt in-region, prove it's non-personal, merge
06 In-jurisdiction AI: compute follows the data, and fails closed when it can't
07 Per-subject cryptographic erasure: the Article 17 answer at the grain of the person
08 Residency lifecycle: strict by default, migration as a governed exception
09 Worked example: a day in the life of a four-region fintech

Executive summary

Every other data-residency story is told at the project level: pick one region and every user in your product lives there. That collapses the moment a product has users in more than one jurisdiction — a regulated multinational is left running one project per country, stitching them together by hand, or accepting the risk of a residency violation for whichever users end up in the wrong place.

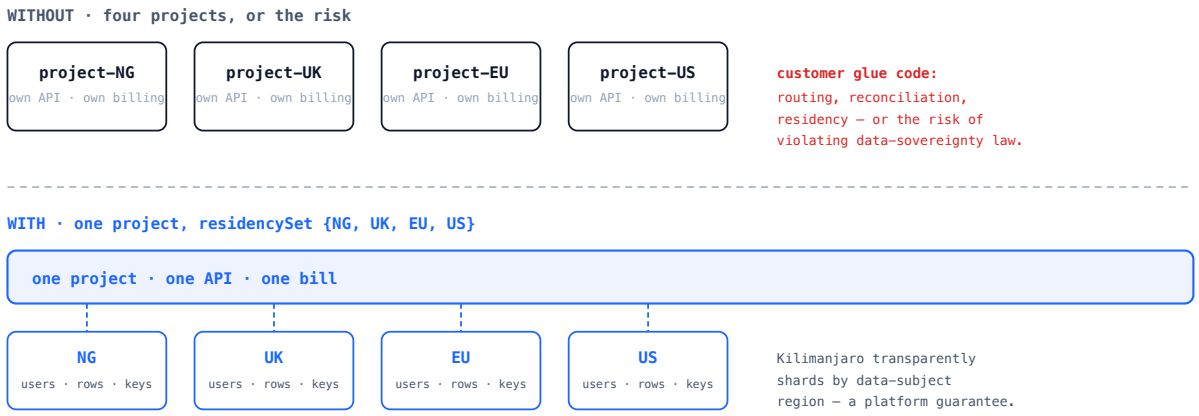
Kilimanjaro moves the residency unit down to the data subject. One project declares the jurisdictions it may place users in; each end user is pinned to one of them at creation. That user's rows, their encryption key, and their agent execution all live in their region — and the guarantee is enforced by key topology, not application policy: the key that decrypts a region's data exists only in that region.

This paper walks the mechanism end to end — the key hierarchy, the routing, the cross-region analytics that never move personal data, in-jurisdiction AI, per-subject cryptographic erasure, and the lifecycle operations — and closes with the exact boundary of what the guarantee does and does not cover.

01 The wall: residency at the project level doesn't survive contact with a second jurisdiction

A regulated multinational — a fintech with end users in Nigeria, the UK, the EU, and the US — faces one hard requirement, phrased differently by CBN, GDPR, the DPDPA, and a dozen sectoral US rules: each user's personal data stays in that user's jurisdiction. Project-level residency answers this for a single-country product and stops answering it the moment the product has users in two.

The usual response is to run one project per jurisdiction and reconcile them in the application layer — four APIs, four billing relationships, and a hand-built residency router the team now owns forever. The alternative is to accept the risk of a misplaced record. Neither is a platform guarantee; both are the customer's engineering problem, and both are why residency, done at the wrong grain, becomes the reason a deal doesn't close.



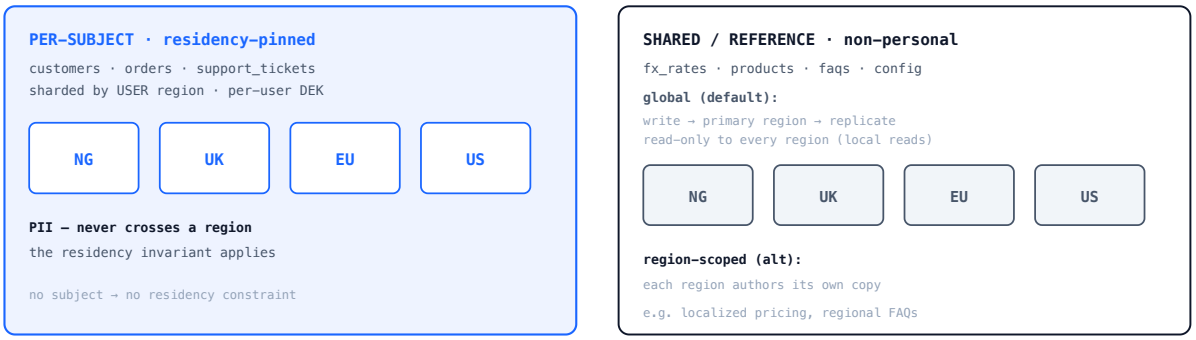
Without per-user residency: four projects, glued together by hand, or the compliance risk of a misplaced user. With: one project, a declared residency set, each user sharded to their region.

Kilimanjaro answers at the grain the regulation actually specifies: the data subject. One project declares a residency set — the jurisdictions it may place users in — and each end user is pinned to exactly one of them. The rest of this paper is that mechanism.

02 The core model: not every table has a data subject

The residency unit is the end user, not the project — and not every row in a project belongs to an end user. A project's tables split into two residency classes. Per-subject data — customers, orders, support tickets — is about or owned by a specific data subject, carries PII, and is sharded to that subject's region under a per-user key. This is what the residency guarantee protects.

Shared or reference data — FX rates, a product catalog, FAQs, feature flags — belongs to no one subject, so the residency invariant simply does not constrain it. By default it is replicated: written to the project's primary region and fanned out read-only to every region, so every region's users get a local, fast read of the same non-personal facts. Where the content genuinely differs by jurisdiction — local pricing, a regulator-mandated disclosure — a table is declared region-scoped instead, authored independently per region.



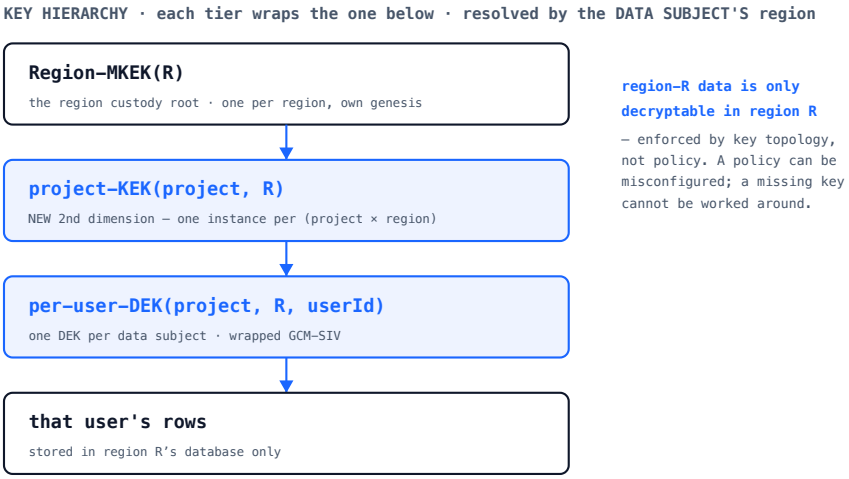
Per-subject data — rows about a data subject — is residency-pinned and sharded by that user's region. Shared or reference data has no subject, so the residency invariant does not apply to it; it is replicated globally by default or authored per-region where it legitimately differs.

The classification is the guardrail, not a formality: a table declared shared or replicated must be certified non-personal. A system table that turns out to hold PII — a beneficiaries list with names, say — is per-subject, not shared, and is residency-pinned like any other personal-data table. Getting this line right is what keeps the fast, replicated path correct.

03 Enforced by key topology, not policy

The guarantee's spine is a single structural fact: the region-R data-encryption key exists only in region R. A policy setting can be misconfigured, forgotten, or bypassed by a bug; a key that was never provisioned outside its region cannot be. This is the same by-construction discipline as Keep's Enforced Boundary — the guarantee is a property of the architecture, not an instruction the runtime is trusted to follow.

Concretely, the existing key chain gains a second dimension and a new bottom tier. The region's master key wraps a key-encryption key scoped to the pair (project, region) — an extension of the existing per-project key sharding from one instance to N. That project-region key in turn wraps a per-user data-encryption key, one per data subject, resolved by the data subject's own region rather than the project's. The user's rows are encrypted under their key, in their region's database. Nothing about the underlying cipher changes — this is provisioning and resolution, not new cryptography, which is why the change carries low implementation risk against a large product surface.



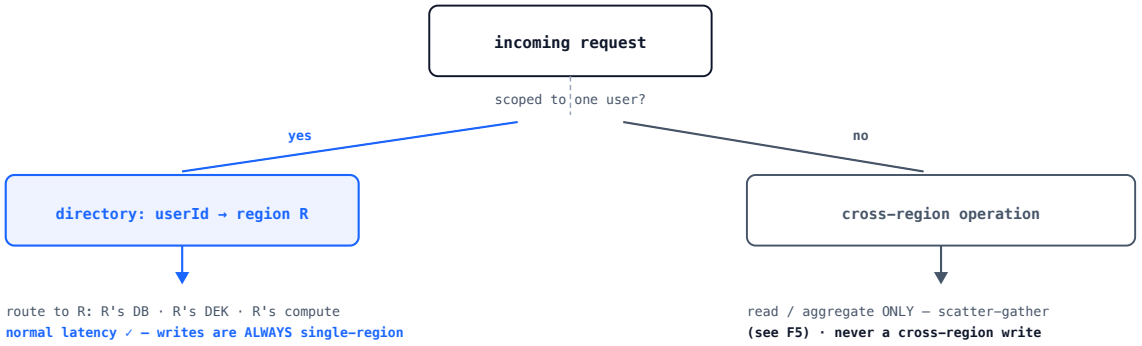
A region's master key wraps a per-(project × region) key, which wraps a per-data-subject key — resolved by the data subject's own region, never the project's. Region-R data is only decryptable where the region-R key exists: in region R.

One consequence falls out for free: destroying a single data subject's key crypto-destroys the data encrypted under it — that subject's alone, with no other subject touched — the subject of the next section.

04 Request routing: single-region by default, cross-region only when asked for

A small, non-personal directory — a user ID mapped to a region — is the only structure that is genuinely global. Every request scoped to a single user consults it and routes straight to that user's region: their database, their key, their compute. Latency is unaffected by how many regions the project spans, because a single-user operation never touches more than one.

That single-region-by-default routing has a structural payoff most distributed-data designs have to fight for: because a user is pinned to exactly one region, every write for that user is single-region by construction. There is no cross-region write transaction, no two-phase commit, no distributed consensus to reason about — the hardest part of a multi-region data system is designed out rather than engineered around.



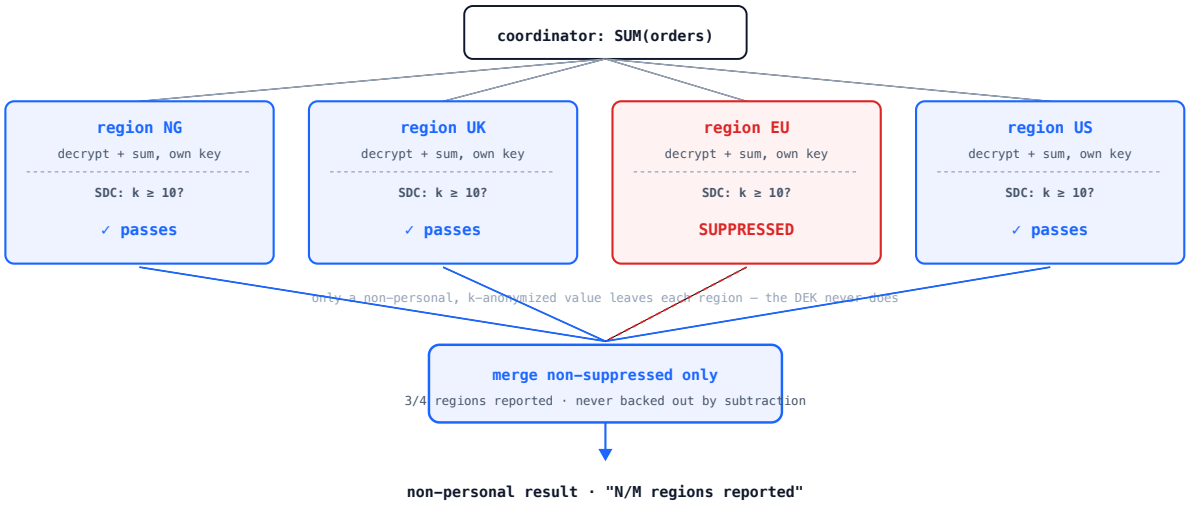
A request scoped to one user routes through a non-PII directory to that user's region for a normal-latency read or write. A request not scoped to a single user is a cross-region operation — read or aggregate only; every write stays single-region.

A request that is not scoped to a single user — a report, a dashboard, an agent batch job over many customers — is a cross-region operation, and it is read or aggregate only. That path is the subject of the next section, because it is exactly where the residency guarantee is easiest to accidentally break, and where the platform is most deliberate about not doing so.

05 Unified analytics without moving PII: scatter, decrypt in-region, prove it's non-personal, merge

"An aggregate is non-personal" is not a safe assumption — it is false for small cohorts. A SUM over one row is that person's exact value; a COUNT of one combined with an AVG reveals an individual; a top-ten ranking over a region with eight active users names all eight.

And it is not enough to count rows: one data subject can own many rows — orders, tickets, events — so a cohort of five people behind two hundred actions is still five people, not two hundred. So a cross-region rollup cannot simply ship each region's number to a coordinator and trust that the number is anonymous by virtue of being an aggregate. It has to be proven, per region, before it leaves.



A cross-region aggregate fans out to every region; each region decrypts locally under its own key and computes its contribution, then a statistical-disclosure-control gate — a k-anonymity floor of at least 10 distinct data subjects — decides whether that contribution may leave. A region below the floor returns SUPPRESSED, never a value; the coordinator merges only the regions that passed.

Kilimanjaro proves it with statistical disclosure control, applied inside the region, before anything crosses the boundary. Each region decrypts its own contribution under its own key — the key never leaves — computes the local aggregate, and checks it against a k-anonymity floor counted over distinct data subjects, not rows: by default, at least ten, configurable upward — lowering it below that default is a separately gated, audited exception with a hard floor of five, never a routine clearance bypass.

A region whose cohort falls short returns a first-class result state, SUPPRESSED, rather than a value, an error, or a zero — the region knows its own subject count and withholds before the coordinator ever sees a number. The coordinator additionally guards against reconstruction: it rejects query pairs that differ by a single record (the classic “total minus everyone-but-X” attack), and it never backs out a suppressed region’s value by subtracting the reported regions from a grand total.

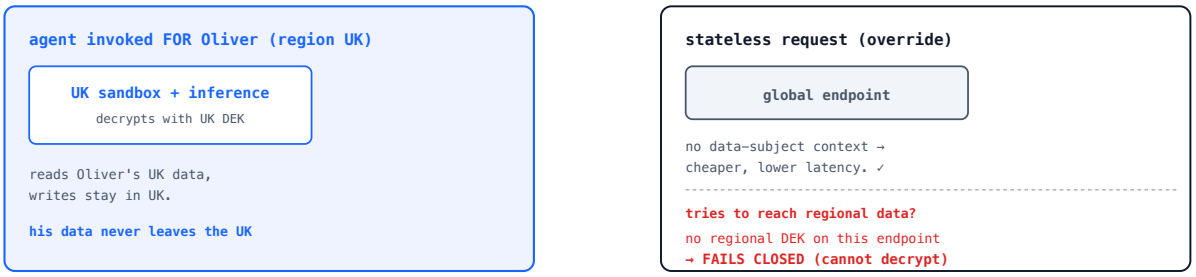
What crosses a region boundary, then, is never a raw record and never an unproven aggregate — it is a value that has already cleared a disclosure-control gate inside the region that computed it.

The coordinator merges only the regions that passed, using the standard composable operations (sums and counts add; averages divide; top-N re-ranks; cardinality merges via mergeable sketches), and reports coverage explicitly: if a region is down or suppressed, the result says so — “N of M regions reported” — rather than presenting a partial number as the complete one. A raw, row-level listing or join across regions is deliberately not offered on this path; it is paged per region instead, because there is no disclosure-control gate that makes a raw row non-personal.

06 In-jurisdiction AI: compute follows the data, and fails closed when it can't

The same principle that governs data governs compute: an agent invoked on behalf of a specific user runs in that user's region, against that user's in-region data, decrypting with that region's key — the inference endpoint and the extension sandbox are regional infrastructure, not a single global pool that happens to fetch regional data over the wire. A cross-user batch job ("summarize every customer") inherits the same scatter-and-merge discipline as a cross-region query.

Not every request needs a data subject's context. A stateless call — translate this pasted paragraph, summarize this inline text — can opt into a global compute endpoint for lower cost and latency, at the request level or as a project default. The important property is what happens if a stateless request nonetheless reaches for regional data: the global endpoint was never issued a regional key, so it cannot decrypt what it fetches.



An agent invoked for a user executes in that user's region, decrypting with that region's key. A stateless request may run on a cheaper global endpoint, but if it ever reaches for regional data, there is no regional key on that endpoint to decrypt it with — it fails closed.

The request fails closed, not because a policy check caught it, but because the key it would need does not exist where it is running. The override is a routing and cost decision; it is not a way to route around the residency guarantee, because the guarantee does not depend on the override being configured correctly.

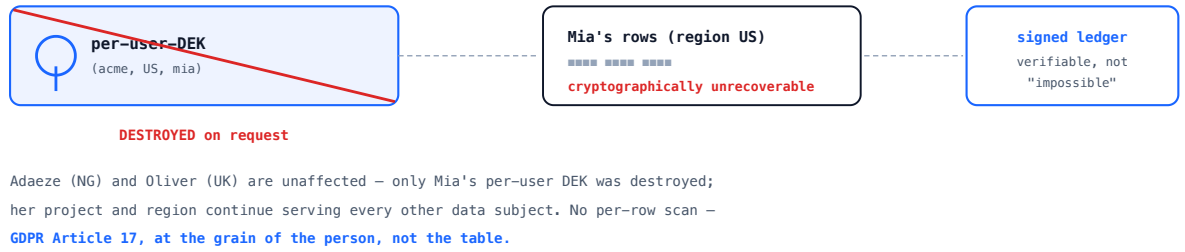
This is also where Kilimanjaro's two sovereignty stories meet without either one borrowing the other's strength. Keep's Enforced Boundary — the Warden as the agent runtime's sole mutation egress — continues to apply inside whichever region the agent is running

in; residency decides where the sandbox runs, Keep decides what it is allowed to do once it’s there. Composing the two is additive, and neither claim is stretched to cover the other.

07 Per-subject cryptographic erasure: the Article 17 answer at the grain of the person

A right-to-erasure request under GDPR Article 17, or its equivalents elsewhere, asks a database question that is hard to answer cleanly at scale: destroy a person's protected data — including from backups — without touching anyone else's data or scanning the whole table to find their rows. Per-user residency answers it, for the data encrypted under that person's key, at the same grain the key hierarchy already operates on.

Because each data subject has their own key, erasing a subject is destroying that one key — nothing else. Every field encrypted under it, in the live store and in any backup, becomes permanently unrecoverable the moment the key is gone; no other data subject's key, region, or rows are touched, and no per-row scan across the schema is required to find the fields encrypted under that key. The event is recorded on the signed audit ledger, so the erasure is not merely asserted — it is a claim a customer or a regulator can independently verify.



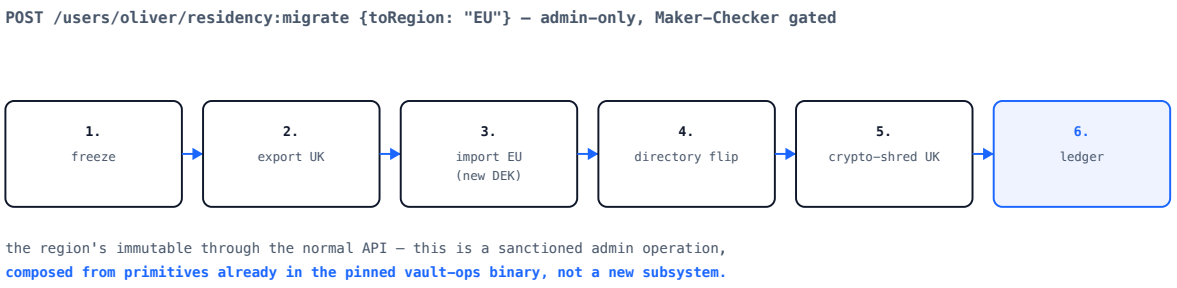
Destroying one data subject's key renders the data encrypted under it — their designated encrypted fields and tokenized values, in their region — cryptographically unrecoverable, verifiable through the signed audit ledger; their remaining, non-designated columns are erased by deletion in the same governed operation, with no effect on any other data subject.

The precise framing matters here more than anywhere else in this paper: the guarantee is "cryptographically unrecoverable, and verifiable," not "impossible to recover under any conceivable condition." The strength on offer is a provable per-subject crypto-shred with a signed receipt — and that is the guarantee the regulation is actually asking for.

08 Residency lifecycle: strict by default, migration as a governed exception

A user's region is resolved once, at creation, against the project's declared residency set — and the default policy is strict and fail-closed: a creation request for a region outside the set is rejected outright, never silently placed in the nearest available region. A silent fallback is exactly the failure mode a residency guarantee exists to prevent — it would place precisely the users the feature protects in the wrong jurisdiction. A project may instead declare an explicit fallback region as a deliberate, on-the-record compliance decision; it is never a hidden default.

Once assigned, a user's region does not change through the ordinary API — which removes an entire category of staleness and consistency question before it can arise. The occasional real need to move a subject (a user relocates jurisdictions) is handled as a sanctioned, admin-only operation, gated by Maker-Checker approval and run in a scheduled window: freeze writes for the subject, export their data from the source region, import and re-encrypt it under a fresh key in the target region, flip the routing directory atomically with the import, and cryptographically shred the source copy — with a signed event recording who approved the move and when.



An admin-gated, Maker-Checker-approved migration: freeze writes, export the subject's data from the source region, import and re-key it in the target region, flip the routing directory atomically with the import, crypto-shred the source, and record a signed event.

Nothing about this operation is new machinery bolted on for residency; it composes primitives — export, import, key rotation, crypto-shred — that already exist in the platform's custody model, at the grain of a single data subject rather than a whole project.

09 Worked example: a day in the life of a four-region fintech

acme-fintech is a single Kilimanjaro project with a declared residency set of Nigeria, the UK, the EU, and the US, and a strict residency policy. Adaeze signs up and is pinned to Nigeria: a directory entry, a key provisioned in Nigeria, and her rows created only in Nigeria's database. Oliver and Mia sign up the same way, pinned to the UK and the US respectively.

Adaeze places an order. The write routes to Nigeria, is encrypted under her Nigeria key, and is stored there — it never touches another region. Later, acme's operations team asks for total order volume across all four regions for the month: the query fans out, each region decrypts and sums locally under its own key, each region's result clears the disclosure-control gate before it leaves (a region with too few active users that month returns SUPPRESSED rather than a number), and the coordinator merges the regions that passed into one non-personal total.



acme-fintech declares a residency set of Nigeria, the UK, the EU, and the US. Users are created, write, get aggregated across regions, get served by an in-region agent, and — on request — erased, all inside one project.

Separately, an agent drafts a support reply for Oliver — it runs in the UK, reading his UK data with his UK key; his data never leaves the UK to do it. And when Mia later requests that her account be deleted, her per-user key is destroyed and her rows deleted: the data encrypted under her key becomes cryptographically unrecoverable and the rest is removed by deletion, the event is recorded on the signed ledger, and nothing about Adaeze's or Oliver's data or region is affected. One project answered all five operations without acme ever building a residency router of its own.

What we disclose

The residency guarantee is precise, and we are precise about its edges. What crosses a region boundary is never raw personal data — only an aggregate that has cleared statistical disclosure control inside the region that computed it: a k-anonymity floor counted over distinct data subjects, not rows, defaulting to at least ten and configurable upward — it is never presented as a per-operator dial, and lowering it below the default is a separately gated, audited exception with a hard floor of five, not a routine clearance bypass — a differencing guard against reconstruction, and explicit partial-coverage reporting rather than a silently incomplete total presented as whole.

We do not say aggregates "freely cross regions" — they cross only once proven non-personal, and a raw cross-region listing or join is deliberately paged per region rather than offered as a free global query, because no disclosure-control gate makes a raw row non-personal.

Per-subject erasure is described exactly as strong as it is: destroying a data subject's key renders the data encrypted under it — their designated encrypted fields and tokenized values — cryptographically unrecoverable, verifiable through a signed audit-ledger record; their remaining, non-designated columns are erased by deletion in the same governed operation, not by the key-shred. We do not claim the data is impossible to recover under every conceivable condition — the guarantee is a provable crypto-shred at the grain of the person, with a receipt, and that is the strength GDPR Article 17 and its analogues actually ask for.

This claim scopes to data encrypted under the per-user key. An artifact an extension derives from a subject's data and stores under the project's own key — embeddings or excerpts a search or retrieval extension writes, for example — is not reached by destroying that one subject's key; it is erased through that extension's own data-handling path, not this per-subject crypto-shred.

The region key hierarchy described here composes with whichever custody tier a project runs on. The hardware and attestation properties of each tier are set out in the Sovereign Key Custody paper.

Two capabilities in this paper keep the limits set out for those systems elsewhere: an in-region Keep deployment inherits Keep's own by-construction posture, and residency migration is an administrator operation, not a self-service one. Cross-region joins beyond the paged model are not offered.

Map your jurisdictions to a residency set

Talk to a Sovereign Engineer about the regions your users are in, the frameworks that govern them, and how a residencySet maps onto your project.

Talk to a Sovereign Engineer →

Browse the compliance frameworks