DPDPA consent is now an engineering problem: building a 'Consent-as-Code' control plane

Thought Leadership • February 18, 2026

First published by Lexology.

By: Gurjot Singh

With India's DPDP Rules notified in November 2025, organisations need more than UI banners they need demonstrable, machine-verifiable consent that binds notice, purpose and processing across distributed systems.

Note: This article is for general information and discussion purposes only and does not constitute legal advice.

Key takeaways

- Treat consent as a versioned policy artifact (not a one-time UI event).
- Decouple 'notice' from product pages by publishing a standalone, immutable Notice Contract that can be referenced and audited.
- Generate consent receipts as evidence objects; store them in an append-only ledger and link them to purpose, data categories and processing flows.
- Enforce consent at runtime using policy decision / enforcement points so every API call is purpose-bound.
- Make withdrawal propagation measurable: event-driven revocation + acknowledgements +reconciliation to prove completeness.

Why DPDPA makes consent a control-plane issue

India's Digital Personal Data Protection Act, 2023 requires consent to be free, specific, informed,unconditional and unambiguous, expressed through a clear affirmative action. In practice, that means consent has to be purpose-specific and demonstrable, not implied or bundled.

Separately, the DPDP Rules, 2025 were notified on 14 November 2025, operationalising the law and raising expectations around clear notices, minimisation and user control mechanisms.

Taken together, these requirements push organisations toward a system where consent can be(a) proven later, (b) enforced consistently across microservices, vendors and data stores, and (c) withdrawn with predictable effect.

The failure pattern: 'consent captured', but not enforceable

Many organisations can show that a user clicked 'I agree' at some point. What they struggle to show is that the click actually constrained downstream processing. Common gaps include:

- Notice drift: the words users saw have changed, but systems cannot prove which version applied.
- Purpose sprawl: consent captured for one purpose is reused for adjacent purposes ('function creep').
- Shadow copies: data replicated into analytics, logs or third-party systems that are not connected to consent state.
- Withdrawal friction: revocation exists in policy, but the product makes it hard, or systems keep processing due to propagation delays.
- Evidence gaps: no durable receipt tying identity, notice version, purpose and time to the processing event that occurred later.

These gaps are not primarily legal they are architectural. They arise when consent is treated as a front-end artefact rather than a distributed control-plane signal.

A practical operating model: Consent-as-Code

Consent-as-Code is an implementation pattern where consent is represented as machine readable, versioned policy that can be evaluated at runtime and audited later. The goal is not to replace legal wording; it is to make legal meaning enforceable in systems.

At a minimum, a Consent-as-Code model defines:

1. A stable notice object (what was told),
2. A consent state object (what was agreed),
3. A runtime evaluation path (how enforcement happens), and
4. An evidence trail (how proof is produced).

Designing the 'Consent Fabric' control plane

A consent program becomes scalable when you separate consent management into a shared service a control plane that every product and data flow consults. A practical reference architecture has five layers:

1. Notice Contracts (publish layer)
2. Consent Service (capture + lifecycle layer)
3. Evidence Ledger (proof layer)
4. Policy Runtime (decision + enforcement layer)
5. Revocation Propagation (withdrawal + reconciliation layer)

1) Notice Contracts: make notice standalone and verifiable

Treat each notice as a first-class artefact with its own lifecycle. Instead of embedding notice text in UI screens (where it drifts), publish a Notice Contract with:

- A unique identifier (notice_id)
- Version and effective dates
- Purposes covered (purpose_id list)
- Data categories per purpose
- Processing context (controllers/processors, cross-border transfers if applicable)
- Language/locale variants
- Hash/signature to make the content tamper-evident

Your UI and APIs then reference notice_id + version, ensuring you can always reproduce 'what the user saw'.

2) Consent receipts: generate durable proof objects

When a user gives consent, generate a receipt that can be stored and later produced for audits,disputes or regulator inquiries. A strong receipt minimally includes:

- data_principal identifier reference (pseudonymous if possible)
- notice_id + version
- purpose_id(s) consented to
- time stamp + channel (web/app/call centre)
- affirmative action metadata (e.g., checkbox toggle, digital signature, OTP proof whererelevant)
- scope constraints (time-bound, region-bound, feature-bound) if used

Receipts should be append-only: never overwrite. Instead, store lifecycle events (grant, update,withdraw) as separate entries.

3) Consent lifecycle as a state machine

Model consent as a lifecycle with explicit events so systems can reason about what changed and when. Common events include:

- CONSENT_GRANTED (for purpose_id set)
- CONSENT_UPDATED (add/remove purposes, change scope)
- CONSENT_WITHDRAWN (for purpose_id set)
- CONSENT_EXPIRED (time-bound consent)
- CONSENT_RECONCILED (post-propagation verification)

This is not bureaucracy: it enables downstream services to subscribe to changes and apply them deterministically.

4) Runtime enforcement: purpose-binding at every decision point

To make consent enforceable, every processing action should declare its purpose. Then a policy run time can decide whether that purpose is permitted for that user and data category at that moment.

In practice, organisations implement a Policy Decision Point (PDP) and Policy EnforcementPoints (PEPs):

- PDP evaluates: user consent state + purpose + data category + context (e.g., child flag,region, processor) ? allow/deny/allow-with-constraints.
- PEP sits in gateways/services/jobs and blocks or filters data access if PDP denies.

This approach prevents 'consent sprawl' because purpose becomes a required parameter not an after thought.

5) Withdrawal propagation: make revocation provable, not just requested

DPDPA expects withdrawal to be supported; the engineering challenge is ensuring that revocation reaches every downstream copy of data and processing pipeline.

A pragmatic pattern is event-driven revocation with acknowledgements:

- Emit a WITHDRAWAL event scoped to purpose_id(s).
- Each downstream system consumes the event and responds with an ACK once it has stopped the relevant processing and applied retention rules.
- Track outstanding ACKs; escalate failures; reconcile periodically (e.g., compare ledger state vs downstream attestations).

The key is measurable completeness: you should be able to answer 'which systems have applied this withdrawal?' with evidence.

Where Consent Managers fit

DPDPA introduces the concept of a 'Consent Manager' a registered entity enabling individuals to give, manage, review and withdraw consent through an accessible and interoperable platform.

Even if you do not integrate with an external Consent Manager on day one, you should design your Consent Fabric so it can ingest and honour consent signals from such interoperable platforms without bespoke engineering each time.

Implementation roadmap: 30-60-90 day plan

Day 0-30: Inventory and stabilise

- Enumerate purposes (purpose taxonomy) and map them to products, data stores andprocessors.
- Identify where consent is captured today and where it is enforced (usually nowhere).
- Draft your first Notice Contracts for the highest-risk processing (marketing, profiling, crosscontext analytics).

Day 31-60: Build proof + enforcement

- Implement a central consent service with consent receipts and an append-only event store.
- Integrate a PDP with one or two high-traffic PEPs (API gateway + one data pipeline).
- Instrument audit logs so every decision is traceable to notice_id/version + purpose_id.

Day 61-90: Make withdrawal measurable

- Add event-driven withdrawal propagation and ACK tracking for priority systems.
- Implement reconciliation: dashboards showing propagation completeness, lag and failures.
- Extend to major processors/vendors and establish contractual obligations for ACK/attestation.

A compliance-ready checklist for engineering and privacy teams

Before you claim 'DPDPA-ready', pressure test your consent program with these questions:

- Can you reproduce the exact notice text a user saw on a given date (versioned, tamper evident)?
- Is every processing action purpose-declared and checked at runtime (PDP/PEP)?
- Do you generate immutable consent receipts and lifecycle events (grant/update/withdraw)?
- Can you show, with evidence, that withdrawal has propagated to each downstream system (ACK/reconciliation)?
- Have you minimised data per purpose and prevented function creep through technical controls (not just policy)?

Closing thought

Consent under DPDPA is no longer a "policy page + checkbox" exercise. Organisations that treat consent as code a control plane signal with proof, enforcement and measurable withdrawal will be better positioned to demonstrate compliance, reduce operational risk and earn user trust.

This article was co-authored by Prabal Pathak, Legal Tech Expert