Vishwaas.ai

Non-Repudiation in Consent Management

How Vishwaas AI Makes Every Consent Decision Cryptographically Provable

Executive Summary

India's Digital Personal Data Protection Act, 2023 creates a new class of legal risk for every Data Fiduciary: the burden of proof is on the organisation, not on the data principal. When a data principal claims they never gave consent, or that their withdrawal was ignored, the organisation must produce incontrovertible evidence to the contrary — or face penalties of up to ₹250 crore per violation.

A database row with a consented = true flag and a timestamp is not evidence. It can be edited. It can be back-dated. It tells a court nothing about what the user was shown, what they agreed to, or whether the record has been altered since it was created.

Non-repudiation is the cryptographic property that makes a record impossible to deny. Vishwaas AI builds non-repudiation into every consent event through four independent, interlocking mechanisms:

- SHA-256 Hash Chain — links every consent record to every other, making insertion, deletion, and modification detectable
- RSA-2048 Digital Signature — binds each record to the organisation's cryptographic identity
- RFC 3161 Timestamp Authority (TSA) Token — provides a court-admissible, third-party-verified timestamp
- DB-Level Append-Only Enforcement — prevents any mutation at the database layer, not just the application layer

Together, these four mechanisms produce a consent ledger that is tamper-evident, independently verifiable, legally defensible, and retained for 7 years as required by DPDP Rules 2025 Rule 4.

1. The Legal Problem: Why a Database Timestamp Is Not Enough

1.1 What DPDP Act Requires

DPDP Act Section 6 requires consent to be "free, specific, informed, unconditional and unambiguous." DPDP Rules 2025 Rule 4 requires that a Consent Manager maintain consent artifacts for 7 years and make them available on request to the data principal and the Data Protection Board of India (DPBI).

The phrase "consent artifact" is significant. An artifact is a durable, verifiable record — not just a database entry. It must answer four questions with certainty:

Question	What Must Be Proved
Who consented?	The identity of the data principal
What they consented to	The exact text of the purpose, notice, and conditions shown to them
When they consented	The precise timestamp — to the second
Whether the record is authentic	That it has not been altered since creation

A conventional database answers the first three questions adequately. It cannot answer the fourth without cryptographic proof.

1.2 The Adversarial Scenario

Consider a data principal filing a complaint with the DPBI: "Acme Corp sent me marketing emails but I never gave consent."

Acme produces a database record: { principal_id: X, purpose: marketing_emails, action: granted, timestamp: 2025-06-01 }.

The DPBI asks: "How do we know this record was not inserted after the complaint was filed?" With a conventional database, there is no answer. A DBA with access could have created the record at any time.

With Vishwaas AI's non-repudiation chain, the answer is definitive: "This record's chain hash is embedded in all subsequent records, its RSA signature was created with the key held in AWS KMS at that moment, and the RFC 3161 TSA token was issued by a third-party timestamp authority that Acme has no ability to control or backdate."

2. Layer 1 — SHA-256 Hash Chain

2.1 What a Hash Chain Is

A hash chain is a sequence of records where each record contains a cryptographic fingerprint of itself and a reference to the fingerprint of the previous record. Any change to any record — however small — changes its fingerprint, which then invalidates every subsequent record in the chain.

This is the same mathematical principle used in Bitcoin's blockchain, applied to consent records.

2.2 How Vishwaas AI Implements the Hash Chain

Every consent record in Vishwaas AI carries two hash fields:

record_hash — the fingerprint of this record's own content:

```
record_hash = SHA-256(
  JSON.stringify({
    id, tenant_id, data_principal_id,
    purpose_code, action, status,
    channel, ip_address,
    consent_text_snapshot,
    previous_record_id,
    created_at
  }, { sortKeys: true })  // deterministic key ordering
)
```

The JSON is serialised with deterministic key ordering so that the same record always produces the same hash, regardless of the order properties are serialised by different JSON libraries.

chain_hash — links this record to the previous one:

chain_hash = SHA-256(record_hash + previous_chain_hash)
The chain_hash of record N depends on the chain_hash of record N-1, which depends on record N-2, all the way back to the genesis record.

2.3 The Genesis Record

The very first consent record in a tenant's chain cannot reference a previous chain hash. Instead, it uses a deterministic genesis anchor:

genesis_chain_hash = SHA-256("VISHWAAS_AI_GENESIS_" + tenant_id)
The tenant ID is a UUID assigned at onboarding and never changes. This binds the genesis of the chain to the specific tenant — meaning a chain cannot be transplanted from one tenant's database to another.

2.4 What Tampering Looks Like

Suppose an attacker with database access tries to change a consent record from GRANTED to WITHDRAWN to conceal marketing activity:

Before tampering:

Record 27: action=GRANTED, record_hash=a3f9..., chain_hash=7bc2...
Record 28: action=GRANTED, record_hash=d4e1..., chain_hash=9fa3...
(chain_hash computed using Record 27's chain_hash)

After tampering:

Record 27: action=WITHDRAWN, record_hash=??new??, chain_hash=??new??
Record 28: chain_hash=9fa3... ← still references Record 27's OLD chain_hash
MISMATCH — chain is broken

Any chain verification run after the tamper will fail at Record 28. The attacker would need to recompute every chain hash from Record 27 onwards — but they would also need to re-sign every record with the RSA key (Layer 2) and obtain new TSA tokens (Layer 3), which they cannot do.

2.5 Chain Verification API

Vishwaas AI exposes a verification endpoint:

```
POST /api/v1/consent/verify-integrity/:id
→ {
   "valid": true,
   "record_id": "...",
   "chain_position": 44,
   "chain_intact": true,
   "verified_at": "2026-03-19T10:30:00Z"
 }
```

The DPO or an auditor can trigger verification for any record at any time. Verification iterates backwards from the specified record to the genesis, recomputing each chain hash and comparing it to the stored value. A single mismatch halts the verification and returns the position of the broken link.

3. Layer 2 — RSA-2048 Digital Signature

3.1 Why the Hash Chain Alone Is Not Sufficient

The hash chain proves that no record was altered after the **chain was established.** But it does not prove **who** created the records or that the chain itself was not fabricated wholesale by an attacker who recalculated all hashes.

Digital signatures solve this. A signature can only be created by the holder of the private key. If the signature on a consent record verifies against Acme Corporation's public key, then Acme's systems — and only Acme's systems — created that record.

3.2 How Vishwaas AI Signs Consent Records

Each customer tenant in Vishwaas AI has a dedicated RSA-2048 key pair. The private key is held in AWS Key Management Service (KMS) in ap-south-1 and never leaves KMS — signing operations are performed inside the HSM.

```
digital_signature = RSA-SHA256-Sign(
  data    = record_hash,        // the fingerprint of this specific record
  key     = tenant_kms_key_arn,  // tenant-specific KMS key; never extracted
  format  = base64              // stored in the consent record
)
```

The public key is stored in the tenants table and is freely available for verification. Any party — including the DPBI — can verify a signature:

```
valid = RSA-SHA256-Verify(
  data      = record_hash,
  signature = digital_signature,  // from the consent record
  publicKey = tenant_public_key   // from the tenants table
)
```

3.3 What the Signature Proves

A valid signature on a consent record proves:

- **Authenticity —** the record was created by Acme Corporation's Vishwaas AI system (only the holder of the KMS private key could have produced this signature)
- **Integrity —** the record_hash has not changed since signing (any change to the record would change the hash, invalidating the signature)
- **Non-repudiation —** Acme cannot deny creating this record without also denying ownership of their own KMS key

3.4 Key Rotation

WS KMS supports automatic annual key rotation. When a key is rotated, new consent records are signed with the new key material. Old records remain verifiable because KMS retains all historical key material and selects the correct version based on the key version ID embedded in the digital_signature field.

4. Layer 3 — RFC 3161 Timestamp Authority Token

4.1 The Problem of Self-Reported Timestamps

The database created_at timestamp on a consent record is set by Acme's own servers. A court cannot rule out the possibility that a server's clock was manipulated, or that a record was backdated.

The RFC 3161 Timestamp Authority (TSA) protocol solves this by involving a trusted third party who cannot be controlled by Acme.

4.2 How RFC 3161 Works

Step 1: Vishwaas AI computes a hash of the consent record

```
timestamp_request_hash = SHA-256(record_hash)
```

Step 2: Vishwaas AI sends a TimeStampRequest to a trusted TSA (e.g. DigiCert TSA, GlobalSign TSA)

```
request = { hash: timestamp_request_hash, policy: ..., nonce: random }
```

Step 3: TSA signs a TimeStampToken with its own private key:

```
tsa_token = {
  version: 1,
  policy: TSA_POLICY_OID,
  messageImprint: { hashAlgorithm: SHA-256, hashedMessage: <hash> },
  serialNumber: <unique>,
  genTime: "2026-03-15T14:22:37.000Z",  ← TSA's authoritative time
  tsa: <TSA distinguished name>
}
signed with TSA's X.509 certificate
```

Step 4: Vishwaas AI stores tsa_token (DER-encoded, base64) in the consent record

4.3 What the TSA Token Proves

The TSA token provides third-party, court-admissible proof that:

- The consent record (identified by its hash) existed at the time stated in the token
- The timestamp was issued by a trusted authority operating under a known certificate policy
- The timestamp cannot have been backdated because the TSA's clock is independently maintained and its tokens are publicly auditable

Even if Acme's entire infrastructure were compromised and all records backdated, the TSA tokens would remain valid only for the dates when they were actually issued — making the fabrication immediately detectable.

4.4 Legal Admissibility in India

RFC 3161 compliant timestamps are recognised under the Information Technology Act, 2000 as admissible electronic records when produced by a Certifying Authority. The TSA timestamp, combined with the RSA signature and hash chain, provides a consent record that satisfies the evidentiary standard for a DPBI adjudication or a civil court proceeding.

5. Layer 4 — DB-Level Append-Only Enforcement

5.1 Why Application-Layer Guards Are Not Enough

The first three layers (hash chain, signature, TSA token) are sufficient to detect tampering. Layer 4 prevents it.

An attacker with database access but without access to the KMS private key and TSA infrastructure cannot forge a convincing tampered record. But they could still delete records from the ledger or attempt to corrupt the chain, requiring legitimate administrators to re-run verification to detect the damage.

Vishwaas AI eliminates this attack surface by revoking mutation rights at the PostgreSQL database layer.

5.2 Implementation

At the database level, the application role crosstrust_app has the following permissions revoked on the consent records table and the audit schema:

```
-- Applied after every Prisma migration:
REVOKE UPDATE, DELETE ON app.consent_records FROM crosstrust_app;
GRANT  INSERT, SELECT  ON app.consent_records TO   crosstrust_app;

REVOKE UPDATE, DELETE ON audit.events FROM crosstrust_app;
GRANT  INSERT, SELECT  ON audit.events TO   crosstrust_app;
```

This means:

- The NestJS API, running as crosstrust_app, cannot issue UPDATE or DELETE SQL statements against these tables — the database will reject them with a permission error regardless of what the application code requests
- A developer who injects malicious SQL through the API cannot update or delete consent records
- Only a PostgreSQL superuser (a separate credential, held only by the DBA) can modify these tables — and such operations would be visible in the PostgreSQL server log

5.3 The Audit Schema

The audit.events table lives in a separate PostgreSQL schema (audit) distinct from the main app schema. This provides an additional isolation boundary:

app schema → application tables (consent_records, dpr_requests, etc.)
→ app role: full CRUD (except consent_records)

audit schema → audit.events, audit.chain_checkpoints
→ app role: INSERT + SELECT only
→ no ORM model for UPDATE/DELETE (Prisma schema excludes mutation operations)

Every mutation across all Vishwaas AI modules writes an event to audit.events. The audit events themselves are hash-chained using the same SHA-256 Merkle design as consent records — providing a tamper-evident log of all system activity, not just consent decisions.

6. The Consent Text Snapshot

Non-repudiation requires proving not just that consent was given, but what the data principal agreed to. Vishwaas AI solves this with the consent_text_snapshot field.

At the moment a consent record is created — whether through the portal, an API call, or a campaign response — the exact multilingual text that was rendered to the data principal is captured and stored immutably in the record:

{
"consent_text_snapshot": {
"en": "I agree that Acme Corporation may use my email address to send me marketing communications about products and services. I understand I can withdraw this consent at any time.",
"hi": "मैं सहमत हूँ कि Acme Corporation मेरे ईमेल पते का उपयोग मुझे उत्पादों और सेवाओं के बारे में मार्केटिंग संचार भेजने के लिए कर सकती है।"
},
"notice_version_id": "ntc_01HXYZ...",
"notice_content_hash": "a3f9b2..."
}

The notice_content_hash is the SHA-256 hash of the privacy notice version that was active at the time of consent. This allows any auditor to verify that the snapshot matches the notice that was published — or detect if the notice was subsequently altered.

This means Acme can answer the question "what exactly did the user agree to?" definitively — not just "they agreed to our privacy policy" but the exact words, in the exact language, linked to the specific version of the notice.

7. The Complete Non-Repudiation Picture

When Vishwaas AI produces a consent record as evidence, it delivers:

Consent Record #44 — Rahul Verma grants marketing_emails consent

id:	cns_01HXABCDEFGH...
data_principal_id:	dp_01HX123... (Rahul Verma)
purpose_code:	marketing_emails
action:	GRANTED
channel:	portal
ip_address:	49.36.xxx.xxx (encrypted at rest)
consent_text_snapshot:	{ "en": "I agree that Acme Corporation..." }
notice_version_id:	ntc_01HXYZ...

record_hash:	a3f9b2c1d4e5f6a7...	(SHA-256 of record content)
chain_hash:	7bc2d3e4f5a6b7c8...	(SHA-256 of record_hash + prev_chain_hash)
digital_signature:	MEQCIBn7r+...	(RSA-2048 / SHA-256, KMS key: arn:aws:kms:...)
tsa_token:	MIIHZgYJKoZIhvcN...	(RFC 3161, DigiCert TSA, 2026-03-15T14:22:37Z)

previous_record_id:	cns_01HXABCDEFGA... (Record #43 — chain linkage)
created_at:	2026-03-15T14:22:37.000Z

Each of the four proof layers answers a distinct legal challenge:

Challenge	Answered By	How
"We have no record of this consent"	Hash chain + DB append -only	Every subsequent record would be invalid without this one
"You could have created this record after the complaint"	RSA signature + TSA token	KMS sign time and TSA issuance time cannot be backdated
"The record has been altered"	SHA-256 record_hash + RSA signature	Any change invalidates both the hash and the signature
"We don't know what the user agreed to"	consent_text_snapshot	Exact text preserved immutably at the moment of consent
"The timestamp could be wrong"	RFC 3161 TSA token	Third-party timestamp authority; not controlled by Acme
"The ledger is incomplete — records are missing"	Hash chain	Missing records break the chain; detectable in O(n) verification

8. Consent Receipt — The Data Principal's Copy

DPDP Rules 2025 Rule 4 contemplates that data principals should be able to obtain a record of their own consent decisions. Vishwaas AI generates a Consent Receipt PDF on demand from the data principal's portal:

CONSENT RECEIPT

Issued by: Acme Corporation (acme-corp.vishwaasai.in)
Issued to: Rahul Verma <rahul.verma@example.test>
Date & Time: 15 March 2026, 2:22 PM IST (RFC 3161 verified)

You granted consent for: Marketing Emails
"I agree that Acme Corporation may use my email address..."

Record ID: cns_01HXABCDEFGH
Record Hash: a3f9b2c1d4e5f6a7...
Signature: Valid (Acme Corporation — RSA-2048)
TSA Verified: Yes (DigiCert, 2026-03-15T14:22:37Z)

The receipt is itself digitally signed and RFC 3161 timestamped at generation time. It can be presented by the data principal as evidence to the DPBI that consent was (or was not) recorded.

9. Audit Trail Non-Repudiation

The same four-layer design extends to the system-wide audit trail. Every administrative action — DPO approving a DPIA, a Grievance Officer completing a DPR, an IT Admin registering a webhook — writes an event to audit.events with:

- A record_hash of the event content
- A chain_hash linking to the previous audit event
- The actor's user_id and role embedded in the event
- The IP address and timestamp
- The audit chain can be verified end-to-end via:

The audit chain can be verified end-to-end via:

GET /api/v1/audit/verify-chain?from=2026-01-01&to=2026-03-31
→ { "valid": true, "events_checked": 2847, "chain_intact": true }

A DPBI inspector or an external auditor can verify the complete integrity of all system activity for any date range in a single API call.

10. Comparison: Vishwaas AI vs. Conventional Consent Databases

Capability	Conventional DB	Vishwaas AI
Consent recorded	✅	✅
Timestamp stored	✅	✅
Text of what user agreed to	❌ (reference only)	✅ (exact snapshot, immutable)
Tamper detection	❌	✅ SHA-256 hash chain
Proof of creator identity	❌	✅ RSA-2048 digital signature
Third-party verified timestamp	❌	✅ RFC 3161 TSA token
DB-level mutation prevention	❌	✅ REVOKE UPDATE, DELETE
Independent chain verifiability	❌	✅ Anyone with public key can verify
7-year retention enforcement	Manual policy	✅ Automated; immutable
Consent receipt for principal	❌	✅ Signed PDF on demand
Audit trail non-repudiation	❌	✅ Same hash chain design
DPBI-ready evidence package	❌	✅ One-click signed PDF export

Conclusion

Vishwaas AI's non-repudiation architecture transforms consent management from a record-keeping exercise into a cryptographic proof system. The four-layer design — SHA-256 hash chain, RSA-2048 digital signatures, RFC 3161 TSA tokens, and DB-level append-only enforcement — ensures that every consent decision is provable, every audit trail is tamper-evident, and every data principal's consent receipt is legally defensible.

For Data Fiduciaries operating under the DPDP Act, this is not a feature — it is the difference between being able to demonstrate compliance and being exposed to ₹250 crore in penalties when a data principal or the DPBI asks the question: "Prove it."

Contact us

+1 888 208 5076 / +91 901 926 6824

sales@crossidentity.com

www.crossidentity.com

Vishwaas.ai