LITE PAPER

ZOMBIE DELETE

Cryptographically Verifiable Deletion Receipts for Data Deletion Assurance

Proof that a record is gone from every declared datastore, and a guard that keeps it gone.

LITE PAPER · VERSION 2 · OCTOBER 2026

Glen Cameron and Dr Stef Savanah

Together Alone Ventures OÜ, Tallinn

zombiedelete.com · verifier.zombiedelete.com · info@together-alone.com

© 2026 Together Alone Ventures OÜ. All rights reserved. This paper describes a technical protocol. It is not legal advice.

Abstract. Cryptographically Verifiable Deletion Receipts (CVDRs) are introduced as evidence artefacts for data deletion. The underlying algorithms enable mathematical verification of any type of computer system state transition, of which a database change (add, modify, delete) represents only one kind. Hashing, Merkle Trees and cryptographic signatures create the CVDRs, which can then be independently verified using zero-knowledge proofs or other cryptographic methods depending on the configuration. The entire process is transparent, so anyone can validate the procedure. This lite paper condenses the Zombie Delete Technical White Paper, version 2.

1. THE PROBLEM

Organisations are increasingly expected not only to destroy data, but to show that they did. A contract ends and the supplier must return or destroy the customer's data. A data room closes. A litigation hold is released. A retention period expires. A licence to training data ends. A person exercises the right to erasure. Most organisations answer with internal logs, screenshots, administrator attestations, signed delete commands or certificates of destruction. These records show who carried out the deletion. None of them shows that the data is gone. Each asks the regulator, auditor, customer or counterparty to trust the organisation that did the deleting.

2. WHAT ZOMBIE DELETE DOES

Zombie Delete (ZD) introduces the Cryptographically Verifiable Deletion Receipt (CVDR): formal evidence of deletion that an independent party can verify. Zombie Delete can also help keep the defined deleted state in place: where tombstoning is used, a write guard blocks restoration under the same identifier on the configured write paths.

The underlying algorithm is broader than deletion. Zombie Delete proves any defined state transition, of which deletion is only one type. The receipt proves that a particular state existed before the transition (the pre-state) and that a defined transition to the post-state occurred, without exposing the data.

This paper uses one familiar example throughout: erasure of a customer profile record under GDPR Article 17. The receipt proves the record was deleted without revealing any personal data; only a record_id identifies the target, itself cryptographically obscured where necessary. We chose the example for clarity, not because privacy law is the main application. Many obligations to destroy data come from contracts, commerce or security.

Scope. Zombie Delete makes no claims about copies of target data stored elsewhere, whether surreptitiously or by backup processes. ZD applies to specified datastores, including backups if the enterprise stipulates them.

Don't trust, verify. ZD logic is completely transparent, so anyone with a copy of a CVDR and publicly available resources can prove its validity from first principles. TAV publishes full technical details, verification instructions and reference software. These materials are not meant to be trusted; a skilled person can confirm them or build an independent verifier. Verification needs only the receipt, the published procedure and the relevant trust material; it does not depend on any TAV service. A convenience instance of the verifier runs at verifier.zombiedelete.com.

3. THE FUNDAMENTAL MODEL

State transition

In its most general form, Zombie Delete records any defined transition between two system states. The simplest example is a database modified by a record addition, deletion or update. ZD does not take snapshots of entire systems. It uses cryptographic signatures, hash commitments, Merkle Trees and zero-knowledge proofs to capture the transition in one mathematical artefact, the CVDR, from which the transition can be verified later.

A customer who asks for their personal data to be deleted receives a CVDR as proof that it occurred. The enterprise keeps a copy of every receipt for its own auditors. Depending on the enterprise's requirements, the target record is either physically deleted or replaced with a tombstone that blocks re-insertion: every field except the record_id is overwritten with protocol-defined null values.

What a CVDR contains

A CVDR is a series of numbers, mostly in hex, stored as a JSON file. Its components vary with the use case, the enterprise set-up and the proof model. In general a CVDR includes some or all of the following.

Component	Description
Metadata	Protocol version, a unique CVDR number, a database identifier, a timestamp.
Target binding	A unique identifier for the transition element: a record_id, lookup token, document filename or object identifier. Where the identifier itself reveals sensitive data, a salted hash or equivalent privacy-preserving value may be used.
Pre-state and post-state commitments	Cryptographic hashes or other evidence capturing the relevant states before and after the transition: serialised state hashes, Merkle roots, authenticated state proofs or equivalent commitments.
Transition event commitment	A commitment to the specific transition, binding the target, the pre- and post-transition evidence and other protocol-defined event data. It identifies what transition occurred, independently of how that claim is later authenticated.
Evidence-generation provenance	Evidence identifying the process that created the CVDR: source or adapter provenance, a module hash or other code commitment, a TEE measurement, a ZK circuit or verifying-key identifier, or a combination.
Authentication evidence and authority	Evidence authenticating the transition claim: a digital signature, blockchain certificate, TEE attestation, zero-knowledge proof or equivalent.

Preventing re-ingestion

Deleting a record once is not enough if the record can come back. Records return through backup restores, data syncs, migrations, cached exports and re-imports. Zombie Delete therefore treats the deleted state as a state to hold, not only a state to reach. Where the enterprise uses tombstoning, every installation that claims protection against re-ingestion includes a write guard as a required step. The guard rejects any write that would restore data under a tombstoned record_id. In web2 configurations the guard is a database-level control such as a trigger. On ICP the canister code enforces it.

Three limits apply. First, a CVDR proves the transition at the time it occurred; it does not by itself prove that the guard is still in place later. Second, the guard blocks re-ingestion under the same identifier only; a record created under a new identifier is outside it. Third, the guard covers only the write paths registered in the installation.

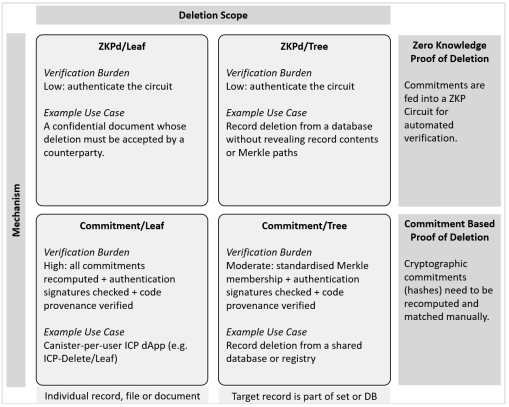
Backup restore needs its own treatment. A restore can overwrite whole tables or databases and reintroduce pre-deletion state. The write guard is an enterprise-implemented control, so keeping tombstones effective across a restore is the enterprise's operational duty: either the tombstones are re-applied before the restored store rejoins the declared scope, or the restore path itself is gated. The applicable Residual Trust Statement records this.

4. PROOF MODELS

The proof models are mapped along two dimensions, scope and mechanism.

Scope is the form of the data target: a stand-alone item such as a file or document, or a member of a set such as a record in a database table. The set case is managed with a Sparse Merkle Tree, which lets the absence or presence of a record be determined efficiently even in very large databases: the Tree model. Individual items use the same commitment techniques without a tree: the Leaf model.

Mechanism is the cryptographic proving structure. Both types rely on hashes as fingerprints. Commitment-based verification comprises several explicit recomputation, authentication and provenance checks. ZKPd reduces the per-receipt check to standard proof verification and also hides the target data and Merkle paths.



Proof models: scope (Leaf, Tree) against mechanism (commitment, zero knowledge).

Commitment-based proofs

Fingerprints of the datastore's pre-state and post-state are captured so that a verifier can check them later. The aim is to prove that a specific target once did exist in the datastore and was transitioned to the protocol-defined deleted state, absence or a tombstone. A hash commits to an input while keeping it secret, so the CVDR can be verified without revealing the target data beyond its identifier. Because the input may itself be sensitive, ZD uses a double hash, and the inspectable certified code lets anyone confirm that the first hash derives from the record whose record_id the CVDR carries.

Zero Knowledge Proof of Delete (ZKPd)

A ZKP works on the same commitments. A ZKP circuit encapsulates the verification rules; the public inputs are the commitments, the private witness is the target data. The enterprise runs the circuit to produce a proof blob; the CVDR carries the proof blob, the public inputs and the verification key (vkey), which is all a verifier needs. The circuit is transparent and can be independently validated. The proof establishes that the circuit relation holds for its public inputs, not that the supplied state root was the datastore's real pre-state, so the state commitment must come from an authenticated or attested source.

5. VERIFICATION

Process validation

Zombie Delete code is open and transparent so anyone can check the proof models and the CVDR derivation process. The CVDR carries the attested identity of the installed code: for commitment-based proofs the module hash. A verifier rebuilds the binary from the published source and build path, hashes it, and matches the module hash in the CVDR. A public release history lets older receipts be re-verified at any future date. For ZKPd the vkey is the fingerprint of the proving configuration and is checked against the circuit source and compile path.

Individual receipt verification

Everything needed to establish the procedure is derivable from the source code. No trust in TAV is required; any residual trust in the evidence authority is configuration-dependent and stated in the Residual Trust Statement. A configuration claims a property described in this paper only where its Residual Trust Statement confirms the property holds for that deployment. The published documentation is complete enough for an independent verifier to execute the procedure from public materials alone; the reference software is a convenience, to be verified rather than trusted.

Two questions are separated. **Verification validity** asks whether the CVDR satisfies the cryptographic and protocol rules that apply to it. **Assurance grade** describes the strength and status of the evidence the receipt relies on, and its authentication authority. For commitment-based proofs the verifier recomputes every committed value, then checks the authentication evidence against the expected authority: a TEE, the blockchain's consensus, or the enterprise's signing key. For ZKPd the proof blob, public inputs and vkey go into a standard verification algorithm that answers accept or reject.

6. WHAT A RECEIPT SHOWS, AND WHAT IT DOES NOT

A CVDR shows	A CVDR does not show
• The target moved from its pre-state to the protocol-defined deleted state, in each declared datastore • When the transition happened, and in what order • The installed code identity attested for the finalisation window, by module hash linked to published source, or the proving configuration associated with the evidence • The authority that authenticated the claim: blockchain consensus, a TEE or the enterprise's own key	• Copies in datastores the enterprise did not declare, including backups unless declared • Copies made before the deletion, including anything a person or a model already read • That the system was not later restored from a snapshot or backup • That earlier byte values cannot be recovered from storage media • Removal from an immutable ledger's transaction history • That the same subject cannot return under a new identifier • That a court in any given jurisdiction will admit the receipt as evidence • Whether the declared datastores are the only datastores that hold the record • Output of a model that saw the record before deletion, including distillation output and synthetic data

Residual Trust Statement

Every Zombie Delete configuration relies on the integrity of hardware, software, signing authorities and protocols. The Residual Trust Statement (RTS) lists those dependencies so the enterprise knows exactly what it can claim. Even standardly trusted mechanisms are listed for completeness: a change-data-capture stream must relay transactions faithfully; TEE hardware must be sound; a blockchain's consensus must be honest. Each line carries its mitigation. In the ICP-Delete/Leaf example, the RTS records that subnet consensus is assumed honest, that the canister controller does not upgrade the canister to reverse tombstones (with the installed module identity attested during finalisation), that tombstoning is a logical overwrite and physical remanence is outside the protocol, that snapshots or backups could hold pre-deletion state, and that the write guard is assumed to remain deployed.

Receipt custody

A CVDR is useful only while it exists. The principle is the same in every configuration: no Zombie Delete process can change or delete an issued receipt, a tampered receipt fails verification, and there is no delete path for receipts. Storage is configuration-specific. In web2 the receipt archive sits outside the customer database that Zombie Delete modifies, and a replayed deletion that would produce different content under an existing receipt_id halts rather than overwrites. In ICP-Delete/Leaf the completed receipt is held in canister stable state. In every configuration the enterprise holds its receipts and must keep them inside its backup scope, because nobody can recreate a lost receipt. TAV holds no receipts and runs no transparency log. Where the enterprise wants an external anchor, it chooses the log or anchor itself.

Long-lived receipts

A receipt may need to hold for the retention period of the obligation it evidences. An issued receipt does not become invalid because a newer hash, signature or proof system is later adopted; it continues to be verified under the protocol_version that created it. If an underlying algorithm is later broken, the receipt's evidential assurance can weaken: a future attacker may forge a receipt or proof that still passes the old verifier. That is a soundness concern, not a disclosure concern, because the receipt never held the data. Where very long-term retention requires it, the receipt holder may separately use standard archival timestamping, such as an RFC 3161 token held in an RFC 4998 evidence record. The CVDR itself remains unchanged.

7. CONFIGURATIONS

Zombie Delete works across web2, web3 and hybrid installations. Configuration-dependent adapters, templates and reference code are provided with the core. Deployment type and proof model are orthogonal: any deployment may use any proof model.

Web2. Traditional datastores such as PostgreSQL and MongoDB, on the enterprise's own servers or in a cloud. Installation does not require existing enterprise code to be modified; ZD code is added, with database-level guards such as triggers. The tombstone blocks re-insertion of the original record and the guard rejects update attempts. A returning customer is on-boarded again under a new record_id.

Web3. The ICP-Delete/Leaf case study runs on the Internet Computer, a mutable blockchain, but Zombie Delete applies to immutable blockchains too. Modern state-machine blockchains keep an explicit current state beside the immutable history, and the transition between explicit states, including a deletion or tombstone, can be evidenced just as for a database. Proving removal from explicit state does not erase data recoverable from transaction history; the RTS says so. For the European view of erasure on blockchains, see the EDPB guidelines of July 2026.

Hybrids. A common design keeps plaintext in a web2 database and only a hash, pointer or permission record on-chain; the main CVDR concerns the database and a second piece of evidence shows the on-chain reference was revoked. Another encrypts data for an immutable chain and keeps the key elsewhere; destroying the key (crypto-shredding) is a before-shred to after-shred transition that Zombie Delete certifies like any other.

Chain Fusion. Internet Computer technology for interacting directly with other blockchains. Zombie Delete, running on ICP, does not execute the native-chain deletion; it verifies the target chain's own cryptographic evidence of the transition and packages the verified result into a CVDR, with no Zombie Delete logic on the native chain. Where a chain cannot provide a state proof, the RTS names the extra assumptions.

8. AI SYSTEMS

AI systems copy data as a matter of course. A source record is embedded into a vector store, indexed for search, written into agent memory and sometimes added to a training set. Deleting the source does not remove these derived copies, and a store's own delete function may not either. A 2026 study recovered soft-deleted embeddings from the raw index files of three HNSW vector databases and reconstructed from them 25.5 per cent of exact names and 46.4 per cent of locations. Zombie Delete applies to AI systems without any change to the model. It treats each derived store as a declared datastore in its own right.

Root-record deletion. The source record is the root from which every derived copy is built. Zombie Delete moves it to the deleted state, tombstones it and issues a CVDR for the source datastore.

Derived stores. The next rebuild of a connected search index, vector store or agent memory reads the tombstone in place of the record, so the rebuilt store does not contain it. Derived stores may need their own CVDRs. The evidence Zombie Delete can obtain is specific to the store: a soft delete at a store's interface is not evidence that the underlying index data is gone, so each configuration uses an adapter for that store class, and its RTS says what a receipt for that store shows. Each derived receipt is dated after the source receipt, and an auditor checks the set as a whole.

Re-ingestion. The tombstone stays at the source, so every later rebuild reads it again, and the write guard blocks any attempt to write the record back under its own identifier. The configuration follows the datastore: conventional databases use the web2 configuration, with receipts signed by the enterprise's deployment key; a derived store held in a canister on the Internet Computer uses the ICP configuration, with the subnet certifying the receipt.

Data starvation. A working model uses a specific record mainly through the stores it reads. With the root removed, each declared store returns nothing. Whatever the model's weights hold of that record is a fixed residue that no source refreshes. This limits how far the record spreads; it does not remove the residue.

What the receipts do not show. Zombie Delete does not claim that a trained model has forgotten anything. The receipts show three narrower things: the root record is gone from every declared store; derived stores rebuilt after the deletion do not contain it; guarded write paths reject its return. They say nothing about any training run; a receipt on the run itself is future work. In this paper, **unlearned** has two meanings only: the model version was trained without the record, which a receipt on the training run would show, or no declared source covered by the receipts remains from which the model can retrieve or refresh the record. It never means that a trained model was made to forget.

9. ROADMAP

The white paper describes the protocol and its configurations, not the release status of individual products. The work below extends the protocol.

- **ZKP Delete.** A full zero-knowledge implementation of the proof models. Circuits are written in Noir; its RTS will state the proving system and setup assumptions.
- **Zombie Protect.** Blocks protected identifiers from re-entering through configured ingestion paths: a registry of prohibited identifiers and a gate on the ingestion path, where an entry for data once held and destroyed cannot be told apart from one for data never held. A separate integration class. Provisional patent filed.
- **Zombie Lock.** A sealed state between keeping and destroying, for restrictions of processing, litigation holds and sanctions freezes: no complete decryption key exists, and the only way to create one is an authenticated transition that records the unsealing. TAV holds no key share. Provisional patent filed.
- **Receipts for AI training.** Corpus commitment, measured data starvation, and verified unlearning. Verified unlearning means the record was deleted from the declared stores, a CVDR proves it, the deleted state is held, and a later training run can be shown to have excluded the record. It is not a claim that a trained model's weights have forgotten the data.
- **Source coverage.** An enterprise-owned, versioned declared-store manifest, never held by TAV, with a reconciliation step against evidence independent of the deletion pipeline where available. Provisional patent filed.
- **Cryptographic agility.** Options under design for ZKPd: dual commitment under a long-lived hash and an in-circuit hash, and hash-based transparent proof systems.
- **Post-quantum signatures.** Hashes and Merkle commitments hold at this size; signatures do not. Under design: a web2 signing profile on ML-DSA (FIPS 204) or SLH-DSA (FIPS 205), stated per protocol_version. The ICP subnet certificate uses BLS; its migration is the network's, and the RTS will say so.
- **Also.** A receipt reference field binding derived receipts to their source; a re-ingestion alarm; published state roots as advance commitments; a TEE signing profile for web2; independent anchoring in a log or timestamp authority the enterprise chooses.

10. INTELLECTUAL PROPERTY

The Zombie Delete patent portfolio comprises 16 provisional patents filed, covering deletion receipts, the tombstone, source coverage, verified unlearning, Zombie Protect and Zombie Lock. The inventors are Glen Cameron and Dr Stef Savanah. The portfolio is assigned to Together Alone Ventures OÜ.

Verification depends on transparency. For each release TAV publishes the source a verifier needs, the build path and the verification reference software; for ZKPd, the circuit source and compile path. Publication for inspection is not an open-source licence and grants no right to use the patented methods.

11. TEAM

Glen Cameron, founder and CEO. Second-time blockchain founder; twenty years as a C-suite communications, strategy and PR executive. **Dr Stef Savanah**, co-founder and Chief Scientist. 192 granted patents, none in the Zombie Delete portfolio; formerly Director of Scientific Research at nChain; PhD in Cognitive Science. **Robin Davies**, COO (fractional). **Antoine Monesma**, Lead Developer, builds all Zombie Delete production code. **Vikram Seth**, Fractional Chief Partnership Officer, former Head of Web3 at Shell. **Zorana Smith**, Head of Sales. **Finja Schnelle**, Admin and Operations. Adviser: **Richard Slater**, former partner, Simmons & Simmons.

12. VERIFY, AND READ MORE

VERIFY A RECEIPT
verifier.zombiedelete.com

REFERENCE SOFTWARE AND FULL WHITE PAPER
info@together-alone.com

PRODUCT
zombiedelete.com

COMPANY
together-alone.com

Together Alone Ventures OÜ, trading as Zombie Delete. Company number 17288376. Tartu mnt 67/1-13b, Harju maakond, 10115 Tallinn, Estonia. This lite paper condenses the Zombie Delete Technical White Paper, version 2, October 2026, which carries the full verification procedure, the ICP-Delete/Leaf case study, the use-case catalogue and references. It describes a technical protocol. It is not legal advice. © 2026 Together Alone Ventures OÜ. All rights reserved.