TECHNICAL WHITE 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.

VERSION 2 · OCTOBER 2026

Glen Cameron and Dr Stef Savanah

Together Alone Ventures OÜ, Tallinn. October 2026

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. Various techniques such as hashing, Merkle Trees, and cryptographic signatures are used to 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 to allow anyone to validate the procedure.

1. INTRODUCTION

Organisations are increasingly expected not only to destroy data, but to show that they did. The duty arises in many settings. 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.

Zombie Delete (ZD) addresses that gap by introducing 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 (section 2).

The underlying algorithm is much broader than deletion and more powerful: Zombie Delete proves any defined state transition, of which deletion is only one type. In the general case, the issued 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 personally identifiable information (PII). Only a record_id identifies the target data, itself cryptographically obscured where necessary. This example also provides the case study in Appendix 1. We chose it for clarity, not because privacy law is the main application. Many obligations to destroy data come from contracts, commerce or security, and Appendix 2 sets out the wider range.

In the deletion use case, Zombie Delete makes no claims regarding copies of target data that may have been stored elsewhere, either surreptitiously or by back-up processes. ZD applies to specified datastores, including back-ups if stipulated by the enterprise.

Zombie Delete borrows the web3 maxim ‘Don’t trust, verify’. ZD logic is completely transparent, so given the requisite technical skills anyone with a copy of a CVDR and publicly available resources can prove its validity from first principles. To make this procedure more convenient we document full technical details, including verification instructions and verification reference software. To emphasise again, these materials are not meant to be trusted; any skilled person can analyse them and confirm their veracity, or they can derive the necessary algorithms from first principles and build their own verification process independently. 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. The reference software is available on request from info@together-alone.com.

2. THE FUNDAMENTAL MODEL

2.1 State Transition

In its most general form, Zombie Delete records any defined transition between two system states. The simplest example is when a database is modified by a record addition, deletion or update, but it could also include changes to system configurations, database triggers, or other environmental modifications. ZD, of course, does not take snapshots of entire systems; rather, it uses cryptographic techniques such as cryptographic signatures, hash commitments, Merkle Trees, and Zero Knowledge Proofs. The transition in question is captured succinctly in the form of a mathematical artefact, the CVDR, from which the transition can be verified at a later date.

For example, a customer of an enterprise who requests their personal information be deleted would receive a CVDR as proof that it occurred. With this CVDR they could personally verify the deletion or provide the artefact to a skilled verifier to perform the verification on their behalf.

1. Receive/detect/observe transition request	End-user deletes their profile in the application front-end. ZD intercepts the deletion request via DB hook.
2. Record target identifier	Record_id is saved as CVDR data.
3. Assemble CVDR metadata	Record protocol version number. Record timestamp. Derive a unique receipt_id.
4. Create commitment to pre-state	Create Sparse Merkle Tree (example process only): Derive salted hashes of all record_ids (determine SMT leaf positions) Replace target leaf value with Hash(salt, record_id, serialised record) Construct Sparse Merkle Tree Compute Merkle root Derive Merkle path for target record Create ZKP public inputs from Merkle root. Create ZKP private witness inputs from Target record and sibling path.
5. Execute state transition	Compute the canonical tombstone record. Commit the update.
6. Create commitment to relevant post-state	Replace target leaf value with Hash(salt, record_id, tombstone record) at same position. Recompute the hashes along the unchanged target Merkle path to obtain the post-deletion Merkle root. Create ZKP public inputs from new Merkle root.
7. Create deletion event commitment	Compute deletion_event_hash over the record_id, pre_state_root, post_state_root, receipt_id and metadata.
8. Produce the cryptographic evidence	Execute ZKP circuit.
9. Establish evidence-generation provenance	Capture TEE attestation of the measured ZD/prover software and bind deletion_event_hash to that execution. Record the ZKP circuit identifier and vkey hash, bound to the attestation.
10. Finalise CVDR	Collate the CVDR fields, including ZK proof blob, circuit and vkey identifiers, TEE attestation evidence and required metadata. Serialise the completed CVDR in the defined JSON format.
11. Store/Export CVDR	Deliver JSON file to end-user (i.e. download to the device). Store CVDR on internal enterprise CVDR table.

Figure 1: Zombie Delete lifecycle with ZKPd/web2 example

Figure 1 shows the ZKPd configuration inside a trusted execution environment. Web2 receipts issued today are signed by the operator (section 5); TEE signing and ZKPd are roadmap items (section 8).

Furthermore, the enterprise would keep a copy of all such receipts for verification by authorised internal or external auditors.

Depending on an enterprise's requirements, the target record might not be physically deleted but instead replaced with a tombstone to block re-insertion. All fields except the record_id are overwritten with 'null' (or, more precisely, protocol-defined canonical null values appropriate to the field type).

The generic Zombie Delete lifecycle, applicable across different proof models (explained in section 3) and configurations (section 5), is depicted in the left column of Figure 1. A clarifying example using the zero knowledge proof model is shown in the right column.

2.2 Cryptographically Verifiable Deletion Receipts

A CVDR is simply a series of numbers displayed mostly in hex format and stored as a JSON file, which is the format expected by our pre-built verification reference software. The components of a CVDR vary depending on factors such as the use case, the enterprise system set-up, and the cryptographic proof model. In general, CVDRs will include some of or all the components described in Table 1.

Table 1: Common CVDR components

Component	Description
Metadata	This may include a protocol version-id; a unique CVDR number; a database identifier; timestamp; etc.
Target binding	A unique identifier for a given transition element: a record_id, lookup token, document filename, object identifier, etc. In the PII deletion example this would be the record_id of the customer profile. Where the identifier reveals sensitive data (for example, say an email address was used as the record identifier) a salted hash or equivalent privacy-preserving value may be used, provided authorised entities can connect it to the target.
Pre-state and post-state commitments	Cryptographic hashes or other evidence capturing the relevant states before and after the transition. Depending on the configuration, these may be serialised state hashes, Merkle roots, authenticated state proofs, or equivalent commitments.
Transition event commitment	A cryptographic commitment to the specific transition, binding the target, relevant pre- and post-transition evidence, and other protocol-defined event data. It identifies *what transition occurred*, independently of how that claim is subsequently authenticated.
Evidence-generation provenance	Evidence identifying or committing to the process and evidence source that created the CVDR. Depending on the configuration this may include source/adapter provenance and an evidence-class identifier, a module hash or other code commitment, a TEE measurement, a ZK circuit or verifying-key identifier, or a combination.
Authentication evidence and authority	Cryptographic evidence authenticating the transition claim. Depending on the configuration this may be a digital signature, blockchain certificate, TEE attestation, zero-knowledge proof, or equivalent evidence.

2.3 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, the tombstone keeps the record_id in place with protocol-defined null values in every other field. Every Zombie Delete 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 (RT6). Second, the guard blocks re-ingestion under the same identifier only. A record created under a new identifier, for example when a former customer signs up again, is outside the guard. 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.

3. PROOF MODELS

The proof models are mapped along two dimensions: scope and mechanism (see Figure 2).

Scope refers to the form of the data target: either a stand-alone item (such as an individual file or document) or a member of a set (such as a record on a database table). The latter scope is managed using a *Sparse Merkle Tree*: a structure that enables either the absence or presence of a record to be determined efficiently in even extremely large databases. We refer to this as the ‘Tree’ model. Individual items use the same underlying commitment techniques (hashes) but have no use for a full tree structure; hence the ‘Leaf’ nomenclature.

Mechanism refers to the cryptographic proving structure. Both types (commitment based and Zero Knowledge Proofs) rely on cryptographic *fingerprints* (hashes). Commitment based verification comprises several explicit recomputation, authentication and provenance checks. ZKPd reduces the per-receipt transition check to standard proof verification. ZKPd has the additional benefit of hiding the target data and Merkle paths, providing an extra layer of security against potential leakage of subtle structural clues such as shared paths, sibling relationships, or patterns across multiple proofs.

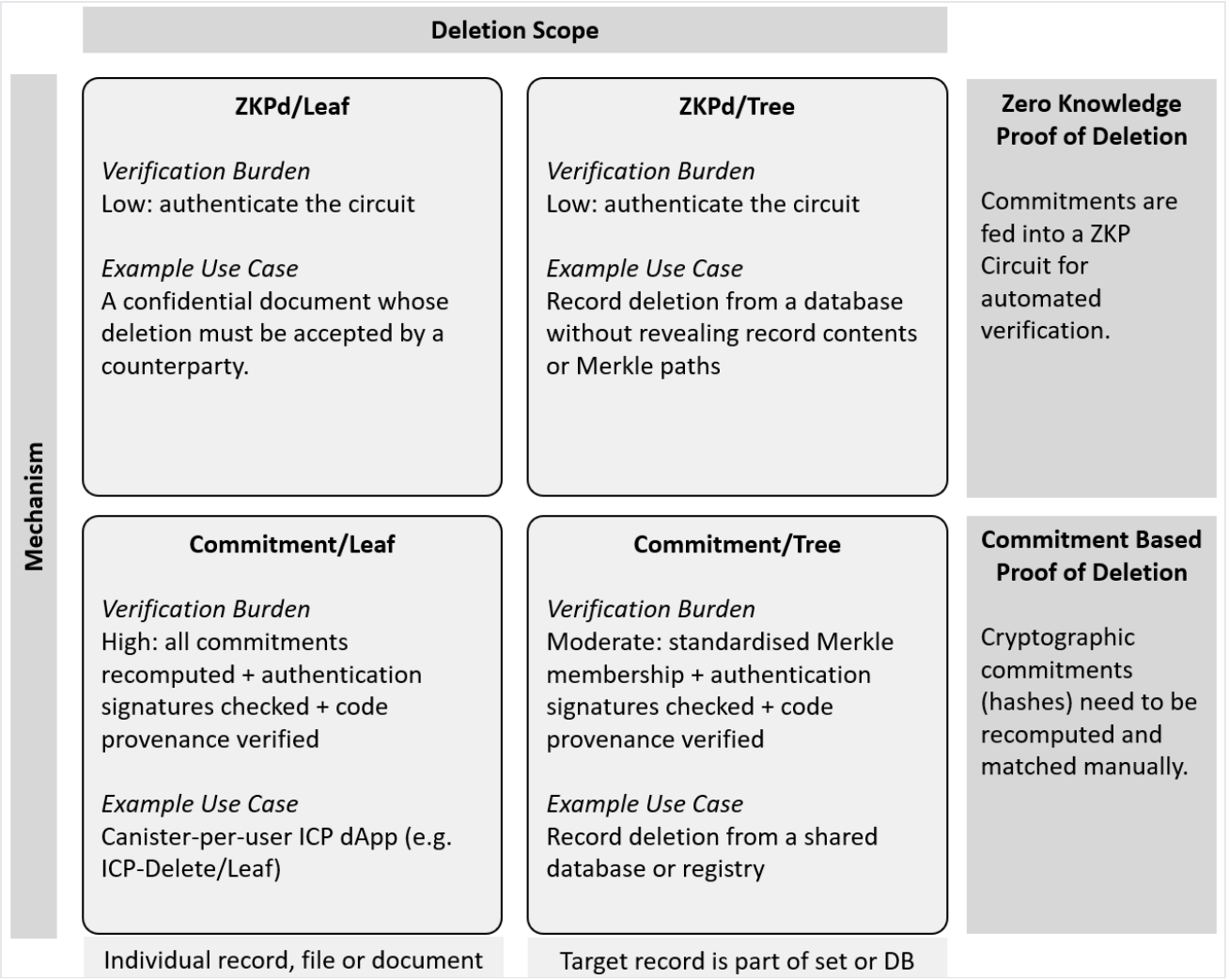


Figure 2: Proof Models

3.1 Commitment Based Proofs

The fundamental principle is to capture fingerprints of the datastore's pre-state (e.g. the state of the DB table, pre-deletion) and post-state in such a way that a verifier can verify them later. The principle applies whether the target is a self-contained item (Leaf mode) or a record in a database table (Tree mode). The aim is to prove that a specific target once *did* exist in the datastore and *was* deleted (or, more precisely, transitioned to the *protocol-defined deleted state*, i.e. absence or a tombstone).

The fingerprinting is accomplished using a 'one-way' mathematical function known as hashing: given an input, the result can always be derived but not vice versa. The hash acts as a *commitment* to a specific input while keeping that input secret. The technique is used so that the proof artefact (the CVDR) can be verified without revealing the target data itself, other than its identifier (such as a record_id).

Normally, verifying a hash requires revealing the input: the verifier feeds it into the same hash function to recompute the hash value and ensures it matches the supplied commitment. However, the input may represent sensitive information (such as PII) that must remain secret, perhaps even to the verifier. We achieve this using a *double hash* (hash of a hash) plus an alternative proof that the first hash was derived from the target data. The alternative proof is furnished by making the certified executing code inspectable so anyone can verify that the first hash is derived from the record whose record_id is included in the CVDR. The full verification procedure is explained in section 4.

3.2 Zero Knowledge Proof of Delete (ZKPd)

A Zero Knowledge Proof works along the same principles as described above, requiring the same commitments. The main difference is in the verification process. ZKPs employ a mathematical algorithm known as a ZKP circuit to encapsulate the verification rules. The circuit inputs comprise both public and private data (referred to formally as public inputs and private witness respectively). The public inputs are the commitments, while the private witness refers to the target data. The circuit is executed by the prover (the enterprise) to produce the proof blob. The CVDR includes the proof blob, public inputs and the cryptographic verification key (vkey), which is all that is needed to perform a verification. The verification process is performed programmatically rather than a series of individual checks as needed for the commitment based models. The circuit itself is transparent and can be independently validated.

The proof establishes that the circuit relation holds for its public inputs; it does not by itself establish that the supplied state root was the datastore's real pre-state. The state commitment must therefore come from an authenticated or attested source (for example, a TEE-attested prover or a certified on-chain root).

4. VERIFICATION

There are two aspects to verification: (i) the once-off validation of the process as a whole; and (ii) the steps required to validate an individual CVDR.

4.1 Process validation

Zombie Delete code is open and transparent so anyone can see and check the workings of the proof models and the CVDR derivation process. Verifiers are invited to inspect the source code itself and confirm that the attested module identity corresponds to that published source. This is possible because the CVDR contains a commitment (hash) of the installed code (the *binary*). For commitment based proofs this is the *module hash*; the ZKP case is somewhat different but analogous.

The module hash confirmation is done by recreating the binary from the publicly available source code and then hashing it. The result should match the module hash in the CVDR. This is a once-off verification for as long as the ZD code remains unchanged but must be repeated if the ZD core software is upgraded. A history of ZD releases is publicly maintained so CVDRs created by earlier versions can always be re-verified against an old module hash at any future date.

Module hash verification requires both the *source code* and the *build path* to be readily available. TAV publishes the necessary details including the version numbers of all build tools and processes (the build path). In some instances (for example, where the procedure involves integrating Zombie Delete logic directly into proprietary enterprise code) the enterprise may need to publish the details or make them available on demand to authorised verifiers such as professional auditors.

For ZKPd, the inspectable source code includes the ZK circuit. The *verification key* (vkey) acts as a fingerprint of the proving configuration and is checked against the circuit source code and compile path, comprising compiler version, gadget-library lockfile and other build dependencies. Where the proving system uses a circuit-specific trusted setup (for example Groth16), the circuit source, compile path, verification key and associated setup artefacts must all correspond.

4.2 Individual CVDR verification

The verification of individual CVDRs depends on the proof model and configuration. In principle, all the information required to establish the verification procedure is derivable from the source code, and a verifier seeking full assurance should confirm the published procedure against that source. No trust in TAV is required; any residual trust in the evidence authority is configuration-dependent and stated in the RTS. A configuration claims a property described in this paper only where its Residual Trust Statement confirms the property holds for that deployment. Zombie Delete additionally publishes verification documentation complete enough (exact fields, derivations, domain tags, encodings, constants, trust roots and provenance materials) that an independent verifier can execute the standard procedure from the documentation and public materials alone. The documentation is a checkable restatement of the source, and the verification reference software is meant as a convenience. Both are expected to be independently verified rather than trusted.

Two distinct questions are addressed. *Verification validity* asks whether the CVDR satisfies the cryptographic and protocol rules applicable to it. *Assurance grade* describes the strength and status of the evidence on which the receipt relies. The CVDR identifies the nature of the evidence supporting the claim and its authentication authority. A cryptographically valid CVDR may rely on evidence of differing strength depending on the configuration. All Zombie Delete configurations include a *Residual Trust Statement* (RTS) that gives the enterprise a full account of the assumptions applicable to its installation.

For commitment based proofs the verifier needs to perform several individual verification steps. In a nutshell, the verifier must first check the protocol adherence and internal consistency of the CVDR by recomputing all the committed values. Then check the authentication signature, also embedded in the CVDR. The authenticating authority should ideally be an automated, independent, trusted mechanism. For example, in web2 installations it might be a Trusted Execution Environment (TEE) with a publicly accessible cryptographic key. For web3 installations the authority is the blockchain's own consensus mechanism. The authenticating mechanism cryptographically validates the transition evidence carried by the CVDR, and the verifier checks it using the verification method and trust material appropriate to that configuration. For example, where authentication uses a digital signature, the verifier checks that the relevant transition evidence was signed by the expected authority using its published verification key.

For ZKPd, CVDR verification is the same in principle but the process is baked into the circuit. The main components of a ZKPd CVDR (proof blob, public inputs, and vkey) are fed into a standard verification algorithm that executes all verification steps and simply responds with accept/reject.

The case study described in Appendix 1 provides an in-depth example, demonstrating the chain of logic that proves the claimed deletion or other state transition.

4.3 Residual Trust Statement

All Zombie Delete configurations rely on the integrity of various hardware, software, signing authorities, and protocols. The RTS exists to delineate those dependencies so enterprises can have full confidence in the claims they make.

As ZD configurations depend on a combination of processes there will usually be several line items in an RTS. To maintain a high level of rigour, even standardly trusted mechanisms are listed for completeness. For example, where a ZD configuration subscribes to a CDC (Change Data Capture) stream, the CDC infrastructure must be trusted to faithfully relay the transaction data; if the process executes within a TEE (Trusted Execution Environment) the integrity of the TEE hardware must be trusted; and so on. An example RTS using the case study configuration from Appendix 1 is shown in Table 2.

Table 2: Example Residual Trust Statement (ICP-Delete/Leaf configuration: see Appendix 1)

#	Assumption	Mitigation
RT1	ICP subnet consensus is honest (2/3 nodes not colluding)	Foundation of ICP security model. Subnet membership governed by NNS.

Table 2: Example Residual Trust Statement (ICP-Delete/Leaf configuration: see Appendix 1) (continued)

#	Assumption	Mitigation
RT2	Canister controller does not upgrade canister to reverse tombstones	V3A/V3B establish the code identity associated with the certified deletion/finalisation event: the module_hash_certificate subnet-attests the installed module identity during the bounded finalisation process; the displayed module_hash is populated from that certificate; V3A authenticates it and V3B links it to published, reproducible source. Governance/DAO controls mitigate subsequent upgrades; a later upgrade does not invalidate the archived receipt and is not detected by standard archival verification
RT3	DataSource adapter's field_manifest faithfully declares all PII fields	Adapter code is auditable. The state encoding specification that defines the PII boundary must be published and version-controlled. Adapter runs inside canister under consensus. If the published boundary omits a PII field, that field is excluded from the hash and survives tombstoning: the receipt is valid but deletion is incomplete. This is an integration correctness concern, not a protocol failure.
RT4	Tombstone execution renders prior PII values irrecoverable	The process performs logical overwrite of PII fields in stable memory. ICP does not guarantee that prior byte values are physically irrecoverable from individual replica storage. The process operates at the logical protocol level; physical data remanence is a platform-level concern outside the protocol's scope. This is consistent with GDPR's standard of "reasonable measures": logical deletion with cryptographic proof is the appropriate benchmark, not physical media scrubbing.
RT5	No canister snapshots or backups exist that could resurrect deleted data	ICP canister snapshots (backup/migration feature) could contain pre-deletion state. An archived CVDR proves the certified historical deletion transition; it does not establish whether the system was subsequently restored. Snapshot/backup control is an operational residual outside the archival receipt claim; the receipt documents a point-in-time deletion.
RT6	The integration's mandatory write guard remains in place.	Resurrection of the tombstoned record on registered write paths is blocked by the deployed code (a required integration control, conformance-tested). A privileged controller can subsequently replace the code enforcing the barrier; this is a residual operational trust assumption. Such a later change does not invalidate an already-issued CVDR and is outside standard archival receipt verification. Identity-level rejoin under a different identifier is not blocked and not claimed.

The current situation for enterprises that do *not* have a formal proving system such as Zombie Delete is a straightforward, unsubstantiated claim; in essence: 'trust us, we deleted it'. CVDRs provide evidence to back the claim. The strength and nature of that evidence are apparent from the CVDR itself, including its authentication evidence and evidence class. The RTS gives the enterprise the fuller account of the assumptions and residual trust applicable to its installation. For example, in cases where a TEE is used it is the hardware's signature that provides the ultimate trust anchor. In blockchain environments the trust anchor is the chain's consensus mechanism.

When the entire ZD process is executed on the enterprise's own servers with no independent authentication service such as a TEE, the signing authority is effectively the enterprise itself (*enterprise self-attestation*). The possibility of a fabricated CVDR therefore remains. However, even this weakest version of CVDRs is a significant improvement on the unsubstantiated claim. It yields a cryptographically bound, specific and checkable claim attributable to the enterprise; an incomplete deletion under a given CVDR becomes a signed false statement.

Table 3: What a CVDR 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 (RT5) • That earlier byte values cannot be recovered from storage media (RT4) • Removal from an immutable ledger's transaction history (section 5) • That the same subject cannot return under a new identifier (RT6) • 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

4.4 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 the web2 configuration the receipt archive sits outside the customer database that Zombie Delete modifies, and if a replayed deletion would produce different content under an existing receipt_id the process halts and does not overwrite. 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.

4.5 Long-lived receipts

A CVDR 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, which identifies the hash functions and proof system in use; for ZKPd the vkey identifies the circuit. If an underlying algorithm is later broken, the receipt's evidential assurance can weaken: a future attacker may be able to 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 evidentiary retention requires it, the receipt holder may separately use standard archival timestamping or evidence-preservation mechanisms, such as an RFC 3161 timestamp token held in an RFC 4998 evidence record. The CVDR itself remains unchanged. That practice belongs to the receipt holder; TAV operates no archival service.

Circuit-friendly hash functions such as the Poseidon family are younger than SHA-2 and under active public cryptanalysis; the Ethereum Foundation's programme on Poseidon runs to December 2026. The hash functions and proof system of each ZKPd release are therefore stated per protocol_version, and cryptographic-agility options are listed in section 8.

5. ZOMBIE DELETE CONFIGURATIONS

Zombie Delete is designed to work across multiple enterprise configurations. The various cryptographic techniques (e.g. hashing functions, Merkle Trees, zero knowledge proofs, etc.) are deployed as appropriate for the use case. Configuration-dependent adapters, templates and reference code are provided along with the core according to the enterprise set-up and their proof requirements. The main deployment categories are web2, web3, and hybrid installations. These are orthogonal to the proof models; a deployment may use any of the proof models depicted in Figure 2 depending on the application and enterprise requirements.

5.1 Web 2 (Traditional Databases)

Web2 refers to traditional set-ups using well-established datastores such as PostgreSQL, MongoDB, and so on. These systems may be built on the enterprise's own servers or in a cloud service. Zombie Delete works across almost any environment but TAV encourages the use of high-security utilities such as TEE (Trusted Execution Environments), HSM (Hardware Security Modules), and KMS (Key Management Services).

The right-side column in Figure 1 shows an example lifecycle where the configuration is a record deletion (more precisely, tombstoning) from a web2 DB (e.g. PostgreSQL) running inside a TEE.

Zombie Delete installations do not require existing enterprise code to be modified. ZD code is added to enterprise deployments and requires new database-level guards, such as DB triggers. The tombstone blocks re-insertion of the original record and the DB guard rejects update attempts. This does not stop a customer voluntarily returning to the enterprise; a returning customer will, however, need to undergo the on-boarding process again from scratch and will be assigned a new record_id as if a new customer.

5.2 Web 3 (Blockchain)

Web3 refers to blockchain-enabled environments. The case study described in Appendix 1 is designed for ICP (Internet Computer Protocol), which is a mutable blockchain, but Zombie Delete applies to immutable blockchains also. 'Immutable' implies that data cannot be deleted. However, this is not an entirely accurate representation of the situation. First, it is rare for blockchains to store any sensitive information as plaintext. As discussed in the 'hybrid' section, a common model is to have plaintext data on traditional web2 servers and only encrypted data or pointers on the blockchain. Second, and more importantly, the Zombie Delete model plays a significant role even in cases where immutable blockchains do store sensitive information in plaintext.

Modern state-machine blockchains maintain an explicit current state in addition to immutable transaction history. The transition between explicit states, including when records are deleted or tombstoned, can be evidenced with Zombie Delete just as for traditional databases. It is true that any past state can be recreated by recourse to historical transactions, however this does not invalidate the Zombie Delete process.

The situation is roughly analogous to web2 where a record is deleted from the primary database while the possibility always exists for back-ups and untrackable data copies to be retained. For these, as for all configurations, the Residual Trust Statement makes clear to the enterprise exactly what can or cannot be claimed with respect to the Zombie Delete process. The enterprise must assess if the RTS satisfies whichever obligation (regulatory or otherwise) they are aiming to fulfil. For the European data protection view of erasure on blockchains, see the EDPB guidelines (ref 7).

5.3 Web2+Web3 Hybrids

Figure 3 shows the various configurations of Zombie Delete where blockchains are involved.

A common design is to keep plaintext data in a traditional web2 database and put only a commitment (hash), pointer, or permission record on-chain. In that situation the main deletion CVDR may concern the off-chain database, while a second piece of evidence shows that the on-chain reference was revoked or removed; in effect, ZD versions execute on both datastores.

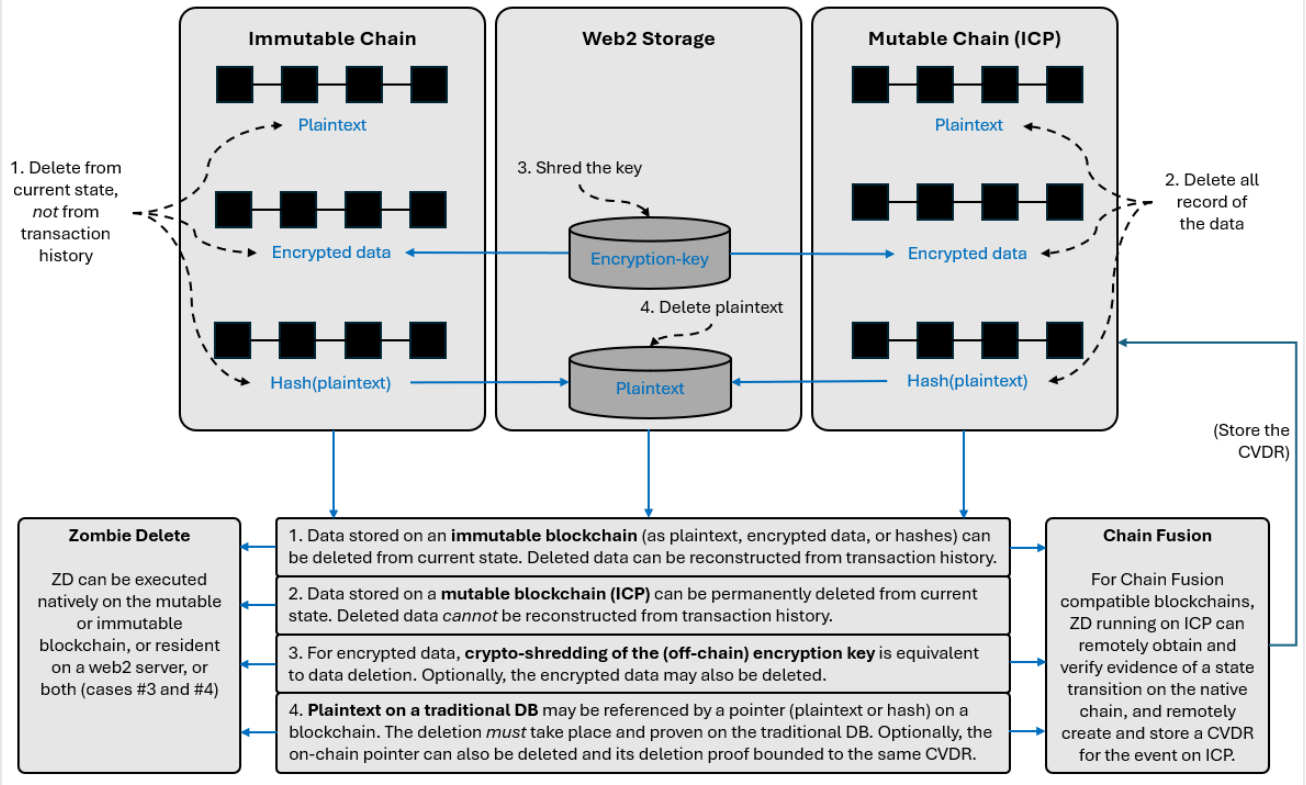


Figure 3: web2+web3 hybrid options

Another approach is to encrypt sensitive data (*ciphertext*) for storage on an immutable blockchain and keep the encryption key in an alternative datastore (traditional or blockchain). Then deleting the key or performing a procedure to make it permanently unusable (often referred to as *crypto-shredding*) renders the ciphertext inaccessible under the usual cryptography assumptions. The *before-**shred*** **to** ***after-**shred*** state transition can be certified by Zombie Delete, just as any other state transition.

5.4 Chain Fusion

Chain Fusion is Internet Computer Protocol (ICP) technology for interacting directly with other blockchains. As Figure 3 shows it enables Zombie Delete, running on ICP, to verify state transitions, such as deletions, occurring on a compatible native chain, without installing any Zombie Delete logic on that chain.

In the native chain case, Zombie Delete participates directly in creating the transition and its evidence. With Chain Fusion, the native application performs an ordinary state change and Zombie Delete acts as an independent, external observer and verifier. In this mode, Zombie Delete does not execute the native-chain deletion; it verifies the target chain's own evidence of the transition and binds that evidence into a CVDR.

A Chain Fusion CVDR is therefore slightly different from a chain native CVDR. Rather than relying on evidence generated by Zombie Delete itself, ZD on Chain Fusion uses the target blockchain's own cryptographic evidence of state. Specific mechanisms are chain-dependent but a common example is a finalised block containing a commitment to the blockchain state together with a proof showing the relevant value before and after the transition. Zombie Delete, running on ICP, checks that evidence and packages the *verified* result into the CVDR. The receipt is therefore verified when it is created and remains independently *re-verifiable* afterwards. Where the target chain cannot provide a cryptographic state proof, the RTS identifies the additional assumptions involved in validating the transition.

Chain Fusion is particularly useful where no changes should be made to the native blockchain application. It also enables a single CVDR to combine related transitions; for example, deletion of sensitive data from an off-chain database together with deletion or revocation of its on-chain reference. As previously discussed, proving removal from a blockchain's explicit state does not erase data that remains recoverable from immutable transaction history. Again, the RTS delineates exactly what any configuration proves.

6. 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 record does not remove these derived copies. A store's own delete function may not remove them either. A 2026 study showed that soft-deleted embeddings stay recoverable from the raw index files of three HNSW vector index implementations. From those vectors the authors reconstructed 25.5 per cent of exact names and 46.4 per cent of locations in biographical text. They also recovered every age and gender marker from synthetic patient records (ref 9).

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 (section 2) 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. The rebuilt store does not contain the record, and derived stores may need their own CVDRs. What evidence Zombie Delete can obtain is specific to the store. Deletion, compaction and rebuild semantics differ from product to product, and a soft delete at a store's interface is not evidence that the underlying index data is gone (ref 9). Each configuration therefore uses an adapter for that store class, which defines the evidence it obtains, and the Residual Trust Statement says what a receipt for that store shows. Where a derived store lives in a datastore Zombie Delete already supports, the enterprise's rebuild drops the derived entry and Zombie Delete issues a separate CVDR for that datastore from the store's change feed or other evidence surface defined by the adapter. Each receipt carries the evidence-generation provenance of its configuration (Table 1). Each scope hashes the record's key under its own scope configuration, so the record hash differs from store to store. The enterprise states which source receipt each derived receipt follows; a reference field that binds them is listed in section 8. Each derived receipt is dated after the source receipt. An auditor checks the set as a whole: recompute each receipt, confirm the provenance and the stated links, and confirm the order.

Re-ingestion. The tombstone stays at the source, so later rebuilds of connected indexes and memory stores read the tombstone again. The write guard (section 2) blocks any attempt to write the record back under its own identifier.

Configuration. The configuration follows the datastore. Where the AI system's stores are conventional databases, the web2 commitment-based configuration applies: processing can remain entirely within the enterprise's own estate and receipts are signed with the enterprise's deployment key. Where a derived store, such as an agent's memory, lives in a canister on the Internet Computer, the ICP configuration applies and the subnet certifies the receipt (section 5). A TEE signing profile for web2 is listed in section 8.

Data starvation. A working model uses a specific record mainly through the stores it reads: the search index, agent memory, tool calls and data refreshes. 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. We call this data starvation. It limits how far the record spreads. It does not remove the residue, and it does not stop the spread entirely.

What the receipts do not show. A model trained on the record before its deletion may keep some of what it learned. 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. The receipts say nothing about any training run. Whether a model version was trained without the record is a claim about the run itself: a receipt on the training run, binding the dataset read and the model version before and after, is future work (section 8). A model that continues from weights that saw the record holds a residue the tombstone does not reach. So does a model trained on the output of such a model, by distillation or on synthetic data: that output is a derived copy with no link to the source record.

In this paper, unlearned has two meanings only. Either the model version was trained without the record, which a receipt on the training run would show (section 8), or no declared source covered by the receipts remains from which the model can retrieve or refresh the record, which the receipts in this section show. It never means that a trained model was made to forget.

Verified unlearning, where this paper and the patent portfolio use the term, means four things: the record was deleted from the declared stores; a CVDR proves that deletion; the deleted state is held, by tombstone and write guard where configured; and a later training run can be shown to have excluded the record. It is not a claim that an already-trained model's weights have forgotten the data.

Further transitions. Zombie Delete certifies any defined state transition (section 2). The same model extends to a retraining or unlearning run. A receipt can bind the dataset used, the model version before and after, and the method. Such a receipt proves what was done, not what the model still knows. Section 8 lists this among our future work.

7. PRODUCT COMPONENTS

A Zombie Delete product is a defined, versioned package of implementation, verification and governance materials. The package is composed of technical components and documentation, both aspects of which can be independently checked and validated by a skilled individual. The exact package varies by configuration, but the following components provide the common product-assurance baseline.

7.1 Technical components

- *CVDR generation modules*. This is the Zombie Delete core function: the versioned code, ZKP circuit, procedure or controlled workflow that derives the CVDR. In a commitment based proof it is the code that performs the pre-state commitment, and post-state-commitment, or Sparse Merkle Tree root and Merkle path. For ZKPd it is the zero knowledge proof circuit that produces the proof blob and accompanying metadata for the CVDR.
- *Integration material*. This includes templates for adapters that act as interfaces between the ZD code and the enterprise-specific installations.
- *Verification reference software*. For the convenience of verifiers, this is a package of programs runnable against an exported CVDR to perform the verification steps. The source code is transparent for anyone to check so no trust of TAV or the enterprise is required.

7.2 Documentation set

- *Installation and integration manual*. Enterprise-facing instructions for deploying the technical components, including security assumptions and required configuration controls.
- *Operations, recovery and archival guide*. Suggested procedures for preserving evidence, handling incomplete issuance, recovering from operational failure, managing keys or credentials, and archiving receipts.

- *Residual Trust Statement*. A detailed description of all assumptions and remaining trust requirements, to ensure the enterprise has full visibility on what may be legitimately claimed.
- *Verification procedure*. Explains how the CVDR constitutes proof and the steps required to verify it. Once again, the intention is complete transparency so anyone with the requisite technical skills can check for themselves that the procedure is valid.
- *Code provenance*. Full transparency requires the ZD source code to be publicly accessible and authenticated so anyone can check the validity of the CVDR generation process.

8. ROADMAP

This paper describes the Zombie Delete 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 in section 3. Circuits are written in Noir. Its Residual Trust Statement will state the proving system and its setup assumptions.
- **Zombie Protect.** Zombie Delete evidences deletion; Zombie Protect blocks protected identifiers from re-entering through configured ingestion paths. A data holder keeps a cryptographic registry of prohibited identifiers and a gate on its ingestion path. Each new record must carry a proof that its identifier is not prohibited. The registry never reveals which identifiers it holds. An entry for data once held and destroyed cannot be told apart from an entry for data never held. Zombie Protect is a separate integration class: Zombie Delete asks nothing of the enterprise's application logic beyond the write guard, while Zombie Protect places a gate on the ingestion path. A provisional patent is filed.
- **Zombie Lock.** Some duties sit between keeping data and destroying it: a restriction of processing, a litigation hold, a sanctions freeze. Zombie Lock places a record in a sealed state in which no complete decryption key exists. The only way to create the key is an authenticated state transition that records the unsealing, so nobody can open the record silently. TAV holds no key share. A provisional patent is filed.
- **Receipts for AI training.** Three receipts. *Corpus commitment*: proof that a training run read a dataset from which the record was absent. *Measured data starvation*: an extraction-test score recorded beside each receipt across model versions. *Verified unlearning*: a receipt on the retrain or the unlearning run, covering the data used, the model before and after, and the method; a receipt for the process, not a claim about what the model still knows.

- **Source coverage.** Every claim in this paper is scoped to declared datastores. A verifier can check what was declared; nothing in a receipt can show that the declaration is complete, and Table 3 says so. Under design: an enterprise-owned, versioned declared-store manifest, held inside the enterprise estate and never by TAV, and a reconciliation step that, where useful and available, compares the manifest with evidence from systems independent of the deletion pipeline, such as data catalogue exports and cloud asset inventories. Its form, a field in the receipt or a separate coverage artefact, is under design. A provisional patent is filed.
- **Cryptographic agility.** Options for ZKPd under design: committing the same state under both a conservative long-lived hash and an in-circuit hash, and hash-based transparent proof systems that need no reference string. The proving backend, curve and setup for each release are stated per protocol_version (section 4.5).
- **Post-quantum signatures.** A CVDR's hashes and Merkle commitments rest on SHA-256, which a quantum computer weakens but does not break at this size. Its signatures do not share that property. Under design: a web2 signing profile using ML-DSA (FIPS 204) or SLH-DSA (FIPS 205), with the signature algorithm stated per protocol_version, and the hash-based proof systems above for ZKPd. The ICP subnet certificate uses BLS; its migration belongs to the network, not to TAV, and the RTS will say so.
- **Receipt reference field.** A derived-store receipt names the source receipt it follows, so a verifier binds the set without the enterprise's statement.
- **Re-ingestion alarm.** A monitoring component on the audit or error stream raises an alarm when the write guard blocks or halts a write.
- **Published state roots.** On request, the daemon writes each state root to a store the enterprise chooses before any deletion, so a receipt's pre-state is a commitment made in advance. Optional; the destination is the enterprise's.
- **TEE signing profile.** A trusted execution environment profile for web2 configurations.
- **Independent anchoring.** An external anchor for receipts signed with the enterprise's own key, such as a transparency log or timestamp authority the enterprise chooses. This gives self-attested receipts an independent date.

9. INTELLECTUAL PROPERTY

The Zombie Delete patent portfolio comprises 16 provisional patents filed. They cover deletion receipts, the tombstone, source coverage, verified unlearning, Zombie Protect and Zombie Lock (section 8). The inventors are Glen Cameron and Dr Stef Savanah. The portfolio is assigned to Together Alone Ventures OÜ.

Verification depends on transparency. A verifier must be able to inspect the code whose module hash a CVDR carries, rebuild it and compare the hashes (section 4). For each release, TAV publishes the source that this check needs, the build path and the verification reference software. Publication for inspection is not an open-source licence. It grants no right to use the patented methods.

For ZKPd, the circuit source and compile path are published with each release, and the verification key is checked against them.

TAV may release some components under an open licence.

10. TEAM

Glen Cameron, founder and CEO. Second-time blockchain founder and previous grant winner. Twenty years as a C-suite communications, strategy and PR executive at global level. Named inventor across the core deletion portfolio.

Dr Stef Savanah, co-founder and Chief Scientist. Holder of 192 granted patents and over 400 pending, none of them part of the Zombie Delete portfolio. Formerly Director of Scientific Research at nChain. PhD in Cognitive Science, Macquarie University.

Robin Davies, COO (fractional). Blockchain expert, financial and regulatory services executive, entrepreneur and growth leader.

Antoine Monesma, Lead Developer. Full-stack web, mobile and Web3 developer. Builds all Zombie Delete production code.

Vikram Seth, investor and Fractional Chief Partnership Officer. Former Head of Web3 at Shell, working on enterprise blockchain for payments, supply chains and sustainability. Founder of V38 and co-founder of 512 Ventures.

Zorana Smith, Head of Sales. **Finja Schnelle**, Admin and Operations Executive.

Adviser

Richard Slater, Legal Consultant. Multi-disciplined senior legal counsel. Former partner, Financial Services Group, Simmons & Simmons, London.

APPENDIX 1: CASE STUDY, ICP-DELETE/LEAF

This configuration is a commitment based proof designed for ICP dApps in which the target data for deletion is stored on a dedicated canister (see bottom left quadrant of Figure 2). This proof model has been selected because it represents the highest verification burden, thus disclosing the process in detail (ZKPd, by contrast, obscures the process by automating it within the ZKP circuit). Accordingly, the case study provides an excellent demonstration of the rigour involved in composing a CVDR and the comprehensiveness of the verification procedure.

The case study involves a straightforward PII deletion request (as might be relevant to GDPR Article 17). The data is not physically deleted but replaced with a tombstone. This means the record still exists keyed by its identifier; all other data fields are replaced with a *null* value (in practice ‘null’ does not literally mean zeroes or blanks but predetermined, protocol-specific and publicly known canonical tombstone values). For the case study this means the dedicated data canister is retained in the tombstone state to block re-insertion.

Table 4 shows the CVDR for ICP-Delete/Leaf along with a description of each component and its usage in the verification process. As for every proof model, ICP-Delete/Leaf is fully and transparently documented, including the CVDR creation process, the verification procedure, and verification reference software. The verifier is not expected to trust the published material; the verification procedure can be derived from first principles. Both the published steps and the reference software can be audited for correctness and completeness.

Table 4: CVDR for ICP-Delete/Leaf

Field	Description	How the verifier uses it
protocol_version	Identifies the CVDR protocol version, defining the field set and derivation/verification rules that apply.	Matches the CVDR to the published protocol version to identify the corresponding verification rules.
trust_root_key_id	Identifies the trust root under which the CVDR's certificates verify. Stamped by the Zombie Delete core at finalisation from its active key identifier: “mainnet” for production or “local-dev” for development/testing.	Selects the corresponding trusted root for certificate verification. “mainnet” uses ICP’s published NNS root; “local-dev” uses the supplied local-replica root used in development and testing environments.
canister_id	The ICP principal of the canister that is the system of record. In Tree mode this canister holds the table of customer records; in Leaf mode, as in this case study, it is a single canister per customer.	Cross-check against the canister identity authenticated by the subnet certificates.
record_id	The application-level identifier for the target data. In Leaf mode this has a one-to-one relationship with canister_id but is usually not the same value.	Used with canister_id and deletion_seq to recompute receipt_id (verification V1)
deletion_seq	A monotonic per-canister event counter, providing deterministic sequencing and contributing to a unique receipt identity.	Used to recompute receipt_id, and also directly as an input to deletion_event_hash (V1)
receipt_id	Unique receipt identifier and storage key, derived from canister_id, record_id and deletion_seq. This is the mechanism that binds the external target identifier (record_id) into the certified chain; the state hashes commit to the target's data, not to its identifier.	Recomputed and compared with the displayed value; then used in recomputing deletion_event_hash, binding the stated target into the certified chain.

Table 4: CVDR for ICP-Delete/Leaf (continued)

Field	Description	How the verifier uses it
timestamp	The ICP consensus time recorded for the deletion event.	Used to recompute deletion_event_hash (V1)
pre_state_hash	A commitment to the record's live state immediately before deletion, captured before the transition is applied. SHA-256 of serialised record data.	(This cannot itself be recomputed from CVDR values as the target data must remain secret). Used to recompute deletion_event_hash (V1)
post_state_hash	A commitment to the post-deletion state (the *tombstone state*), computed after the transition is applied. SHA-256 of the protocol-defined serialised tombstone state of the target record.	Recomputed from the defined tombstone state and checked against the CVDR value. Used to recompute deletion_event_hash (V1)
tombstone_hash	A commitment to the defined deletion outcome: SHA-256 (canister_id, protocol-defined serialised tombstone state, timestamp, deletion_seq).	Diagnostic only; not part of the verification chain. If verification fails, can be recomputed to help localise the inconsistency.
deletion_event_hash	A single commitment binding the pre-state, post-state, receipt identity, event time and deletion sequence.	Recomputed from the fields above and compared with the displayed value (V1); then compared with the certified_data authenticated by bls_certificate (V2)
bls_certificate	ICP subnet certificate over the canister's certified data, retrieved through the canister certificate query and stored in the completed CVDR.	Validated against the selected ICP trust root, including confirmation that the signing subnet was authorised by that root and was responsible for the stated canister, and the certified_data it authenticates must equal deletion_event_hash
module_hash_certificate	ICP subnet certificate over the exact system-state path containing the canister's installed module hash, retrieved during finalisation and stored in the completed CVDR.	Validated against the selected ICP trust root, including confirmation that the signing subnet was authorised for the stated canister_id (V3A). The certified path must be exactly /canister/<canister_id>/module_hash (ref 8), and the certified value must equal the module_hash displayed in the CVDR.
module_hash	The SHA-256 fingerprint of the complete installed canister software (host application plus embedded Zombie Delete engine). It is populated from module_hash_certificate when the CVDR is completed.	Checked against the certificate, then compared with the hash reproduced from the published source, dependencies and build procedure, allowing the attested software to be independently inspected (V3B).

Figure 4 is a schematic showing how the CVDR is created, from which the verification procedure can be inferred. The following is a high-level summary. The checks are:

- V1 internal derivation and consistency (receipt_id and deletion_event_hash recomputed from the displayed fields)
- V2 certificate authentication under the selected trust root and certified_data = deletion_event_hash
- V3 the code-identity/provenance family:
- V3A subnet-attested deployed module identity
- V3B reproducible source/build provenance

The verifier's first step (V1) is to recompute the receipt's protocol-defined derivations, including receipt_id and deletion_event_hash, and compare them with the displayed values. This ensures that the CVDR has been properly formed and satisfies the validity rules.

> *V1 proves the CVDR is internally consistent, but how do we know the values are true? Anyone can create an internally consistent CVDR from scratch using fake data.*

As depicted in Figure 4, deletion_event_hash is published as the canister's certified data and certified by the ICP subnet, thereby establishing by blockchain consensus that this exact deletion event was recorded by the target canister. In V2 the verifier checks the subnet certificate (bls_certificate) included in the CVDR and confirms the certified value equals the recomputed deletion_event_hash.

> *That only proves the canister ran a process to create the CVDR but how do we know what that process was? For example, how do we know that process properly fingerprinted the pre-state and post-state?*

ICP has an inbuilt utility to provide the module_hash (a fingerprint of the canister binary) and to certify it with the module_hash_certificate. This attests the installed code identity associated with the deletion and finalisation process. Both are included in the CVDR and can be validated. The source code is published by TAV (and/or the enterprise dApp operator) and can be examined to check the algorithm. For example, the source code will expose the process of computing the pre_state_hash and post_state_hash for the given record_id. The pre_state_hash itself cannot be recomputed because the data must remain private but record_id is bound into the certified deletion event via receipt_id, and the code identity attested for the finalisation interval (V3A/V3B) is the code whose published source defines how pre_state_hash is computed.

> *How do we link the source code to the module_hash?*

The binary can be rebuilt directly from the published source code (given the build path, which is published alongside it) and hashed to recompute the module_hash. The recomputed module_hash should be a byte-for-byte exact match for the CVDR module_hash. This reproducible-build check is performed once for each distinct module_hash and can then be reused for all receipts generated by that module.

The finer details have been omitted here for clarity but are fully explained in the Zombie Delete documentation. The CVDR also contains a protocol_version, which identifies the applicable field definitions, derivation rules and verification procedure, so receipts remain tied to the correct protocol specification as Zombie Delete evolves.

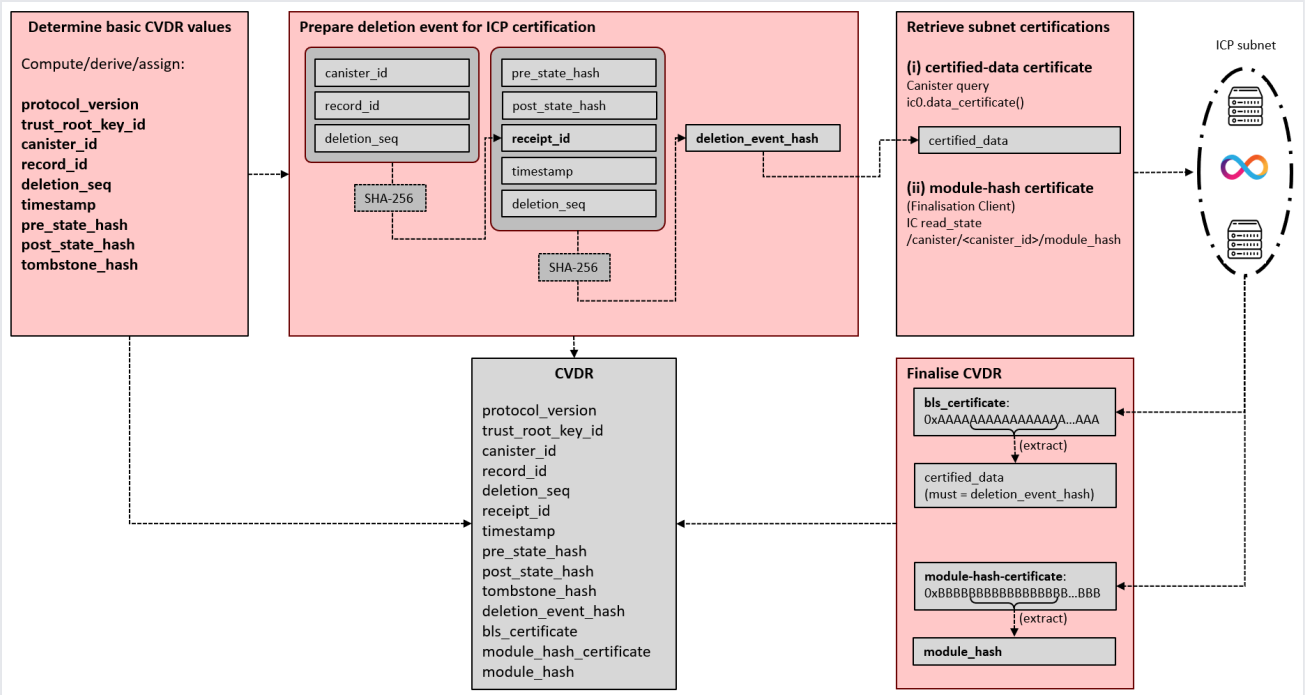


Figure 4: flow diagram for the case study (ICP-Delete/Leaf)

APPENDIX 2: ZOMBIE DELETE USE CASES

Many obligations to destroy data sit outside privacy law. The table orders use cases by market, with privacy erasure last.

Area	Example	What the CVDR shows
Post-breach clean-up	After an incident, exposed records, keys and credentials are removed from declared systems	Each removal happened, when, and which attested code was installed
Healthcare and clinical	When electronic patient records reach disposal, the HIPAA Security Rule requires procedures for their final disposition (ref 6); clinical-trial records reach the end of retention	The records left each declared datastore on a stated date
Defence and national security	Controlled data is destroyed at contract end or on transfer	Destruction the other party can check without seeing the data
Litigation and investigations	A legal hold is released, or discovery material must be returned or destroyed	Deletion that is dated and checkable

Table (continued)

Area	Example	What the CVDR shows
SaaS and vendor exit	A processor deletes customer data at contract end (ref 1, Article 28(3)(g)); a financial entity exits an ICT provider (ref 2)	The supplier's deletion, checkable by the customer
KYC, AML and payments	Due diligence records reach the end of five-year retention (ref 3); cardholder data passes its retention period (ref 5)	Deletion on schedule
Mergers and acquisitions	Divestiture, clean-room separation or a closed data room	The separated datastore or data room was cleared
Cross-border and counterparty	A data transfer ends, or a recipient must delete a mis-sent file	The recipient's deletion, checkable by the sender
Dual-use and export control	Controlled technical data must be destroyed	Destruction without disclosure of the data
Blockchain and Web3	Credential or soulbound token revocation, NFT burn, DAO membership removal	The transition in the chain's explicit state (section 5)
AI systems	Source record deletion with index, vector store and agent memory rebuilds; the end of a training-data licence	One receipt for each declared store (section 6)
Marketing and adtech	Consent is withdrawn, and the tombstone doubles as a suppression list	The record was deleted from the declared store and same-ID re-ingestion is blocked on guarded write paths
Privacy erasure	Right to erasure and retention expiry (ref 1); bulk deletion requests through California's DROP (ref 4)	The subject's record left each declared datastore
Beyond deletion	Key rotation, certificate and access revocation, change-control evidence, employee offboarding	Any defined state transition (section 2)

REFERENCES

Sources last checked 4 October 2026.

1. Regulation (EU) 2016/679 (GDPR), Articles 5(1)(e), 17 and 28(3)(g). EUR-Lex
2. Regulation (EU) 2022/2554 (DORA), Article 28(8), exit strategies. EUR-Lex

3. Regulation (EU) 2024/1624 (AMLR), Article 77: keep records for five years, then delete the personal data. Applies from 10 July 2027. EUR-Lex
4. California Delete Act (SB 362). From 1 August 2026, data brokers must process deletion requests from the Delete Request and Opt-out Platform (DROP) at least every 45 days. Independent audits start on 1 January 2028 and repeat every three years. CalPrivacy, DROP for data brokers; CPPA, information for data brokers
5. PCI Security Standards Council, PCI DSS v4.0.1, Requirement 3.2.1. PCI SSC document library
6. HIPAA Security Rule, 45 CFR 164.310(d)(2)(i), disposal. eCFR
7. EDPB, Guidelines 02/2025 on processing of personal data through blockchain technologies, version 2.0, adopted 7 July 2026.
8. DFINITY, IC interface specification: certified data, and the state tree path /canister/<canister_id>/module_hash.
9. Chakraborttii, García Alvarado, Abdulofizova and Dwivedi, Ghost Vectors: Soft-Deleted Embeddings Remain Reconstructible in HNSW Vector Databases, arXiv:2606.18497, 16 June 2026.

APPENDIX 3: WHAT CHANGED SINCE VERSION 1

Version 1 of this paper (March 2026) was a product and market paper. Version 2 is the protocol paper. The main changes:

- **The general model.** Deletion is one instance of a certified state transition. The CVDR is defined over any pre-state to post-state transition (section 2), not over deletion alone.
- **The claim is bounded.** Version 1 said that a reinsertion of a tombstoned record was detectable by any verifier. Version 2 makes no detect claim. The write guard blocks re-ingestion on registered write paths; Table 3 and the Residual Trust Statement state what a receipt shows and what it does not.
- **Verification is specified (section 4).** Version 1 described verification. Version 2 gives the procedure: process validation by module hash and reproducible build, individual receipt verification (V1 to V3B), the Residual Trust Statement (RT1 to RT6), receipt custody, and the treatment of long-lived receipts.
- **Proof models and configurations (sections 3 and 5).** Proof models are mapped on scope (Leaf, Tree) and mechanism (commitment, zero knowledge). Configurations cover web2, web3, hybrids and Chain Fusion. Version 1 presented these as product codes.
- **AI systems (section 6).** New. Root-record deletion, derived stores as declared datastores, data starvation, and a definition of unlearned that never means a model was made to forget.

- **Proof system.** Version 1 specified Groth16. Version 2 fixes no backend: the proving system and its setup assumption are stated per protocol version, and cryptographic ageing is treated in section 4.5.
- **Source and verification.** Full source, build path and reference verifier are published for each release, and anyone can verify without any TAV service. Publication is not an open-source licence. A convenience verifier runs at verifier.zombiedelete.com.
- **Intellectual property.** The portfolio is stated as 16 provisional patents, with the inventors named and the assignment to Together Alone Ventures OÜ stated.
- **Roadmap (section 8).** ZKP Delete, Zombie Protect, Zombie Lock, receipts for AI training, source coverage, cryptographic agility, post-quantum signatures and independent anchoring. The Great Clean-Up and PQR-Delete of version 1 are now covered by independent anchoring, cryptographic agility and post-quantum signatures.
- **Scope.** Version 2 confines itself to the protocol. Delivery models, regulatory context and the competitive landscape are published separately.