

Zero-Trust, Defense-Grade Financial & Health Data Analytics Platform

Technical Specification · Version 3.0 · October 2026

0. Document Control

Field	Value
Document Identifier	SPEC-ZDE-2026-v3.0
Title	Zero-Trust, Defense-Grade Financial & Health Data Analytics Platform
Version	3.0 (consolidated)
Status	APPROVED FOR PRODUCTION
Target Compliance Domains	DoD IL5/IL6, HIPAA Security Rule, SEC Rule 17a-4, FINRA, NIST SP 800-53 Rev. 5
Effective Date	October 2026
Document Owner	Rob Nock
Supersedes	"Defense-Grade Production Specification: Zero-Trust Financial & Health Data Analytics Platform" (part 1 and part 2 drafts)
Consolidated	9 October 2026

Prepared for Rob Nock. Status and target compliance domains are as approved by the document owner. Formal authorizations are out of scope (see section 8).

Table of Contents

0. Document Control..... 1

0.1 Revision History 3

0.2 Revision Notes / Changelog 3

1. System Overview & Architecture..... 5

1.1 Architecture Diagram 5

1.2 Core Architecture Mandates..... 5

2. Functional Specification..... 6

FS-1 Governed Cell-Level Read Access 6

FS-2 Ephemeral Enclave Processing (Zero-Spill)..... 6

FS-3 Controlled Aggregation Exports (k-Anonymity / Differential Privacy)..... 6

FS-4 Audit Lineage & Non-Repudiation 6

3. Technical Stack 7

4. Security Controls & Data Spill Prevention Protocols 8

4.1 Network Boundary Security 8

4.2 AWS IAM Read-Only Policy Definition 8

4.3 Hardware-Backed Cryptographic Signing Pipeline (Asymmetric, RSASSA-PSS-SHA-256 via KMS) 9

5. Reference Implementation 11

5A. Canonical Provenance Payload (provenance.py) 11

5B. Auditor Verification (verify_report.py)..... 12

5C. Zero-Egress Viewport & Watermarked Pixel Stream (viewport.py)..... 14

6. Infrastructure-as-Code Hardening..... 17

6.1 KMS Signing Key and Key Policy (Terraform)..... 17

6.2 Kernel Hardening (/etc/sysctl.d/99-defense-hardening.conf)..... 19

7. Incident Response & Real-Time SIEM Triggers 21

8. Operational Sign-Off..... 22

Appendix A. Glossary..... 23

Appendix B. Control Mapping to NIST SP 800-53 Rev. 5 24

0.1 Revision History

Version	Date	Description
Draft (part 1)	2026	Overview, mandates, FS-1..FS-4, controls table, signing formula (truncated)
Draft (part 2)	2026	Sections 5B, 5C, 6, 7 and operational sign-off
3.0 (partial)	October 2026	New v3.0 header, document control, sections 1–4.1, start of 4.2 (truncated)
3.0 (consolidated)	9 October 2026	Single consolidated document in the v3.0 structure; corrections and completions per 0.2

0.2 Revision Notes / Changelog

Technical corrections applied in this consolidation:

- 1. Signing engine renamed and corrected.** "HMAC Signing Engine (KMS Asymmetric Key)" is now **"Asymmetric Signing Engine (RSASSA-PSS-SHA-256 via KMS)"** in the diagram, mandates, stack table, and section 4.3. HMAC is a symmetric MAC: a verifier would need the same secret key, so it cannot provide non-repudiation. References to "HMAC signing keys" and "HMAC/signatures" were also removed.
- 2. Unified signed payload.** The signer and verifier now use one canonical payload: `JSON {query_hash, user_arn, timestamp, s3_object_versions, row_count}` with sorted keys, separators (',', ':'), and UTF-8 encoding. It is used in the formula (4.3.1), the signer (5A), and the verifier (5B). This replaces the ambiguous `SHA256(QueryHash || Timestamp || S3objectVersion || RowCount)` concatenation in part 1 and the four-field payload in part 2's 5B, which omitted `s3_object_versions`. The digest is signed with `MessageType=DIGEST` to avoid KMS's 4 KB RAW limit.
- 3. KMS key policy and rotation.** Added `EnableRootAccountAdministration` (account root, standard) and `AllowKeyAdministrationNoCryptoUse` (KeyAdmin role, no Sign/Verify) to prevent key-management lockout. Also added a `DenySignExceptEnclaveRole` guard and an auditor `kms:GetPublicKey` statement. Rotation: asymmetric KMS keys do not support automatic rotation, so `enable_key_rotation = false` and a manual new-key-plus-alias procedure is documented (6.1). Part 1's "automatic rotation" now applies only to the symmetric SSE-KMS data keys.
- 4. sysctl comments corrected and settings added.** `kernel.yama.ptrace_scope = 3` disables ptrace entirely, including for root, and cannot be lowered until reboot. `dev.tty.ldisc_autoload = 0` restricts auto-loading of TTY line-discipline modules. Added `kernel.kptr_restrict = 2`, `kernel.dmesg_restrict = 1`, and `kernel.perf_event Paranoid = 3` (with a mainline-kernel caveat). Added `swapoff`, `fstab`, and `swap.target` masking, `systemd-coredump` controls, and a note that `vm.swappiness` alone does not disable swap (FS-2). Comments were moved onto their own lines, as `sysctl.d` requires.
- 5. Watermark.** The watermark is now tiled diagonally (a rotated brick-pattern overlay at 15 % opacity) with an opaque footer label. Added a note that pixel rendering deters text scraping but not camera capture or OCR; the watermark provides traceability (5C).
- 6. Confidential-computing platforms.** Intel SGX is noted as deprecated on many current Xeon and client parts, and AMD SEV-SNP and Intel TDX are listed as current options. Clarified that Nitro Enclaves isolate memory from the parent instance through the Nitro Hypervisor, not SEV or SGX. "Hardware attestation (AWS Nitro TPM)" was corrected to Nitro Enclave attestation documents (PCRs), with NitroTPM for parent-instance measured boot (FS-2, section 3).

7. **WORM retention.** S3 Object Lock in Compliance mode now states a 7-year retention (SEC Rule 17a-4) in FS-4, section 3, and section 6.1. The v3.0 audit row read "CloudWatch"; it now reads CloudTrail + CloudWatch Logs, consistent with FS-4.
8. **Status and claims.** "Status: APPROVED FOR PRODUCTION" and the target compliance domains are retained as written. "Guarantees zero data spills" is now "is designed to prevent data spills". The sign-off now reads "designed to satisfy the control objectives of ...", and a scope exclusion was added: DoD PA/ATO for IL5/IL6, the HIPAA risk analysis, and the SEC 17a-4(f) third-party undertaking are out of scope. Text completed because of source truncation is marked *[completed by editor]*.

Completions and additional clarifications:

1. **Section 4.2 IAM policy** *[completed by editor]*: completed from the truncation point. Adds `s3:ListBucket` (prefix-scoped), `aws:SourceVpce` and `aws:SecureTransport` conditions, `kms:Decrypt` via S3 only, explicit Deny on `s3:PutObject`, `DeleteObject`, `DeleteObjectVersion`, and `PutObjectAcl`, and a Deny for insecure transport.
2. **Section 4.3.2 execution sequence** *[completed by editor]*: completed from "Generate Result Set" through k-anon/DP gate → canonical hash → `kms:Sign` → audit write → watermarked render, with fail-closed ordering (audit before render).
3. **Section 5A** *[completed by editor]*: new canonical provenance module and example record. **Section 5B** now also covers offline verification via `kms:GetPublicKey` and the `cryptography` library (PSS, MGF1-SHA-256, salt length = digest length).
4. **Architecture diagram**: the ASCII diagram was replaced with a Mermaid source block and a rendered vector figure. **FS-3**: clarified that k-anonymity and differential privacy are distinct mechanisms. **TLS 1.3 enforcement**: now specified through `aws:SecureTransport` plus an `s3:TlsVersion` bucket-policy condition. **Section 7**: four SIEM triggers added and labeled. **Appendices A and B**: glossary and NIST SP 800-53 Rev. 5 mapping added.

Editorial conventions: text marked *[completed by editor]* was written to complete content that was truncated in the source drafts. Paragraphs labeled **Editor's note** are clarifications added during consolidation.

1. System Overview & Architecture

This specification defines the engineering architecture for a Zero-Trust, Zero-Data-Egress (ZDE) analytics platform that operates directly against sensitive financial and health datasets stored in AWS S3. The architecture is designed to prevent data spills across storage, transport, execution, and output display.

1.1 Architecture Diagram

Figure 1 shows the trust boundary. Everything except the analyst viewport sits inside the VPC trust boundary. The only thing that crosses it is a watermarked pixel stream.

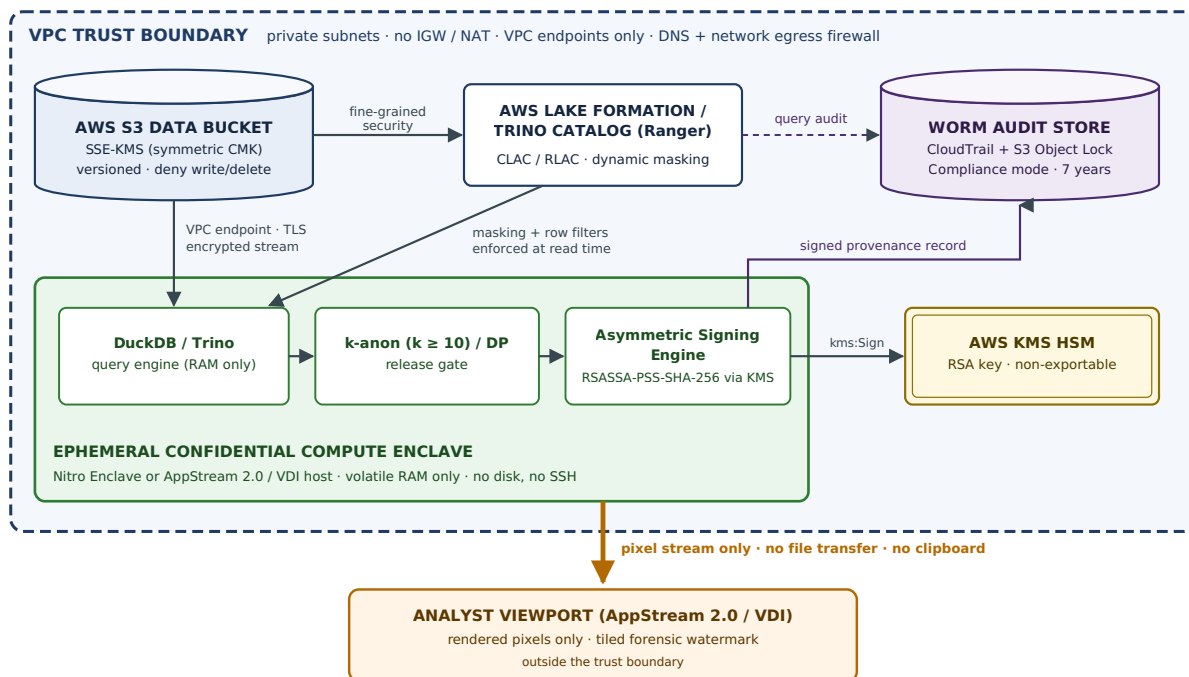


Figure 1. SPEC-ZDE-2026 trust boundary and data flow (rendered from the Mermaid source in the Markdown edition).

1.2 Core Architecture Mandates

- **Zero Raw Disk Writes.** Raw rows, unencrypted caches, and temporary tables never touch persistent workstation or server disk. All transformations run in volatile RAM inside an isolated enclave or a remote VDI host. Swap and core dumps are disabled (see 6.2).
- **Cell-Level and Row-Level Access Control (CLAC/RLAC).** Users query data only through a centralized governance layer (AWS Lake Formation or Apache Ranger). PII, account numbers, and medical identifiers are masked dynamically at read time, based on the IAM role and execution context.
- **Hardware-Backed Cryptographic Lineage.** Output provenance is signed inside the AWS KMS HSM boundary with a non-exportable asymmetric RSA key pair (RSASSA-PSS-SHA-256). Signing keys are never exposed to application memory. Callers use short-lived STS credentials.
- **Zero-Egress Viewport.** Analysts see results only as a read-only, watermarked pixel stream (AWS AppStream 2.0 / WorkSpaces). File download, local disk caching, printing, and clipboard are disabled through the streaming stack settings and OS policy (Group Policy / driver level).

2. Functional Specification

FS-1 Governed Cell-Level Read Access

Analysts run standard ANSI SQL against datasets stored in S3 in Parquet format. Every query goes through the centralized governance catalog.

- **Masking:** unprivileged analysts receive redacted fields. For example, `account_number` renders as `XXXX-XXXX-1234`.
- **Row-level security:** results are limited to the analyst's assigned regional or departmental scope.

FS-2 Ephemeral Enclave Processing (Zero-Spill)

SQL runs on hardware-isolated compute: AWS Nitro Enclaves or dedicated VDI hosts.

- **Isolation model (corrected).** *Nitro Enclaves* are isolated from their parent EC2 instance by the Nitro Hypervisor. The enclave has its own vCPUs and memory, and no persistent storage, interactive access, or external networking. Only a local vsock channel connects it to the parent. Even root or an administrator on the parent instance cannot read enclave memory. This isolation does **not** come from AMD SEV or Intel SGX, and the two should not be confused. *Dedicated VDI or confidential-VM hosts* may also use CPU memory encryption. Current options are **AMD SEV-SNP** (available on selected EC2 families, e.g. M6a/C6a/R6a) and **Intel TDX** (offered by other cloud providers and on-premises platforms; confirm availability before deployment). **Intel SGX** has been deprecated on many current Xeon and client processors and is not a target platform for this design.
- **Attestation.** Nitro Enclaves produce attestation documents signed by the Nitro Hypervisor PKI, carrying measurements (PCR0–PCR8) of the enclave image. These measurements are logged under FS-4. NitroTPM can additionally provide measured boot for parent instances.
- **Kernel safeguards.** Swap, core dumps, ptrace, and kernel memory or profiling interfaces are disabled on enclave and host kernels (section 6.2).

FS-3 Controlled Aggregation Exports (k-Anonymity / Differential Privacy)

Before rendering or export, every result set passes an automated release gate. Result sets that contain raw individual-level records, or any aggregate group with $N < 10$ (k-anonymity threshold $k = 10$), are rejected to prevent re-identification.

Editor's note: k-anonymity (minimum group size, i.e. suppression) and differential privacy (calibrated noise under a privacy budget ϵ) are separate mechanisms. The $k \geq 10$ rule is always enforced. Where configured, released aggregates also carry DP noise, and the gate tracks the cumulative ϵ per analyst and dataset.

FS-4 Audit Lineage & Non-Repudiation

For every query, the following are written to append-only, tamper-evident storage (AWS CloudTrail plus S3 Object Lock in **Compliance mode, 7-year retention**): the SQL statement, IAM context, hardware attestation measurements, row count, the S3 object versions scanned, and the KMS signature over the canonical provenance payload (4.3).

3. Technical Stack

Layer	Technology	Technical Function	Compliance & Security Alignment
Data Lake Storage	AWS S3 (Standard / Express One Zone)	Master repository for encrypted analytical datasets	SSE-KMS with symmetric customer-managed keys (automatic rotation enabled) in FIPS 140-3 Level 3 validated HSMs. Bucket policies enforce HTTPS/TLS 1.3 only (<code>aws:SecureTransport</code> plus an <code>s3:TlsVersion</code> minimum) and explicitly deny write/delete for analytical roles.
Governance Catalog	AWS Lake Formation / Apache Ranger	CLAC, RLAC, dynamic data masking	Enforces the HIPAA Minimum Necessary Standard and DoD CMMC 2.0 access restrictions.
Isolated Compute	AWS Nitro Enclaves / AppStream 2.0	Ephemeral compute running DuckDB / Trino	No persistent storage, no SSH. Nitro Enclave attestation documents (PCR measurements); NitroTPM for parent-instance measured boot.
Identity & Access	AWS IAM Roles Anywhere / OIDC federation	Short-lived session credentials (max 1 h)	X.509 certificate authentication with hardware-protected private keys (mTLS at the access tier). Zero static or long-lived credentials.
Key Management	AWS KMS (asymmetric, HSM-backed)	Digital signatures over query-output provenance	RSASSA-PSS-SHA-256 signing inside the KMS HSM boundary (<code>kms:Sign</code>). Private keys are non-exportable; public keys are distributed for verification.
Audit & Compliance	AWS CloudTrail + CloudWatch Logs / S3 Object Lock	WORM audit store	S3 Object Lock in Compliance mode with 7-year retention, supporting SEC Rule 17a-4(f) and HIPAA audit requirements.

4. Security Controls & Data Spill Prevention Protocols

4.1 Network Boundary Security

All compute runs in private VPC subnets with explicit routing controls. There is no internet gateway and no NAT gateway.

- **VPC S3 gateway endpoints.** S3 access is restricted to internal traffic through the gateway endpoint (`com.amazonaws.<region>.s3`). The bucket policy rejects requests that do not arrive through the designated VPC endpoint or VPC ID.
- **Egress firewall rules.** DNS Firewall and AWS Network Firewall explicitly block external endpoints, including generative-AI APIs and public cloud services (e.g. `*.openai.com`, `*.anthropic.com`, `bedrock-runtime.*.amazonaws.com`). The baseline is default-deny, with allow-lists only for required AWS service endpoints.
- **Editor's note:** with no internet path, the workload reaches KMS, STS, CloudWatch Logs and similar services through VPC interface endpoints, each with its own endpoint policy.

4.2 AWS IAM Read-Only Policy Definition

This policy is attached to all analytical execution roles. The source draft ended after `"s3:GetObjectVersion"`,. Everything after that line was *[completed by editor]* to the stated design: read-only access to the data prefix, only through the VPC endpoint, only over TLS, with explicit denies on write, delete, and ACL changes. ARNs are placeholders (account 123456789012).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowS3ReadOnlySecureData",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Resource": "arn:aws:s3:::zde-secure-data-bucket/analytics/*",
      "Condition": {
        "StringEquals": { "aws:SourceVpc": "vpce-0a1b2c3d4e5f67890" },
        "Bool": { "aws:SecureTransport": "true" }
      }
    },
    {
      "Sid": "AllowListDataPrefixOnly",
      "Effect": "Allow",
      "Action": "s3:ListBucket",
      "Resource": "arn:aws:s3:::zde-secure-data-bucket",
      "Condition": {
        "StringLike": { "s3:prefix": ["analytics/", "analytics/*"] },
        "StringEquals": { "aws:SourceVpc": "vpce-0a1b2c3d4e5f67890" },
        "Bool": { "aws:SecureTransport": "true" }
      }
    },
    {
      "Sid": "AllowDecryptViaS3Only",
      "Effect": "Allow",
      "Action": "kms:Decrypt",
      "Resource": "arn:aws:kms:us-east-1:123456789012:key/11111111-2222-3333-4444-555555555555",
    }
  ]
}
```

```

    "Condition": {
      "StringEquals": { "kms:ViaService": "s3.us-east-1.amazonaws.com" }
    },
  },
  {
    "Sid": "DenyAllWriteDeleteAndAclChanges",
    "Effect": "Deny",
    "Action": [
      "s3:PutObject",
      "s3:DeleteObject",
      "s3:DeleteObjectVersion",
      "s3:PutObjectAcl"
    ],
    "Resource": [
      "arn:aws:s3:::zde-secure-data-bucket",
      "arn:aws:s3:::zde-secure-data-bucket/*"
    ]
  },
  {
    "Sid": "DenyInsecureTransport",
    "Effect": "Deny",
    "Action": "s3:*",
    "Resource": [
      "arn:aws:s3:::zde-secure-data-bucket",
      "arn:aws:s3:::zde-secure-data-bucket/*"
    ],
    "Condition": {
      "Bool": { "aws:SecureTransport": "false" }
    }
  }
]
}

```

Notes *[completed by editor]*:

- IAM condition operators within one `Condition` block are ANDed. This policy keys on `aws:SourceVpce`, the specific gateway endpoint. If the deployment pins the VPC instead, replace it with `"aws:SourceVpc": "vpc-0a1b2c3d4e5f67890"`. Both keys are present only on requests that traverse a VPC endpoint, so requests from outside the VPC never match the Allow statements.
- `s3:ListBucket` is scoped with `s3:prefix` to the `analytics/` prefix. `s3:GetObjectVersion` is required because provenance (4.3) records exact object versions.
- `kms:Decrypt` is required to read SSE-KMS objects. It is limited to calls made through S3 (`kms:ViaService`).
- The S3 bucket policy should mirror these explicit denies for all principals: deny when `aws:SecureTransport` is false, when `aws:SourceVpce` does not match, or when `s3:TlsVersion` is below the mandated minimum (`NumericLessThan "1.3"`; verify client support before enforcing). Explicit Deny statements override any Allow granted elsewhere.

4.3 Hardware-Backed Cryptographic Signing Pipeline (Asymmetric, RSASSA-PSS-SHA-256 via KMS)

4.3.1 Signing formula

The signed message is a single canonical byte string. Both the signer (5A) and the verifier (5B) build it with the same function.

```
CanonicalPayload = UTF-8( JSON( {
    "query_hash":      hex(SHA-256(UTF-8(sql_text_as_submitted))),
    "row_count":       <integer>,
    "s3_object_versions": [ {"bucket", "key", "version_id"}, ... ],
    "timestamp":       "<RFC 3339 UTC, e.g. 2026-10-09T17:15:00Z>",
    "user_arn":        "<IAM principal ARN>"
  },
  sort_keys = true, separators = (",", ":") ) )

- exactly these five keys; keys sorted; no whitespace; UTF-8
- s3_object_versions sorted by (bucket, key, version_id)

Digest      = SHA-256(CanonicalPayload)

Signature = KMS.Sign(KeyId      = <full key ARN>,
                     Message    = Digest,
                     MessageType = DIGEST,
                     SigningAlgorithm = RSASSA_PSS_SHA_256)
# PSS, MGF1-SHA-256, salt length = 32 bytes (digest length)
```

MessageType = DIGEST is used because RAW messages are limited to 4,096 bytes, and a query that scans many object versions can exceed that. The signature is identical to signing the canonical payload with MessageType = RAW, so offline verifiers can check it directly against the payload bytes (5B).

4.3.2 Technical execution sequence

The source draft ended after "Generate Result Set". Steps 5–9 were *[completed by editor]*. Every step fails closed: if a step fails, nothing reaches the viewport.

#	Stage	Component	Action	Fail-closed behaviour
1	Query	AppStream 2.0 session → query service	Analyst submits plaintext ANSI SQL under a short-lived (≤ 1 h) STS session.	Reject if credentials are expired or not federated.
2	Masking	Lake Formation / Ranger	Applies column masking (CLAC) and row filters (RLAC) for the caller's role.	Deny if no grant covers the requested columns or rows.
3	Enclave execution	DuckDB / Trino in Nitro Enclave or VDI host	Reads Parquet objects over the VPC endpoint (TLS) into RAM and records each <code>version_id</code> .	Abort if attestation fails or a read is denied.
4	Result set	Enclave (RAM only)	Materializes the masked, filtered result set in volatile memory.	Never spilled to disk (no swap, no temp files).
5	k-anon / DP gate	Release gate	Rejects row-level output or any group with $N < 10$; applies DP noise where configured.	Reject and raise a SIEM alert (section 7).
6	Canonical payload hash	Signing engine	Builds the canonical JSON (4.3.1) and computes SHA-256.	Abort on any schema mismatch.
7	<code>kms:Sign</code>	AWS KMS HSM	Signs the digest with RSASSA_PSS_SHA_256 using the provenance key.	No signature means no release.
8	Audit write	CloudTrail + S3 Object Lock	Writes the provenance record and signature to the WORM bucket. CloudTrail independently logs the <code>Sign</code> call.	No render until the audit write is confirmed.
9	Viewport render	AppStream 2.0 / VDI	Renders the approved result as a tiled-watermark pixel stream (5C).	No file, clipboard, or print channel.

5. Reference Implementation

All code below is reference-grade Python 3.11+ (boto3, cryptography, numpy, pandas, opencv-python-headless). Placeholder ARNs use account 123456789012. Sections 5A–5C share one canonical-payload function, so signer and verifier cannot drift apart.

5A. Canonical Provenance Payload (provenance.py)

[completed by editor] This section was absent from the source drafts. It implements correction 2 and the signing formula in 4.3.1.

```

"""5A - Canonical provenance payload + in-enclave KMS signing (RSASSA-PSS-SHA-256)."""
import hashlib
import json
from datetime import datetime, timezone

SIGNING_ALGORITHM = "RSASSA_PSS_SHA_256"
SIGNED_FIELDS = ("query_hash", "user_arn", "timestamp", "s3_object_versions", "row_count")

def sha256_hex(data: bytes) -> str:
    return hashlib.sha256(data).hexdigest()

def canonical_payload(provenance: dict) -> bytes:
    """Return the exact bytes that are hashed and signed.

    Canonical form: JSON object with exactly the five SIGNED_FIELDS, keys sorted,
    separators ('', ':'), no insignificant whitespace, UTF-8 encoded.
    s3_object_versions is a list of {"bucket", "key", "version_id"} objects,
    sorted by (bucket, key, version_id) so list order is deterministic.
    """
    versions = sorted(
        [{"bucket": v["bucket"], "key": v["key"], "version_id": v["version_id"]}
         for v in provenance["s3_object_versions"]],
        key=lambda v: (v["bucket"], v["key"], v["version_id"]),
    )
    body = {
        "query_hash": provenance["query_hash"],
        "user_arn": provenance["user_arn"],
        "timestamp": provenance["timestamp"], # RFC 3339 UTC, e.g. 2026-10-09T17:15:00Z
        "s3_object_versions": versions,
        "row_count": int(provenance["row_count"]),
    }
    return json.dumps(body, sort_keys=True, separators=(",", ":"),
                      ensure_ascii=False).encode("utf-8")

def build_provenance(sql_text: str, user_arn: str, s3_object_versions: list,
                    row_count: int) -> dict:
    """Assemble the unsigned provenance record inside the enclave."""
    return {
        "query_hash": sha256_hex(sql_text.encode("utf-8")),
        "user_arn": user_arn,
        "timestamp": datetime.now(timezone.utc).strftime("%Y-%m-%dT%H:%M:%SZ"),
        "s3_object_versions": s3_object_versions,
        "row_count": row_count,
    }

def sign_provenance(provenance: dict, kms_client, kms_key_arn: str) -> dict:
    """Hash the canonical payload and sign the digest inside the KMS HSM boundary.
```

```

MessageType='DIGEST' avoids the 4,096-byte limit on RAW messages when many
S3 object versions are scanned. The private key never leaves KMS.
"""
digest = hashlib.sha256(canonical_payload(provenance)).digest()
resp = kms_client.sign(
    KeyId=kms_key_arn, # full key ARN, never an alias
    Message=digest,
    MessageType="DIGEST",
    SigningAlgorithm=SIGNING_ALGORITHM,
)
signed = dict(provenance)
signed.update({
    "payload_sha256": digest.hex(),
    "hsm_signature": resp["Signature"].hex(),
    "signing_key_arn": resp["KeyId"],
    "signing_algorithm": SIGNING_ALGORITHM,
})
return signed

```

Example signed provenance record (stored in the WORM audit bucket and attached to the rendered report). The `query_hash` shown is a placeholder: it is the SHA-256 of an empty string.

```

{
  "query_hash": "e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855",
  "user_arn": "arn:aws:iam::123456789012:role/DefenseAnalystRole",
  "timestamp": "2026-10-09T17:15:00Z",
  "s3_object_versions": [
    {
      "bucket": "zde-secure-data-bucket",
      "key": "analytics/claims/2026/09/part-0000.parquet",
      "version_id": "3HL4kqtJlcpXroDTdMJ.rmSpXd3dIbrHY"
    },
    {
      "bucket": "zde-secure-data-bucket",
      "key": "analytics/claims/2026/09/part-0001.parquet",
      "version_id": "Ho6vPy1uHrLr2Q0fJ0sQ5FRz9p3eQZ1a"
    }
  ],
  "row_count": 142,
  "payload_sha256": "18b2f7ef1674fc856f499e8b21e969b45e7b869cc066dbf6a1c971f76c222c89",
  "hsm_signature": "3a8f9c...<512 hex chars: 256-byte RSA-2048 signature>",
  "signing_key_arn": "arn:aws:kms:us-east-1:123456789012:key/aaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
  "signing_algorithm": "RSASSA_PSS_SHA_256"
}

```

The exact bytes that are hashed and signed for this record:

```

{"query_hash": "e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855", "row_count": 142, "s3_object_versions": [{"bucket": "zde-secure-data-bucket", "key": "analytics/claims/2026/09/part-0000.parquet", "version_id": "3HL4kqtJlcpXroDTdMJ.rmSpXd3dIbrHY"}, {"bucket": "zde-secure-data-bucket", "key": "analytics/claims/2026/09/part-0001.parquet", "version_id": "Ho6vPy1uHrLr2Q0fJ0sQ5FRz9p3eQZ1a"}], "timestamp": "2026-10-09T17:15:00Z", "user_arn": "arn:aws:iam::123456789012:role/DefenseAnalystRole"}

```

5B. Auditor Verification (`verify_report.py`)

Auditors verify the RSASSA_PSS_SHA_256 signature produced by AWS KMS without access to private keys or enclave memory. There are two modes:

- **Online:** `kms:Verify`. KMS checks the signature, and the auditor needs only `kms:Verify` on the key.

- **Offline:** fetch the public key once with `kms:GetPublicKey`, archive it alongside the audit records, and verify locally with the Python `cryptography` library (PSS, MGF1-SHA-256, salt length = digest length). This mode keeps working after the KMS key is retired and needs no AWS connectivity, which matters for 7-year retention.

```
"""5B - Auditor / compliance CLI: verify a KMS RSASSA-PSS-SHA-256 provenance signature.
```

```
Online: kms:Verify (needs kms:Verify on the key; private key never exposed).
```

```
Offline: kms:GetPublicKey once, then verify locally with `cryptography`  
        (PSS, MGF1-SHA256, salt length = digest length = 32 bytes).
```

```
"""
```

```
import argparse
import hashlib
import json
import sys
```

```
import boto3
from botocore.exceptions import ClientError
from cryptography.exceptions import InvalidSignature
from cryptography.hazmat.primitives import hashes, serialization
from cryptography.hazmat.primitives.asymmetric import padding
```

```
from provenance import SIGNING_ALGORITHM, canonical_payload
```

```
def verify_online(provenance: dict, kms_key_arn: str, region: str) -> bool:
```

```
    """Verify via the KMS Verify API (signature checked inside KMS)."""
```

```
    kms = boto3.client("kms", region_name=region)
```

```
    digest = hashlib.sha256(canonical_payload(provenance)).digest()
```

```
    try:
```

```
        resp = kms.verify(
            KeyId=kms_key_arn,
            Message=digest,
            MessageType="DIGEST",
            Signature=bytes.fromhex(provenance["hsm_signature"]),
            SigningAlgorithm=SIGNING_ALGORITHM,
        )
```

```
        return bool(resp["SignatureValid"])
```

```
    except ClientError as err:
```

```
        # KMS raises KMSInvalidSignatureException for a bad signature.
```

```
        if err.response["Error"]["Code"] == "KMSInvalidSignatureException":
```

```
            return False
```

```
        raise
```

```
def fetch_public_key_pem(kms_key_arn: str, region: str) -> bytes:
```

```
    """Export the public half (DER SubjectPublicKeyInfo) and return it as PEM."""
```

```
    kms = boto3.client("kms", region_name=region)
```

```
    resp = kms.get_public_key(KeyId=kms_key_arn)
```

```
    if SIGNING_ALGORITHM not in resp["SigningAlgorithms"]:
```

```
        raise ValueError("Key does not support RSASSA PSS SHA 256")
```

```
    public_key = serialization.load_der_public_key(resp["PublicKey"])
```

```
    return public_key.public_bytes(serialization.Encoding.PEM,
                                   serialization.PublicFormat.SubjectPublicKeyInfo)
```

```
def verify_offline(provenance: dict, public_key_pem: bytes) -> bool:
```

```
    """Verify locally with no AWS access, using an archived public key."""
```

```
    public_key = serialization.load_pem_public_key(public_key_pem)
```

```
    try:
```

```
        public_key.verify(
            bytes.fromhex(provenance["hsm_signature"]),
            canonical_payload(provenance), # library hashes with SHA-256
            padding.PSS(mgf=padding.MGF1(hashes.SHA256()),
                        salt_length=padding.PSS.DIGEST_LENGTH),
            hashes.SHA256(),
```

```

    )
    return True
except InvalidSignature:
    return False

def main() -> int:
    ap = argparse.ArgumentParser(description="Verify ZDE report provenance")
    ap.add_argument("provenance_json")
    ap.add_argument("--region", default="us-east-1")
    ap.add_argument("--public-key-pem", help="offline mode: archived public key")
    args = ap.parse_args()

    with open(args.provenance_json, encoding="utf-8") as fh:
        prov = json.load(fh)
        key_arn = prov["signing_key_arn"] # verify against the recorded key ARN

    if args.public_key_pem:
        with open(args.public_key_pem, "rb") as fh:
            ok = verify_offline(prov, fh.read())
    else:
        ok = verify_online(prov, key_arn, args.region)

    if ok:
        print(f"[VERIFIED] Signature valid for query_hash={prov['query_hash']} "
              f"user={prov['user_arn']} rows={prov['row_count']}")
        return 0
    print("[CRITICAL] SIGNATURE INVALID - quarantine artifact and raise SIEM alert",
          file=sys.stderr)
    return 2

if __name__ == "__main__":
    sys.exit(main())

```

Usage:

```

# Online (inside the audit VPC, AuditorRole credentials)
python verify_report.py report-provenance.json --region us-east-1

# Offline: export the public key once, archive it with the audit records
aws kms get-public-key --key-id "$SIGNING_KEY_ARN" \
  --query PublicKey --output text | base64 -d > signing-key.der
openssl pkey -pubin -inform DER -in signing-key.der -out signing-key.pem
python verify_report.py report-provenance.json --public-key-pem signing-key.pem

```

Expected output: [VERIFIED] Signature valid for query_hash=... (exit code 0) or [CRITICAL] SIGNATURE INVALID ... (exit code 2), which triggers the SIEM rule in section 7.

5C. Zero-Egress Viewport & Watermarked Pixel Stream (viewport.py)

Approved query output is rendered into an in-memory display buffer inside AppStream 2.0 / VDI and streamed as pixels only. The watermark is **tiled diagonally** across the whole frame (a rotated overlay at 15 % opacity, in a brick pattern). An opaque footer repeats the same label, so the label survives cropping.

```

"""5C - Zero-egress viewport: render results to an in-RAM pixel buffer with a
diagonally tiled, per-user forensic watermark. Nothing is written to disk."""
import math

import cv2
import numpy as np
import pandas as pd

```

```
FONT = cv2.FONT_HERSHEY_SIMPLEX
```

```
def _tiled_watermark_mask(h: int, w: int, text: str, angle_deg: float) -> np.ndarray:
    """Return a float mask in [0, 1] containing `text` tiled and rotated."""
    side = int(math.ceil(math.hypot(h, w))) # cover the canvas after rotation
    layer = np.zeros((side, side), dtype=np.uint8)
    scale, thick = 0.6, 1
    (tw, th), _ = cv2.getTextSize(text, FONT, scale, thick)
    step_x, step_y = tw + 80, th + 70
    for row, y in enumerate(range(th, side, step_y)):
        offset = (row % 2) * (step_x // 2) # brick pattern
        for x in range(-offset, side, step_x):
            cv2.putText(layer, text, (x, y), FONT, scale, 255, thick, cv2.LINE_AA)
    rot = cv2.getRotationMatrix2D((side / 2, side / 2), angle_deg, 1.0)
    layer = cv2.warpAffine(layer, rot, (side, side))
    y0, x0 = (side - h) // 2, (side - w) // 2
    return layer[y0:y0 + h, x0:x0 + w].astype(np.float32) / 255.0

def render_watermarked_viewport(df: pd.DataFrame, user_arn: str, timestamp: str,
                                width: int = 1200, line_height: int = 25,
                                opacity: float = 0.15,
                                angle_deg: float = 30.0) -> np.ndarray:
    """Render a (masked, k-anon-approved) result set to a BGR uint8 frame."""
    lines = df.to_string(index=False).splitlines()
    height = max(line_height * (len(lines) + 3), 300)
    canvas = np.full((height, width, 3), 255, dtype=np.uint8)

    char_w = 9 # fixed pitch keeps DataFrame columns aligned
    for i, line in enumerate(lines):
        y = line_height * (i + 1)
        for j, ch in enumerate(line[: (width - 20) // char_w]):
            cv2.putText(canvas, ch, (10 + j * char_w, y), FONT, 0.45,
                        (20, 20, 20), 1, cv2.LINE_AA)

    label = f"CONFIDENTIAL - {user_arn} - {timestamp}"
    mask = _tiled_watermark_mask(height, width, label, angle_deg)[..., None]
    color = np.array([0, 0, 160], dtype=np.float32) # BGR dark red
    out = canvas.astype(np.float32) * (1 - opacity * mask) + color * (opacity * mask)
    out = np.clip(out, 0, 255).astype(np.uint8)

    # Opaque footer so the trace label survives cropping of the tiled area.
    cv2.rectangle(out, (0, height - line_height), (width, height), (235, 235, 235), -1)
    cv2.putText(out, label, (10, height - 8), FONT, 0.45, (0, 0, 120), 1, cv2.LINE_AA)
    # Returned buffer goes straight to the streaming frame buffer.
    # Never call cv2.imwrite() or persist this array.
    return out
```

region	account_number	n_patients	avg_claim_usd
EAST	XXXX-XXXX-1234	142	1234.50
WEST	XXXX-XXXX-1234	57	987.25
NORTH	XXXX-XXXX-1234	31	2210.00

Figure 2. Reference output of `render_watermarked_viewport()` on synthetic, masked data ($k \geq 10$ groups).

Limitations (corrected). Pixel-only rendering deters text scraping, copy/paste, and file exfiltration. It does **not** prevent an analyst from photographing the screen or running OCR on a captured image. The watermark does not prevent disclosure. What it provides is **traceability**: any captured image carries the principal ARN and timestamp, which can be matched to the signed provenance record and the audit trail. OpenCV's Hershey fonts render ASCII only, so any non-ASCII values must be transliterated before rendering.

6. Infrastructure-as-Code Hardening

6.1 KMS Signing Key and Key Policy (Terraform)

```
# 6.1 - Provenance signing key (asymmetric, HSM-backed, non-exportable)
locals {
  account_id      = "123456789012"
  key_admin_role  = "arn:aws:iam::123456789012:role/KeyAdminRole"
  enclave_role    = "arn:aws:iam::123456789012:role/EnclaveExecutionRole"
  auditor_role    = "arn:aws:iam::123456789012:role/AuditorRole"
  signing_alg     = "RSASSA_PSS_SHA_256"
}

resource "aws_kms_key" "defense_signing_key" {
  description      = "SPEC-ZDE-2026 provenance signing key (RSASSA-PSS-SHA-256)"
  key_usage        = "SIGN_VERIFY"
  customer_master_key_spec = "RSA_2048" # consider RSA_3072 for a longer horizon
  deletion_window_in_days = 30
  enable_key_rotation = false # unsupported for asymmetric keys: rotate manually (6.1)
  multi_region      = false
  policy            = data.aws_iam_policy_document.signing_key_policy.json

  tags = {
    System = "SPEC-ZDE-2026"
    Usage  = "provenance-signing"
  }
}

resource "aws_kms_alias" "defense_signing" {
  name          = "alias/zde-provenance-signing"
  target_key_id = aws_kms_key.defense_signing_key.key_id
}

data "aws_iam_policy_document" "signing_key_policy" {
  # Standard: account root retains control, enabling IAM-based administration
  # and preventing an unmanageable key.
  statement {
    sid      = "EnableRootAccountAdministration"
    effect   = "Allow"
    actions  = ["kms:*"]
    resources = ["*"]
    principals {
      type       = "AWS"
      identifiers = ["arn:aws:iam:${local.account_id}:root"]
    }
  }

  # Key administrators manage the key lifecycle but cannot use it (no Sign/Verify).
  statement {
    sid      = "AllowKeyAdministrationNoCryptoUse"
    effect   = "Allow"
    actions  = [
      "kms:Create*", "kms:Describe*", "kms:Enable*", "kms:List*",
      "kms:Put*", "kms:Update*", "kms:Revoke*", "kms:Disable*",
      "kms:Get*", "kms:Delete*", "kms:TagResource", "kms:UntagResource",
      "kms:ScheduleKeyDeletion", "kms:CancelKeyDeletion",
    ]
    resources = ["*"]
    principals {
      type       = "AWS"
      identifiers = [local.key_admin_role]
    }
  }
}
```

```

# Only the enclave execution role may sign, only PSS-SHA-256, only digests.
statement {
  sid      = "AllowEnclaveSigningOnly"
  effect   = "Allow"
  actions  = ["kms:Sign"]
  resources = ["*"]
  principals {
    type      = "AWS"
    identifiers = [local.enclave_role]
  }
  condition {
    test      = "StringEquals"
    variable  = "kms:SigningAlgorithm"
    values    = [local.signing_alg]
  }
  condition {
    test      = "StringEquals"
    variable  = "kms:MessageType"
    values    = ["DIGEST"]
  }
}

# Auditors verify online and export the public key for offline verification.
statement {
  sid      = "AllowAuditorVerification"
  effect   = "Allow"
  actions  = ["kms:Verify"]
  resources = ["*"]
  principals {
    type      = "AWS"
    identifiers = [local.auditor_role]
  }
  condition {
    test      = "StringEquals"
    variable  = "kms:SigningAlgorithm"
    values    = [local.signing_alg]
  }
}

statement {
  sid      = "AllowAuditorPublicKeyExport"
  effect   = "Allow"
  actions  = ["kms:GetPublicKey", "kms:DescribeKey"]
  resources = ["*"]
  principals {
    type      = "AWS"
    identifiers = [local.auditor_role]
  }
}

# Defense in depth: no other principal can sign, even if an IAM policy allows it.
statement {
  sid      = "DenySignExceptEnclaveRole"
  effect   = "Deny"
  actions  = ["kms:Sign"]
  resources = ["*"]
  principals {
    type      = "AWS"
    identifiers = ["*"]
  }
  condition {
    test      = "ArnNotEquals"
    variable  = "aws:PrincipalArn"
    values    = [local.enclave_role]
  }
}
}

```

Key policy design (corrected).

- `EnableRootAccountAdministration` (standard) and `AllowKeyAdministrationNoCryptoUse` keep the key manageable. Without an administration path, a key policy can leave the key permanently unmanageable. The `KeyAdmin` role has no `kms:Sign` or `kms:Verify` permissions.
- `DenySignExceptEnclaveRole` stops any other identity from signing, even one that is granted `kms:Sign` through an IAM policy delegated by the root statement.
- Restrict `kms:ScheduleKeyDeletion` and `kms:DisableKey` on this key with an SCP. Signatures must remain verifiable for the full 7-year audit retention period.
- Partition note: for AWS GovCloud (US), used for IL5, ARNs use `arn:aws-us-gov:`. IL6 workloads run in AWS Secret Region partitions, and ARNs must be adapted accordingly.

Key rotation (corrected). AWS KMS does not support automatic rotation for asymmetric keys, so `enable_key_rotation` must be `false`. Rotate manually, at least annually or on suspected compromise:

1. Create a new `aws_kms_key` with the same spec and policy (e.g. `defense_signing_key_2027`).
2. Repoint `alias/zde-provenance-signing` to the new key. Signers resolve the alias at startup, but they record the **resolved key ARN** in each provenance record (`signing_key_arn`).
3. Remove `kms:Sign` from the old key's policy and keep `kms:Verify` and `kms:GetPublicKey`. Export the old key's public key to the WORM audit store.
4. Keep the old key enabled, or at least its archived public key available, for the full 7-year retention period. Verifiers always use the key ARN recorded in the record, never the alias.

Symmetric data-encryption CMKs (SSE-KMS for the data bucket) do support, and should use, automatic rotation.

6.2 Kernel Hardening (/etc/sysctl.d/99-defense-hardening.conf)

This applies to enclave image kernels, parent instances, and VDI hosts. In `sysctl.d` files, comments must be on their own lines.

```
# /etc/sysctl.d/99-defense-hardening.conf
# SPEC-ZDE-2026-v3.0 section 6.2. Apply with: sysctl --system

# --- Core dumps (FS-2: no process memory may reach disk) ---
# Never produce core dumps for setuid/privileged processes.
fs.suid_dumpable = 0
# Pipe any core dump to a program that discards it.
kernel.core_pattern = |bin/false

# --- Process memory inspection ---
# 3 = "no attach": ptrace is disabled for ALL processes, including root
# (CAP_SYS_PTRACE). Irreversible: cannot be lowered again until reboot.
# Breaks gdb/strace by design.
kernel.yama.ptrace_scope = 3

# --- Kernel attack-surface reduction ---
# Restrict auto-loading of TTY line-discipline kernel modules to callers
# with CAP_SYS_MODULE (unprivileged users can no longer trigger them).
dev.tty.ldisc_autoload = 0
# Hide kernel pointer addresses in /proc and other interfaces from all
# users, including root.
kernel.kptr_restrict = 2
# Restrict the kernel ring buffer (dmesg) to CAP_SYSLOG.
kernel.dmesg_restrict = 1
# Restrict perf_event_open (memory and profiling side channel).
# 3 = deny all unprivileged use. This value is honoured only by kernels
# carrying the Debian/Ubuntu/Android patch; mainline treats >= 2 as its
# most restrictive level. Verify on the target kernel.
kernel.perf_event_paranoid = 3

# --- Filesystem protections ---
# Restrict O_CREAT opens of FIFOs and regular files in world-writable
# sticky directories (e.g. /tmp) that the caller does not own.
fs.protected_fifos = 2
fs.protected_regular = 2

# --- Memory layout ---
# Full ASLR (stack, mmap, VDSO and heap).
kernel.randomize_va_space = 2

# --- Swap (FS-2) ---
# Minimise swapping on hosts where a swap device cannot be removed.
# NOTE: this does NOT disable swap; swap devices are removed below.
vm.swappiness = 0
```

Swap and core-dump controls (FS-2). Setting `vm.swappiness` alone does not disable swap. It only biases the kernel away from swapping. Swap must be removed entirely:

```
# Disable and remove all swap (FS-2: RAM contents must never be paged to disk)
swapoff -a
sed -i.bak -E '/^[^#].*\sswap\s/s/^/# disabled per SPEC-ZDE-2026 6.2: /' /etc/fstab
systemctl mask swap.target
# Disable systemd-coredump storage and per-process core limits
mkdir -p /etc/systemd/coredump.conf.d
printf '[Coredump]\nStorage=none\nProcessSizeMax=0\n' \
  > /etc/systemd/coredump.conf.d/99-zde.conf
echo '* hard core 0' > /etc/security/limits.d/99-zde-nocore.conf

# Verify
sysctl --system
test -z "$(swapon --show)" && echo "swap: none"
sysctl kernel.yama.ptrace_scope kernel.kptr_restrict kernel.dmesg_restrict
```

7. Incident Response & Real-Time SIEM Triggers

Findings are routed to AWS Security Hub and the SIEM. Rows marked *(editor addition)* extend the source table.

Trigger	Detection source	Severity	Automated response
k-anonymity rejection (N < 10)	Release-gate log → CloudWatch Logs → Security Hub custom finding	Medium	Log IAM context and <code>query_hash</code> . Escalate to High on repeated rejections by the same principal (possible differencing attack).
Memory dump attempt / ptrace call	auditd rules (ptrace, process_vm_readv, core-dump events); GuardDuty Runtime Monitoring	Critical	Terminate the enclave and revoke the IAM session (deny on <code>aws:TokenIssueTime</code>).
Non-VPC S3 access attempt	CloudTrail S3 data events (AccessDenied without matching vpce); GuardDuty S3 Protection	Severity 1	Immediate SOC escalation.
KMS signature verification failure	verify_report.py exit code 2; kms:Verify returning KMSInvalidSignatureException	High	Quarantine the exported report artifact and open an integrity investigation.
kms:Sign by a principal other than EnclaveExecutionRole <i>(editor addition)</i>	CloudTrail management events (EventBridge rule)	Critical	Disable the principal and review key policy and SCPs.
Audit-store tamper attempt <i>(editor addition)</i>	CloudTrail: PutBucketPolicy, PutObjectLockConfiguration, DeleteObject* on the audit bucket	High	Alert the ISSO. The Compliance-mode lock blocks deletion regardless.
Signing-key deletion or disable <i>(editor addition)</i>	CloudTrail: ScheduleKeyDeletion, DisableKey	Critical	Cancel the deletion (CancelKeyDeletion) and escalate.
Blocked egress DNS query <i>(editor addition)</i>	Route 53 Resolver DNS Firewall logs	High	Isolate the workload and investigate.

8. Operational Sign-Off

This architecture provides defense in depth across storage, transport, execution, and export. No unencrypted data is stored locally, processing runs only in hardware-isolated and attested enclaves, and query-output provenance is signed by an HSM-backed asymmetric key. The platform is **designed to satisfy the control objectives of** DoD Cloud Computing SRG Impact Levels 5 and 6, the HIPAA Security Rule, SEC Rule 17a-4 and FINRA recordkeeping requirements, and NIST SP 800-53 Rev. 5.

Scope exclusion. This specification is not, and does not substitute for, a formal authorization or compliance determination. The following are out of scope: DoD Provisional Authorization (DISA) and the mission-owner ATO for IL5/IL6, the HIPAA risk analysis and risk management required under 45 CFR 164.308(a)(1), and the SEC Rule 17a-4(f) designated third party or undertaking requirements and the associated compliance assessment of the WORM store.

Role	Name	Signature	Date
System Owner	Rob Nock		
Security Architect			
Information System Security Manager (ISSM)			
Compliance / Privacy Officer			

Appendix A. Glossary

Term	Definition
CLAC	Cell-Level Access Control: permissions and dynamic masking applied to individual columns or cells, based on the caller's role and context.
RLAC	Row-Level Access Control: filters that restrict which rows a principal can see (e.g. by region or department).
k-anonymity	A property of released data in which every record or group is indistinguishable from at least $k - 1$ others. Here, any group with $N < 10$ is suppressed ($k = 10$).
DP	Differential Privacy: a mathematical privacy guarantee achieved by adding calibrated noise to query results. Cumulative privacy loss is bounded by a budget ϵ .
WORM	Write Once, Read Many: storage in which records cannot be modified or deleted during the retention period (here, S3 Object Lock in Compliance mode, 7 years).
HSM	Hardware Security Module: tamper-resistant hardware that generates, stores, and uses keys without exposing them. AWS KMS HSMs are FIPS 140-3 Level 3 validated.
IL5 / IL6	DoD Cloud Computing SRG Impact Levels. IL5 covers higher-sensitivity CUI and National Security Systems (unclassified). IL6 covers classified information up to SECRET.
RSASSA-PSS	RSA Signature Scheme with Appendix, Probabilistic Signature Scheme (RFC 8017). Used here with SHA-256, MGF1-SHA-256, and a 32-byte salt.
PCR	Platform Configuration Register: a measurement of enclave or boot components, reported in attestation documents.
VPCE	VPC Endpoint: a private connection from a VPC to an AWS service, with no internet path.

Appendix B. Control Mapping to NIST SP 800-53 Rev. 5

This is an indicative mapping to standard control identifiers. It is not a control-implementation statement or an authorization package.

Family	Control	Title	How this specification addresses it
AC	AC-3	Access Enforcement	Lake Formation / Ranger CLAC and RLAC (FS-1); IAM read-only policy (4.2)
AC	AC-4	Information Flow Enforcement	VPC endpoints, egress firewall, pixel-only viewport (4.1, 5C)
AC	AC-6	Least Privilege	Read-only execution roles; separation of KeyAdmin, signer, and auditor roles (4.2, 6.1)
AC	AC-17	Remote Access	Analyst access only through managed AppStream 2.0 / VDI sessions (1.2)
AU	AU-2 / AU-3	Event Logging / Content of Audit Records	SQL, IAM context, attestation, row count, and object versions logged (FS-4)
AU	AU-6	Audit Record Review, Analysis, and Reporting	Security Hub / SIEM triggers (7)
AU	AU-9	Protection of Audit Information	CloudTrail + S3 Object Lock in Compliance mode (FS-4)
AU	AU-10	Non-repudiation	KMS RSASSA-PSS signatures over canonical provenance (4.3, 5A-5B)
AU	AU-11	Audit Record Retention	7-year WORM retention (FS-4)
AU	AU-12	Audit Record Generation	Per-query provenance record and CloudTrail Sign events (4.3.2)
SC	SC-4	Information in Shared System Resources	RAM-only processing; no swap or core dumps (FS-2, 6.2)
SC	SC-7	Boundary Protection	Private VPC, no IGW/NAT, DNS and network firewall (4.1)
SC	SC-8	Transmission Confidentiality and Integrity	aws:SecureTransport enforcement; TLS to S3 and KMS (4.2)
SC	SC-12 / SC-13	Cryptographic Key Establishment and Management / Cryptographic Protection	KMS key policy and manual rotation; FIPS-validated HSMs (6.1)
SC	SC-28	Protection of Information at Rest	SSE-KMS on the data and audit buckets (3)
SC	SC-39	Process Isolation	Nitro Enclaves / confidential compute (FS-2)
SI	SI-4	System Monitoring	GuardDuty, CloudTrail, and auditd detections (7)
SI	SI-7	Software, Firmware, and Information Integrity	Enclave attestation (PCRs); signed provenance verification (FS-2, 5B)
SI	SI-16	Memory Protection	ASLR, ptrace disabled, kptr/dmesg restrictions (6.2)
SI	SI-19	De-identification	Dynamic masking; k-anonymity / DP release gate (FS-1, FS-3)
IA	IA-2	Identification and Authentication (Organizational Users)	OIDC federation / IAM Roles Anywhere (3)
IA	IA-3	Device Identification and Authentication	Hardware-backed X.509 client certificates (3)
IA	IA-5	Authenticator Management	Short-lived (≤ 1 h) credentials; no static keys (3)