Agent Context Isolation for Private Data Analysis

Mary Maller
Inversed Tech

Abstract

AI agents operating over shared databases introduce a privacy risk: private data that enters an agent's context window may be reproduced in a subsequent interaction with a different user, regardless of intent or policy. We present **CAPE** (Context-Aware Private Execution), a protocol that allows users to obtain AI-assisted analysis of permissioned data without the data ever entering the agent's context.

The general idea is that the agent is given access to synthetic data that models the structure of the real data with all sensitive values replaced by simulated ones. It uses this to generate, test, and debug a script that achieves the user's aims and that can be executed in a sandboxed environment. The script is never executed by the agent and thus the real data never enters its context. The script is instead executed by a trusted gateway component, which we call Threshold, that returns the result directly to the user. Threshold is purpose-built software that, unlike generative AI, can be formally audited.

The agent must obtain user approval before Threshold will execute the script. The user approves via a purpose-built client, which assembles and signs an execution token without agent involvement. Threshold delivers the result directly to the user over a Server-Sent Events connection. The agent receives only an HTTP 202 acknowledgement before any data is accessed, and no further signal.

All events are recorded in a tamper-proof append-only audit log whose integrity can be verified by external parties without access to the full log.

This design note describes the protocol's threat model, connection lifecycle, security properties, and audit design, and illustrates it using the Chinook database as a case study.

Contents

1 Introduction

AI agents operating through a shared database can be a useful tool. However, once private data enters an agent's context window, the human loses all control over how that data is used. It may be maliciously used in a training model, or it may be unintentionally leaked in a subsequent interaction with a different user. As a real world example, in 2023, Samsung engineers accidentally leaked confidential source code and internal meeting notes by pasting them into ChatGPT for analysis [5]. It is difficult even for honest providers to design an LLM without these privacy risks.

This design note describes a protocol for agent context isolation that we call **CAPE** (Context-Aware Private Execution). CAPE allows users to request AI-assisted analysis of data without exposing that data to the agent. The agent queries are limited to a synthetic database. A synthetic database is a copy of the database structure where all private data fields have been randomly generated and have no statistical resemblance to the real data. The agent generates a script, potentially via an interactive session with the user, testing and iterating against the synthetic database until it is ready to execute on the real data. The agent learns no private information during this process because the only data it has access to is synthetic.

Upon finalising the script, the agent requests execution from a gateway service we call Threshold. Threshold is a proxy server that sits between AI agents and the systems they access and enforces per-agent permissions. When the agent requests an execution it receives an acknowledgement, but no additional output. Threshold executes scripts in a sandboxed environment and delivers the result directly to the user. Threshold stores all communications between itself and the agents in a tamper-proof log that can be reviewed by auditors or consulted during incident response. Threshold is purpose-built software, not an LLM, and therefore auditors can ensure that it is implemented in a manner that does not leak user data.

To ensure good user experience, we introduce the user's client as a third component. The user's client handles the user-facing side of the protocol: it manages the user's signing key (never shared with the agent), presents scripts for human review, and interacts with the agents and Threshold on the user's behalf. Like Threshold, the client is purpose-built software and not an LLM. Upon receiving an approval request for an execution, the client opens a direct SSE channel with Threshold to which the result will be delivered, bypassing the agent completely.

The key property CAPE achieves is *context isolation*: private data never enters the agent's context window and therefore cannot leak into future interactions, regardless of who the agent subsequently serves. This satisfies the spirit of GDPR Article 25 by making data protection a structural property of the system rather than a policy requirement.

We use the Chinook database [21] as a running case study throughout this document. Chinook is an open-source sample database modelling a digital music store, widely used in commercial and educational contexts. Its schema and permission structure are described in Section 2; a complete worked example applying CAPE to Chinook is given in Section 6.

2 Case Study: The Chinook Database

Chinook [21] is an open-source sample database modelling a digital music store. It is widely used in commercial and educational settings. The schema comprises 11 tables covering the store's catalogue, customers, employees, and sales transactions.

2.1 Schema overview

Table	Contents
`Artist`	Artist names
`Album`	Albums linked to artists
`Track`	Tracks with duration, price, genre, and media type
`Genre`	Genre classifications
`MediaType`	Media format types
`Playlist`	Named playlists
`PlaylistTrack`	Playlist-to-track join table
`Customer`	Customer name, email, phone, billing address
`Employee`	Employee name, title, hire date, reporting structure
`Invoice`	Invoice header: customer, date, billing address, total
`InvoiceLine`	Line items: track, unit price, quantity

2.2 Read permission structure

We partition the schema into three permission tiers based on commercial sensitivity.

Tier	Tables	Rationale
Public	`Artist`, `Album`, `Track`, `Genre`, `MediaType`, `Playlist`, `PlaylistTrack`	Catalogue data; no personally identifiable information, no commercial sensitivity
Personal	`Customer`, `Employee`	Names, contact details, addresses, organisational structure
Financial	`Invoice`, `InvoiceLine`	Transaction history and revenue figures

An agent operating on behalf of any user may freely query the public tier. The personal and financial tiers require read permission. The agent does not carry those permissions; all personal and financial data is accessed exclusively through CAPE.

3 Threat Model

The threat model addresses risks arising from an AI agent that handles requests from multiple users over time, where those users have different read permissions on a shared database. We identify four security properties that a protocol must achieve: *private execution*, *human approval binding*, *execution rate binding*, and *secure logging*.

3.1 Trust boundaries

Actor	Trusted?
Threshold (Gateway)	Yes. Purpose-built software; not an LLM; follows strict protocol rules
User's client	Yes. Purpose-built software; holds signing key, never shared with agent
User	Yes. Within their read permissions
AI agent	No. Treated as untrusted; receives no private data

3.2 Assumptions

To simplify our model we make two assumptions that might not hold in practice.

- **No side channels exist.** The agent cannot measure the length of time a user takes between requests; nor the power consumption of their machine; nor other side channels that might leak information about the state of the execution.
- **Honest user.** The user does not communicate the execution result to the agent after receiving it. CAPE prevents accidental leakage by design; it does not constrain what a user chooses to share deliberately.

3.3 Private Execution

We require that agents learn no private data from their interactions in CAPE. This includes secondary threats that arise if the agent receives any signal correlated with the private data after execution. Even a binary success/failure response can be exploited: a script of the form

```
if total_revenue < 10000:
    raise Exception()
```

causes the error channel to function as a 1-bit oracle over private data. More generally, any feedback the agent receives about execution outcome can be used to extract information through repeated queries.

Under our model assumptions, we say a protocol achieves *private execution* if for any probabilistic polynomial-time agent $\mathcal{A}$ and any two private datasets D_0, D_1 consistent with the public schema S:

$$\left| \Pr[\mathcal{A}(\text{view}_0) = 1] - \Pr[\mathcal{A}(\text{view}_1) = 1] \right| \leq \text{negl}(\lambda)$$

where view_b is the agent's view in an execution where the private dataset is D_b and λ is the security parameter. That is, no efficient agent can distinguish an execution over D_0 from one over D_1.

3.4 Human approval binding

We say a protocol achieves *human approval binding* if no efficient agent can cause Threshold to execute a script the user has not explicitly approved, except with negligible probability.

3.5 Execution rate binding

Without access control on execution, an agent could trigger a large number of database queries, causing availability or cost problems.

We say a protocol achieves *execution rate binding* if no efficient agent can cause Threshold to perform more executions than the user has explicitly authorised in that session, except with negligible probability.

3.6 Secure logging

The audit log records every script submission and execution outcome. Without tamper protection, a compromised Threshold could suppress or alter log entries to conceal misbehaviour — executing a different script than the one the user approved, or omitting an execution from the record entirely.

We adopt the following security definitions from Dowling et al. [9], Figure 9. A logging scheme LS is *tamper-evident* if no probabilistic polynomial-time adversary $\mathcal{A}$ can win either of the following experiments with non-negligible probability:

proof-coll: $\mathcal{A}$ outputs $(e, \vec{E}, F, \vec{M}, pk)$; the experiment returns 1 iff

$$\begin{aligned}&\mathsf{CheckEntries}(\vec{E}, F, pk) = 1 \wedge \\ &\mathsf{CheckMembership}(e, F, \vec{M}, pk) = 1 \ \wedge \ e \notin \vec{E}.\end{aligned}$$

entry-cons: $\mathcal{A}$ outputs $(\vec{E}_0, \vec{E}_1, F_0, F_1, \vec{C}, pk)$; the experiment returns 1 iff

$$\begin{aligned}&\mathsf{CheckConsistency}(F_0, F_1, \vec{C}, pk) = 1 \wedge \\ &\mathsf{CheckEntries}(\vec{E}_0, F_0, pk) = 1 \wedge \\ &\mathsf{CheckEntries}(\vec{E}_1, F_1, pk) = 1 \ \wedge \ \vec{E}_0 \not\prec \vec{E}_1.\end{aligned}$$

The first experiment captures collision resistance of proofs: $\mathcal{A}$ cannot produce a valid membership proof for an entry not contained in the list represented by the same fingerprint. The second captures consistency of entries: $\mathcal{A}$ cannot produce a consistency proof between two fingerprints where the entries of the first are not a prefix of the entries of the second.

4 The CAPE Protocol

4.1 Overview

CAPE (Context-Aware Private Execution) separates the roles of script authorship (the agent) and script execution (Threshold). The agent never touches private data; Threshold never exposes it to the agent. The user is the only party that sees both the script and its output.

The protocol involves a third component: the **user's client**. The client is purpose-built software — not an LLM — that implements the user-facing side of the protocol. It manages the user's private signing key, which is never shared with the agent. Like Threshold, the client follows strict protocol rules: it signs payloads only when the user explicitly approves, enforces the `user_id` identity binding, and does not relay private data to the agent. The separation between agent and client is a security boundary: an agent that could influence the client's key material or signing behaviour could self-authorise executions.

The database schema is treated as public. The agent uses it to write correct scripts without requiring any access to the underlying data.

4.2 Execution flow

1. **Script development.** The agent writes an analysis script using the public schema. It iterates against the synthetic dataset until satisfied the script is correct.

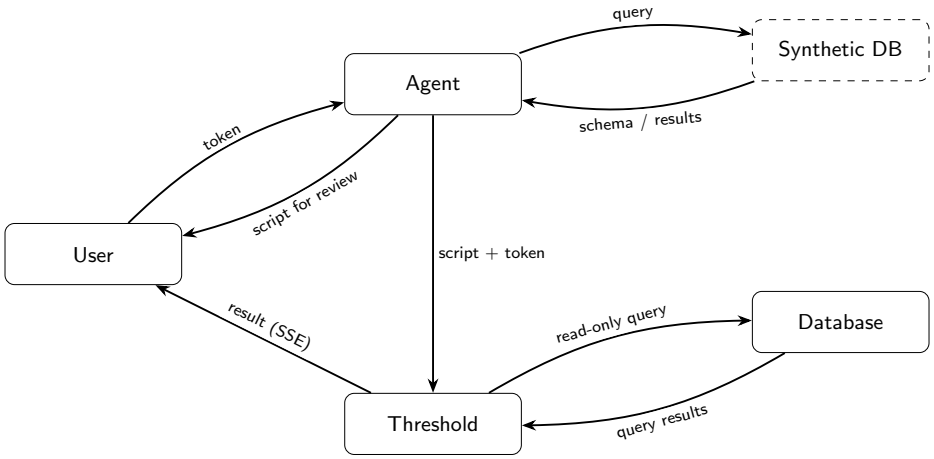

Figure 1: CAPE protocol flow. The user reviews and approves the script via the client; the agent then submits it to Threshold and receives only an HTTP 202 acknowledgement. The result travels exclusively from Threshold to the user via SSE, never passing through the agent.

2. **Script review and approval.** The agent presents the script together with its proposed execution parameters — timeout and resource bounds — to the user for review. If the agent does not propose bounds it must supply defaults from its own configuration. The user sees the script text and parameters in human-readable form and approves or rejects in a single interaction. The user never reviews raw cryptographic values such as the script hash or `execution_id`; these are managed transparently by the client. If the user rejects, the agent must revise the script or parameters and re-present; no token is issued and no execution occurs.

   If the user approves, the client handles the token mechanics without agent involvement: it hashes the script, generates a cryptographically pseudorandom `execution_id` (minimum 128 bits), appends `user_id` from its own certificate, and signs the complete payload:

   $$\text{token} = \text{Sign}_{\text{user}}\big(\texttt{hash(script)} \| \texttt{execution\_id} \| \texttt{execution\_timeout} \| \texttt{resource\_bounds} \| \texttt{user\_id}\big)$$

   The client uses a hybrid signing scheme, producing both an ECDSA P-256 signature and an ML-DSA signature (NIST FIPS 204) over the same payload; Threshold considers a token valid only if both signatures verify. This ensures the approval remains binding if either algorithm is broken, including by a future quantum adversary capable of breaking ECDSA. Because `user_id` is appended by the client and covered by both signatures, the agent cannot substitute a different identity. `execution_timeout` commits the maximum wall-clock time Threshold will allow the script to run; this may be long — for example, a complex analysis intended to run overnight. Threshold may not execute the script under looser limits than those committed in the token. The client provides the token and `execution_id` for use in the next step.

3. **Result stream open.** Before handing the token to the agent, the user's client opens a Server-Sent Events (SSE) connection to Threshold:

`GET /admin/cape/stream/{execution_id}`

authenticated with the user's admin credentials. Threshold authenticates the user and verifies that they hold sufficient read permissions to receive the execution result; if either check fails the connection is refused. Threshold also checks that `execution_id` is not already present in the used-token store; if it is, the connection is refused, preventing a second SSE connection from being parked against an already-used token. Threshold records the time the connection was opened and enforces a configurable submission window: if the agent does not submit within this window, Threshold closes the connection with an `expired` event. The submission window is a Threshold configuration parameter; it is not part of the token and the agent has no influence over it. Threshold then holds the connection open, waiting for the corresponding execution to complete. The result will be pushed directly over this connection and not written to any persistent store.

4. **Submission.** The client passes the token to the agent. The agent submits the pair (*script*, *token*) to Threshold and receives an HTTP 202 acknowledgement. This is the last communication the agent receives regarding this request.

5. **Validation.** Threshold verifies:

   - the token carries valid signatures from the issuing user — both ECDSA and ML-DSA — whose certificate chains Threshold verifies against its configured trust roots; certificate revocation checking is an implementation concern outside the scope of this protocol;
   - `user_id` in the token payload matches the identity in the signing certificate, confirming the user appended their own identity and the agent did not substitute it;
   - `hash(script)` matches the hash committed in the token;
   - `execution_id` matches the open SSE connection for this user;
   - the submission window has not elapsed (enforced from the SSE open timestamp, not the token);
   - the user's read permissions are sufficient to access the target data;
   - `execution_id` has not been used in a prior submission. Threshold acquires an exclusive lock on its used-token store, checks for the presence of `execution_id`, and if absent inserts it before releasing the lock. A token whose `execution_id` is already present is rejected as a replay. Used-token entries are pruned once the submission window for their `execution_id` has elapsed, bounding the size of the store.

   If any check fails, the request is rejected. The agent is not notified of the reason. Threshold extracts `resource_bounds` and `execution_timeout` from the token and applies them to the execution environment before the script begins.

6. **Execution.** Threshold executes the script it received and verified in step 5 — not a cached or transformed version — in a sandboxed environment against a read-only database connection scoped to the user's read permissions. The read-only constraint is enforced at the database role level, not by policy.

7. **Result delivery.** The output is pushed directly over the user's open SSE connection. The connection then closes. The result is not written to the audit log or any persistent store. The agent receives nothing.

4.3 Execution environment

CAPE does not specify a script language. The choice — SQL, Python, sandboxed JavaScript, or any other — is a deployment decision that does not affect the protocol's privacy properties. What matters is that the execution environment satisfies the following interface requirements, regardless of language:

- **Read-only database access.** Enforced at the database role level, not by the script runtime. The role granted to Threshold for CAPE executions must carry no write privileges.
- **No network access.** The script must not be able to make outbound network connections. A script that exfiltrates query results to an external endpoint would bypass the SSE delivery guarantee entirely.
- **No filesystem access.** The script must not be able to read from or write to the host filesystem.
- **Execution isolation.** Each script execution must be isolated from others. One execution must not be able to observe or influence the state of another. Because all executions are read-only, a user may hold multiple simultaneous open SSE connections under different `execution_id`s without risk of conflicting writes or inconsistent state. Concurrent executions are permitted.
- **Resource bounds.** CPU time and memory usage must be bounded per execution and are committed into the token alongside `execution_timeout`. Bounds are proposed by the agent and approved by the user at step 2; if the agent does not propose bounds, it is expected to supply defaults from its own configuration. Threshold enforces the approved limits. The distinction between `execution_timeout` (a hard wall-clock limit enforced by Threshold) and CPU time (enforced by the execution environment) is intentional: a long-running script may consume little CPU while waiting on I/O, and a user may wish to permit this — for example, an overnight batch analysis.
- **Language-specific static validation.** Before execution against either the synthetic or live dataset, Threshold performs language-specific checks — at minimum, syntax validation and confirmation that no write operations are present. The specific checks are an implementation concern; the requirement that they occur before any data is accessed is not. Because Threshold holds no write privileges on the database, any write operation that evades static validation will fail at execution time with a database-level permission error.

4.4 Connection lifecycle

In all cases below, Threshold writes an outcome entry to the tamper-proof audit log recording the terminal state of the execution. Script submission is recorded as an intent entry at the time it occurs, following the write-ahead log pattern: an intent is committed before execution begins, and an outcome is written for every terminal state. Token issuance is not recorded by Threshold because Threshold does not participate in it — the token is produced by the user's client and first seen by Threshold at submission.

Submission window elapses before agent submits. If the agent does not submit within Threshold's configured submission window, Threshold pushes an `expired` event over the open SSE connection and closes it. The outcome is logged with status `expired`. To retry, the user opens a new stream and the client issues a new token.

Execution timeout reached. If the script is still running when `execution_timeout` elapses, Threshold aborts execution, discards any partial result, pushes a `timeout` event over the SSE connection, closes it, and logs the outcome with status `timeout`. The agent is not notified.

No open connection at submission time. Threshold validates at step 5 that an SSE connection is open for the given `execution_id` before executing the script. If no connection is present, the submission is rejected and logged with status `denied`. The agent is not notified of the reason. To retry, the user opens a new stream and the client issues a new token.

User disconnects before execution completes. If the SSE connection drops while the script is still running, Threshold aborts execution, discards any partial result, and logs the outcome with status `cancelled`. Nothing is buffered. To retry, the user opens a new stream, the client issues a new token, and the agent resubmits.

User cancels execution. The user may cancel a running execution at any time by sending an authenticated request:

```
DELETE /admin/cape/execute/{execution_id}
```

Threshold aborts the script, closes the SSE connection, discards any partial result, and logs the outcome with status `cancelled`. The agent is not notified.

Execution fails. If the script raises an error or the database connection fails, Threshold pushes an error event over the SSE connection, closes it, and logs the outcome with status `error`. The error is visible only to the user. The agent, having already received HTTP 202, is not informed.

Execution succeeds. Threshold pushes the result over the SSE connection, closes it, and logs the outcome with status `ok`. The script hash and `execution_id` are recorded, linking the outcome back to the submission intent entry.

Threshold crashes mid-execution. If Threshold crashes after writing the intent entry but before execution completes, no outcome entry is written. The intent entry remains in the log with no corresponding outcome. An unresolved intent — an intent entry with no matching outcome — is itself informative: it records that execution was initiated and that delivery of the result cannot be confirmed. This is a deliberate design choice. Writing a crash outcome on restart would require Threshold to read the log at startup to identify unresolved intents; leaving them dangling avoids granting Threshold read access to the log entirely.

4.5 Audit log

Every CAPE event is recorded in a tamper-proof audit log maintained by Threshold. Tamper-proof means the log is structured as an append-only Merkle tree following the Certificate Transparency model (RFC 9162). Each entry is a leaf in the tree; the tree root is periodically signed and published as a Signed Tree Head (STH). Any external party can verify that a specific entry is genuinely present using an inclusion proof ($O(\log n)$ hashes), and that the log has only ever grown — never had entries removed or altered — using a consistency proof between two STHs. Neither proof requires access to the full log.

CAPE produces two categories of entry, following the write-ahead log pattern described above:

Intent entries are written at script submission, before execution begins. Threshold writes one intent entry per submission, recording the `execution_id`, the script hash, the acting user, and a timestamp. The intent entry exists in the log even if the corresponding execution never completes.

Outcome entries are written once for every terminal state. An outcome entry carries a `ref_seq` linking it back to its corresponding submission intent entry, and a status drawn from: `ok`, `error`, `cancelled`, `denied`, `expired`, or `timeout`. Together the intent and outcome entries provide a complete, linked record of every execution attempt.

Log entries record metadata — who, what, when, and status — but not the content of query results. Threshold is the sole author of all log entries; the agent has no code path to write, modify, or suppress an entry.

4.6 Properties

- **Context isolation.** Private data never enters the agent's context window.
- **No oracle channel.** The agent receives no signal correlated with execution output or private data values.
- **Script binding.** The execution token is cryptographically bound to the exact script the user reviewed. A modified script requires a new token and a new user approval.
- **Access enforcement.** Threshold connects to the database under a role limited to the data the issuing user is authorised to read.
- **Auditability.** Script submission and every execution outcome are recorded as linked intent and outcome entries in the tamper-proof audit log. Inclusion and consistency proofs allow external parties to verify log integrity without access to the full log.
- **User identity binding.** The `user_id` in the token payload is set by the user's client from its own certificate and cross-checked by Threshold against the signing certificate identity. The agent cannot substitute a different identity: a token carrying a mismatched `user_id` is rejected.

5 Testing Execution on Synthetic Data

A synthetic dataset is maintained with the same schema as the live database, populated entirely with fabricated records — not anonymised or sampled from real data. Because the synthetic dataset contains no sensitive information, the agent has direct read-only access to it and does not go through Threshold — it may query the schema, introspect table structure, and test scripts freely without any approval mechanism.

The agent uses the synthetic dataset to develop and test its script before submitting it for real execution. This provides a debugging feedback loop without any privacy risk: the agent can observe output format, catch logic errors, and verify behaviour on edge cases (nulls, type mismatches, boundary values) that the synthetic data is constructed to include. The quality of this assurance depends directly on how well the synthetic data covers the distribution of the live data: synthetic data that is too clean or too uniform will not surface scripts that silently fail or produce

misleading results on real inputs. Operators are responsible for maintaining synthetic datasets that are representative enough to make testing meaningful.

In the Chinook case study, the synthetic dataset mirrors the full schema. Fabricated customer names, addresses, and invoice records allow the agent to verify that a revenue analysis script joins correctly and produces well-formed output, without touching any real customer or transaction data.

Before execution on either dataset, Threshold performs language-specific static validation — at minimum, syntax checking and confirmation that no write operations are present. The specific checks depend on the script language in use; the requirement that they occur before any data is accessed is a protocol invariant.

6 Case Study: Applying CAPE to Chinook

This section applies the CAPE protocol to the Chinook database introduced in Section 2, walking through a complete protocol execution with concrete artefacts at each step.

6.1 Scenario

A user with permission to read the financial tier wishes to analyse revenue by genre over the past year. The query requires joining `Invoice`, `InvoiceLine`, `Track`, and `Genre` — spanning both the financial and public tiers.

6.2 Step 1: Script development

The agent inspects the public schema and constructs the following script, iterating against the synthetic Chinook dataset (see Section 5) until it produces well-formed output:

```
SELECT g.Name        AS Genre,
       SUM(il.UnitPrice * il.Quantity) AS Revenue
FROM   Invoice      i
JOIN   InvoiceLine  il ON il.InvoiceId = i.InvoiceId
JOIN   Track        t  ON t.TrackId    = il.TrackId
JOIN   Genre        g  ON g.GenreId    = t.GenreId
WHERE  i.InvoiceDate >= CURRENT_DATE - INTERVAL '1 year'
GROUP  BY g.Name
ORDER  BY Revenue DESC;
```

The agent verifies on the synthetic dataset that the join produces one row per genre, that the `Revenue` column is non-null, and that the ordering is correct. No real customer or invoice data is accessed at this stage.

6.3 Step 2: Script review and approval

The agent presents the script to the user together with proposed execution parameters. The client displays the following for the user to review:

```
Script:
  SELECT g.Name AS Genre, SUM(il.UnitPrice * il.Quantity) AS Revenue
  FROM Invoice i
  JOIN InvoiceLine il ON il.InvoiceId = i.InvoiceId
```

```
  JOIN Track t        ON t.TrackId    = il.TrackId
  JOIN Genre g        ON g.GenreId    = t.GenreId
  WHERE i.InvoiceDate >= CURRENT_DATE - INTERVAL '1 year'
  GROUP BY g.Name
  ORDER BY Revenue DESC;

Timeout:  30 seconds
CPU:      10 seconds
Memory:   128 MB

Approve? [y/n]
```

The user reads the script, confirms it performs only a revenue aggregation, and approves. The client generates an `execution_id`, hashes the script, appends the user's identity from its certificate, and signs the token. The agent never sees the token payload being assembled.

6.4 Steps 3–7: Execution

The client opens an SSE connection to Threshold. Threshold verifies the user's financial-tier read permissions and records the connection timestamp. The client hands the token to the agent, which submits the `(script, token)` pair to Threshold. The agent receives:

```
HTTP/1.1 202 Accepted
```

This is the last message the agent receives. It has no information about whether execution succeeded, what the result contained, or when the script ran.

Threshold validates the token signatures, cross-checks the script hash, confirms the `execution_id` is fresh, and executes the script under a database role scoped to the user's financial-tier read permissions.

6.5 Result

The revenue breakdown is pushed over the SSE connection directly to the user (figures illustrative):

```
Genre               Revenue
------------------  -------
Rock                826.65
Latin               382.14
Metal               261.36
Alternative & Punk  244.08
Jazz                212.32
...
```

The agent sees none of these figures. The result is not written to the audit log or any persistent store; the log records only that a script with the approved hash was submitted and completed with status `ok`.

7 Related Work

The problem of preventing sensitive data from entering an AI agent's context sits at the intersection of several active research areas. We survey the most relevant prior work and explain how each approach differs from the protocol described here.

7.1 Confidential Computing and TEE-Based Inference

The dominant industry approach to private AI inference is to run the model inside a *Trusted Execution Environment* (TEE) — a hardware-isolated enclave (Intel SGX, AMD SEV, NVIDIA H100 Confidential GPU) such that the data is encrypted even during inference, and neither the cloud operator nor the infrastructure can observe it. Recent academic work in this vein includes confidential prompting for cloud LLM inference [14], analysis of performance and cost tradeoffs across CPU and GPU TEEs [8], and a dual-privacy construction combining TEEs with cryptographic techniques [11]. A practical deployment survey is given in [19]. Industry platforms pursuing this approach include Opaque Systems [17], built on research from the UC Berkeley RISELab; Anjuna Security [4], which offers enclave-based confidential AI infrastructure; and Edgeless Systems [12], whose Continuum platform runs LLM inference on confidential NVIDIA H100 GPUs.

The TEE approach solves a different problem from the one addressed here. It protects data from the *infrastructure operator* — the cloud provider or system administrator who controls the hardware. Inside the enclave, the LLM processes data in plaintext; the TEE prevents the operator from observing it, but the agent itself has full access to the decrypted content.

For stateless inference — a single request with context discarded after the response — this containment is sufficient: private data does not carry over to future sessions. However, agentic deployments typically maintain persistent state: conversation history, memory stores, RAG indices, or tool call logs. In such systems a TEE does not prevent data from one user's session appearing in another's, because both sessions operate inside the same enclave with access to the same memory layer. The protocol described here addresses this gap: by ensuring private data never enters the agent's context at all, it eliminates cross-session leakage regardless of whether a memory layer is present.

7.2 Agent Isolation and Sandboxing

IsolateGPT [24], presented at NDSS 2025, proposes a hub-and-spoke architecture in which LLM-powered applications execute in isolated environments with well-defined interfaces to the host system. The system defends against prompt injection, data exfiltration by compromised apps, and inadvertent exposure across app boundaries, with reported overhead below 30%. CELLMATE [6] applies a similar sandboxing approach to browser-based AI agents. The ACE framework [1] provides a broader security architecture for LLM-integrated application systems, formalising trust boundaries across components.

These works focus on isolating the *execution environment* of an agent from the host system, rather than isolating the agent's *context* from private data. An agent in a sandboxed environment may still receive sensitive data as input and retain it across interactions.

7.3 Access Control for AI Agents

AgentBound [2] is the first access control framework designed specifically for MCP servers. Inspired by the Android permission model, it supports declarative access policies with reported 80.9% automatic policy generation accuracy from code, at negligible runtime overhead. Cerbos [7] offers

dynamic fine-grained authorisation for MCP servers. Protecto [20] implements context-based access control (CBAC) for agents, making access decisions based on the identity of the requesting agent, the stated purpose, and the data classification. Oracle's Deep Data Security, introduced in Database 26ai [18], embeds identity-aware access control directly in the database layer for agentic workloads.

Access control frameworks govern *which data an agent may query*. They do not address what happens to that data once it has been retrieved — in particular, they do not prevent it from persisting in the agent's context and leaking into future interactions. The protocol described here is complementary: access control determines what the agent is permitted to read; context isolation ensures that even permitted data does not accumulate in the agent's context window.

7.4 Differential Privacy

Differential privacy (DP) adds calibrated noise to query outputs so that the presence or absence of any individual record cannot be inferred from the result. Google's VaultGemma [15] is a 1B-parameter LLM trained with user-level differential privacy. Amazon has applied DP to the decoding stage of LLM inference [3]. However, DP provides a statistical guarantee over a *population* of queries and is not a natural fit for the single-user analysis scenario considered here. It also requires the privacy budget to be set in advance, and recent work has demonstrated that LLM feedback can reduce the effective privacy guarantee in practice [10].

7.5 Cryptographic Approaches

Fully homomorphic encryption (FHE) allows computation over encrypted data without decryption [13]. Secure multi-party computation (SMPC) distributes computation across parties such that no single party sees the full input [22]. Zero-knowledge proofs (ZKPs) allow a prover to demonstrate that a computation was performed correctly without revealing inputs [25]. These techniques provide strong cryptographic guarantees but remain computationally expensive for large models and general-purpose analysis scripts. The protocol described here achieves context isolation without cryptographic overhead, relying instead on architectural separation between script authorship and execution.

7.6 Summary Comparison

Table 1 summarises the six approaches against the properties most relevant to agent context isolation.

7.7 Governing Agent Capabilities

Recent work on cryptographic capability binding for AI agents [16] proposes signing the complete capability set of an agent at deployment time, making its authorised operations verifiable. This is complementary to the execution token mechanism described in Section 4, which binds a token to the hash of a specific script rather than to a standing capability declaration. The systems security foundations survey in [23] provides a comprehensive treatment of trust, attestation, and principal hierarchies in agentic systems.

	Context isolation	Operator privacy	Oracle-free	Hardware-independent	No crypto overhead	Per-execution approval	Auditable
TEE-based inference	∼	✓	×	×	✓	×	×
Agent sandboxing	×	×	×	✓	✓	×	×
Access control	×	×	×	✓	✓	×	∼
Differential privacy	∼	×	∼	✓	∼	×	×
FHE / ZKP / SMPC	✓	✓	✓	✓	×	×	∼
CAPE (this work)	✓	×	✓	✓	✓	✓	✓

Table 1: Comparison of related approaches. ✓ = satisfied; × = not satisfied; ∼ = partially or conditionally. Context isolation: private data never enters the agent's context window. Oracle-free: the agent receives no signal correlated with execution output. Per-execution approval: every execution requires a signed token bound to the exact script bytes. Auditable: execution events are recorded in a tamper-proof log verifiable by external parties. TEE-based inference achieves context isolation only for stateless deployments. Differential privacy reduces but does not eliminate the oracle channel, and adds overhead at training time. Access control and FHE/ZKP/SMPC may support audit logging but do not provide externally verifiable tamper-proof logs as a protocol property. CAPE does not protect operator privacy against a compromised Threshold.

8 Discussion and Conclusion

CAPE provides a structural guarantee that private data never enters the agent's context window. It does not prevent a user from deliberately sharing results with the agent after receiving them. The protocol is designed to eliminate *accidental* leakage — the structural risk that an agent retains and reproduces private data across sessions — not to constrain what a user chooses to do with their own query results.

By design, the database schema is public. In deployments where the schema itself is sensitive, additional measures would be required before applying CAPE.

Threshold is the single point of trust. A compromised Threshold could log results, forward them to the agent, or execute a script other than the one the user approved. Operators should treat the integrity of the Threshold process as the primary security concern. This concentration of trust is deliberate: unlike the AI agent, whose behaviour is probabilistic, context-dependent, and cannot be fully predicted from its inputs, Threshold is purpose-built software. Its complete behaviour is specified by its source code; it has no generative capacity to hallucinate, fabricate results, or be prompted into unexpected actions. The same is true of the user's client. Relocating trust from an agent to Threshold and the client is therefore a narrowing of the trust surface — from a system whose behaviour is inherently difficult to bound, to components that can be formally specified, audited, and verified.

CAPE achieves its guarantee through architectural separation rather than cryptography: the agent writes code, Threshold runs it, and the result travels only to the user. The user's signature over the token payload is a first-class security mechanism: it is what prevents the agent from self-authorising execution. Without the user's private key, the agent cannot produce a valid token,

and Threshold will not proceed. The interactive token binds execution to a specific script and a specific user, preventing the agent from triggering queries autonomously. The SSE delivery model is designed so that the result is not written to any persistent store, eliminating any artefact the agent could retrieve. Together these properties close the context leakage channel without requiring hardware enclaves, homomorphic encryption, or differential privacy.

The Chinook case study illustrates that the protocol is viable with an off-the-shelf relational database and standard web infrastructure. CAPE is intentionally script-language agnostic: the execution environment requirements are stated as interface properties (read-only access, no network or filesystem access, isolation, resource bounds), leaving the language choice as a deployment decision.

References

[1] ACE: A security architecture for LLM-integrated application systems, 2025. https://arxiv.org/abs/2504.20984.

[2] AgentBound: Fine-grained access control for MCP servers, 2024. https://arxiv.org/abs/2510.21236.

[3] Amazon Science. Differentially private decoding in large language models. Amazon Science. https://assets.amazon.science/, 2024.

[4] Anjuna Security. Confidential AI infrastructure. https://www.anjuna.io/, 2025.

[5] Bloomberg. Samsung bans staff's AI use after spotting ChatGPT data leak. Bloomberg News, 2 May 2023, 2023. https://www.bloomberg.com/news/articles/2023-05-02/samsung-bans-chatgpt-and-other-generative-ai-use-by-staff-after-leak.

[6] ceLLMate: Sandboxing browser AI agents, 2024. https://arxiv.org/abs/2512.12594.

[7] Cerbos. Dynamic authorisation for AI agents and MCP servers. https://www.cerbos.dev/blog/mcp-authorization, 2025.

[8] Confidential LLM inference: Performance and cost across CPU and GPU TEEs, 2025. https://arxiv.org/abs/2509.18886.

[9] Benjamin Dowling, Felix Günther, Udyani Herath, and Douglas Stebila. Secure logging schemes and certificate transparency. In *21st European Symposium on Research in Computer Security (ESORICS)*, pages 140–158, 2016. https://eprint.iacr.org/2016/452.

[10] Differential privacy reversal via LLM feedback. https://medium.com/@instatunnel/it-162aee1dbfe5, 2026.

[11] Towards confidential and efficient LLM inference with dual privacy protection, 2025. https://arxiv.org/abs/2509.09091.

[12] Edgeless Systems. Continuum confidential AI. https://www.edgeless.systems/, 2025.

[13] Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. IACR Cryptology ePrint Archive, Report 2021/783, 2021. https://eprint.iacr.org/2021/783.

[14] In Gim, Caihua Li, and Lin Zhong. Confidential prompting: Privacy-preserving LLM inference on cloud, 2024. https://arxiv.org/abs/2409.19134.

[15] Google Research. VaultGemma: The world's most capable differentially private LLM. Google Research Blog. https://research.google/blog/vaultgemma-the-worlds-most-capable-differentially-private-llm/, 2025.

[16] Governing dynamic capabilities: Cryptographic binding for AI agents, 2026. https://arxiv.org/abs/2603.14332.

[17] Opaque Systems. Confidential AI platform. https://www.opaque.co/, 2025.

[18] Oracle. Oracle deep data security: Identity-aware data access control for agentic AI. Oracle Database 26ai. https://blogs.oracle.com/database/, 2025.

[19] Privacy-preserving LLM inference in practice. IACR Cryptology ePrint Archive, Report 2026/105, 2026. https://eprint.iacr.org/2026/105.

[20] Protecto. Context-based access control for AI agents. https://www.protecto.ai/, 2025.

[21] Luis Rocha. Chinook database. GitHub. https://github.com/lerocha/chinook-database, 2023.

[22] Secure multiparty generative AI, 2024. https://arxiv.org/abs/2409.19120.

[23] Systems security foundations for agentic computing. IACR Cryptology ePrint Archive, Report 2025/2173, 2025. https://eprint.iacr.org/2025/2173.

[24] Yuhao Wu, Franziska Roesner, Tadayoshi Kohno, Ning Zhang, and Umar Iqbal. IsolateGPT: An execution isolation architecture for LLM-based agentic systems. In *Network and Distributed System Security Symposium (NDSS)*, 2025. https://arxiv.org/abs/2403.04960.

[25] A survey of zero-knowledge proof based verifiable machine learning, 2025. https://arxiv.org/abs/2502.18535.