Managing Secrets Across Trust Boundaries in a Federated Workflow Execution Framework

Jeevan Varghese
jeevan.varghese@student.uva.nl
University of Amsterdam
The Netherlands

Natallia Kokash
n.kokash@uva.nl
University of Amsterdam
The Netherlands

Ana Oprescu
a.m.oprescu@uva.nl
University of Amsterdam
The Netherlands

Adam S. Z. Belloum
a.s.z.belloum@uva.nl
University of Amsterdam
The Netherlands

Abstract

Federated workflow systems enable organizations to execute applications across independently operated infrastructures; however, securely delivering secrets across these boundaries remains a practical challenge: existing frameworks such as Brane, a compute-to-data orchestrating framework, dispatch tasks to remote sites without protecting the credentials they carry. We present a secret management mechanism that instead keeps secrets under the data owning domain's control, combining confidential secret delivery, capability based access control, and domain-local policy enforcement into an end-to-end protection path that removes the central scheduler from the secret's confidentiality boundary. We implement the mechanism as an extension to Brane and evaluate it on a multi-site testbed: against thirteen identified threats, eight are mitigated and two partially mitigated, while tamper testing confirms fail-closed behavior.

Keywords: federated computing, agentic AI, secret management, workflow orchestration, capability tokens, hybrid public-key encryption, policy enforcement, zero trust

1 Introduction

A task dispatched to a federated workflow execution framework frequently needs a *secret* to do its work: an API key, or a database password. Within a single administrative domain this has been addressed by tools such as HashiCorp Vault or Kubernetes Secrets [20]. Federation breaks the assumption those tools rely on: once dispatched to a remote site, a task's secret crosses into a domain with its own operators, and the requester loses direct control over who sees it. The central scheduler sits on the secret's path yet it is typically operated by neither the user nor the receiving domain, and a container does not hide its contents from the host that runs it: a value in an environment variable or process memory is visible to a privileged host operator or a co-resident workload [39].

The leakage of credentials into public repositories and container/Kubernetes deployments has been measured at scale [27, 36, 39]. Whether single-domain secret tools remain sufficient once a secret must reach a partially trusted remote site is an open question that only sharpens as schedulers evolve into the multi-tenant, cross-provider dispatch layers agentic and serverless AI systems require.

Our longer-term motivation is agentic workflows over such a federation: an LLM-backed agent deployed on a worker node can locate, harmonize, and filter data that never leaves its site, contribute to a jointly trained model, supply evidence for a finding, or leave a record for later audit — provided it discloses nothing beyond the authorized process, while the control plane authorizes which LLM endpoints a site may call and centrally records every call for audit. Analyzing Brane [41] against this goal, we found that the absence of secret passing is the first obstacle even for the simplest agentic pattern, *agent-as-a-task*, where a stateless agent handles one dispatched step without retaining state; FedTheses, a federated thesis-search prototype running LLM agents across university repositories, is exactly this class [26] and identified secure API-key passing as a main obstacle, which this paper removes. The richer *agent-as-a-service* pattern, where a long-lived agent retains memory and state at a site, requires a broader roadmap and is out of scope here.

We study secret management in federated workflow execution. Taking Brane as a representative framework, we ask:

RQ1. How do threats to secret confidentiality and integrity under federated dispatch differ from a centralized, single-domain baseline?

RQ2. What properties must secret storage, delivery, scoping, and revocation provide in a multi-domain, container-based architecture?

RQ3. What mechanisms can act as release preconditions, under what assumptions, and where do they fail?

RQ4. What is the cost of secret access-control checks on the dispatch path, in runtime and in authoring complexity?

We contribute a threat model for secret delivery in federated execution, defining trust boundaries, adversaries, and assets relative to a centralized baseline (Section 4). We derive security requirements and trace them to the threats they address (Section 5). Based on these requirements, we present a layered design for confidential, policy-governed delivery that combines short-lived capability tokens, hybrid public-key encryption (HPKE) sealing [3] to remove the controller from the confidentiality boundary, per-site policy checks before decryption, output redaction, and mutual Transport Layer Security (TLS) (Section 5). We implement the design by extending the Brane's middleware and evaluate it on a multi-site testbed with respect to functional behavior, confidentiality evidence, threat coverage, and performance overhead (Section 6). Finally, we analyze remote attestation and image signing as alternative preconditions and discuss why their limitations motivate a policy-based design (Section 7).

2 Background

A 2026 measurement of public GitHub reported roughly 29 million newly leaked secrets in a single year, among them 24,008 unique secrets sitting in Model Context Protocol (MCP) configuration files, a

Table 1. Documented credential-exposure patterns in agentic and tool-calling systems, the engineering error behind each, and the corresponding entry in our threat model and requirements.

Exposure pattern	Reported evidence	Engineering error	Threat
Credentials hardcoded in agent or tool configuration	24,008 unique secrets in MCP configuration files in one year, 2,117 confirmed valid [17]	Setup guides normalise passing keys as command-line arguments or storing them in JSON, so the credential is copied wherever the configuration is copied.	T3, SR-A
Credentials in process output, logs, and traces	73.5% of leaks in a study of 17,022 agent skills came from debug printing [9]	Standard output is fed back into the model context and into framework logs; no redaction discipline is applied at the boundary between task and framework.	T4, SR-D
Credential theft through coerced tool calls	MCP clients coerced into exfiltrating environment variables and system files [18, 31]	The executing side inherits ambient credentials and performs no per-release authorisation, so any tool call that reaches it can spend them.	T10, SR-C
Broad, long-lived agent authority	Goal hijack and excessive agency head the OWASP agentic risk taxonomy [30]	Coarse, static credentials are issued once for a whole session instead of being scoped to a step and expiring with it.	T7, SR-F
Weak identity binding between routing and execution	Threat modelling of MCP, A2A, and related protocols finds identity binding for executable components unenforced [2]	The routing layer's identity is accepted as the executing side's authorisation, so a redirected request carries authority it was never granted.	T8, SR-F
Supply-chain harvesting of host credentials	The s1ngularity compromise harvested 2,349 credentials from 1,079 developer systems, driving installed AI command-line tools to enumerate them [16]	Long-lived tokens sit in the environment of every process on the host, so one compromised package inherits all of them.	T6, T10 (deferred)

pattern encouraged by setup guides that pass API keys as command-line arguments or store them inline in JSON [17]. A large-scale study of LLM agent skills found debug logging to be the dominant leakage vector, behind 73.5% of the credential exposures identified, because agent frameworks feed process output straight back into a model's context [9]. Table 1 groups such findings by the engineering error behind them and maps each onto the threat it becomes on a dispatch path. Every row is an error the framework can make on the workflow author's behalf, which is why we treat secret handling as a property of the execution framework rather than of a workflow.

A *federated workflow execution framework* lets several organizations run a workflow jointly without pooling their data. Rather than moving datasets to a central cluster, it ships containerized tasks to workers near each dataset and coordinates their execution across sites, so each organization keeps administrative control of its own data and policies [23, 24]. This is structurally close to federated *function-as-a-service*: a scheduler dispatches a job as a short-lived containerized unit of work to whichever site should run it. funcX applies this model to federated function serving for science [8], and the same dispatch shape underlies emerging agent platforms that fan a workflow's steps, or an agent's tool calls, out to workers or accelerators owned by different providers.

Four terms recur throughout: a *workflow* is the user's submitted program; a *task* is one containerized step; a *domain* (or site) is a participating location running its own worker and holding its own data; and a *handle* is the name a task uses for a secret it consumes (*e.g.*, API key).

We use *Brane* [6, 41] as our representative example, developed for privacy-preserving processing of data across organizations that cannot centralize it [23, 41]. A user compiles a workflow with the `brane` CLI; the driver (`brane-drv`) runs it on a workflow VM and dispatches each task to the worker (`brane-job`) of its assigned domain, decided by the planner (`brane-plr`), over a per-node proxy; a global registry (`brane-api`) holds package and dataset metadata, and each domain runs a policy checker (`brane-chk`) and local registry [7]. Every task input therefore lands, at least briefly, on a machine run by another organization; Figure 1 traces this path. Under a single operator a credential never leaves one boundary; federation breaks this, since each worker belongs to a different organization and the control plane is a further, separate party with no authority over the sites it serves [23, 24], while the receiving domain's operator or a co-resident workload can observe a container's contents on its own host [39].

Our design draws on four established primitives: *mutual TLS*, extending TLS [32] so both endpoints authenticate via X.509 certificates [10]; a *capability token*, a JSON Web Token [22] realizing a compact, signed, short-lived bearer credential naming permitted actions [12]; HPKE, combining key encapsulation, derivation, and authenticated encryption so a sender can seal a value only the private-key holder can open; and *eFLINT*, a normative-rules language evaluated at runtime [43], on which `brane-chk` is built [15, 23].

3 Related work

Brane dispatches containerized tasks by data location and governance constraint, treating each site as its own trust domain [23, 41]. funcX applies the same architecture to federated function serving, authenticating users centrally but leaving application-level secrets out of scope [8]. In the clinical domain, vantage6 realises the Personal Health Train pattern, shipping containerized algorithms to institutional data stations under its own access-control system [28]. All three bring computation to data, and none treats the credential a task consumes as a first-class protected object; each assumes an operator provisions it at the site beforehand.

Vault and Kubernetes Secrets keep credentials out of source and images [20], but protection is bounded by a single domain. KubeKeeper shows Kubernetes Secrets are routinely over-exposed to pods and adds per-pod, per-secret access control [33]. Sealed secrets and `age` encrypt to a recipient's public key so a relaying operator cannot read it [5, 42], close in spirit to our seal, but scoped to one operator's cluster and without gating release on the receiving organization's own policy.

Production deployments assemble comparable functionality from several commercial categories — vaults (e.g., HashiCorp Vault, CyberArk), identity platforms (Okta, Auth0), agent-governance tools, and guardrail/observability products — but each operates either inside a single administrative domain or at the model's input/output, not on the dispatch path, and so answers who may retrieve a credential rather than whether it may be released to this task, at this site, now.

Trusted Execution Environments (TEEs) such as Intel Software Guard Extensions (SGX) and AMD Secure Encrypted Virtualization with Secure Nested Paging (SEV-SNP) isolate code and data

even from a compromised host [1, 11]. A complementary approach establishes trust in a machine's state rather than isolating execution: Keylime roots trust in a Trusted Platform Module (TPM)-based measured boot, verifying that a machine started in a known, unmodified configuration [35]; Thijsman *et al.* extend such attestation to edge worker nodes in Kubernetes [40]. Attestation answers whether a node is currently in a trustworthy state, a natural precondition for releasing a secret to it, but, to our knowledge, has not been connected to a scheduler's secret-delivery path.

Secure Production Identity Framework For Everyone (SPIFFE) issues short-lived, auto-rotating cryptographic identities to workloads rather than relying on static credentials [38] — the same short-lived-capability principle our token applies, following the macaroon model of bearer tokens with embedded, verifiable restrictions [4]. Sigstore links software artifact signatures to short-lived identities [29], verifying *how* code was built but not whether a secret should reach a specific task at dispatch. Industry guidance converges on the same principle, urging short-lived, identity-driven access for every service account, continuous integration (CI) job, and agent in place of static credentials [17].

Existing approaches leave gaps that our design addresses. Federated frameworks delegate secret management to whoever deploys them, with no built-in mechanism. Cloud-native and commercial tools protect only a single cluster or administrative domain. Agentic-security tools detect credential exposure after the fact but do not change how secrets are delivered. Attestation, identity, and signing systems verify a node, workload, or image, but do not connect that verification to a release decision across organizational boundaries. We bridge this gap in the middleware by assembling three mechanisms. Firstly, each secret is sealed with HPKE to the target worker's key. Secondly, every request is scoped by a short-lived capability token. Finally, decryption is gated on a policy check performed by the receiving domain.

4 Threat model

The system under study is the Brane deployment explained in Section 2. A secret travels from the user's client, through the central controller, to the container that consumes it (Figure 1). Brane lets task parameters be marked as secrets, but does not protect them. In unmodified Brane, every task argument travels as plaintext, and sensitivity is never recorded downstream of the client: the DSL's (domain-specific language's) `secret` annotation on a parameter is dropped during compilation, so nothing after the client can tell a credential from any other string. The baseline source tree confirms the gap: dispatch runs over plaintext HTTP/2, the worker's gRPC job service authenticates no caller, and the policy reasoner accepts whatever identity the client's own submitted workflow claims, stamped with a constant, unverified signature.

TB0 (client–controller) is local and trusted. TB1 (controller–worker) is the cross-site hop where the secret first leaves the owner's hands; the controller is *honest-but-curious* [25], expected to forward work, not read it. TB2 (worker–checker) is where the receiving domain decides whether to release a secret. TB3 (worker host–container) is where cryptography stops helping: the host must see the plaintext to inject it.

We consider a network attacker on TB1 [13], the honest-but-curious controller, an unauthorized client crafting requests against

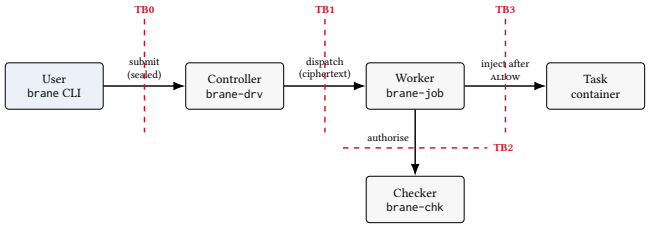

Figure 1. The secret-carrying dispatch path and its four trust boundaries: client–controller (TB0), cross-site controller–worker (TB1), worker–checker authorization (TB2), and worker host–container (TB3).

worker endpoints, and a malicious non-root worker operator observing host-visible state; a root operator reading live container memory is residual risk, scoped out along with side-channel, privilege-escalation, and physical attacks. The model protects task secrets, the authorization decision, the capability token, the sealing/signing keys, the policy database, and package images, across every channel a secret can surface through (arguments, environment, logs, failure output).

A motivating scenario, T-EW ("empty wallet"), threads through the paper: a researcher's paid API token, needed by a task at a partner site, leaks at six points in unmodified Brane (plaintext transport, debug logs, container arguments recoverable via base64, the environment, and echoed failure output) before the partner operator, without entering the container, replays it for unrelated work.

Table 2 catalogs thirteen threats, STRIDE-classified (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege) [37], rated by likelihood multiplied by impact, and marked as *mitigated*, *partial*, or *deferred* with the middleware extension presented in this work. T1–T5 are targeted most directly, T6–T13 round out the model. Eight are mitigated (T1–T5, T7, T8, T13), two partially (T9, T12), three deferred (T6, T10, T11); T-EW composes the mitigated T1, T3, T4. Threats listed in this table are mapped back to the exposure patterns of Table 1.

Three goals drive the requirements for our proposed solution:

- *Confidentiality*: the secret remains protected along its entire path, from issuance to use.
- *Integrity of the authorization decision*: any failure in the decision process denies access rather than allowing it, following the fail-safe defaults principle [34].
- *Controlled runtime exposure*: the secret is revealed only at the point where it is actually needed, and nowhere else.

Our solution works under assumptions which can also be seen as limitations. First, we trust the worker host against its own root operator: no software-based defense protects a secret from an adversary who already has root access to the machine. Second, we exclude microarchitectural attacks, privilege-escalation exploits, and physical attacks on hardware from our threat model.

5 Design and Implementation

The threat model revealed one hard constraint: the controller, sitting on the secret's path, must route work without ever holding the secret it routes. Authenticating the channel alone does not achieve

Table 2. Threat catalogue (STRIDE, severity, coverage). Full per-threat evidence appears in the extended report.

ID	STRIDE	Sev.	Threat (status)
T1	Info. disc.	Hi	Plaintext dispatch link (TB1). *Mitigated:* mTLS, per-site CA.
T2	Spoof./EoP	Hi	Unauthenticated worker RPCs. *Mitigated*: worker-call token.
T3–T4	Info. disc.	Hi	Secrets in argv/env/failure output. *Mitigated*: env. injection, redacted logs, byte-count on failure.
T5	Tamper./EoP	Hi	Checker handles dropped ⇒ fail-open. *Mitigated:* handles joined, fail closed.
T7–T8	Spoof./EoP	Hi	Replayable/long-lived token; confused deputy [19]. *Mitigated:* short-lived, nonce- and session-bound tokens.
T13	Tamper.	Med	Misrouted secret via unbound env. vars. *Mitigated:* handle-scoped, location-bound injection.
T9, T12	Tamp./Spoof./DoS	Hi/Med	Unsigned images; checker DoS → allow. *Partial*: upload auth added, signing deferred; T5 fix narrows DoS to degraded-allow.
T6, T10, T11	Info. disc.	Hi/Med	Secrets at rest; result exfiltration; cross-workflow leakage. *Deferred*, named as future work.

this, since the controller is a legitimate endpoint. Conversely, hiding the value from the controller is insufficient if the receiving domain can open any secret delivered to its worker without further authorization. We therefore design the mechanism as a set of complementary protections, following a least-trust principle [34], and implement it as a fork of the Brane framework[1].

The client creates a short-lived, scoped capability token and seals the secret for the intended worker into a *sealed secret bundle*: the requested secret names, their encrypted values, and the token. This bundle is transported over mutually authenticated channels alongside the task request. The controller (`brane-drv`) forwards the bundle as part of dispatch but never decrypts or otherwise processes its contents, since secrets are encrypted specifically for the worker expected to execute the task. At the receiving site, the worker first asks the local policy checker whether the requested secrets may be released; only after an ALLOW decision does it decrypt them. If the policy denies the request, the worker returns without accessing the secret material at all. This separates *authorization* (which secrets a task may receive, decided by site policy) from *secret access* (decryption, performed only by the worker) — so neither the controller nor any intermediate component ever needs the underlying credentials. The protection layers are summarized in Table 4 and mapped to the dispatched path in Figure 2.

Once released, a secret is injected directly into the task container as an environment variable rather than a command-line argument, so it is invisible to `ps` (T3), and is kept in memory using zeroizing wrappers to limit its lifetime. Brane's container execution interface was extended to accept these protected environment variables, populated only after successful policy evaluation; the same mechanism applies uniformly to any credential type (e.g., an LLM API key) independent of the application running inside the container. Results, workflow bodies, and failed deserializations are logged by length or byte count rather than content (T4), and secret values are excluded from normal logging and error reporting. This does not conceal a value a malicious package deliberately copies into its own output (T10), nor hide it from a privileged host operator reading the environment via `docker inspect` or live memory (TB3); both remain residual risks of the threat model in Section 4.

Worker dispatch and check RPCs require a worker-call token (T2); the package-upload endpoint requires an operator token (T9, partially: image signing is deferred); and the checker logs every release decision with its handle and verdict (SR-G, SR-E). Short-lived, audience-specific authorization tokens, together with per-worker encryption keys, prevent a secret bundle intended for one service or worker from being reused elsewhere.

[1] https://github.com/Jeeva-uva/brane-nightly-secret

The extension is optional: when a workflow does not use secrets, the dispatch and execution paths remain unchanged. It provides the secret-delivery functionality required by both deployment models discussed above; in particular, a containerized agent can now run as a normal Brane task while obtaining its model credentials locally at the worker. Supporting a long-running agent-as-a-service deployment requires an additional Brane container-lifecycle and service-invocation mechanism, but not a different secret-management mechanism.

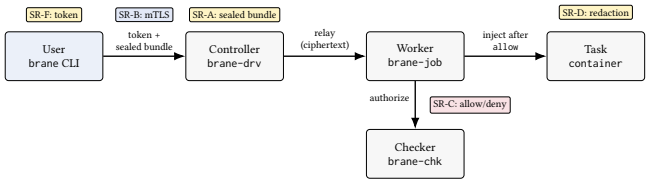

Figure 2. Secret protection in Brane. The client creates a scoped capability and seals the secret for the target worker. The controller relays the opaque bundle, while the worker releases the secret only after a local policy decision and injects it into the task container.

6 Evaluation

We evaluate on a testbed with a control plane and two worker domains, `site-a` and `site-b`, re-provisioned from scratch, running stock and modified Brane side by side on the same cluster and binaries with `sk-EVALONLY-` values. The workload is a trivial `hello_world` echo, run as a single-task workflow (A) and a dual-site workflow (B), each with (A′, B′) and without a sealed secret; ten interleaved runs per configuration, first five discarded as warm-up, reporting the median of ten per-run medians.

Functional validation (RQ1–RQ3). Under a policy allowing only `api_key`, a request for that handle succeeds end to end (seal, relay, check, decrypt, inject); a request for an unlisted handle (`db_pw`) is denied by the count-equality gate before execution, on an identical code path differing only in policy. The live run evidences T1 (mTLS), T3/T4 (absent plaintext, env. injection), SR-C (allow/deny), and SR-A (controller-blind capture, tamper-probe).

Searching the controller's 162-line log for a unique marker planted in a sealed secret yields zero hits; the same search against the worker log yields zero. A tamper-probe against each worker's deployed keypair confirms the seal fails closed: a correct key opens a properly sealed value, while a tampered ciphertext, tampered encapsulated key, wrong-location context, or wrong-handle context all fail identically on both sites.

Table 3. Security requirements and the corresponding design mechanisms.

ID	Requirement	Design mechanism
SR-A	Secret reaches the worker unreadable by any intermediary, including the controller.	Sealed bundle encrypted for the target worker
SR-B	Every cross-site dispatch hop is mutually authenticated and encrypted.	Mutual TLS
SR-C	Secret release occurs only after explicit authorization by the receiving domain; failure denies.	Per-site policy checker and allow/deny decision
SR-D	Secret values remain outside arguments, logs, environment dumps, and failure output.	Direct injection and redaction
SR-E	Each threat's status is assessed against captured evidence.	Security evaluation (Section 6)
SR-F	Authority is short-lived, session-bound, and restricted to the requested secret handles.	Capability token
SR-G	Each release decision is recorded for later review.	Release-decision logging

Table 4. Overview of the secret-delivery mechanism.

Layer	Role in secret protection
Capability	The client creates a short-lived token identifying the session and the secret handles that may be used. The token provides authorization but does not contain the secret.
Sealed delivery	Each secret is encrypted for the intended worker before leaving the client. The controller therefore receives only an opaque encrypted value and can forward it without learning the secret.
Authenticated transport	Communication between Brane components uses mutually authenticated and encrypted channels, protecting the request while it is in transit.
Policy-controlled release	The worker consults the receiving domain's policy before accessing the encrypted secret. Only an explicit ALLOW decision permits decryption; a denial or failed check results in no secret access.
Protected injection	After authorization, the worker decrypts the secret and supplies it directly to the task container as an environment variable. Secret values are kept out of logs and diagnostic messages and are cleared from memory when no longer needed.
Auditability	Secret-release decisions are recorded so that access can be reviewed after execution.

Performance (RQ4). With no secrets involved, the modified version of Brane performs the same as stock Brane, within normal measurement noise (10.729 vs. 10.711 seconds for workflow A; 16.882 vs. 16.852 seconds for workflow B). The added mTLS encryption, worker-call authentication, and policy-checker consultation don't cost anything measurable on their own.

When a sealed secret is used, median latency rises by 0.74% for workflow A (10.729 → 10.809 s) and 0.86% for workflow B (16.882 → 17.027 s) — both well under the 20% threshold set as acceptable. Across twenty repeated runs, the overhead varied between 0.57% and 1.40%; a 95% confidence interval puts the true mean overhead at 0.76% (0.66–0.85%) for A and 1.00% (0.82–1.17%) for B.

A sweep of payload sizes from 1 to 64 KB showed no relationship between payload size and overhead, confirming the added cost comes from a fixed cryptographic handshake rather than scaling with data size. In terms of developer effort, using the feature requires only a single –secret NAME=VALUE flag and one line of eFLINT policy. Initial setup takes roughly 7 seconds per worker for a one-time key ceremony, which reuses the existing per-site certificate authority.

Limitations. All three tokens share one embedded HS256 key across audiences (per-audience JWK keys [21] are future work); the short controller–proxy hop within one node is plaintext, which the design does not rest on since the value is sealed end to end; there is no token revocation or replay cache within the token's lifetime; the seal target set is fixed client-side rather than derived from the planner; and redaction covers the framework's own log and result surfaces, not a package that deliberately copies a secret into its own output (T10).

7 Discussion

Federation introduces two adversaries absent from centralized deployment — a partially trusted controller and a receiving host administered by another organization — both exploitable in our threat model. The requirements in Table 3 are met for delivery, transport, scoping, release, and redaction; storage-at-rest and revocation (T6) remain partial, since revocation relies on short token lifetimes. The per-domain pre-decryption policy check, at about 1% overhead, complements attestation and image signing: those establish node state and artifact provenance, while policy determines whether release is authorized. By sealing secrets to the receiving worker, scoping requests with short-lived tokens, and delegating release to the receiving domain, the design keeps secrets confidential across trust boundaries.

The studied dispatch pattern—an orchestrator routing short-lived containerized tasks to independently administered workers—is representative of federated FaaS [8] and agentic systems using MCP or A2A-style tool calls. When such tasks require credentials, the routing layer is on the credential's path but should not learn its value, while the executing domain should retain control over release. Our mechanism therefore provides a candidate building block for zero-trust secret delivery in ephemeral, multi-tenant agent runtimes, with low and size-independent overhead.

In an MCP-style workflow, a planner may route a tool requiring a user-held API key to a partner site; our mechanism applies directly, sealing the key to that site, scoping it to a short-lived token, and gating release on the provider's own policy before decryption, with the release logged and the value kept out of output and traces [9]. This addresses the common problem of credentials crossing orchestration and agent layers [2, 30], although the executing provider necessarily sees the plaintext.

Threats to validity. The controller-blind property follows primarily from the absence of a recipient key and decryption path in `brane-drv`, supported by log analysis and tamper probes. The evaluation is limited to one framework, cluster, and workload, and does not include formal protocol verification or a user study of authoring complexity. The main unresolved threat is TB3: a privileged host operator requires a TEE for protection.

8 Conclusions and Future Work

Federation breaks the assumption that a single administrator protects a workflow's secrets: credentials cross an independently operated scheduler to a host outside the data owner's control. We address this with five complementary mechanisms: sealed delivery, scoped capability tokens, per-domain policy checks, output redaction, and mutual TLS. The contribution is the composition of established mechanisms into a policy-governed secret-delivery pattern for federated dispatch, implemented in Brane but applicable beyond to serverless and agentic systems that move credentials across organizational boundaries.

Future work includes nonce replay protection, per-audience JWKs, key rotation, post-quantum Key Encapsulation Mechanisms (KEMs), deriving seal targets from scheduler assignments, and integration with existing secret stores. Richer purpose- and provenance-aware policies [14, 24] are another natural extension.

Acknowledgements

This work was supported by the European Union's Horizon Europe research and innovation programme under grant agreement No. 101298951 (FRAME project) and by the DigitAfrica initiative (https://www.digitafrica.eu/). Views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] Advanced Micro Devices. 2020. *AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More.* Technical Report. Advanced Micro Devices, Inc.

[2] Z. Anbiaee, M. Rabbani, M. Mirani, G. Piya, I. Opushnyev, A. Ghorbani, and S. Dadkhah. 2026. Security Threat Modeling for Emerging AI-Agent Protocols: A Comparative Analysis of MCP, A2A, Agora, and ANP. arXiv preprint arXiv:2602.11327. https://arxiv.org/abs/2602.11327.

[3] R. Barnes, K. Bhargavan, B. Lipp, and C. Wood. 2022. *Hybrid Public Key Encryption.* RFC 9180. Internet Research Task Force (IRTF).

[4] A. Birgisson, J. G. Politz, Ú. Erlingsson, A. Taly, M. Vrable, and M. Lampson. 2014. Macaroons: Cookies with Contextual Caveats for Decentralized Authorization in the Cloud. In *Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS).* Internet Society. doi:10.14722/ndss.2014.23212

[5] Bitnami Labs. 2017. *Sealed Secrets for Kubernetes.* https://github.com/bitnami-labs/sealed-secrets

[6] Brane Framework. 2026. Brane: A Specification. https://braneframework.github.io/. Accessed: 2026-06.

[7] Brane Framework. 2026. Brane: The User Guide. https://braneframework.github.io/manual/. Accessed: 2026-06.

[8] R. Chard, Y. N. Babuji, Z. Li, T. J. Skluzacek, A. Woodard, B. Blaiszik, I. T. Foster, and K. Chard. 2020. funcX: A Federated Function Serving Fabric for Science. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing (HPDC).* ACM, 65–76. doi:10.1145/3369583.3392683

[9] Z. Chen, Y. Zhang, Y. Liu, G. Deng, Y. Li, Y. Zhang, J. Ning, L. Y. Zhang, L. Ma, and Z. Li. 2026. Credential Leakage in LLM Agent Skills: A Large-Scale Empirical Study. arXiv preprint arXiv:2604.03070. https://arxiv.org/abs/2604.03070.

[10] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. 2008. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.* RFC 5280. Internet Engineering Task Force (IETF).

[11] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. https://eprint.iacr.org/2016/086 Cryptology ePrint Archive, Paper 2016/086.

[12] J. B. Dennis and E. C. Van Horn. 1966. Programming Semantics for Multiprogrammed Computations. *Commun. ACM* 9, 3 (1966), 143–155. doi:10.1145/365230.365252

[13] D. Dolev and A. C. Yao. 1983. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory* 29, 2 (1983), 198–208. doi:10.1109/TIT.1983.1056650

[14] C. A. Esterhuyse, T. Müller, and L. T. van Binsbergen. 2025. JustAct+: A Framework for Auditable Multi-Agent Systems Regulated by Inter-Organisational Policies. arXiv preprint arXiv:2502.00138. https://arxiv.org/abs/2502.00138.

[15] C. A. Esterhuyse, T. Müller, L. T. van Binsbergen, and A. S. Z. Belloum. 2022. Exploring the Enforcement of Private, Dynamic Policies on Medical Workflow Execution. In *2022 IEEE 18th International Conference on e-Science (e-Science).* IEEE.

[16] GitGuardian. 2025. *The Nx "s1ngularity" Attack: Inside the Credential Leak.* Accessed: 2026-09.

[17] GitGuardian. 2026. *The State of Secrets Sprawl 2026.* Technical Report. GitGuardian. https://www.gitguardian.com/state-of-secrets-sprawl-report-2026. Accessed: 2026-09.

[18] Y. Guo, P. Liu, W. Ma, Z. Deng, X. Zhu, P. Di, X. Xiao, and S. Wen. 2025. Systematic Analysis of MCP Security. arXiv preprint arXiv:2508.12538. https://arxiv.org/abs/2508.12538.

[19] N. Hardy. 1988. The Confused Deputy: (or why capabilities might have been invented). *ACM SIGOPS Operating Systems Review* 22, 4 (1988), 36–38. doi:10.1145/54289.871709

[20] HashiCorp. 2026. *What is Vault?* Accessed: 2026-06-17.

[21] M. Jones. 2015. *JSON Web Key (JWK).* RFC 7517. Internet Engineering Task Force (IETF).

[22] M. Jones, J. Bradley, and N. Sakimura. 2015. *JSON Web Token (JWT).* RFC 7519. Internet Engineering Task Force (IETF).

[23] J. Alsayed Kassem, T. Müller, C. A. Esterhuyse, M. G. Kebede, A. Osseyran, and P. Grosso. 2025. The EPI framework: A data privacy by design framework to support healthcare use cases. *Future Generation Computer Systems* 165 (2025), 107550. doi:10.1016/j.future.2024.107550

[24] N. Kokash, A. Belloum, and P. Grosso. 2026. Compliance Management for Federated Data Processing. arXiv preprint arXiv:2602.19360. https://arxiv.org/abs/2602.19360.

[25] Y. Lindell. 2017. How to Simulate It – A Tutorial on the Simulation Proof Technique. In *Tutorials on the Foundations of Cryptography.* Springer. doi:10.1007/978-3-319-57048-8_6

[26] Mingjiang Ma. 2026. *FedTheses: federated thesis search with LLM agents over the Model Context Protocol.* https://github.com/MingjiangMa/fedtheses

[27] M. Meli, M. R. McNiece, and B. Reaves. 2019. How Bad Can It Git? Characterizing Secret Leakage in Public GitHub Repositories. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium (NDSS).* Internet Society. doi:10.14722/ndss.2019.23418

[28] A. Moncada-Torres, F. Martin, M. Sieswerda, J. van Soest, and G. Geleijnse. 2020. VANTAGE6: an open source priVAcy preserviNg federaTed leArninG infrastructurE for Secure Insight eXchange. In *AMIA Annual Symposium Proceedings*, Vol. 2020. American Medical Informatics Association, 870–877.

[29] Z. Newman, J. S. Meyers, and S. Torres-Arias. 2022. Sigstore: Software Signing for Everybody. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS).* ACM, 2353–2367. doi:10.1145/3548606.3560596

[30] OWASP GenAI Security Project. 2026. *OWASP Top 10 for Agentic Applications 2026.* OWASP Foundation. Accessed: 2026-09.

[31] B. Radosevich and J. Halloran. 2025. MCP Safety Audit: LLMs with the Model Context Protocol Allow Major Security Exploits. arXiv preprint arXiv:2504.03767. https://arxiv.org/abs/2504.03767.

[32] E. Rescorla. 2018. *The Transport Layer Security (TLS) Protocol Version 1.3.* RFC 8446. Internet Engineering Task Force (IETF).

[33] M. Rostamipoor, A. Sadeghi, and M. Polychronakis. 2025. KubeKeeper: Protecting Kubernetes Secrets Against Excessive Permissions. In *2025 IEEE 10th European Symposium on Security and Privacy (EuroS&P).* IEEE.

[34] J. H. Saltzer and M. D. Schroeder. 1975. The Protection of Information in Computer Systems. *Proc. IEEE* 63, 9 (1975), 1278–1308. doi:10.1109/PROC.1975.9939

[35] N. Schear, P. T. Cable, T. M. Moyer, B. Richard, and R. Rudd. 2016. Bootstrapping and Maintaining Trust in the Cloud. In *Proceedings of the 32nd Annual Conference on Computer Security Applications (ACSAC).* ACM, 430–440. doi:10.1145/2991079.2991104

[36] M. S. I. Shamim, F. A. Bhuiyan, and A. Rahman. 2020. XI Commandments of Kubernetes Security: A Systematization of Knowledge Related to Kubernetes Security Practices. In *2020 IEEE Secure Development Conference (SecDev).* IEEE, 58–64. doi:10.1109/SecDev45635.2020.00025

[37] A. Shostack. 2014. *Threat Modeling: Designing for Security.* Wiley.

[38] SPIFFE Project. 2026. *SPIFFE: Secure Production Identity Framework for Everyone.* Cloud Native Computing Foundation project. Accessed: 2026-06-18.

[39] S. Sultan, I. Ahmad, and T. Dimitriou. 2019. Container Security: Issues, Challenges, and the Road Ahead. *IEEE Access* 7 (2019), 52976–52996. doi:10.1109/ACCESS.2019.2911732

[40] J. Thijsman, M. Sebrechts, F. De Turck, and B. Volckaert. 2024. Trusting the Cloud-Native Edge: Remotely Attested Kubernetes Workers. In *2024 33rd International Conference on Computer Communications and Networks (ICCCN).* IEEE. arXiv:2405.10131.

[41] O. Valkering, R. Cushing, and A. Belloum. 2021. Brane: A Framework for Programmable Orchestration of Multi-Site Applications. In *2021 IEEE 17th International Conference on e-Science (e-Science).* IEEE, 277–282. doi:10.1109/eScience51609.2021.00056

[42] Filippo Valsorda. 2019. *age: A Simple, Modern and Secure File Encryption Tool.* https://github.com/FiloSottile/age

[43] L. T. van Binsbergen, C. A. Esterhuyse, and T. Müller. 2025. Reflections on the design, applications and implementations of the normative specification language eFLINT. arXiv preprint arXiv:2511.12276. https://arxiv.org/abs/2511.12276.