whitepaper

V1.0 · 2026

Zero-Trust Whitepaper

A reference for security, platform, and infrastructure leaders evaluating a2a cloud. Threat model, isolation guarantees, cryptographic primitives, audit surfaces, and key management for governed AI agent execution.

SECTION 1

Threat model

The runtime treats five actors as distinct principals: the *agent* being executed, any *peer agent* it calls or is called by, every external *tool* it touches, the human *user* initiating the task, and the platform *operator* hosting the run. Each has its own identity, its own authority surface, and its own audit trail. None inherits trust from another.

Trust boundaries sit between every adjacent pair. The user→agent boundary is enforced by an auth edge with a service identity and rate limits. The agent→peer boundary is enforced by signed grants. The agent→tool boundary is enforced by scoped capability tokens and a workspace client. The runtime→host boundary is enforced by a libkrun microVM. Nothing crosses a boundary without an explicit, signed claim.

We **explicitly distrust** three things: LLM output (it can be adversarial, hallucinated, or prompt-injected); peer agent intent (a callee may be compromised, misconfigured, or deliberately misbehaving); and tool side-effects (a tool may corrupt state, exfiltrate data, or refuse to terminate). The runtime is built so that any of the three failing in isolation cannot escalate.

SECTION 2

Zero-trust runtime design

The runtime ships with no ambient credentials. There is no shared service account, no environment-wide API key, no filesystem the agent process can read by default. Every action that touches state outside the agent's own memory requires a grant, and every grant is rejected unless its signature verifies, its audience matches, and its expiry is in the future.

Grants are scoped along three axes: *skill* (which capability is invocable), *time* (a TTL — default five minutes), and *target* (which workspace bucket and which callee). A grant authorizing read of one bucket cannot be used to read another, even by the same agent in the same run.

Every grant is signed by the issuer before it leaves the control plane. Signature verification happens at the receiving side *before* any tool dispatch or file read. There is no path through the runtime that consumes authority without first verifying its envelope.

SECTION 3

MicroVM isolation guarantees

Each run executes inside a libkrun microVM. The boundary is a hardware-virtualized one — not a container, not a chroot, not a seccomp profile layered over the host kernel. A compromised agent cannot read host memory, attach to other tenants, or escape into the orchestrator process.

The filesystem inside the microVM is scoped to the workspace bucket named in the run's grant. There is no path from the agent process to `/etc/secrets`, to host configuration, or to neighbouring tenants' data. Host network is disabled by default; egress is mediated by the platform's policy layer.

The lifecycle is ephemeral. The microVM is created at run start, torn down at run end, and discarded — its disk image, its memory, and its process tree do not survive into the next invocation. Persistence is explicit, goes through the workspace client, and is recorded in the receipt.

SECTION 4

Grant signing

Grants are signed claims that bind an issuer, an audience, a scope, and a lifetime. Production deployments sign with Ed25519; local/dev deployments fall back to HMAC-SHA256 over a shared platform secret. The verifying side selects the algorithm based on which key material is configured — there is no algorithm field in the envelope to downgrade.

The wire format is two base64url segments joined by a dot: `base64url(payload).base64url(sig)`. The payload is canonical JSON over the field set below; the signature covers the exact payload bytes. Implementation lives in `a2a_pack/grants.py`.

GRANT PAYLOAD FIELDS	
`grant_id`	Random 64-bit identifier. Replay-detection key in the audit log.
`agent_caller`	Issuing agent identity (issuer in code). Anchors provenance.
`target`	Audience the grant is bound to. Verified by the receiver before any tool call.
`skills[]`	Allowed skill names. Anything outside this set is rejected at the runtime boundary.
`not_before`	Earliest unix second the grant is valid (issued_at in code).
`expires_at`	Latest unix second. Default TTL is 5 minutes. Past this, the runtime treats the grant as forged.
`nonce`	Per-grant random token. Combined with grant_id to prevent replay across runs.

A grant that fails signature verification, audience match, or expiry check raises `GrantInvalid` and is logged as a security event. The runtime never falls through to a default-allow.

SECTION 5

Receipt provenance

When a run completes — success, error, cancellation, or partial — the runtime seals an `ExecutionReceipt`. The receipt is the run's ground truth: it records what ran, what it touched, what it produced, and under whose authority. Implementation lives in `a2a_pack/receipts.py`.

The envelope uses the same wire format as grants: `base64url(payload).base64url(sig)`. Ed25519 in production, HMAC-SHA256 fallback for dev. Verification does not check freshness — receipts are historical artifacts and are expected to verify long after the run.

EXECUTION RECEIPT FIELDS	
`receipt_id`	Stable identifier for the run. Indexed by the receipt store.
`agent_name / agent_version`	Which agent + build produced the output.
`caller / task_id`	Who invoked the run; the task it belongs to.
`skill_name`	Which skill on the agent card was executed.
`input_hash`	sha256 of canonical-JSON inputs. Replay key.
`input_preview / result_preview`	Truncated previews safe to inline in API responses.
`grant_ids[]`	Every grant the run consumed. Cross-references the grant log.
`file_ops`	reads, writes, bytes_read, bytes_written, and truncated path previews.
`tool_calls[]`	name, args_hash, status, elapsed_ms. Args are stored by hash, not payload.
`artifacts[]`	Workspace artifacts the skill produced (path, mime_type, bytes).

`handoffs[]`	Each agent-to-agent call: callee, skill, grant_id, status, elapsed_ms.
`status / error_type`	ok, error, cancelled, partial.
`eval_score / reviewer`	Optional evaluation score and human reviewer attestation.
`started_at / ended_at / elapsed_ms`	Run timing. Stable inputs reproduce identical envelopes modulo timing + nonce.
`nonce`	Per-receipt random token sealed into the signature.

Tool-call arguments are stored by `args_hash`, never by payload. This keeps receipts small enough to inline in API responses and avoids leaking sensitive arguments into the audit log surface.

SECTION 6

Audit + replay design

Receipts double as the event source for replay. A receipt captures the input hash, the grant set, the ordered tool calls (by hash), the file operations, and the handoffs. Given the same inputs and the same grants, the runtime can re-execute a run and validate its tool-call sequence and file-op sequence against the original receipt. There is no separate audit log schema — the receipt *is* the audit log entry.

The receipt store is append-only. Default retention is 90 days hot, 7 years cold-archive for tenants under compliance frameworks that require long-tail retrievability; both are configurable per tenant. Receipts are content-addressed by `receipt_id` and queryable by caller, task, agent, skill, and time range.

The determinism boundary is honest. LLM inference is stochastic — we do not promise byte-identical model output across replays. What we *do* promise is that tool I/O, file operations, and handoffs are deterministically reproducible from the receipt: the captured argument hashes and result previews are sufficient to detect divergence between the original run and a replay, and to flag the divergence at the tool boundary, not after silent state drift.

SECTION 7

Key management

The runtime reads its key material from environment variables. The full set, by role:

`A2A_PLATFORM_SECRET`	HMAC-SHA256 shared secret. Dev/local fallback for both grants and receipts when no asymmetric key is configured. Not for production.
`A2A_GRANT_SIGNING_KEY`	Ed25519 private key. Held only by trusted platform components that mint grants.
`A2A_GRANT_VERIFYING_KEY`	Ed25519 public key. Distributed to every agent and runtime that needs to verify a grant.
`A2A_RECEIPT_SIGNING_KEY`	Ed25519 private key used by the runtime to seal receipts at run completion.
`A2A_RECEIPT_VERIFYING_KEY`	Ed25519 public key. Distributed to downstream verifiers, auditors, and compliance integrations.

Rotation: the grant signing key is rotated quarterly under standard operation; receipt signing keys are rotated annually with overlap, since verifiers must accept the previous key long enough to validate archived receipts. Both rotations follow a publish-new-then-revoke-old protocol; verifying keys are distributed as a small set rather than a single value, so old material can be retired without invalidating in-flight tokens. Compromise of a signing key triggers immediate revocation, re-issuance of all in-flight grants, and a recorded incident receipt.