Service Account Replacement: Eliminating Long-Lived Shared Secrets in 90 Days

Machine Identity / Last updated 2026-06-11 / https://www.scrambleid.com/learn/service-account-replacement

In one sentence: Long-lived service-account passwords, API keys, OAuth client secrets, and SSH keys are the dominant cause of non-human identity breach; the practical replacement architecture is cloud-native workload identity for in-cloud workloads, sender-constrained tokens for cross-boundary paths, and OIDC federation for CI/CD, with a 90-day target of ≥ 95% migration off long-lived secrets.

TL;DR (canonical)

- **The problem is structural, not operational.** Long-lived shared secrets accumulate in repos, vaults, environment files, and developer laptops. They leak. Once leaked, they remain valid for years and replay from anywhere.
- **The replacement architecture is three patterns:**
  1. **Cloud-native workload identity** for workloads inside a cloud (AWS IRSA, GCP WIF, Azure Managed Identity, SPIFFE/SPIRE for cross-cloud).
  2. **Sender-constrained tokens** for cross-boundary service-to-service (RFC 7523 JWT client assertions plus RFC 8705 mTLS or RFC 9449 DPoP).
  3. **OIDC federation** for CI/CD pipelines (GitHub Actions, GitLab CI, others) issuing short-lived credentials at job execution.
- **The migration sequence:** inventory, classify by blast radius, migrate highest-leverage paths first (production admin, CI/CD with admin scope, cross-tenant integrations), replace with workload identity and sender-constrained tokens, decommission old credentials.
- **The KPI target:** ≥ 95% of machine identities migrated off client secrets within 90 days. Owner coverage 100%. Mean time to rotate keys ≤ 24 hours.
- **The vault doesn't disappear.** It moves from primary control plane to exception store.

Why service-account secrets are the dominant NHI breach vector

Three structural reasons:

1. Volume and accumulation. A modern enterprise easily has tens of thousands of service accounts across cloud, CI/CD, integrations, and legacy systems. Most enterprises do not have a complete inventory.

2. Inevitability of leakage. A long-lived secret will eventually leak. The cumulative leakage probability over a year approaches certainty for any organization above modest scale. The question is not whether but when, and which one.

3. Unbounded blast radius once leaked. A leaked service-account secret remains valid for the duration of its lifetime, which is often years. It is not bound to a specific workload or network; it can be replayed from anywhere on the internet, by any actor who finds it. Multiple high-profile breaches in 2022-2024 (CircleCI, the Microsoft SAS-token exposure, Okta support tooling, others) started with leaked long-lived NHI credentials.

The fix is not "rotate secrets faster." Rotation cycle reduces the time-window of leak validity, but doesn't eliminate the structural problem. The fix is to remove the long-lived shared secret from the architecture entirely.

The three replacement patterns

Pattern 1: cloud-native workload identity for in-cloud workloads

The most leveraged change. Cloud providers offer mechanisms where a workload can prove its identity to the cloud's IAM without holding any persistent secret:

Cloud	Mechanism	When to use
AWS	IAM Roles for Service Accounts (IRSA) for EKS workloads; IAM Roles Anywhere for off-cloud workloads; STS AssumeRole with web identity for federated workloads	Default for workloads in AWS; only fallback when constrained
GCP	Workload Identity Federation; Workload Identity for GKE; Service Account impersonation via short-lived tokens	Default for GCP workloads
Azure	Managed Identity (system-assigned and user-assigned); Microsoft Entra ID Workload Identity for AKS; federated credentials	Default for Azure workloads
Cross-cloud / hybrid	SPIFFE/SPIRE for workload attestation across heterogeneous environments	When workloads span clouds or run on-prem

The pattern: the workload's runtime (the cluster, the cloud-managed service) attests its identity to the cloud's IAM. The cloud issues a short-lived credential scoped to what the workload needs. There is no persistent secret in source control or in a vault.

For a side-by-side comparison of the three major cloud providers' patterns plus SPIFFE/SPIRE, see Cloud Workload Identity Compared.

Pattern 2: sender-constrained tokens for cross-boundary paths

Even with cloud-native workload identity, service-to-service calls across boundaries (SaaS-to-on-prem, partner integrations, cross-cloud) need credentials in flight. The architecture:

- **RFC 7523 JWT client assertion:** the client authenticates to the IDP's token endpoint with a signed JWT instead of a shared secret.
- **RFC 8705 mTLS:** access tokens bound to the client's TLS certificate. The token is only valid when presented over a mutually-authenticated TLS connection with that certificate.
- **RFC 9449 DPoP:** access tokens bound to a client's public key, with each request signed by the corresponding private key. Used where mTLS is impractical.

The IDP validates the JWT assertion, issues a short-lived access token (≤ 300 seconds default in ScrambleID), and binds the token via mTLS or DPoP. A leaked token without the matching PoP material returns `invalid_token` at the resource server. See Sender-Constrained Tokens (mTLS, DPoP) and client_secret vs JWT vs mTLS for deeper coverage.

Pattern 3: OIDC federation for CI/CD pipelines

CI/CD pipelines are a particularly leaky surface for long-lived credentials. The traditional pattern: the CI provider holds an admin API key for the cloud or production system. The pipeline runs, the key is loaded into the runner's environment, the pipeline calls the cloud, the key returns to wherever it was stored. Each leak vector (compromised CI provider, leaked runner config, exfiltrated environment) exposes the credential.

The replacement: OIDC federation. The CI provider's runner authenticates to the cloud at the moment of action with a credential whose lifetime is the duration of the job:

- **GitHub Actions:** native OIDC token federated to AWS via IAM Roles, to GCP via Workload Identity Federation, to Azure via federated credentials.
- **GitLab CI/CD:** ID tokens with the same federation pattern.
- **Jenkins, CircleCI, Buildkite, others:** OIDC support varying by maturity.

For deep coverage including cloud-by-cloud configuration, see GitHub Actions OIDC Federation Across Clouds.

The migration sequence

A concrete ordered approach for a platform team:

Phase 1: Inventory (week 1–2)

Build a real inventory of NHIs. Most enterprises start with:

- **Vault audit.** Every entry in HashiCorp Vault, AWS Secrets Manager, GCP Secret Manager, Azure Key Vault, custom vault systems.
- **IAM policy mining.** Service accounts in cloud IAM with their grants and last-used data.
- **CI/CD configuration analysis.** Every pipeline, every secret reference, every credential mounted into runners.
- **Source-control secret scanning.** Use a tool like trufflehog, gitleaks, or vendor-provided secret scanning to find committed secrets (which are by definition compromised; rotate before migration).
- **Environment-variable inventory.** Production, staging, dev environments and the secrets they hold.
- **Application-config inventory.** Anything in `config.yaml`, `appsettings.json`, etc.

The inventory is rarely complete on the first pass. Iterate.

Phase 2: Classify by blast radius (week 2–3)

For each NHI, score:

- **Production vs non-production.** Production-impacting first.
- **Read vs write vs admin.** Admin first.
- **Customer-data-touching.** Customer-data-touching first.
- **Cross-tenant or cross-org.** Cross-boundary first.
- **CI/CD with admin scope.** Highest priority.
- **Used in last N days.** Active first; dormant clients investigate (decommission or document).

A typical priority output:

Tier	Examples	Migration timeline
1 (highest)	Production database admin, IAM admin, payment-system service accounts, CI/CD with prod-admin scope	Weeks 3-4
2	Cross-tenant integrations, external partner APIs, CI/CD with prod-write scope	Weeks 4-8
3	Internal service-to-service in production	Weeks 6-10
4	Lower-impact, non-prod, dormant	Weeks 8-12

Phase 3: Migrate (weeks 3–12)

For each NHI in priority order:

1. **Determine the right replacement pattern.** In-cloud → workload identity. Cross-boundary → sender-constrained tokens. CI/CD → OIDC federation.
2. **Stand up the replacement in parallel.** New workload identity / token / OIDC trust relationship while the old credential continues to work.

3. **Cut over the workload to the new pattern.** Verify functionality.
4. **Monitor for failures.** A failed migration that leaves the workload broken is worse than the original credential.
5. **Decommission the old credential.** Revoke. Remove from vault. Remove from environment. Delete from source.
6. **Audit.** Confirm the credential is gone and the workload is operating on the new pattern.

Phase 4: Decommission and audit (week 12+)

- **Owner attestation.** Every remaining NHI has an explicit owner who attests it should still exist.
- **The 5% explicit decision.** For NHIs that haven't been migrated, document why: legacy system that doesn't support modern auth, third-party integration where the partner is the constraint, special-case workflow needing architectural review. The remaining is by deliberate decision, not by oversight.
- **Continuous discovery.** New NHIs created after the migration must enter the inventory at registration. The IDP enforces ownership; new clients without owners are rejected.

KPIs and acceptance tests

The KPIs worth tracking through the migration:

KPI	Target	Why it matters
% of machine identities without secrets	≥ 95% within 90 days	The headline migration metric
Owner coverage	100%	Orphan accounts are how migration fails
Token issuance and validation latency	Low-latency, measured against your baseline	The replacement architecture must perform; credentials are short-lived, so issuance happens constantly
PoP failure rate	Any > 0 alerted; > 0.05% critical	Misconfiguration detection
JWT replay attempts	0 successful	Replay is the canary for token theft
Mean time to rotate keys	≤ 24 hours	Demonstrates operational maturity
Time from compromise alert to revocation	Minutes, not hours	Incident response readiness

Where the migration commonly stalls

Legacy systems that don't support modern auth. Mainframe systems, older enterprise software, integration platforms designed before OAuth 2.0. The decision: invest in a broker (a service that

fronts the legacy system with modern auth and presents a standards-compliant interface to the rest of the architecture), retire the legacy system, or accept-and-document the residual risk.

Vendor SaaS without OAuth client-credentials support. Some SaaS vendors only offer API-key authentication. The decision: pressure the vendor to support modern auth (most are responsive to enterprise customers), use a broker, or accept-and-document.

CI/CD pipelines with deeply embedded admin keys. Migration requires careful sequencing to avoid breaking active pipelines. The pattern: stand up OIDC federation in parallel, migrate one repo at a time, validate each, then deprecate the old keys.

Cross-tenant integrations. Cross-organization migrations require coordination with the counterparty's security team. Plan for the longer cycle.

The "we can't break production" objection. Real, but solvable. The pattern is parallel operation (old credential + new credential both work for a transition period) rather than cut-over. The transition period closes when the new pattern is verified.

Operational considerations

Discovery is iterative. The inventory will surface NHIs you didn't know about. Plan for it.

Migration breaks things. Even careful migrations cause incidents. Have rollback plans. Have monitoring. Schedule migrations during low-risk windows where possible.

Cultural shift. Engineers who are used to "just put a secret in the vault" need to learn the new patterns. Documentation, training, and team-by-team enablement matter.

Cost. Cloud-native workload identity is generally free or low-cost; the operational savings (no secret-rotation runbooks, no incident-response when secrets leak) typically outweigh the migration cost.

Compliance posture during migration. Auditors will ask about the migration. Document the plan, the progress, and the residual risk for any NHIs that haven't been migrated.

What this is not

Not a one-time project. New NHIs get created. New systems get adopted. The migration is a continuous discipline. The IDP enforces the new patterns at registration; the inventory is continuously updated.

Not a vault replacement. The vault remains useful for genuine secrets (third-party API keys for vendors that don't support modern auth, certificates that need distribution, encrypted-at-rest material). The vault is the exception store, not the default store.

Not a panacea for credential management. Workload identity, sender-constrained tokens, and OIDC federation eliminate one specific class of failure (long-lived shared secret leakage). Other classes

(over-privileged accounts, missing revocation, audit gaps) remain and must be addressed in their own right.

Standards alignment

Standard	Relevance
RFC 7523	JWT client assertion as the authentication primitive
RFC 8705	mTLS for sender-constrained tokens
RFC 9449	DPoP for app-layer PoP
RFC 9700	OAuth 2.0 Security Best Current Practice
SPIFFE/SPIRE	Cross-cloud workload attestation
NIST SP 800-207	Zero Trust applied to non-human identity
OAuth 2.0 / OIDC	Authorization-server foundation
SOC 2 / ISO/IEC 27001	Operational controls for key management
FedRAMP IA controls	Authentication for federal-facing CSPs

Key Takeaway

Service-account replacement is the practical effort to eliminate long-lived shared secrets (passwords, API keys, OAuth client secrets, SSH keys) from the machine-to-machine authentication architecture. The replacement is three patterns: cloud-native workload identity for in-cloud workloads (AWS IAM Roles for Service Accounts/IRSA, GCP Workload Identity Federation, Azure Managed Identity, SPIFFE/SPIRE cross-cloud), sender-constrained tokens for cross-boundary paths (RFC 7523 JWT client assertions plus RFC 8705 mTLS or RFC 9449 DPoP), and OIDC federation for CI/CD pipelines (GitHub Actions, GitLab CI). The migration sequence: inventory all NHIs, classify by blast radius, migrate highest-leverage paths first (production admin, CI/CD with admin scope, cross-tenant integrations), validate each, and decommission old credentials. KPI target: ≥ 95% of machine identities migrated off client secrets within 90 days, owner coverage 100%, mean time to rotate keys ≤ 24 hours. The vault remains useful as the exception store but moves out of the default-control-plane role; new credentials are issued at the moment of action with lifetimes measured in minutes, not years.

FAQ

Why are service-account passwords a problem?

Service-account passwords (and the API keys, OAuth client secrets, and SSH keys that play the same role) are long-lived shared secrets. They accumulate in source control, vault entries, environment files, CI/CD configurations, and developer laptops. They get committed to public repos, exposed in logs, harvested from stolen laptops, and exfiltrated from compromised vaults. Once leaked, they remain valid for the duration of their lifetime (often years) and can be replayed from anywhere on the internet. Multiple high-profile breaches (CircleCI 2023, Okta support tooling 2023, others) have started with leaked long-lived NHI credentials.

What replaces a service-account password?

Three patterns. For workloads inside a cloud, cloud-native workload identity (AWS IAM Roles for Service Accounts/IRSA, GCP Workload Identity Federation, Azure Managed Identity, or SPIFFE/SPIRE for cross-cloud). For service-to-service paths across boundaries, sender-constrained access tokens (mTLS per RFC 8705 or DPoP per RFC 9449) issued via JWT client assertions per RFC 7523. For CI/CD pipelines, OIDC federation between the CI provider (GitHub Actions, GitLab CI) and the cloud, issuing short-lived credentials at job execution. The pattern in all three: no persistent shared secret; credentials are issued at the moment of action with a lifetime measured in minutes.

How do I prioritize what to migrate first?

Prioritize by blast radius. Production-impacting service accounts (database admin, IAM admin, secrets-manager admin, payment-relevant systems) first. CI/CD pipeline credentials with admin scope second (a leaked CI key can sit unnoticed for months before use). Cross-tenant or cross-organization integrations third. Internal lower-impact service accounts last. The principle: the higher the blast radius if the credential leaks, the higher the priority for replacement.

What's a reasonable timeline for service-account replacement?

ScrambleID's KPI target is ≥ 95% of machine identities migrated off client secrets within 90 days. Mature platform teams achieve this with a focused effort. The remaining 5% is typically legacy systems, third-party integrations that haven't supported modern auth yet, or special-case workflows that need explicit architectural review. Owner attestation cycles surface the remaining cases for explicit decision: migrate, retire, or accept-and-document.

Do I still need a secrets vault?

Yes, but for different things. The vault remains useful for storing secrets that genuinely have to exist (third-party API keys for vendors that don't yet support modern auth, certificates that need to be distributed to specific endpoints, encrypted-at-rest material). The vault is no longer the primary

control plane for service-to-service authentication; that's the IDP and the cloud-native workload identity systems. The vault becomes the exception store, not the default store.

How do I know I actually migrated everything?

Inventory and KPI tracking. ScrambleID's M2M control plane tracks: percentage of M2M clients with no client secret (target ≥ 95%), owner coverage (target 100%), tokens issued per client per day (anomaly detection on dormant clients suddenly active), JWT replay attempts (Overwatch metric A11), and key rotation cadence. Discovery scans (vault audits, IAM policy mining, environment-variable inventory in CI/CD) surface clients that exist outside the inventory. The remaining 5% is by attestation, not by accident.

References (public)

- RFC 7523 (JWT Profile for OAuth 2.0): https://datatracker.ietf.org/doc/html/rfc7523
- RFC 8705 (OAuth 2.0 Mutual-TLS): https://datatracker.ietf.org/doc/html/rfc8705
- RFC 9449 (OAuth 2.0 DPoP): https://www.rfc-editor.org/rfc/rfc9449.html
- RFC 9700 (OAuth 2.0 Security BCP): https://www.rfc-editor.org/rfc/rfc9700.html
- SPIFFE/SPIRE: https://spiffe.io/
- AWS IAM Roles for Service Accounts (IRSA): https://docs.aws.amazon.com/eks/latest/userguide/iam-roles-for-service-accounts.html
- Google Cloud Workload Identity Federation: https://cloud.google.com/iam/docs/workload-identity-federation
- Azure Managed Identities: https://learn.microsoft.com/en-us/entra/identity/managed-identities-azure-resources/
- NIST SP 800-207 (Zero Trust): https://csrc.nist.gov/publications/detail/sp/800-207/final

---

Related reading

- What Is Non-Human Identity (NHI)?
- M2M Authentication Without Secrets
- Sender-Constrained Tokens (mTLS, DPoP)
- Cloud Workload Identity Compared
- GitHub Actions OIDC Federation Across Clouds
- client_secret vs JWT Client Assertion vs mTLS
- What Is AI Agent Identity?
- Compliance Mapping: NIST and CISA