# Agent Identity in ATB - Complete Guide

How AI Agents Get SPIFFE Identities via SPIRE

ATB Team

January 2026

## Contents

# Agent Identity in ATB: Complete Guide

## Executive Summary

This guide explains how AI agents in ATB (Agent Trust Broker) obtain their cryptographic identities using SPIFFE/SPIRE. Unlike traditional systems that use static API keys or secrets, ATB agents get short-lived, automatically-renewed certificates based on **where and how they run**.

**Key Benefits:**

- No secrets to manage or rotate
- Certificates valid for only 10 minutes
- Identity tied to workload attestation
- Private keys never touch disk

---

# Part 1: How the Agent Gets Its First SPIFFE ID

## Overview

The identity issuance process has 4 main steps:

| Step | Actor | Action |
|---|---|---|
| 0 | Admin | Registers workload entry in SPIRE Server |
| 1 | Kubernetes | Starts agent pod with namespace/SA |
| 2 | Agent | Calls SPIRE Workload API via Unix socket |
| 3 | SPIRE Agent | Attests workload and issues X.509 certificate |

---

### Step 0: Admin Pre-Registers Workload Entry

Before any agent can get an identity, an admin registers an **entry**:

```
spire-server entry create \
  -spiffeID spiffe://example.org/agent/sales-bot \
  -parentID spiffe://example.org/spire/agent/k8s_psat/demo-cluster \
  -selector k8s:ns:ai-agents \
  -selector k8s:sa:sales-bot-sa
```

**Parameters:**

| Parameter | Value | Meaning |
|---|---|---|
| -spiffeID | spiffe://example.org/agent/sales-bot | Identity to issue |
| -parentID | spiffe://...demo-cluster | Which SPIRE Agent can issue |
| -selector k8s:ns:ai-agents | Kubernetes namespace | Pod must be here |
| -selector k8s:sa:sales-bot-sa | ServiceAccount | Pod must use this |

**This creates a rule:** "Any pod in namespace `ai-agents` using ServiceAccount `sales-bot-sa` gets the SPIFFE ID `spiffe://example.org/agent/sales-bot`"

---

### Step 1: Agent Workload Starts

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sales-bot
  namespace: ai-agents
spec:
  template:
    spec:
      serviceAccountName: sales-bot-sa
      containers:
        - name: agent
          image: company/sales-bot:v1.2
          volumeMounts:
            - name: spire-agent-socket
              mountPath: /run/spire/sockets
              readOnly: true
      volumes:
        - name: spire-agent-socket
          hostPath:
            path: /run/spire/sockets
            type: Directory
```

---

### Step 2: Agent Requests SVID

**Python:**

```python
from pyspiffe.workloadapi import X509Source

source = X509Source(
    workload_api_addr="unix:///run/spire/sockets/agent.sock"
)
svid = source.svid
print(f"My SPIFFE ID: {svid.spiffe_id}")
# Output: spiffe://example.org/agent/sales-bot
```

**Go:**

```go
source, _ := workloadapi.NewX509Source(ctx,
    workloadapi.WithClientOptions(
        workloadapi.WithAddr("unix:///run/spire/sockets/agent.sock"),
    ),
)
```

```
svid, _ := source.GetX509SVID()
fmt.Printf("My SPIFFE ID: %s\n", svid.ID)
```

---

### Step 3: SPIRE Agent Attests

SPIRE Agent checks:

1. **WHO** is calling? (Unix socket peer credentials)
2. **WHAT** Kubernetes context? (Query K8s API)
   - Namespace: ai-agents (MATCH)
   - ServiceAccount: sales-bot-sa (MATCH)
3. **MATCH** against entries? YES
4. **DECISION**: Issue SVID

---

### Step 4: X.509-SVID Issued

| Field | Value |
|---|---|
| Subject Alternative Name | URI: spiffe://example.org/agent/sales-bot |
| Validity | 10 minutes (auto-renewed) |
| Public Key | Ed25519 |
| Private Key | In memory only (never on disk) |

---

## Part 2: Security Analysis

### Traditional vs SPIFFE Approach

| Traditional | SPIFFE/SPIRE |
|---|---|
| API keys in env vars | No secrets to manage |
| Manual rotation | Auto-rotated every 10 min |
| Secrets can leak | Identity from attestation |
| Anyone with secret = access | Must be right workload |

### Attack Scenarios

| Attack | Why It Fails |
|---|---|
| Steal certificate | Expires in 10 minutes |
| Copy to another machine | Private key in memory only |
| Impersonate from different pod | Attestation checks namespace/SA |
| Create fake pod | Need K8s RBAC for namespace |

---

## Part 3: Different Attestation Methods

### Kubernetes (Most Common)

```
-selector k8s:ns:ai-agents
-selector k8s:sa:sales-bot-sa
-selector k8s:pod-label:app:sales-bot
```

### Docker

```
-selector docker:label:ai.agent:sales-bot
```

### AWS

```
-selector aws:iamrole:arn:aws:iam::123456789:role/SalesBotRole
```

### Unix

```
-selector unix:uid:1001
-selector unix:path:/opt/agents/sales-bot
```

---

## Part 4: Complete Example

### 1. Setup SPIRE

```
helm install spire-server spire/spire-server -n spire-system
helm install spire-agent spire/spire-agent -n spire-system
```

### 2. Register Entry

```
kubectl exec -n spire-system spire-server-0 -- \
  spire-server entry create \
    -spiffeID spiffe://example.org/agent/sales-bot \
    -parentID spiffe://example.org/spire/agent/k8s_psat/demo-cluster \
    -selector k8s:ns:ai-agents \
    -selector k8s:sa:sales-bot-sa
```

### 3. Create ServiceAccount

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sales-bot-sa
  namespace: ai-agents
```

### 4. Deploy Agent

```
apiVersion: apps/v1
kind: Deployment
```

```yaml
metadata:
  name: sales-bot
  namespace: ai-agents
spec:
  replicas: 1
  selector:
    matchLabels:
      app: sales-bot
  template:
    metadata:
      labels:
        app: sales-bot
    spec:
      serviceAccountName: sales-bot-sa
      containers:
        - name: agent
          image: company/sales-bot:v1.2
          env:
            - name: SPIFFE_ENDPOINT_SOCKET
              value: "unix:///run/spire/sockets/agent.sock"
          volumeMounts:
            - name: spire-agent-socket
              mountPath: /run/spire/sockets
              readOnly: true
      volumes:
        - name: spire-agent-socket
          csi:
            driver: "csi.spiffe.io"
            readOnly: true
```

## 5. Agent Code

```python
from pyspiffe.workloadapi import X509Source
import os

def main():
    socket = os.environ.get(
        "SPIFFE_ENDPOINT_SOCKET",
        "unix:///run/spire/sockets/agent.sock"
    )
    source = X509Source(workload_api_addr=socket)

    svid = source.svid
    print(f"I am: {svid.spiffe_id}")
    # Output: spiffe://example.org/agent/sales-bot
```

---

## Summary

The agent **never stores secrets**. Its identity comes from:

- **Where it runs** (Kubernetes namespace)
- **How it runs** (ServiceAccount)
- **Cryptographic proof** (SPIRE attestation)

The entire process is automatic once the admin registers the workload entry.

---

## Related Documentation

- How AgentAuth Works
- SPIFFE Integration Guide
- Security Hardening
- Audit Logging