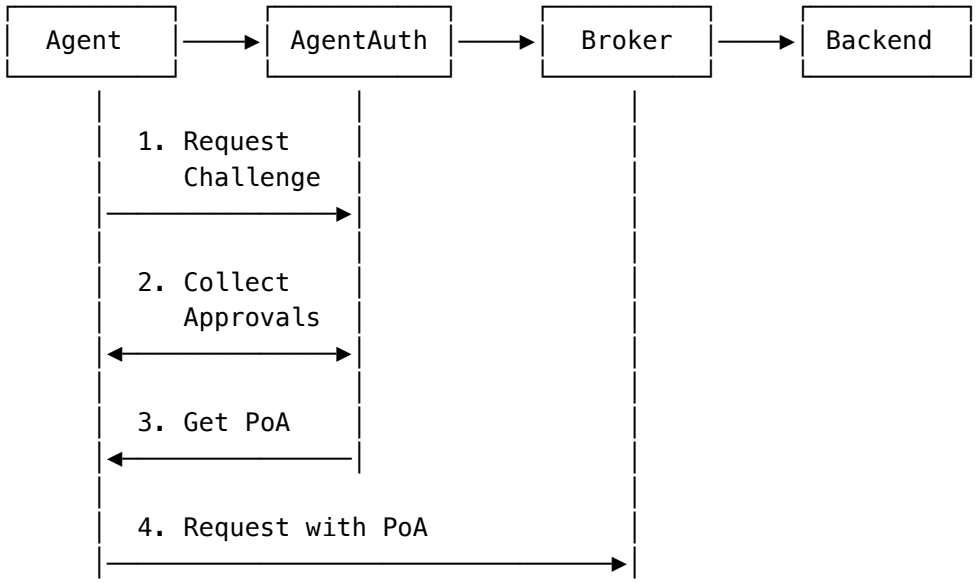
Authentication & Authorization Guide

This guide explains how ATB authenticates agents and authorizes their actions using Proof-of-Authorization (PoA) tokens.

Overview

ATB uses a **zero-trust** model where every action must be explicitly authorized:



Identity: SPIFFE/SPIRE

Every workload in ATB has a cryptographic identity via SPIFFE.

SPIFFE ID Format

`spiffe://<trust-domain>/<workload-path>`

Examples: - `spiffe://prod.company.com/ns/agents/sa/claude-assistant` - `spiffe://prod.company.com/ns/connector`

How Identity Works

1. **SPIRE Agent** runs on each node
2. **Workloads** request SVIDs (SPIFFE Verifiable Identity Documents)
3. **X.509-SVID** used for mTLS connections
4. **JWT-SVID** can be used for API authentication

Configuring SPIFFE in ATB

The broker extracts the SPIFFE ID from the client certificate:

```yaml
# values.yaml
broker:
  tls:
    mode: "spiffe"  # Use SPIRE for workload identity
    spireSocketPath: "/run/spire/sockets/agent.sock"
```

Proof-of-Authorization (PoA) Tokens

PoA tokens are short-lived, signed JWTs that authorize specific actions.

PoA Token Structure

```
{
  "header": {
    "alg": "RS256",
    "typ": "JWT",
    "kid": "key-2026-01"
  },
  "payload": {
    "iss": "atb-agentauth",
    "sub": "spiffe://example.org/agent/demo",
    "aud": "atb-broker",
    "iat": 1736679600,
    "exp": 1736679900,
    "jti": "poa_abc123xyz",

    "act": "crm.contact.update",
    "con": {
      "max_records": 10,
      "allowed_fields": ["name", "email"]
    },
    "leg": {
      "basis": "contract",
      "ref": "MSA-2026-001",
      "jurisdiction": "US",
      "accountable_party": {
        "type": "human",
        "id": "user@example.com"
      }
    },
    "apr": [
      {
        "approver_id": "manager@example.com",
        "approved_at": "2026-01-12T10:00:00Z",
        "required": true
      }
    ]
  }
}
```

PoA Claims Explained

Claim	Required	Description
`iss`	Yes	Issuer (AgentAuth service)
`sub`	Yes	Subject (agent's SPIFFE ID)
`aud`	Yes	Audience (ATB Broker)
`iat`	Yes	Issued at (Unix timestamp)
`exp`	Yes	Expiration (Unix timestamp)
`jti`	Yes	Unique token ID (for replay protection)

Claim	Required	Description
`act`	Yes	Action being authorized
`con`	No	Constraints (limits, filters)
`leg`	Yes	Legal basis for the action
`apr`	Depends	Approvals (required for medium/high risk)

Legal Basis (`leg`)

Every PoA must include a legal basis explaining why the action is permitted:

```
{
  "leg": {
    "basis": "contract",      // contract, consent, legitimate_interest, legal_obligation
    "ref": "MSA-2026-001",    // Reference to legal document
    "jurisdiction": "US",      // Legal jurisdiction
    "accountable_party": {
      "type": "human",         // human, organization
      "id": "user@example.com" // Who is accountable
    }
  }
}
```

Constraints (`con`)

Constraints limit what the action can do:

```
{
  "con": {
    "max_amount": 10000,
    "currency": "USD",
    "allowed_vendors": ["VENDOR001", "VENDOR002"],
    "max_records": 100,
    "exclude_fields": ["ssn", "credit_card"]
  }
}
```

Approval Flow

Medium-Risk Actions (Single Approval)

```
# 1. Create challenge
POST /v1/challenge
{
  "agent_spiffe_id": "spiffe://example.org/agent/demo",
  "action": "crm.contact.update",
  "legal_basis": { ... }
}

# Response
{
  "challenge_id": "ch_abc123",
  "requires_approval": true,
  "required_approvers": 1,
  "risk_tier": "medium"
```

```
}

# 2. Submit approval
POST /v1/challenge/ch_abc123/approve
{
  "approver_id": "manager@example.com",
  "reason": "Approved for support case #1234"
}

# Response includes PoA token
{
  "poa": "eyJhbGciOiJSUzI1NiI...",
  "expires_at": "2026-01-12T12:05:00Z"
}
```

High-Risk Actions (Dual Control)

High-risk actions require **two distinct approvers**:

```
# 1. Create challenge
POST /v1/challenge
{
  "agent_spiffe_id": "spiffe://example.org/agent/demo",
  "action": "sap.payment.execute",
  "legal_basis": { ... }
}

# Response
{
  "challenge_id": "ch_xyz789",
  "requires_approval": true,
  "required_approvers": 2,
  "risk_tier": "high"
}

# 2. First approval
POST /v1/challenge/ch_xyz789/approve
{
  "approver_id": "finance-manager@example.com"
}

# Response (still pending)
{
  "status": "pending",
  "approvals": 1,
  "required": 2
}

# 3. Second approval (different person!)
POST /v1/challenge/ch_xyz789/approve
{
  "approver_id": "cfo@example.com"
}
```

```
# Response (complete)
{
  "poa": "eyJhbGciOiJSUzI1NiI...",
  "expires_at": "2026-01-12T12:05:00Z"
}
```

Dual Control Rules

- Two approvers must be **distinct** (different `approver_id`)
- The **requester** cannot be an approver
- Both approvals must happen before the challenge expires
- Approval order doesn't matter

Token Validation

The broker validates PoA tokens against multiple checks:

1. Signature Verification

```
Token signed with → AgentAuth private key
Broker verifies with → AgentAuth public key (via JWKS)
```

JWKS endpoint: `GET /v1/.well-known/jwks.json`

2. Claims Validation

Check	Failure Reason
`exp` in past	`token_expired`
`iat` in future	`token_not_yet_valid`
`aud` != broker	`invalid_audience`
`sub` != agent	`subject_mismatch`
`act` mismatch	`action_not_authorized`

3. Replay Protection

Each `jti` (token ID) is cached until expiration. Reusing a token returns:

```
{
  "error": "token_already_used",
  "message": "This PoA token has already been consumed"
}
```

4. Policy Evaluation

OPA evaluates the request against the policy:

```
decision = {
  "allow": allow,
  "risk_tier": risk_tier,
  "reasons": reasons
}

allow {
```

```
  valid_poa
  valid_legal_basis
  constraints_satisfied
  approvals_sufficient
}
```

Token Lifetime

Setting	Default	Description
`POA_TTL_SECONDS`	300 (5 min)	How long a PoA token is valid
`CHALLENGE_TTL_SECONDS`	300 (5 min)	How long to collect approvals

For high-security environments, reduce these values:

```yaml
agentauth:
  env:
    POA_TTL_SECONDS: "60"      # 1 minute tokens
    CHALLENGE_TTL_SECONDS: "120"  # 2 minutes to approve
```

Key Management

Signing Keys

AgentAuth signs PoA tokens with an Ed25519 or RSA private key:

```bash
# Generate Ed25519 key (recommended)
openssl genpkey -algorithm ed25519 -out signing.key

# Create Kubernetes secret
kubectl create secret generic atb-agentauth-signing-key \
  --from-file=ed25519_privkey_pem=signing.key \
  -n atb
```

Key Rotation

1. Generate new key with new `kid`
2. Add new key to JWKS
3. Wait for old tokens to expire
4. Remove old key from JWKS

Security Best Practices

1. Minimize Token Lifetime

```yaml
POA_TTL_SECONDS: "60"  # Shorter = more secure
```

2. Use Specific Actions

```
// Good: Specific action
{ "act": "crm.contact.update" }

// Bad: Overly broad
{ "act": "crm.*" }
```

3. Always Include Constraints

```
{
  "con": {
    "max_records": 10,
    "allowed_fields": ["name", "email"]
  }
}
```

4. Require Legal Basis

Every action should have a documented legal basis for GDPR/compliance:

```
{
  "leg": {
    "basis": "contract",
    "ref": "Customer Agreement Section 4.2",
    "jurisdiction": "EU"
  }
}
```

5. Monitor for Anomalies

Set up alerts for: - High volume of denied requests - Unusual approval patterns - Token reuse attempts - After-hours high-risk actions

See Observability Guide for alerting configuration.