Authentication Patterns for Agents and MCP Services

Using JWK Endpoints with Embedded Certificates

Technical Documentation v1.0

Author: Shankar Kalyanarman

Table of Contents

1. Executive Summary

The Model Context Protocol (MCP) recently evolved to support comprehensive OAuth 2.1-based authentication, addressing enterprise security requirements for protecting sensitive data in transit. This document provides actionable guidance for implementing secure authentication and encryption patterns for MCP services using JWK endpoints.

Key Findings: MCP servers should operate purely as Resource Servers, delegating authentication to external authorization servers rather than implementing custom authentication logic. This aligns with the principle of separation of concerns and leverages battle-tested identity providers.

2. MCP Authentication Architecture

The 2025 MCP specification update introduced OAuth 2.1 integration with mandatory PKCE and Dynamic Client Registration, fundamentally changing how authentication works in the MCP ecosystem.

2.1 Core Components

Component	Role	Key Responsibilities
MCP Server	Resource Server	• Validates OAuth tokens • Hosts JWK endpoints • Implements certificate rotation • Encrypts sensitive payloads
AI Agent	OAuth Client	• Implements OAuth 2.1 with PKCE • Manages token lifecycle • Supports dynamic registration • Handles multi-server connections
Authorization Server	Identity Provider	• Authenticates users • Issues OAuth tokens • Manages consent • Supports dynamic client registration

3. JWK Endpoint Design

For JWK endpoint design, RFC 7517 provides clear guidance on hosting both encryption ("enc") and authentication ("auth") keys. The standard pattern involves a single JWKS endpoint returning multiple keys distinguished by the `use` parameter.

3.1 JWK Endpoint Structure

```
{
  "keys": [
    {
      "kty": "RSA",
      "use": "sig",
      "kid": "auth-2025-01",
      "alg": "RS256",
      "n": "...",
      "e": "AQAB"
    },
    {
      "kty": "RSA",
      "use": "enc",
      "kid": "enc-2025-01",
      "alg": "RSA-OAEP-256",
      "n": "...",
      "e": "AQAB"
    }
  ]
}
```

Critical Security Constraint: RFC 7517 explicitly prohibits using the same key for both signing and encryption due to cross-protocol attack vulnerabilities. Always generate distinct key pairs for each purpose, using the `use` parameter to clearly delineate their roles.

3.2 Key Parameters

Parameter	Purpose	Values
kty	Key Type	RSA, EC, OKP
use	Key Usage	sig (signing), enc (encryption)
kid	Key ID	Unique identifier (e.g., "auth-2025-01")

alg	Algorithm	RS256, ES256, RSA-OAEP-256

4. 3-Certificate Rotation Model

Maintaining three active certificates simultaneously emerges as the optimal pattern for seamless rotation without service interruption. This model, successfully implemented by CockroachDB and major cloud providers, maintains certificates in three distinct states.

4.1 Certificate States

1. **Primary (Active)** - Currently signing new tokens and encrypting data
2. **Secondary (Standby)** - Available for validation, ready to become primary
3. **Retired (Grace Period)** - Still valid for verification but no longer used for new operations

4.2 Rotation Timeline

Phase	Timing	Action
Certificate Generation	60-75 days before expiry	Create new certificate in standby mode
Certificate Promotion	5-15 days after generation	Promote standby to primary
Grace Period	Max token lifetime + buffer	Keep retired cert for validation
Final Removal	After all tokens expire	Remove oldest certificate

4.3 Implementation State Machine

```
STAGED → ENABLED → DISABLED → DELETED
```

Best Practice: HashiCorp Vault's PKI engine with multiple issuer support provides an excellent reference implementation, allowing a single PKI mount to manage multiple Certificate Authority certificates with automatic default issuer selection.

5. Hybrid Encryption

Research conclusively demonstrates that hybrid encryption approaches are superior to asymmetric-only solutions for API security contexts. The evidence from NIST standards, academic papers, and industry implementations overwhelmingly supports combining asymmetric and symmetric cryptography.

5.1 Recommended Approach

Component	Algorithm	Purpose
Key Exchange	ECDH with P-256 or X25519	Initial key establishment
Symmetric Encryption	AES-256-GCM or ChaCha20-Poly1305	Data encryption
Authentication	ECDSA	Digital signatures and identity verification
Session Management	Short-lived keys	Rotating every 24 hours or per session

5.2 Performance Benefits

Performance Gains:

- 100x+ performance improvement for bulk data processing
- Enhanced forward secrecy through ephemeral key usage
- Industry alignment with TLS 1.3 and Signal Protocol
- Compliance with FIPS 140-2 and Common Criteria requirements

5.3 Nested JWT Pattern

For protecting PII data, implement the nested JWT pattern following the sign-then-encrypt order:

1. Create a JWS (JSON Web Signature) containing the sensitive claims
2. Encrypt the entire JWS using JWE (JSON Web Encryption)
3. Set cty: "JWT" in the JWE header to indicate nested structure

```
{
  "alg": "RSA-OAEP-256",
  "enc": "A256GCM",
  "cty": "JWT",
  "kid": "enc-2025-01"
}
```

6. Agent Perspective

From the agent's viewpoint, the authentication landscape presents unique challenges and opportunities. Modern frameworks like Cloudflare's Agents SDK have simplified this complexity by providing built-in authentication flows, tool discovery, and connection management.

6.1 Authentication Flow

1. **Initial request without credentials** - The agent attempts to access a protected endpoint
2. **401 Unauthorized response** - The MCP server replies with authentication challenge
3. **OAuth flow initiation** - The agent generates PKCE parameters
4. **User authorization** - Browser opens for user authentication and consent
5. **Token exchange** - Agent exchanges authorization code for access tokens
6. **Authenticated requests** - All subsequent requests include the Bearer token

6.2 Dynamic Client Registration

Dynamic Client Registration allows AI agents to autonomously discover, register, and authenticate with MCP servers at runtime. This eliminates the M×N registration problem where every agent would need manual registration with every MCP server.

```
POST /register HTTP/1.1
Content-Type: application/json
Accept: application/json
Host: mcp.server.com

{
  "client_name": "My AI Agent",
  "redirect_uris": ["http://localhost:8080/callback"],
  "grant_types": ["authorization_code"],
  "response_types": ["code"],
  "token_endpoint_auth_method": "none"
}
```

6.3 Multi-Server Connection Management

Key Capabilities:

- Automatic namespacing - Tools from different servers are namespaced to avoid conflicts
- Connection state tracking - Automatic reconnection on connection loss
- Real-time capability updates - Agents receive notifications when server capabilities change
- Transport abstraction - Support for SSE, HTTP, and upcoming Streamable HTTP

7. Implementation Recommendations

7.1 JWK Endpoint Configuration

- Host your JWKS endpoint at `/.well-known/jwks.json`
- Set appropriate caching headers: `Cache-Control: max-age=300`
- Serve over HTTPS with strong TLS configuration
- Include both current and upcoming keys during rotation periods
- Use distinct key IDs (kid) for each key version
- Implement rate limiting and monitoring

7.2 Key Management Strategy

1. Generate separate key pairs for signing and encryption operations
2. Use Hardware Security Modules (HSMs) for production key storage
3. Implement automated rotation with the 3-certificate model
4. Monitor key usage with comprehensive audit logging
5. Plan for emergency rotation with documented procedures

7.3 Security Architecture

- Delegate authentication to established identity providers (Azure AD, Auth0, Okta)
- Validate all tokens against the `aud` (audience) and `iss` (issuer) claims
- Implement proper scope verification based on OAuth 2.1 standards
- Use stateless JWT sessions where possible for scalability

8. Security Considerations

8.1 Token Validation

```
// Required validation steps for every request
1. Verify token signature using public key from JWK endpoint
2. Check token expiration (exp claim)
3. Validate issuer (iss) matches expected authorization server
4. Confirm audience (aud) includes this resource server
5. Verify scope claims match required permissions
6. Check token binding if implemented
```

8.2 Common Vulnerabilities to Avoid

Vulnerability	**Mitigation**
Key Confusion Attacks	Never use same key for signing and encryption
Token Replay	Implement jti (JWT ID) tracking for one-time tokens
Insufficient Scope Validation	Always verify scope claims before granting access
Missing Token Binding	Consider DPoP for proof-of-possession

9. Performance Optimization

9.1 Caching Strategies

- Cache JWKS responses respecting HTTP cache headers
- Implement local token validation cache (5-minute TTL)
- Use connection pooling for JWK endpoint requests
- Consider CDN distribution for JWKS endpoints

9.2 Encryption Optimization

Performance Tips:

- Use field-level encryption for large payloads with mixed sensitivity
- Implement connection pooling for cryptographic operations
- Separate cryptographic processing from API gateway for scale
- Consider hardware acceleration for AES operations

10. Conclusion

Implementing secure authentication for MCP services requires careful orchestration of multiple components from both server and client perspectives. For servers: OAuth 2.1 authorization flows, JWK endpoint design with dual-purpose keys, the 3-certificate rotation model, and hybrid encryption for data protection. For agents: dynamic client registration, robust token management, and connection multiplexing capabilities.

The ecosystem is rapidly evolving with strong support from major cloud providers and identity platforms. Success depends on leveraging established identity providers for authentication, implementing proper key separation and rotation strategies, and choosing hybrid encryption approaches that balance security with performance.

Key Takeaways:

- Use OAuth 2.1 with established identity providers
- Implement separate keys for signing and encryption
- Adopt the 3-certificate rotation model
- Choose hybrid encryption for optimal performance
- Enable dynamic client registration for scalability

Authentication Patterns for Agents and MCP Services - Technical Documentation v1.0

Author: Shankar Kalyanarman