# Dokumentasjon: API Key-autentisering i .NET 8 uten middleware (Attribute + Authorization Filter)

## 1. Introduksjon

Denne veiledningen viser hvordan du sikrer API-endepunkter i **.NET 8** ved hjelp av **API Key** uten å bruke middleware. I stedet brukes en `[ApiKey]`**-attributt** sammen med et **authorization filter** (`IAuthorizationFilter` / `IAsyncAuthorizationFilter`) som validerer nøkkelen før controller-handlingen kjøres.

**Når passer API Keys?**

- Tjeneste-til-tjeneste-tilgang (skript, integrasjoner, IoT).
- Enkle scenarier der du vil begrense tilgang uten brukerinnlogging.
- Du ønsker å **identifisere klientapplikasjoner**, ikke sluttbrukere.
- Du vil kunne **spore bruken** pr. klient og eventuelt differensiere **tilgang (scopes)**.

> Merk: API Keys identifiserer normalt **applikasjoner**. For sluttbrukerautentisering, vurder OAuth2/OIDC eller JWT.

---

## 2. Oversikt over løsningen

- `[ApiKey]` attributt på controller/handling for å aktivere sjekken.
- `ApiKeyAuthorizationFilter` utfører validering av nøkkelen.
- `IApiKeyValidator` kapsler valideringslogikk (fra `appsettings`, database, osv.).
- **Konfigurerbart header-navn** (standard: `x-api-key`).

---

## 3. Konfigurasjon (appsettings.json)

**Enkel (én nøkkel):**

```
{
  "ApiKeyOptions": {
    "HeaderName": "x-api-key",
    "Keys": [
      { "key": "sk_test_123", "appId": "test-app", "scopes": [ "users:read" ] }
    ]
  }
}
```

**Avansert (flere nøkler m/scopes):**

```
{
  "ApiKeyOptions": {
    "HeaderName": "x-api-key",
    "Keys": [
      { "key": "sk_test_123", "appId": "test-app", "scopes": [ "users:read" ] },
      { "key": "sk_live_ABC", "appId": "reporter", "scopes": [ "users:read",
"users:write" ] }
    ]
  }
}
```

> **Sikkerhet:** Lagre aldri rå nøkler i repo for prod. Bruk Secret Manager/KeyVault. Vurder hashing i lagring (se validator under).

---

## 4. Domenemodeller og Options

```
public class ApiKeyItem
{
    public string Key { get; set; } = "";
    public string AppId { get; set; } = "";
    public string[] Scopes { get; set; } = Array.Empty<string>();
}

public class ApiKeyOptions
{
    public string HeaderName { get; set; } = "x-api-key";
    public List<ApiKeyItem> Keys { get; set; } = new();
}
```

Registrer options i `Program.cs`:

```
builder.Services.Configure<ApiKeyOptions>
(builder.Configuration.GetSection("ApiKeyOptions"));
```

---

## 5. Attributt og Filter

### 5.1. `[ApiKey]`-attributt (ServiceFilter/TypeFilter)

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;

[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method)]
public sealed class ApiKeyAttribute : TypeFilterAttribute
{
    public ApiKeyAttribute() : base(typeof(ApiKeyAuthorizationFilter)) { }
}
```

> `TypeFilterAttribute` registrerer avhengigheter gjennom DI automatisk. Alternativt kan du bruke `ServiceFilterAttribute` og registrere filteret eksplisitt.

5.2. Authorization-filter (async-variant)

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;

public class ApiKeyAuthorizationFilter : IAsyncAuthorizationFilter
{
    private readonly IApiKeyValidator _validator;
    private readonly IOptions<ApiKeyOptions> _options;

    public ApiKeyAuthorizationFilter(IApiKeyValidator validator,
IOptions<ApiKeyOptions> options)
    {
        _validator = validator;
        _options = options;
    }

    public async Task OnAuthorizationAsync(AuthorizationFilterContext context)
    {
        var headerName = _options.Value.HeaderName ?? "x-api-key";
        var hasHeader =
context.HttpContext.Request.Headers.TryGetValue(headerName, out var provided);

        if (!hasHeader || string.IsNullOrWhiteSpace(provided))
        {
            context.Result = new UnauthorizedObjectResult(new { error = $"
{headerName} header is missing." });
            return;
        }

        var result = await _validator.ValidateAsync(provided!);

        if (!result.Ok)
        {
            context.Result = new UnauthorizedObjectResult(new { error = "Invalid
API Key." });
            return;
        }

        // (Valgfritt) Legg inn appId/scopes i HttpContext.Items for senere bruk
        context.HttpContext.Items["appId"] = result.AppId;
        context.HttpContext.Items["scopes"] = result.Scopes;
    }
}
```

## 6. Validator

### 6.1. Grensesnitt + returtype

```
public record ApiKeyValidationResult(bool Ok, string AppId, IEnumerable<string>
Scopes);

public interface IApiKeyValidator
{
    Task<ApiKeyValidationResult> ValidateAsync(string providedApiKey,
CancellationToken ct = default);
}
```

### 6.2. Enkelt oppsett (lese rett fra config)

```
using System.Security.Cryptography;
using System.Text;

public class ConfigApiKeyValidator : IApiKeyValidator
{
    private readonly IOptions<ApiKeyOptions> _options;

    public ConfigApiKeyValidator(IOptions<ApiKeyOptions> options)
    {
        _options = options;
    }

    public Task<ApiKeyValidationResult> ValidateAsync(string providedApiKey,
CancellationToken ct = default)
    {
        // Sammenlikn direkte (enkelt, men rå nøkkel i config)
        var match = _options.Value.Keys.FirstOrDefault(k => k.Key ==
providedApiKey);

        if (match is null)
            return Task.FromResult(new ApiKeyValidationResult(false, "",
Array.Empty<string>()));

        return Task.FromResult(new ApiKeyValidationResult(true, match.AppId,
match.Scopes));
    }
}
```

### 6.3. Avansert (hash-lagring)

```
public class HashedApiKeyValidator : IApiKeyValidator
{
    private readonly IYourDbContext _db; // tabell: ApiKeys { AppId, KeyHash,
IsActive, Scopes[] }
    public HashedApiKeyValidator(IYourDbContext db) => _db = db;

    public async Task<ApiKeyValidationResult> ValidateAsync(string providedApiKey,
CancellationToken ct = default)
    {
        var hash = ToSha256Hex(providedApiKey);
        var rec = await _db.ApiKeys
            .Where(x => x.KeyHash == hash && x.IsActive)
            .Select(x => new { x.AppId, Scopes = x.Scopes.Select(s => s.Value) })
            .FirstOrDefaultAsync(ct);

        if (rec is null) return new ApiKeyValidationResult(false, "",
Array.Empty<string>());
        return new ApiKeyValidationResult(true, rec.AppId, rec.Scopes);
    }

    private static string ToSha256Hex(string input)
    {
        var bytes = SHA256.HashData(Encoding.UTF8.GetBytes(input));
        return Convert.ToHexString(bytes);
    }
}
```

---

## 7. Registrering i `Program.cs`

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

// Options
builder.Services.Configure<ApiKeyOptions>
(builder.Configuration.GetSection("ApiKeyOptions"));

// Validator (velg én)
builder.Services.AddScoped<IApiKeyValidator, ConfigApiKeyValidator>();
// builder.Services.AddScoped<IApiKeyValidator, HashedApiKeyValidator>();

// Filter (kreves for TypeFilter/ServiceFilter)
builder.Services.AddScoped<ApiKeyAuthorizationFilter>();

var app = builder.Build();

app.MapControllers();
app.Run();
```

---

## 8. Bruke attributtet i Controller

```
[ApiController]
[Route("api/[controller]")]
public class UsersController : ControllerBase
{
    // Sikkerhet: krever gyldig API Key
    [ApiKey]
    [HttpGet("all")]
    public ActionResult<IEnumerable<string>> GetAll()
    {
        var appId = HttpContext.Items["appId"] as string ?? "unknown";
        return Ok(new[] { "ducdang", $"requested-by:{appId}" });
    }

    // Eksempel: enkel scope-sjekk på tvers av actions
    [ApiKey]
    [HttpPost]
    public IActionResult Create([FromBody] object payload)
    {
        var scopes = (IEnumerable<string>) (HttpContext.Items["scopes"] ??
Array.Empty<string>());
        if (!scopes.Contains("users:write"))
            return Forbid(); // 403 – mangler rett scope

        return Ok(new { status = "created" });
    }
}
```

---

## 9. Klienteksempler (cURL og Postman)

**cURL (standard headernavn):**

```
curl -H "x-api-key: sk_test_123" https://api.dittdomene.no/api/users/all
```

**Postman:**

1. Gå til **Headers**.
2. Legg til `x-api-key` med verdien fra konfigurasjonen.
3. Kjør forespørsel.

---

## 10. Feilhåndtering og tilbakemeldinger

- **401 Unauthorized** – manglende eller ugyldig nøkkel.
- **403 Forbidden** – autentisert klient, men mangler nødvendige scopes (dersom du sjekker dette i handlingen eller via et eget filter).
- Returner konsistente **ProblemDetails**/JSON-feil for bedre DX (ev. egen `ProblemDetailsFactory`).

---

## 11. Beste praksis

- **HTTPS** alltid. API Keys sendes i klartekst på transportlaget.
- **Rotasjon:** tillat flere aktive nøkler per app med utløpsdato.
- **Rate limiting** pr. `appId` (ASP.NET Rate Limiting/YARP/NGINX).
- **Logging:** logg `appId` og viktig metadata (ikke selve nøkkelen).
- **Least privilege:** gi kun nødvendige scopes.
- **Ikke i URL:** aldri send API Keys som query-param.
- **Ikke i frontend:** unngå å bake API Keys inn i klientkode for nettleser.

---

## 12. Enhetstesting (skisse)

```
[Fact]
public async Task MissingHeader_Returns401()
{
    var ctx = new DefaultHttpContext();
    var actionCtx = new ActionContext(ctx, new RouteData(), new
ActionDescriptor());
    var authCtx = new AuthorizationFilterContext(actionCtx, new
List<IFilterMetadata>());
    var validator = A.Fake<IApiKeyValidator>();
    A.CallTo(() => validator.ValidateAsync(A<string>._, A<CancellationToken>._))
        .Returns(new ApiKeyValidationResult(false, "", Array.Empty<string>()));

    var opts = Options.Create(new ApiKeyOptions { HeaderName = "x-api-key" });
    var filter = new ApiKeyAuthorizationFilter(validator, opts);

    await filter.OnAuthorizationAsync(authCtx);

    Assert.IsType<UnauthorizedObjectResult>(authCtx.Result);
}
```

---

## 13. Vanlige variasjoner

- **Flere header-navn**: støtt både `Authorization: ApiKey <key>` og `x-api-key`.
- **Globalt filter**: legg til `options.Filters.Add<ApiKeyAttribute>()` for hele APIet, og overstyr med `[AllowAnonymous]` der det trengs.
- **Policy-basert scopesjekk**: lag et eget `IAuthorizationHandler` som leser `HttpContext.Items["scopes"]` og håndhever krav.

---

## 14. Oppsummering av flyt

1. Klient sender request med `x-api-key: <key>`.
2. `[ApiKey]` aktiverer `ApiKeyAuthorizationFilter`.
3. Filter leser header, kaller `IApiKeyValidator.ValidateAsync`.
4. Ved suksess legges `appId/scopes` i `HttpContext.Items` og handlingen kjøres.
5. Ved feil returneres **401**, eller **403** ved manglende scope.

---

## 15. Tillegg: Middleware (valgfritt – ikke nødvendig i denne løsningen)

For en sentralisert variant kan du bruke middleware. Denne dokumentasjonen fokuserer **ikke** på middleware, men du kan kombinere tilnærmingene ved behov (f.eks. middleware for grov blokkering, filter for finmasket policy).

---

## 16. Hurtig sjekkliste

- [ ] `ApiKeyOptions` + `appsettings.json` på plass
- [ ] `IApiKeyValidator` registrert
- [ ] `ApiKeyAuthorizationFilter` registrert
- [ ] `[ApiKey]` lagt på kontroller/handling
- [ ] cURL/Postman testet
- [ ] Logging/rate limiting/scope-sjekk vurdert