API keys

Long-lived tokens for programmatic access with optional scopes.

Anonymous · 09/09/2026

API keys

Long-lived tokens for scripts, CI pipelines, and third-party integrations. Distinct from JWT (short-lived session) and from verification tokens (one-use).

Creating a key

```
use Derafu\PlatformBundle\Identity\Contract\Service\ApiKeyManagerInterface;

$generated = $apiKeyManager->create(
    user: $currentUser,
    name: 'CI Pipeline',
    scopes: ['read:projects', 'write:reports'],
    expiresAt: new \DateTimeImmutable('+1 year'),
);

// $generated->plaintext — the full token (dik_...). Show once, never again.
// $generated->apiKey    — the persisted entity with displayPrefix.
```

Emits no event — the UI is responsible for displaying the plaintext token immediately after creation.

Token format

```
dik_<40 hex chars>
```

- `dik_` — “Derafu Identity Key” prefix, never appears in the database.
- 40 hex chars = 160 bits of entropy from `random_bytes(20)`.
- **SHA-256 hash** is stored in the database (`tokenHash`).
- Only the first 8 chars after `dik_` are stored as `displayPrefix` for UI (“ends in …ab3f”) — the full plaintext is irrecoverable after creation.

Authentication

The `ApiKeyAuthenticator` fires on requests with an `Authorization` header matching `Bearer dik_...`.

No session is created — every API request re-validates the token.

```
Authorization: Bearer dik_3a7c9b2f...
```

Session users (browser logins) are never affected by API key logic.

Scopes

Scopes are app-defined strings stored in a JSON column on the `ApiKey` entity. An empty scopes array means “unrestricted.”

Defining allowed scopes

Implement `PermissionProviderInterface` to advertise your app’s scopes (they appear in the UI when creating keys):

```
use Derafu\PlatformBundle\Identity\Contract\Service\PermissionProviderInterface;

final class MyAppScopes implements PermissionProviderInterface
{
    public function getPermissions(): array
    {
        return [
            'read:projects'  => 'Read project data',
            'write:projects' => 'Create and update projects',
            'admin:billing'  => 'Manage billing settings',
        ];
    }
}
```

The service is auto-discovered via `PermissionProviderInterface`’s `AutoconfigureTag`.

Enforcing scopes on endpoints

Use the `#[RequiresScope]` attribute on API Platform state processors:

```
use Derafu\PlatformBundle\Identity\Attribute\RequiresScope;

#[RequiresScope('write:projects')]
final class CreateProjectProcessor implements ProcessorInterface
{
    // ...
}
```

A compiler pass (`ProcessorScopePass`) builds a static class → scopes map at compile time. The

`ApiKeyAuthenticator` rejects requests where the key lacks the required scope with `403 Forbidden`.

Session users bypass scope checks entirely — scopes restrict API tokens, not interactive users.

Rate limiting

Per-key fixed-window rate limiting. Disabled by default; enable via config:

```
derafu_platform:
    identity:
        rate_limit:
            enabled: true
            default_limit: 1000
            default_window: 3600   # 1 hour
            scope_limits:
                'write:projects':
                    limit: 100
                    window: 3600
```

When enabled, every API-key-authenticated response includes:

```
X-RateLimit-Limit: 1000
X-RateLimit-Remaining: 993
X-RateLimit-Reset: 1718000000
```

When the quota is exceeded, the response is `429 Too Many Requests` with a JSON error body. Session users are never rate-limited.

The default implementation uses PSR-6 cache (works with any adapter, including Redis or Memcached). For high-concurrency environments, a `RedisApiKeyRateLimiter` using atomic `INCR` is also provided. Swap it by binding `ApiKeyRateLimiterInterface` to the Redis implementation:

```
services:
    Derafu\PlatformBundle\Identity\Contract\Service\ApiKeyRateLimiterInterface:
        alias: Derafu\PlatformBundle\Identity\Service\RedisApiKeyRateLimiter
```

Listing and revoking keys

```
// All keys for a user
$keys = $apiKeyRepository->findByUser($user);

// Revoke (hard delete)
$apiKeyManager->revoke($apiKey);
```

The bundle’s settings UI (IdentityUi module) provides pages for listing, creating, and revoking API keys.

Last updated on 09/09/2026