Docs

Certificate Project

Derafu Certificate

Anonymous · 24/08/2026 · #php

Table of Contents

Certificate Project

Derafu Certificate

Anonymous · 24/08/2026 · #php

Derafu Certificate

Last updated on 24/08/2026

Introduction

Library for digital certificates

Anonymous · 24/08/2026

Library for digital certificates

A comprehensive PHP library for working with digital certificates, providing tools for loading, validating and generating certificates.

Features

- **Certificate Loading**: Load certificates from files, data, arrays or keys.
- **Certificate Validation**: Validate certificates against specific requirements.
- **Certificate Generation**: Create self-signed certificates for testing.
- **Certificate Information**: Extract key information from certificates (ID, name, email, etc.).
- **Key Management**: Work with public and private keys, modulus, and exponent.

Installation

```
composer require derafu/certificate
```

Basic Usage

Loading a Certificate

```
use Derafu\Certificate\Service\CertificateLoader;

// Create a loader.
$loader = new CertificateLoader();

// Load from a file.
$certificate = $loader->loadFromFile('/path/to/certificate.p12', 'password');

// Load from data.
$certificate = $loader->loadFromData($certificateData, 'password');
```

```
// Load from array.
$certificate = $loader->loadFromArray([
    'cert' => $publicKey,
    'pkey' => $privateKey
]);

// Load from keys.
$certificate = $loader->loadFromKeys($publicKey, $privateKey);
```

Accessing Certificate Information

```
// Get basic certificate information.
$id = $certificate->getId(); // e.g., "12345678-9"
$name = $certificate->getName(); // e.g., "John Doe"
$email = $certificate->getEmail(); // e.g., "john.doe@example.com"

// Check certificate validity.
$isActive = $certificate->isActive(); // true if certificate is valid.
$expirationDays = $certificate->getExpirationDays(); // days until expiration.

// Get validity dates.
$validFrom = $certificate->getFrom(); // e.g., "2025-01-01T00:00:00"
$validTo = $certificate->getTo(); // e.g., "2026-01-01T00:00:00"

// Get certificate issuer.
$issuer = $certificate->getIssuer(); // e.g., "Example CA"

// Get key components.
$modulus = $certificate->getModulus();
$exponent = $certificate->getExponent();

// Get raw keys.
$publicKey = $certificate->getPublicKey(); // with headers.
$privateKey = $certificate->getPrivateKey(); // with headers.
$cleanPublicKey = $certificate->getPublicKey(true); // without headers.
$cleanPrivateKey = $certificate->getPrivateKey(true); // without headers.
```

Validating a Certificate

```
use Derafu\Certificate\Exception\CertificateException;
use Derafu\Certificate\Service\CertificateValidator;

$validator = new CertificateValidator();

try {
    $validator->validate($certificate);
    echo "Certificate is valid";
} catch (CertificateException $e) {
    echo "Certificate validation failed: " . $e->getMessage();
```

```
}
```

Creating a Fake Certificate for Testing

```
use Derafu\Certificate\Service\CertificateLoader;
use Derafu\Certificate\Service\CertificateFaker;

$loader = new CertificateLoader();
$faker = new CertificateFaker($loader);

// Create a fake certificate with default values.
$certificate = $faker->createFake();

// Create a fake certificate with custom values.
$certificate = $faker->createFake(
    id: '12345678-9',
    name: 'John Doe',
    email: 'john.doe@example.com',
    password: 'secure_password'
);

// Export to PKCS#12 format.
$pkcs12Data = $certificate->getPkcs12('password');
file_put_contents('certificate.p12', $pkcs12Data);
```

Using the Service

The `CertificateService` provides a unified interface to all library functionality:

```
use Derafu\Certificate\Service\CertificateLoader;
use Derafu\Certificate\Service\CertificateFaker;
use Derafu\Certificate\Service\CertificateValidator;
use Derafu\Certificate\Service\CertificateService;

// Create the service with its dependencies.
$loader = new CertificateLoader();
$faker = new CertificateFaker($loader);
$validator = new CertificateValidator();
$service = new CertificateService($faker, $loader, $validator);

// Use the service for certificate operations.
$certificate = $service->loadFromFile('/path/to/certificate.p12', 'password');
$service->validate($certificate);

// Create a fake certificate for testing.
$fakeCertificate = $service->createFake(
    '12345678-9',
    'John Doe',
```

```
    'john.doe@example.com'
);
```

Advanced Usage

Creating a Self-Signed Certificate

For more control over certificate generation:

```
use Derafu\Certificate\SelfSignedCertificate;
use Derafu\Certificate\Service\CertificateLoader;

// Create a self-signed certificate with custom values.
$selfSigned = new SelfSignedCertificate();
$selfSigned->setSubject(
    C: 'US',
    ST: 'California',
    L: 'San Francisco',
    O: 'Example Organization',
    OU: 'IT Department',
    CN: 'John Doe',
    emailAddress: 'john.doe@example.com',
    serialNumber: '12345678-9'
);

$selfSigned->setIssuer(
    CN: 'Example CA'
);

$selfSigned->setValidity(365); // Valid for 1 year.
$selfSigned->setPassword('secure_password');

// Get the certificate array.
$certArray = $selfSigned->toArray();

// Load as a Certificate object.
$loader = new CertificateLoader();
$certificate = $loader->loadFromArray($certArray);
```

Working with Asymmetric Keys

```
use Derafu\Certificate\AsymmetricKeyHelper;

// Normalize a public key (add headers if missing).
$normalizedPublicKey = AsymmetricKeyHelper::normalizePublicKey($rawPublicKey);

// Normalize a private key (add headers if missing).
```

```
$normalizedPrivateKey = AsymmetricKeyHelper::normalizePrivateKey($rawPrivateKey);

// Generate a public key from modulus and exponent.
// Requirements: composer require phpseclib/phpseclib
$publicKey = AsymmetricKeyHelper::generatePublicKeyFromModulusExponent(
    $modulus,
    $exponent
);
```

Last updated on 24/08/2026

Certificate Class

Certificate Class

Anonymous · 24/08/2026

Certificate Class

The `Certificate` class is the core component of the Derafu Certificate library, representing a digital certificate with its public and private keys and providing methods to access certificate details and properties.

Overview

The `Certificate` class implements the `CertificateInterface` and provides functionality to:

- Access public and private keys.
- Extract certificate metadata (ID, name, email, issuer, etc.).
- Check certificate validity.
- Get certificate validity dates.
- Retrieve cryptographic components (modulus, exponent).
- Generate PKCS#12 data.

Usage

Creating a Certificate

To create a `Certificate` instance, you need a public key (certificate) and a private key:

```
use Derafu\Certificate\Certificate;

$certificate = new Certificate($publicKey, $privateKey);
```

The constructor automatically normalizes the keys using `AsymmetricKeyHelper`.

Getting Keys

```
// Get both keys as an array.
$keys = $certificate->getKeys();
// Result: ['cert' => '...public key...', 'pkey' => '...private key...']
```

```
// Get both keys without headers/footers.
$cleanKeys = $certificate->getKeys(clean: true);

// Get just the public key.
$publicKey = $certificate->getPublicKey();
// Alternatively.
$publicKey = $certificate->getCertificate();

// Get just the private key.
$privateKey = $certificate->getPrivateKey();

// Get clean keys (without headers/footers).
$cleanPublicKey = $certificate->getPublicKey(clean: true);
$cleanPrivateKey = $certificate->getPrivateKey(clean: true);
```

Certificate Metadata

```
// Get certificate ID (usually a document/tax number).
$id = $certificate->getId(); // e.g., "12345678-9"
$id = $certificate->getId(forceUpper:false); // with original case, e.g., "12345678-k"

// Get certificate owner's name.
$name = $certificate->getName();

// Get certificate owner's email.
$email = $certificate->getEmail();

// Get certificate issuer.
$issuer = $certificate->getIssuer();
```

Certificate Validity

```
// Check if certificate is valid (not expired).
$isActive = $certificate->isActive();

// Check if certificate is valid at a specific date.
$isActiveOn = $certificate->isActive('2025-06-01');

// Get certificate validity period.
$validFrom = $certificate->getFrom(); // e.g., "2025-01-01T00:00:00"
$validTo = $certificate->getTo(); // e.g., "2026-01-01T00:00:00"

// Get total validity period in days.
$totalDays = $certificate->getTotalDays();

// Get days remaining until expiration.
$daysRemaining = $certificate->getExpirationDays();

// Get days remaining from a specific date.
```

```
$daysRemainingFrom = $certificate->getExpirationDays('2025-06-01');
```

Cryptographic Components

```
// Get certificate raw data.
$data = $certificate->getData();

// Get private key details.
$privateKeyDetails = $certificate->getPrivateKeyDetails();

// Get modulus and exponent.
$modulus = $certificate->getModulus();
$exponent = $certificate->getExponent();

// With custom word wrapping.
$modulus = $certificate->getModulus(wordwrap: 75);
$exponent = $certificate->getExponent(wordwrap: 75);
```

Exporting a Certificate

```
// Export certificate as PKCS#12 data.
$pkcs12 = $certificate->getPkcs12('password');

// Save to a file.
file_put_contents('certificate.p12', $pkcs12);
```

Error Handling

The `Certificate` class throws `CertificateException` when it encounters problems such as:

- Unable to extract the certificate ID.
- Unable to find the certificate name.
- Unable to find the certificate email.
- Unable to access the modulus or exponent.

Example:

```
use Derafu\Certificate\Exception\CertificateException;

try {
    $id = $certificate->getId();
    $name = $certificate->getName();
    $email = $certificate->getEmail();
} catch (CertificateException $e) {
    echo "Certificate error: " . $e->getMessage();
}
```

Implementation Details

ID Extraction

The `getId()` method attempts to extract the certificate ID through multiple methods:

1. First, it checks the `serialNumber` field in the subject.
2. If not found, it looks for the ID in the Subject Alternative Name (SAN) extension.
3. If still not found, it throws a `CertificateException`.

Certificate Validation

The `isActive()` method checks if the certificate is valid at the current date (or a specified date) by comparing it against the certificate's validity period.

Cryptographic Components

The `getModulus()` and `getExponent()` methods extract these values from the private key details and return them as base64-encoded strings, which can be used for various cryptographic operations.

Self-Signed Certificate

SelfSignedCertificate Class

Anonymous · 24/08/2026

SelfSignedCertificate Class

The `SelfSignedCertificate` class provides functionality to generate self-signed certificates for testing and development purposes. It allows customization of certificate properties such as subject, issuer, validity period, and password protection.

Overview

This class creates a digital certificate that includes:

1. A Certificate Authority (CA) certificate.
2. A user certificate signed by the CA.
3. Both packaged together in PKCS#12 format.

The generated certificate is suitable for testing electronic signature functionality without requiring a certificate from a commercial Certificate Authority.

Basic Usage

```
use Derafu\Certificate\SelfSignedCertificate;
use Derafu\Certificate\Service\CertificateLoader;

// Create with default settings.
$selfSigned = new SelfSignedCertificate();
$certificateArray = $selfSigned->toArray();

// Load as a Certificate object.
$loader = new CertificateLoader();
$certificate = $loader->loadFromArray($certificateArray);
```

Configuration Methods

Setting the Subject

The subject represents the owner of the certificate:

```
$selfSigned->setSubject(
    C: 'US',                              // Country code (2 letters).
    ST: 'California',                     // State/Province.
    L: 'San Francisco',                   // Locality/City.
    O: 'Example Corporation',             // Organization.
    OU: 'IT Department',                  // Organizational Unit.
    CN: 'John Doe',                       // Common Name (full name).
    emailAddress: 'john.doe@example.com', // Email address.
    serialNumber: '12345678-9',           // ID/Serial number (tax ID, etc.).
    title: 'System Administrator'         // Title.
);
```

Only `CN`, `emailAddress`, and `serialNumber` are strictly required. The method will throw a `CertificateException` if any of these are empty.

Setting the Issuer

The issuer represents the Certificate Authority that issues the certificate:

```
$selfSigned->setIssuer(
    C: 'US',                            // Country code
    ST: 'Washington',                   // State/Province
    L: 'Seattle',                       // Locality/City
    O: 'Example CA',                    // Organization
    OU: 'Certificate Authority',        // Organizational Unit
    CN: 'Example Root CA',              // CA Name
    emailAddress: 'ca@example.com',     // CA Email
    serialNumber: '87654321-0'          // CA ID
);
```

Setting the Validity Period

```
// Set validity to 730 days (2 years).
$selfSigned->setValidity(730);
```

The default validity period is 365 days (1 year).

Setting the Password

```
// Set password for the private key.
$selfSigned->setPassword('secure_password');
```

The default password is `'i_love_derafu'`.

Generating the Certificate

Getting the Certificate Array

```
// Get the certificate as an array with 'cert' and 'pkey' elements.
$certificateArray = $selfSigned->toArray();

// The array contains:
// [
//     'cert' => '...public key certificate...',
//     'pkey' => '...private key...'
// ]
```

Getting the Raw PKCS#12 Data

```
// Get the toPkcs12() method result (PKCS#12 binary data).
$pkcs12Data = $selfSigned->toPkcs12();

// Save to a file.
file_put_contents('certificate.p12', $pkcs12Data);
```

Certificate Structure

The generated certificate consists of:

1. An issuer (CA) certificate that is self-signed.
2. A subject certificate that is signed by the issuer.
3. Both certificates' private keys.

The PKCS#12 container includes both certificates and is password-protected using the specified password.

Default Values

If you create a `SelfSignedCertificate` without any customization, it will use the following default values:

Default Subject

- Country: `CL` (Chile).
- State: `Colchagua`.
- Locality: `Santa Cruz`.
- Organization: `Intergalactic Robots Organization`.

- Organizational Unit: `Technology`.
- Common Name: `Daniel Bot`.
- Email: `daniel.bot@example.com`.
- Serial Number: `11222333-9`.
- Title: `Bot`.

Default Issuer

- Country: `CL` (Chile).
- State: `Colchagua`.
- Locality: `Santa Cruz`.
- Organization: `Derafu`.
- Organizational Unit: `Technology`.
- Common Name: `Derafu Test Certificate Authority`.
- Email: `fakes-certificates@derafu.org`.
- Serial Number: `76192083-9`.

Other Defaults

- Validity: 365 days.
- Password: `i_love_derafu`.

Implementation Details

The `SelfSignedCertificate` class uses the PHP OpenSSL extension to:

1. Generate a key pair for the issuer.
2. Create a Certificate Signing Request (CSR) for the issuer.
3. Self-sign the issuer's CSR to create the issuer certificate.
4. Generate a key pair for the subject.
5. Create a CSR for the subject.
6. Sign the subject's CSR with the issuer's certificate.
7. Package everything into a PKCS#12 container.

The process mimics a real CA-issued certificate but is entirely self-contained and suitable for testing purposes.

Last updated on 24/08/2026

Asymmetric Key Helper

AsymmetricKeyHelper Class

Anonymous · 24/08/2026

AsymmetricKeyHelper Class

The `AsymmetricKeyHelper` class provides utilities for working with RSA certificates and keys. It offers methods to normalize public and private keys and to generate public keys from modulus and exponent values.

Overview

When working with digital certificates and electronic signatures, you often need to handle RSA key components in different formats. The `AsymmetricKeyHelper` class solves common problems related to key formatting and generation:

1. Adding standard PEM headers and footers to raw key data.
2. Converting between key components (modulus, exponent) and full key format.

Key Normalization

Normalizing a Public Key

```
use Derafu\Certificate\AsymmetricKeyHelper;

// Raw public key without headers.
$rawPublicKey = "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvTLIKu... (more base64
data)";

// Normalize the public key (add headers and footers).
$normalizedPublicKey = AsymmetricKeyHelper::normalizePublicKey($rawPublicKey);

// Result:
// -----BEGIN CERTIFICATE-----
// MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvTLIKu...
// (more base64 data, wrapped, by default, at 64 characters per line)
// -----END CERTIFICATE-----
```

Normalizing a Private Key

```
// Raw private key without headers.
$rawPrivateKey = "MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkA... (more base64 data)";

// Normalize the private key (add headers and footers).
$normalizedPrivateKey = AsymmetricKeyHelper::normalizePrivateKey($rawPrivateKey);

// Result:
// -----BEGIN PRIVATE KEY-----
// MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkA...
// (more base64 data, wrapped, by default, at 64 characters per line)
// -----END PRIVATE KEY-----
```

Custom Line Length

Both normalization methods support a custom line length for wrapping the key data:

```
// Wrap lines at 75 characters instead of the default 64.
$normalizedPublicKey = AsymmetricKeyHelper::normalizePublicKey($rawPublicKey, 75);
$normalizedPrivateKey = AsymmetricKeyHelper::normalizePrivateKey($rawPrivateKey, 75);
```

Key Generation

Generating a Public Key from Modulus and Exponent

You can generate a complete public key if you have the modulus and exponent components:

```
// Base64-encoded modulus and exponent.
$modulus = "wKhpaf5AYomI+0/tLxJtvjHVCveRYYZ9j0yDlL...";
$exponent = "AQAB";

// Generate public key.
$publicKey = AsymmetricKeyHelper::generatePublicKeyFromModulusExponent(
    $modulus,
    $exponent
);

// Result is a complete public key in PKCS1 format:
// -----BEGIN RSA PUBLIC KEY-----
// MIIBCgKCAQEAwKhpaf5AYomI+0/tLxJtvjHVCveRYYZ9j0yDlL...
// -----END RSA PUBLIC KEY-----
```

This method requires the `phpseclib3/phpseclib` library. If the library is not installed, it will throw a `LogicException` with instructions to install it.

Common Use Cases

Preparing Keys for Use with OpenSSL Functions

Many OpenSSL functions require properly formatted keys with headers and footers:

```
// Normalize keys before using with OpenSSL.
$normalizedPublicKey = AsymmetricKeyHelper::normalizePublicKey($rawPublicKey);
$normalizedPrivateKey = AsymmetricKeyHelper::normalizePrivateKey($rawPrivateKey);

// Now use with OpenSSL functions.
$signature = openssl_sign($data, $signature, $normalizedPrivateKey,
OPENSSL_ALGO_SHA256);
$result = openssl_verify($data, $signature, $normalizedPublicKey,
OPENSSL_ALGO_SHA256);
```

Reconstructing Public Keys

In some systems, especially with hardware security modules (HSMs) or smart cards, you might only have access to the modulus and exponent components:

```
// Reconstruct the public key from components.
$publicKey = AsymmetricKeyHelper::generatePublicKeyFromModulusExponent(
    $modulus,
    $exponent
);

// Use for verification.
$result = openssl_verify($data, $signature, $publicKey, OPENSSL_ALGO_SHA256);
```

Implementation Details

Public Key Normalization

The `normalizePublicKey()` method:

1. Checks if the key already contains the “BEGIN CERTIFICATE” header.
2. If not, adds the appropriate BEGIN/END headers.
3. Wraps the content to the specified line length.
4. Returns the normalized key.

Private Key Normalization

The `normalizePrivateKey()` method:

1. Checks if the key already contains the “BEGIN PRIVATE KEY” header.
2. If not, adds the appropriate BEGIN/END headers.
3. Wraps the content to the specified line length.
4. Returns the normalized key.

Public Key Generation

The `generatePublicKeyFromModulusExponent()` method:

1. Decodes the base64-encoded modulus and exponent.
2. Creates BigInteger instances from the binary data.
3. Loads these values into a RSA key object using phpseclib.
4. Exports the key in PKCS1 format.
5. Returns the complete public key string.

Last updated on 24/08/2026

Certificate Service

CertificateService Class

Anonymous · 24/08/2026

CertificateService Class

The `CertificateService` is the main entry point for working with digital certificates in the Derafu Certificate library. It provides a unified interface to all the key functionality like loading, validating, and creating certificates.

Overview

The `CertificateService` follows the service pattern, acting as a facade for:

- `CertificateFaker`: For creating test certificates.
- `CertificateLoader`: For loading certificates from various sources.
- `CertificateValidator`: For validating certificates.

By using the service, you can access all the library's functionality through a single interface.

Basic Usage

Setting Up the Service

```
use Derafu\Certificate\Service\CertificateLoader;
use Derafu\Certificate\Service\CertificateFaker;
use Derafu\Certificate\Service\CertificateValidator;
use Derafu\Certificate\Service\CertificateService;

// Create dependencies.
$loader = new CertificateLoader();
$faker = new CertificateFaker($loader);
$validator = new CertificateValidator();

// Create the service.
$service = new CertificateService($faker, $loader, $validator);
```

Loading Certificates

The service provides methods to load certificates from various sources:

```
// Load from a PKCS#12 file.
$certificate = $service->loadFromFile('/path/to/certificate.p12', 'password');

// Load from PKCS#12 data.
$certificate = $service->loadFromData($pkcs12Data, 'password');

// Load from a certificate array.
$certificate = $service->loadFromArray([
    'cert' => $publicKey,
    'pkey' => $privateKey
]);

// Load directly from key strings.
$certificate = $service->loadFromKeys($publicKey, $privateKey);
```

Validating Certificates

```
use Derafu\Certificate\Exception\CertificateException;

try {
    $service->validate($certificate);
    echo "Certificate is valid!";
} catch (CertificateException $e) {
    echo "Validation failed: " . $e->getMessage();
}
```

Creating Test Certificates

```
// Create with default values.
$certificate = $service->createFake();

// Create with custom values.
$certificate = $service->createFake(
    id: '12345678-9',
    name: 'John Doe',
    email: 'john.doe@example.com'
);

// Use the certificate.
$id = $certificate->getId();
$name = $certificate->getName();
$isValid = $certificate->isActive();
```

Complete Example

```
// Initialize the service.
```

```
$loader = new CertificateLoader();
$faker = new CertificateFaker($loader);
$validator = new CertificateValidator();
$service = new CertificateService($faker, $loader, $validator);

// Create a test certificate.
$certificate = $service->createFake(
    id: '12345678-9',
    name: 'John Doe',
    email: 'john.doe@example.com'
);

// Extract certificate data.
echo "Certificate ID: " . $certificate->getId() . "\n";
echo "Certificate Name: " . $certificate->getName() . "\n";
echo "Valid until: " . $certificate->getTo() . "\n";
echo "Days remaining: " . $certificate->getExpirationDays() . "\n";

// Validate the certificate.
try {
    $service->validate($certificate);
    echo "Certificate is valid\n";
} catch (CertificateException $e) {
    echo "Invalid certificate: " . $e->getMessage() . "\n";
}

// Export the certificate.
$pkcs12Data = $certificate->getPkcs12('password');
file_put_contents('certificate.p12', $pkcs12Data);

// Later, load the certificate back.
$loadedCertificate = $service->loadFromFile('certificate.p12', 'password');
```

Customizing the Service

Custom Validator

You can create a custom validator by implementing the `CertificateValidatorInterface`:

```
use Derafu\Certificate\Contract\CertificateInterface;
use Derafu\Certificate\Contract\CertificateValidatorInterface;
use Derafu\Certificate\Exception\CertificateException;

class MyCustomValidator implements CertificateValidatorInterface
{
    public function validate(CertificateInterface $certificate): void
    {
        // Implement your validation logic.
```

```
        if (!$certificate->isActive()) {
            throw new CertificateException(
                'Certificate is expired.'
            );
        }

        // Add specific validation for your use case.
        if ($certificate->getExpirationDays() < 30) {
            throw new CertificateException(
                'Certificate will expire in less than 30 days.'
            );
        }
    }
}

// Use your custom validator.
$customValidator = new MyCustomValidator();
$service = new CertificateService($faker, $loader, $customValidator);
```

Service Components

CertificateLoader

The `CertificateLoader` component handles loading certificates from various sources:

- PKCS#12 files.
- PKCS#12 binary data.
- Array of keys.
- Individual public and private keys.

CertificateFaker

The `CertificateFaker` component provides a way to create self-signed certificates for testing purposes:

- Create certificates with custom subject information.
- Create certificates with custom issuer information.
- Create certificates with custom validity periods.

Under the hood, it uses the `SelfSignedCertificate` class.

CertificateValidator

The `CertificateValidator` component validates certificates according to specific requirements:

- Checks if the certificate ID (RUN) is properly formatted.

- Ensures that “K” in the ID is uppercase.
- Verifies that the certificate is not expired.

The default validator is designed for Chilean certificates used with the SII (Servicio de Impuestos Internos), but you can implement your own validator for different requirements.

Last updated on 24/08/2026