Signature Service

SignatureService Class

Anonymous · 09/09/2026

SignatureService Class

The `SignatureService` class is the main entry point for working with digital signatures in the Derafu Signature library. It provides a unified interface for generating and validating signatures for both general data and XML documents.

Overview

The `SignatureService` implements the `SignatureServiceInterface`, which combines the functionality of:

- `SignatureGeneratorInterface`: For creating digital signatures.
- `SignatureValidatorInterface`: For validating digital signatures.

This service acts as a facade over the signature generation and validation components, making it easy to use the library's full functionality through a single interface.

Basic Usage

Setting Up the Service

```
use Derafu\Signature\Service\SignatureGenerator;
use Derafu\Signature\Service\SignatureService;
use Derafu\Signature\Service\SignatureValidator;
use Derafu\Xml\Service\XmlDecoder;
use Derafu\Xml\Service\XmlEncoder;
use Derafu\Xml\Service\XmlService;
use Derafu\Xml\Service\XmlValidator;

// Initialize required XML services.
$xmlEncoder = new XmlEncoder();
$xmlDecoder = new XmlDecoder();
$xmlValidator = new XmlValidator();
$xmlService = new XmlService($xmlEncoder, $xmlDecoder, $xmlValidator);

// Create generator and validator.
```

```
$generator = new SignatureGenerator($xmlService);
$validator = new SignatureValidator($generator, $xmlService);

// Create the signature service.
$signatureService = new SignatureService($generator, $validator);
```

Signing Data

```
// Sign simple data with a private key.
$data = 'Data to be signed';
$signature = $signatureService->sign($data, $privateKey);

// Optional: Specify a different signature algorithm.
$signature = $signatureService->sign($data, $privateKey, OPENSSL_ALGO_SHA256);
```

Validating a Signature

```
// Validate a signature using a public key.
$isValid = $signatureService->validate($data, $signature, $publicKey);

// Optional: Specify the same signature algorithm used for signing.
$isValid = $signatureService->validate($data, $signature, $publicKey,
OPENSSL_ALGO_SHA256);

if ($isValid) {
    echo "Signature is valid!";
} else {
    echo "Signature is invalid!";
}
```

Signing XML

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

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

// Sign an XML string.
$xmlString = '<root><element>data</element></root>';
$signedXml = $signatureService->signXml($xmlString, $certificate);

// Sign an XmlDocument object.
$xmlDoc = new XmlDocument();
$xmlDoc->loadXml($xmlString);
$signedXml = $signatureService->signXml($xmlDoc, $certificate);
```

```
// Sign a specific element in the XML (identified by ID).
$xmlWithIds = '<root><element ID="myElement">data</element></root>';
$signedXml = $signatureService->signXml($xmlWithIds, $certificate, 'myElement');
```

Validating XML Signatures

```
use Derafu\Signature\Exception\SignatureException;

try {
    // Validate a signed XML document.
    $signatureService->validateXml($signedXml);
    echo "XML signature is valid!";
} catch (SignatureException $e) {
    echo "XML signature validation failed: " . $e->getMessage();
}
```

Advanced Usage

Calculating Digest Values

```
// Calculate the digest value for an XML document.
$digestValue = $signatureService->generateXmlDigestValue($xmlDoc);

// Calculate the digest value for a specific element.
$digestValue = $signatureService->generateXmlDigestValue($xmlDoc, 'elementId');
```

Working with Signature Nodes

```
// Extract signature node from a signed XML.
$signatureXml = $signedXml; // The signature node XML.
$signatureNode = $signatureService->createSignatureNode($signatureXml);

// Validate just the digest value (content integrity).
try {
    $signatureService->validateXmlDigestValue($xmlDoc, $signatureNode);
    echo "XML content integrity verified!";
} catch (SignatureException $e) {
    echo "Digest validation failed: " . $e->getMessage();
}

// Validate just the signature value (signer authenticity).
try {
    $signatureService->validateXmlSignatureValue($signatureNode);
    echo "Signature authenticity verified!";
} catch (SignatureException $e) {
    echo "Signature validation failed: " . $e->getMessage();
```

```
}
```

API Reference

Data Signing and Validation

```
// Generate a digital signature for data.
public function sign(
    string $data,
    string $privateKey,
    string|int $signatureAlgorithm = OPENSSL_ALGO_SHA1
): string;

// Validate a digital signature for data.
public function validate(
    string $data,
    string $signature,
    string $publicKey,
    string|int $signatureAlgorithm = OPENSSL_ALGO_SHA1
): bool;
```

XML Signing and Validation

```
// Sign an XML document.
public function signXml(
    XmlDocumentInterface|string $xml,
    CertificateInterface $certificate,
    ?string $reference = null
): string;

// Validate an XML signature.
public function validateXml(XmlDocumentInterface|string $xml): void;

// Calculate the digest value for an XML document or element.
public function generateXmlDigestValue(
    XmlDocumentInterface $doc,
    ?string $reference = null
): string;
```

Signature Node Operations

```
// Create a signature node from XML.
public function createSignatureNode(string $xml): SignatureInterface;

// Validate the digest value of a signature.
public function validateXmlDigestValue(
```

```
    XmlDocumentInterface|string $xml,
    SignatureInterface $signatureNode
): void;

// Validate the signature value of a signature.
public function validateXmlSignatureValue(
    SignatureInterface $signatureNode
): void;
```

Implementation Details

Dependency Injection

The `SignatureService` uses dependency injection to receive its required components:

```
public function __construct(
    private readonly SignatureGeneratorInterface $generator,
    private readonly SignatureValidatorInterface $validator
)
```

This design allows for flexibility and testability, as the generator and validator components can be replaced with custom implementations if needed.

Delegation Pattern

The service uses the delegation pattern to forward method calls to the appropriate components:

- Signing methods are delegated to the `SignatureGeneratorInterface` implementation.
- Validation methods are delegated to the `SignatureValidatorInterface` implementation.

This separation of concerns keeps the codebase clean and maintainable.

Error Handling

Validation methods throw `SignatureException` when validation fails, providing detailed error messages about what went wrong:

- Invalid digest values (content has been modified).
- Invalid signature values (signature was not created with the expected private key).
- Missing signature nodes.
- Malformed XML.

These exceptions should be caught and handled appropriately in your application code.

Last updated on 09/09/2026