Docs

Signature Project

Derafu Signature

Anonymous · 24/08/2026 · #php

Table of Contents

Signature Project

Derafu Signature

Anonymous · 24/08/2026 · #php

Derafu Signature

Last updated on 24/08/2026

Introduction

Library for digital signatures

Anonymous · 24/08/2026

Library for digital signatures

A comprehensive PHP library for creating and validating digital signatures, with special focus on XML digital signatures (XML-DSIG).

Features

- **Digital Signatures**: Sign and validate any data with RSA key pairs.
- **XML Signatures**: Full support for XML Digital Signatures (XML-DSIG).
- **Signature Verification**: Validate signatures against public keys.
- **Reference Support**: Sign specific sections of XML documents using ID references.
- **Integration**: Works seamlessly with Derafu Certificate and Derafu XML libraries.

Installation

```
composer require derafu/signature
```

Basic Usage

Signing Data

```
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;

// Set up the signature service.
$xmlEncoder = new XmlEncoder();
$xmlDecoder = new XmlDecoder();
```

```
$xmlValidator = new XmlValidator();
$xmlService = new XmlService($xmlEncoder, $xmlDecoder, $xmlValidator);

$signatureGenerator = new SignatureGenerator($xmlService);
$signatureValidator = new SignatureValidator($signatureGenerator, $xmlService);
$signatureService = new SignatureService($signatureGenerator, $signatureValidator);

// Sign simple data.
$privateKey = '...';
$data = 'Hello, world!';
$signature = $signatureService->sign($data, $privateKey);

// Validate the signature.
$isValid = $signatureService->validate($data, $signature, $publicKey);
```

Signing XML Documents

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

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

// Load XML to sign.
$xml = file_get_contents('document.xml');

// Sign the entire XML document.
$signedXml = $signatureService->signXml($xml, $certificate);

// Sign a specific element in the XML document (identified by ID).
$signedXml = $signatureService->signXml($xml, $certificate, 'elementId');

// Save the signed XML.
file_put_contents('signed_document.xml', $signedXml);
```

Validating XML Signatures

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

// Load signed XML.
$signedXml = file_get_contents('signed_document.xml');

try {
    // Validate the XML signature.
    $signatureService->validateXml($signedXml);
    echo "Signature is valid!";
```

```
} catch (SignatureException $e) {
    echo "Signature validation failed: " . $e->getMessage();
}
```

Advanced Usage

Detailed XML Signature Validation

For more detailed control over the validation process:

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

// Validate the digest value (integrity of the signed content).
$signatureService->validateXmlDigestValue($xmlDocument, $signatureNode);

// Validate the signature value (authenticity of the signer).
$signatureService->validateXmlSignatureValue($signatureNode);
```

Calculating Digest Values

```
use Derafu\Xml\XmlDocument;

// Load XML document.
$xmlDoc = new XmlDocument();
$xmlDoc->loadXml($xml);

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

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

XML-DSIG Implementation Details

The library implements XML Digital Signatures according to the W3C XML Signature Syntax and Processing specification:

1. The `Signature` element is created with the following components:
    - `SignedInfo`: Contains information about what was signed.
    - `SignatureValue`: Contains the actual signature value.
    - `KeyInfo`: Contains information about the key used to validate the signature.

2. Canonicalization is performed using the C14N algorithm (http://www.w3.org/TR/2001/REC-xml-c14n-20010315).

3. Signatures are created using RSA-SHA1 (http://www.w3.org/2000/09/xmldsig#rsa-sha1).

4. Digests are created using SHA1 (http://www.w3.org/2000/09/xmldsig#sha1).

XML-DSIG Structure

When signing an XML document, the resulting signature will have the following structure:

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <CanonicalizationMethod
Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <Reference URI="#elementId">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>...</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>...</SignatureValue>
  <KeyInfo>
    <KeyValue>
      <RSAKeyValue>
        <Modulus>...</Modulus>
        <Exponent>...</Exponent>
      </RSAKeyValue>
    </KeyValue>
    <X509Data>
      <X509Certificate>...</X509Certificate>
    </X509Data>
  </KeyInfo>
</Signature>
```

Integration with Other Derafu Libraries

This library is designed to work seamlessly with other Derafu libraries:

- **Derafu Certificate**: For handling digital certificates and key pairs.
- **Derafu XML**: For handling XML documents and operations.

Last updated on 24/08/2026

Signature Class

Signature Class

Anonymous · 24/08/2026

Signature Class

The `Signature` class is the core component of the Derafu Signature library, representing an XML digital signature node according to the XML-DSIG standard.

Overview

The `Signature` class implements the `SignatureInterface` and represents the XML `<Signature>` element that contains all the information related to a digital signature in an XML document:

- The digest value of the signed content.
- The signature value.
- The public key information for signature verification.

This class is used both when creating new signatures and when validating existing ones.

Usage

Creating a New Signature Node

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

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

// Create a signature node.
$signatureNode = new Signature();

// Configure with digest value, certificate, and optional reference.
$signatureNode->configureSignatureData(
    digestValue: 'bWFpbkRpZ2VzdFZhbHVlQmFzZTY0',
    certificate: $certificate,
```

```
    reference: 'documentId'  // Optional, specify to sign a specific element.
);
```

Working with an Existing Signature Node

```
// Typically you'd use the SignatureValidator to create a signature node from XML.
$signatureNode = $signatureService->createSignatureNode($signatureXml);

// Access signature properties.
$reference = $signatureNode->getReference();
$digestValue = $signatureNode->getDigestValue();
$signatureValue = $signatureNode->getSignatureValue();
$x509Certificate = $signatureNode->getX509Certificate();
```

API Reference

Setting and Getting Data

```
// Set raw data structure.
$signatureNode->setData($dataArray);

// Get the current data structure.
$dataArray = $signatureNode->getData();
```

Configuring the Signature

```
// Configure all signature components at once.
$signatureNode->configureSignatureData(
    digestValue: 'base64DigestValue',
    certificate: $certificate,
    reference: 'elementId'
);
```

Working with XML

```
// Set the XML representation of the signature.
$signatureNode->setXml($xmlDocument);

// Get the XML representation of the signature.
$xmlDocument = $signatureNode->getXml();
```

Setting Signature Value

```
// Set the calculated signature value (after signing the SignedInfo element).
```

```
$signatureNode->setSignatureValue('base64SignatureValue');
```

Getting Signature Components

```
// Get the reference URI (without # prefix).
$reference = $signatureNode->getReference();

// Get the digest value.
$digestValue = $signatureNode->getDigestValue();

// Get the X.509 certificate (without headers/footers).
$certificate = $signatureNode->getX509Certificate();

// Get the signature value.
$signatureValue = $signatureNode->getSignatureValue();
```

Data Structure

The `Signature` class maintains an internal data array that represents the XML structure of the signature. This structure follows the XML-DSIG standard:

```
[
    'Signature' => [
        '@attributes' => [
            'xmlns' => 'http://www.w3.org/2000/09/xmldsig#',
        ],
        'SignedInfo' => [
            '@attributes' => [
                'xmlns:xsi' => 'http://www.w3.org/2001/XMLSchema-instance',
            ],
            'CanonicalizationMethod' => [
                '@attributes' => [
                    'Algorithm' => 'http://www.w3.org/TR/2001/REC-xml-c14n-20010315',
                ],
            ],
            'SignatureMethod' => [
                '@attributes' => [
                    'Algorithm' => 'http://www.w3.org/2000/09/xmldsig#rsa-sha1',
                ],
            ],
            'Reference' => [
                '@attributes' => [
                    'URI' => '', // Reference URI, empty for entire document.
                ],
                'Transforms' => [
                    'Transform' => [
                        '@attributes' => [
```

```
                            'Algorithm' =>
'http://www.w3.org/2000/09/xmldsig#enveloped-signature',
                        ],
                    ],
                ],
                'DigestMethod' => [
                    '@attributes' => [
                        'Algorithm' => 'http://www.w3.org/2000/09/xmldsig#sha1',
                    ],
                ],
                'DigestValue' => '', // Will contain the digest value.
            ],
        ],
        'SignatureValue' => '', // Will contain the signature value.
        'KeyInfo' => [
            'KeyValue' => [
                'RSAKeyValue' => [
                    'Modulus' => '', // Will contain the certificate modulus.
                    'Exponent' => '', // Will contain the certificate exponent.
                ],
            ],
            'X509Data' => [
                'X509Certificate' => '', // Will contain the certificate.
            ],
        ],
    ],
]
```

Implementation Details

XML Invalidation

The `Signature` class automatically invalidates the XML representation when data is modified:

```
// This will cause the internal XML to be invalidated.
$signatureNode->setData($newData);

// This will also invalidate the XML.
$signatureNode->setSignatureValue($newSignatureValue);
```

After invalidation, the XML must be regenerated (typically by the `SignatureGenerator` class) before `getXml()` can be called again.

Reference URIs

Reference URIs are handled according to the XML-DSIG standard:

- An empty URI (`""`) means the entire document is signed.
- A URI starting with `#` refers to an element with the specified ID.
- The `getReference()` method returns the reference without the `#` prefix.
- The `configureSignatureData()` method automatically adds the `#` prefix if not present.

Transformation Algorithm

The transformation algorithm changes based on whether a reference is provided:

- With a reference: `http://www.w3.org/TR/2001/REC-xml-c14n-20010315` (standard C14N).
- Without a reference: `http://www.w3.org/2000/09/xmldsig#enveloped-signature` (enveloped signature transformation).

This ensures that the signature is correctly calculated for both whole-document signatures and element signatures.

Last updated on 24/08/2026

Signature Generator

SignatureGenerator Class

Anonymous · 24/08/2026

SignatureGenerator Class

The `SignatureGenerator` class is responsible for creating digital signatures, both for general data and specifically for XML documents according to the XML-DSIG standard.

Overview

The `SignatureGenerator` implements the `SignatureGeneratorInterface` and provides mechanisms to:

1. Sign any data using a private key.
2. Sign XML documents using a certificate.
3. Calculate digest values for XML documents or specific elements within them.

This class is a core component of the Derafu Signature library and is typically used through the `SignatureService` facade.

Basic Usage

Initialization

```
use Derafu\Signature\Service\SignatureGenerator;
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 the signature generator.
$generator = new SignatureGenerator($xmlService);
```

Signing Data

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

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

Signing XML Documents

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

// 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 = $generator->signXml($xmlString, $certificate);

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

Calculating Digest Values

```
// Create an XML document.
$xmlDoc = new \Derafu\Xml\XmlDocument();
$xmlDoc->loadXml('<root><element ID="myElement">data</element></root>');

// Calculate digest value for the entire document.
$digestValue = $generator->generateXmlDigestValue($xmlDoc);

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

API Reference

Data Signing

```
/**
 * Sign the provided data using a private key.
 *
 * @param string $data Data to be signed.
 * @param string $privateKey Private key to be used for signing.
```

```
     * @param string|int $signatureAlgorithm Algorithm to be used for signing (default
    SHA1).
     * @return string Digital signature in base64.
     * @throws SignatureException If the signing operation fails.
     */
    public function sign(
        string $data,
        string $privateKey,
        string|int $signatureAlgorithm = OPENSSL_ALGO_SHA1
    ): string;
```

XML Signing

```
    /**
     * Sign an XML document using RSA and SHA1.
     *
     * @param XmlDocumentInterface|string $xml XML document to be signed.
     * @param CertificateInterface $certificate Digital certificate to be used for
    signing.
     * @param ?string $reference Reference to which the signature is made. If not
     * specified, the digest of the entire XML document will be signed.
     * @return string String XML with the generated signature included in the
     * "Signature" tag at the end of the XML (last element within the root node).
     * @throws SignatureException If any problem occurs while signing.
     */
    public function signXml(
        XmlDocumentInterface|string $xml,
        CertificateInterface $certificate,
        ?string $reference = null
    ): string;
```

Digest Value Generation

```
    /**
     * Generate the SHA1 ("DigestValue") of a node of the XML with a certain
     * reference. This can be used later to generate the XML signature.
     *
     * If no reference is specified, the "DigestValue" will be calculated over the
     * entire XML (root node).
     *
     * @param XmlDocumentInterface $doc XML document to be signed.
     * @param ?string $reference Reference to which the signature is made.
     * @return string Data of the XML that must be digested.
     * @throws XmlException If the reference is not found in the XML.
     */
    public function generateXmlDigestValue(
        XmlDocumentInterface $doc,
        ?string $reference = null
    ): string;
```

Implementation Details

Digital Signature Process

For general data, the signing process is straightforward:

1. The data is signed using the private key and the specified algorithm.
2. The resulting binary signature is base64-encoded.
3. The base64-encoded signature is returned.

XML Signature Process

For XML documents, the signing process follows the XML-DSIG standard:

1. The XML document is loaded and parsed.
2. If a reference is provided, the referenced element is located.
3. The digest value (SHA1 hash) of the canonicalized (C14N) content is calculated.
4. A `Signature` node is created with the digest value and certificate information.
5. The `SignedInfo` element of the signature is canonicalized.
6. The canonicalized `SignedInfo` is signed using the certificate's private key.
7. The signature value is added to the `SignatureValue` element.
8. The complete signature node is added to the XML document.
9. The signed XML document is returned.

Canonicalization

The class uses the `C14NWithIso88591Encoding` method for canonicalization, which:

1. Applies XML canonicalization (C14N) according to the W3C standard.
2. Converts the result to ISO-8859-1 encoding.
3. Ensures consistent representation across different systems.

This process is crucial for ensuring that the same signature is generated regardless of the XML document's formatting or encoding.

Reference Handling

When a reference is provided:

1. The reference must be an ID attribute value in the XML document.
2. The digest is calculated only for the referenced element.
3. The signature's `Reference` element includes a URI attribute (`#elementId`).
4. The transform algorithm is set to standard C14N.

When no reference is provided:

1. The digest is calculated for the entire document (excluding any existing signature).
2. The signature's `Reference` element has an empty URI attribute.
3. The transform algorithm is set to "enveloped signature transformation".

Signature Node Creation

The class creates a `Signature` node with the following components:

1. `SignedInfo`: Contains information about what data was signed.
    - `CanonicalizationMethod`: Specifies the C14N algorithm.
    - `SignatureMethod`: Specifies RSA-SHA1.
    - `Reference`: Points to the signed content and includes the digest value.
2. `SignatureValue`: Contains the actual signature of the `SignedInfo` element.
3. `KeyInfo`: Contains information about the certificate used for signing.
    - `KeyValue/RSAKeyValue`: Contains the modulus and exponent from the certificate.
    - `X509Data/X509Certificate`: Contains the certificate itself.

Last updated on 24/08/2026

Signature Validator

SignatureValidator Class

Anonymous · 24/08/2026

SignatureValidator Class

The `SignatureValidator` class is responsible for validating digital signatures, both for general data and specifically for XML documents that follow the XML-DSIG standard.

Overview

The `SignatureValidator` implements the `SignatureValidatorInterface` and provides mechanisms to:

1. Validate signatures for any data against a public key.
2. Validate signatures in XML documents.
3. Extract and parse signature nodes from XML documents.
4. Validate specific aspects of XML signatures (digest values and signature values).

This class is a core component of the Derafu Signature library and is typically used through the `SignatureService` facade.

Basic Usage

Initialization

```
use Derafu\Signature\Service\SignatureGenerator;
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 the signature generator (needed by validator).
$generator = new SignatureGenerator($xmlService);
```

```
// Create the signature validator.
$validator = new SignatureValidator($generator, $xmlService);
```

Validating Data Signatures

```
// Validate a signature for some data.
$data = 'Data that was signed';
$signature = 'base64EncodedSignature';
$publicKey = 'publicKeyPEM';

$isValid = $validator->validate($data, $signature, $publicKey);

// Optional: Specify the signature algorithm.
$isValid = $validator->validate($data, $signature, $publicKey, OPENSSL_ALGO_SHA256);

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

Validating XML Signatures

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

// Validate an XML document with a signature.
$signedXml = file_get_contents('signed_document.xml');

try {
    $validator->validateXml($signedXml);
    echo "XML signature is valid!";
} catch (SignatureException $e) {
    echo "XML signature validation failed: " . $e->getMessage();
}
```

Working with Signature Nodes

```
// Extract and parse a signature node from XML.
$signatureXml = '<Signature
xmlns="http://www.w3.org/2000/09/xmldsig#">...</Signature>';
$signatureNode = $validator->createSignatureNode($signatureXml);

// Access signature components.
$reference = $signatureNode->getReference();
$digestValue = $signatureNode->getDigestValue();
$signatureValue = $signatureNode->getSignatureValue();
$x509Certificate = $signatureNode->getX509Certificate();
```

Detailed XML Signature Validation

```
// Create a signature node from a signed XML document.
$signatureNode = $validator->createSignatureNode($signatureXml);

// Validate just the digest value (content integrity).
try {
    $validator->validateXmlDigestValue($xmlDoc, $signatureNode);
    echo "Content integrity verified!";
} catch (SignatureException $e) {
    echo "Content may have been tampered with: " . $e->getMessage();
}

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

API Reference

Data Signature Validation

```
/**
 * Validate the digital signature of data.
 *
 * @param string $data Data to be verified.
 * @param string $signature Digital signature of the data in base64.
 * @param string $publicKey Public key of the signature of the data.
 * @param string|int $signatureAlgorithm Algorithm used to sign (default SHA1).
 * @return bool `true` if the signature is valid, `false` if it is invalid.
 * @throws SignatureException If there was an error while validating.
 */
public function validate(
    string $data,
    string $signature,
    string $publicKey,
    string|int $signatureAlgorithm = OPENSSL_ALGO_SHA1
): bool;
```

XML Signature Validation

```
/**
```

```
     * Validate the validity of an XML signature using RSA and SHA1.
     *
     * @param XmlDocumentInterface|string $xml XML string to be validated.
     * @return void
     * @throws SignatureException If there was an error while validating.
     */
    public function validateXml(XmlDocumentInterface|string $xml): void;
```

Signature Node Handling

```
    /**
     * Creates the `Signature` instance from a string XML with the signature node.
     *
     * @param string $xml String with the XML of the `Signature` node.
     * @return SignatureInterface
     */
    public function createSignatureNode(string $xml): SignatureInterface;
```

Detailed Validation Methods

```
    /**
     * Validate the DigestValue of the signed data.
     *
     * @param XmlDocumentInterface|string $xml Document to be validated.
     * @param SignatureInterface $signatureNode Signature node to be validated.
     * @return void
     * @throws SignatureException If the DigestValue is invalid.
     */
    public function validateXmlDigestValue(
        XmlDocumentInterface|string $xml,
        SignatureInterface $signatureNode
    ): void;

    /**
     * Validate the signature of the `SignedInfo` node of the XML using the X509
     * certificate.
     *
     * @param SignatureInterface $signatureNode Signature node to be validated.
     * @throws SignatureException If the XML signature is invalid.
     */
    public function validateXmlSignatureValue(
        SignatureInterface $signatureNode
    ): void;
```

Implementation Details

Signature Validation Process

For general data signatures, the validation process is as follows:

1. The public key is normalized (headers and footers are added if missing).
2. The base64-encoded signature is decoded to binary.
3. The `openssl_verify` function is called with the data, decoded signature, and public key.
4. If the result is 1, the signature is valid; if 0, it's invalid; if -1, an error occurred.

XML Signature Validation Process

For XML signatures, the validation process follows the XML-DSIG standard:

1. The XML document is loaded and parsed.
2. All `Signature` elements in the document are located.
3. For each signature element:
    - A `Signature` node object is created from the element's XML.
    - The digest value is validated to ensure content integrity.
    - The signature value is validated to ensure signer authenticity.

Digest Value Validation

The digest value validation ensures that the content being signed hasn't been modified:

1. The XML document is loaded.
2. The reference from the signature node is extracted (if any).
3. The digest value is calculated for the referenced content or the entire document.
4. The calculated digest value is compared to the one in the signature.
5. If they don't match, a `SignatureException` is thrown.

Signature Value Validation

The signature value validation ensures that the signature was created with the corresponding private key:

1. The `SignedInfo` element is extracted and canonicalized.
2. The base64-encoded signature value is obtained from the `SignatureValue` element.
3. The public key is extracted from the `X509Certificate` element.
4. The signature is validated using the canonicalized `SignedInfo`, the signature value, and the public key.
5. If the validation fails, a `SignatureException` is thrown.

Error Handling

The `SignatureValidator` provides detailed error messages when validation fails:

- For data signatures, it indicates when an error occurred during validation.
- For XML signatures, it indicates whether the problem is with the digest value (content integrity) or the signature value (signer authenticity).
- For missing signatures or malformed XML, it provides clear error messages.

These detailed error messages help diagnose the exact cause of validation failures.

Dependency on SignatureGenerator

The `SignatureValidator` requires an instance of `SignatureGeneratorInterface` to calculate digest values for XML documents. This dependency ensures that the same digest calculation algorithm is used for both signing and validating.

XML Handling

The class uses the Derafu XML library for XML operations:

- Loading and parsing XML documents.
- Extracting signature elements.
- Canonicalizing XML for digest and signature validation.
- Converting between XML and PHP arrays.

This integration ensures consistent handling of XML across the library.

Last updated on 24/08/2026

Signature Service

SignatureService Class

Anonymous · 24/08/2026

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.

XML-DSIG Standard

XML Digital Signature (XML-DSIG)

Anonymous · 24/08/2026

XML Digital Signature (XML-DSIG)

This document provides an in-depth explanation of how the Derafu Signature library implements the XML Digital Signature (XML-DSIG) standard, including the structure of signature elements, the signing process, and the validation process.

XML-DSIG Overview

XML Digital Signatures provide integrity, message authentication, and signer authentication for XML data. The XML-DSIG standard is defined by the W3C in the XML Signature Syntax and Processing specification.

Signature Structure

The Derafu Signature library creates XML signatures with the following structure:

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <CanonicalizationMethod
Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <Reference URI="#elementId">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>base64EncodedDigestValue</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>base64EncodedSignatureValue</SignatureValue>
  <KeyInfo>
    <KeyValue>
      <RSAKeyValue>
        <Modulus>base64EncodedModulus</Modulus>
        <Exponent>base64EncodedExponent</Exponent>
      </RSAKeyValue>
    </KeyValue>
```

```
    <X509Data>
      <X509Certificate>base64EncodedCertificate</X509Certificate>
    </X509Data>
  </KeyInfo>
</Signature>
```

Key Components

1. **SignedInfo**: Contains information about what data is being signed.
    - **CanonicalizationMethod**: Specifies how the XML is normalized before signing.
    - **SignatureMethod**: Specifies the algorithm used for signing (RSA-SHA1).
    - **Reference**: Points to the data being signed.
        - **Transforms**: Describes transformations applied to the data before digesting.
        - **DigestMethod**: Specifies the algorithm used for the digest (SHA1).
        - **DigestValue**: Contains the base64-encoded digest of the data.
2. **SignatureValue**: Contains the base64-encoded signature of the canonicalized SignedInfo element.
3. **KeyInfo**: Contains information about the key used to validate the signature.
    - **KeyValue/RSAKeyValue**: Contains the RSA key parameters (modulus and exponent).
    - **X509Data/X509Certificate**: Contains the X.509 certificate used for signing.

Signing Process

The Derafu Signature library implements XML signing following these steps:

1. Preparing the Data

- If a reference ID is specified, the referenced element is located in the XML document.
- Otherwise, the entire XML document is used (excluding any existing Signature elements).

2. Calculating the Digest Value

- The data is canonicalized using the C14N algorithm.
- The canonicalized data is converted to ISO-8859-1 encoding.
- The SHA1 digest of the data is calculated and base64-encoded.

3. Creating the SignedInfo Element

- The SignedInfo element is created with the appropriate CanonicalizationMethod, SignatureMethod, and Reference elements.

- The DigestValue is included in the Reference element.

4. Calculating the Signature Value

- The SignedInfo element is canonicalized and converted to ISO-8859-1 encoding.
- The canonicalized SignedInfo is signed using the private key from the certificate.
- The resulting signature is base64-encoded and included in the SignatureValue element.

5. Including Key Information

- The public key components (modulus and exponent) are extracted from the certificate.
- The certificate itself is included in the X509Certificate element.

6. Adding the Signature to the Document

- The complete Signature element is added to the XML document, typically as the last child of the root element.

Validation Process

The Derafu Signature library validates XML signatures following these steps:

1. Locating Signature Elements

- All Signature elements in the XML document are located.
- Each signature is validated independently.

2. Validating the Digest Value

- The reference in the signature is extracted.
- The referenced data (or the entire document) is canonicalized and converted to ISO-8859-1 encoding.
- The SHA1 digest of the data is calculated and base64-encoded.
- The calculated digest is compared to the DigestValue in the signature.
- If they don’t match, the content integrity check fails.

3. Validating the Signature Value

- The SignedInfo element is canonicalized and converted to ISO-8859-1 encoding.
- The X.509 certificate is extracted from the X509Certificate element.
- The signature value is validated using the canonicalized SignedInfo, the SignatureValue, and the public key from the certificate.
- If the validation fails, the signer authenticity check fails.

Reference Types

The Derafu Signature library supports two types of references:

1. Element References

- Specified by a URI attribute with a value starting with `#` (e.g., `URI="#elementId"`).
- Only the referenced element is signed.
- The transform algorithm is set to standard C14N.

2. Whole Document References

- Specified by an empty URI attribute (`URI=""`).
- The entire document is signed, excluding any Signature elements.
- The transform algorithm is set to “enveloped signature transformation”.

Canonicalization

The Derafu Signature library uses the following canonicalization algorithms:

1. XML Canonicalization (C14N)

- Algorithm: `http://www.w3.org/TR/2001/REC-xml-c14n-20010315`
- Ensures consistent XML representation regardless of formatting differences.
- Applied to data before calculating digests and to SignedInfo before calculating signatures.

2. Enveloped Signature Transform

- Algorithm: `http://www.w3.org/2000/09/xmldsig#enveloped-signature`
- Excludes the signature itself when signing the entire document.
- Prevents circular references in the signing process.

Working with References and IDs

When using element references, the referenced element must have an `ID` attribute:

```
<root>
  <element ID="myElement">data</element>
</root>
```

The reference in the signature would be:

```
<Reference URI="#myElement">
```

```
  <!-- ... -->
</Reference>
```

The `SignatureGenerator.signXml()` method accepts the ID as a parameter:

```
$signedXml = $signatureGenerator->signXml($xml, $certificate, 'myElement');
```

Security Considerations

Digest Algorithm

The library uses SHA1 for digest calculation. While SHA1 is considered cryptographically weak for certain applications, it remains the standard algorithm specified in the XML-DSIG specification.

Signature Algorithm

The library uses RSA-SHA1 for signature calculation, which is the standard algorithm specified in the XML-DSIG specification.

Certificate Handling

The library includes the entire X.509 certificate in the signature, which allows for complete validation including certificate chain verification (although this is not currently implemented in the library).

Compatibility

The XML signatures generated by the Derafu Signature library follow the W3C XML-DSIG standard and should be compatible with other XML-DSIG implementations. However, different implementations may have subtle differences in canonicalization or other aspects of the signing process.

Common Issues

Invalid References

If a reference ID is specified but doesn't exist in the XML document, the signing process will fail.

Malformed XML

If the XML document is not well-formed, both signing and validation will fail.

Character Encoding

The library uses ISO-8859-1 encoding for canonicalized XML to ensure consistent digest and signature calculation across different systems.

Example Usage

Signing with a Reference

```
$xml = '<root><element ID="myElement">data</element></root>';
$signedXml = $signatureService->signXml($xml, $certificate, 'myElement');
```

Signing the Entire Document

```
$xml = '<root><element>data</element></root>';
$signedXml = $signatureService->signXml($xml, $certificate);
```

Validating a Signature

```
try {
    $signatureService->validateXml($signedXml);
    echo "Signature is valid!";
} catch (SignatureException $e) {
    echo "Signature validation failed: " . $e->getMessage();
}
```

Last updated on 24/08/2026