Signature Validator

SignatureValidator Class

Anonymous · 09/09/2026

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 09/09/2026