Asymmetric Key Helper

AsymmetricKeyHelper Class

Anonymous · 09/09/2026

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 09/09/2026