Logius Standard - Draft

API Design Rules Module: Encryption

Logius Standard

Draft December 15, 2025

This version:

https://logius-standaarden.github.io/API-mod-encryption/

Latest published version:

https://logius-standaarden.github.io/API-mod-encryption/

Latest editor's draft:

https://logius-standaarden.github.io/API-mod-encryption/

Editor:

Logius Standaarden (Logius)

Author:

Peter Haasnoot (Logius)

Participate:

GitHub Logius-standaarden/API-mod-encryption

File an issue

Commit history

Pull requests

This document is also available in these non-normative format: PDF

Status of This Document

This is a draft that could be altered, removed or replaced by other documents. It is not a recommendation approved by TO.

Table of Contents

§ Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

The key word *MUST* in this document is to be interpreted as described in BCP 14 [*RFC2119*] [*RFC8174*] when, and only when, it appears in all capitals, as shown here.

Abstract

This document is part of the *Nederlandse API Strategie*.

The Nederlandse API Strategie consists of a set of distinct documents.

Status	Description & Link
Informative	Inleiding NL API Strategie
Informative	Architectuur NL API Strategie
Informative	Gebruikerswensen NL API Strategie
Normative	API Design Rules (ADR v2.0)
Normative	Open API Specification (OAS 3.0)
Normative	NL GOV OAuth profiel
Normative	Digikoppeling REST API koppelvlak specificatie
Normative module	GEO module v1.0

Before reading this document it is advised to gain knowledge of the informative documents, in particular the Architecture.

This ADR Module contains the requirements for ADR REST-API encryption based on JWE.

1. Introduction

This module specifies the use of JWE for HTTP payload encryption.

This module is applicable when there is a need for end to end message payload confidentiality between client application and server application. In a complex IT landscape the path between client and server can go over several intermediary components/systems in which case end to end confidentiality can be especially relevant. (In this case TLS is terminated in each step on the path and does not protect the http-message in transport fully end to end). A specific example is when there is confidential data that is processed and routed by an intermediary organization which is not allowed to access the contents of the message.

2. Normative Design Rules

2.1 Summary

Design rules are technical rules, which should be tested automatically.

List of technical rules

- /encryption/jwe: Use JSON Web Encryption (JWE)

2.2 Design rules

> Technical
>
> **/encryption/jwe**: Use JSON Web Encryption (JWE)
>
> **Statement**
>
> For HTTP payload encryption *JSON Web Encryption (JWE)* *MUST* be applied with the following requirements.
>
> - The request is sent to Service Provider with `content-type: application/jose+json`.

- An encrypted request needs to pass application/jose+json as the value for the Content-Type and Accept headers:

```
Content-Type: application/jose+json
Accept: application/jose+json
```

- When the encrypted request uses an unsupported algorithm, the Service Provider rejects the request with a 400 HTTP response.
- Use for encryption the public key from the X.509 certificate of the other party
- Use the following parameters in the JWE protected header:

```
{
  "alg": "RSA-OAEP",
  "enc": "A256GCM",
  "typ": "JWE"
}
```

- JWE compact serialization format is used

The following algorithms *MUST* be applied used.

- Key Management: RSA-OAEP
- Content encryption: A256GCM

§ 2.3 Basic JWE proces flow

The basic flow for encryption using JWE is as follows.

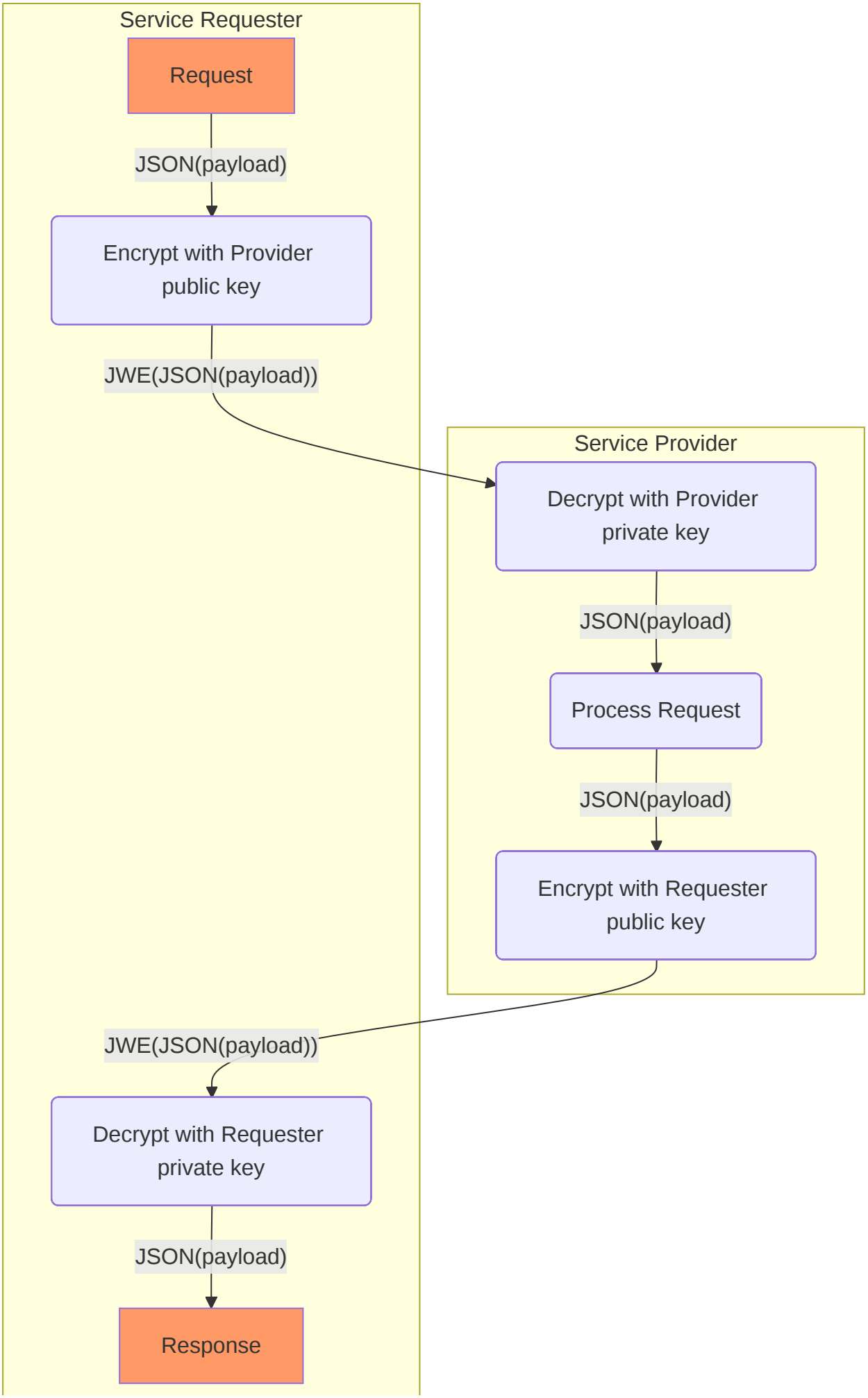
Service Requester
Request
JSON(payload)
Encrypt with Provider public key
JWE(JSON(payload))
Service Provider
Decrypt with Provider private key
JSON(payload)
Process Request
JSON(payload)
Encrypt with Requester public key
JWE(JSON(payload))
Decrypt with Requester private key
JSON(payload)
Response

Figure 1 Encryption

1. Service Requester encrypts payload using Service Provider public encryption key:
2. Service Provider decrypts the request using the corresponding Service Provider private encryption key.
3. Service Provider performs the request and then generates an encrypted response;
4. Service Requester decrypts response using Requester private key

A. Tooling

This section is non-normative.

Libraries implementing JWT and the JOSE specs JWS, JWE, JWK, and JWA are listed here.

https://openid.net/developers/jwt-jws-jwe-jwk-and-jwa-implementations/

B. References

B.1 Normative references

[RFC2119]
Key words for use in RFCs to Indicate Requirement Levels. S. Bradner. IETF. March 1997. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc2119

[rfc7516]
JSON Web Encryption (JWE). M. Jones; J. Hildebrand. IETF. May 2015. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc7516

[RFC8174]
Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words. B. Leiba. IETF. May 2017. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc8174