Encryption

Open API for FSP Interoperability Specification

Encryption

Open API for FSP Interoperability Specification

Table of Contents

Table of Tables

1 Preface

This section contains information about how to use this document.

1.1 Conventions Used in This Document

The following conventions are used in this document to identify the specified types of information

Type of Information	Convention	Example
Elements of the API, such as resources	Boldface	**/authorization**
Variables	Italics within angle brackets	*<ID>*
Glossary terms	Italics on first occurrence; defined in *Glossary*	The purpose of the API is to enable interoperable financial transactions between a *Payer* (a payer of electronic funds in a payment transaction) located in one *FSP* (an entity that provides a digital financial service to an end user) and a *Payee* (a recipient of electronic funds in a payment transaction) located in another FSP.
Library documents	Italics	User information should, in general, not be used by API deployments; the security measures detailed in *API Signature* and *API Encryption* should be used instead.

1.2 Document Version Information

Version	Date	Change Description
1.0	2019-02-22	Initial version
1.1	2020-05-09	This version contains the below changes: - ExstensionList elements in Section 4 have been updated based on the issue "Interpretation of the Data Model for the ExtensionList element", to fix the data model of the extensionList Object.

2 Introduction

This document details security methods to be implemented for Open API (Application Programming Interface) for FSP (Financial Service Provider) Interoperability (hereafter cited as "the API") to ensure confidentiality of API messages between an API client and the API server.

In information security, *confidentiality* means that information is not made available or disclosed to unauthorized individuals, entities, or processes (excerpt from ISO27000[1]). For the API, confidentiality means that some sensitive fields in the payload of an API message cannot be accessed or identified in an unauthorized or undetected manner by the intermediaries involved in the API communication. That is, if some fields of an API message are encrypted by the API client, then only the expected API recipient can decrypt those fields.

JSON Web Encryption (JWE, defined in RFC 7516[2]) must be applied to the API to provide end to end message confidentiality. When an API client sends an HTTP request (such as an API request or callback message) to a counterparty, the API client can determine whether there are sensitive fields in the API message to be protected according to the regulation or local schema. If there is a field to be protected, then the API client uses JWE to encrypt the value of that field. Subsequently, the cipher text of that field will be transmitted to the counterparty.

To support encryption for multiple fields of an API message, JWE is extended in this document to adapt to the requirements of the API.

[1] http://www.27000.org/ - The ISO 27000 Directory

[2] https://tools.ietf.org/html/rfc7516 - JSON Web Encryption (JWE)

2.1 Open API for FSP Interoperability Specification

The Open API for FSP Interoperability Specification includes the following documents.

2.1.1 General Documents

- *Glossary*

2.1.2 Logical Documents

- *Logical Data Model*
- *Generic Transaction Patterns*
- *Use Cases*

2.1.3 Asynchronous REST Binding Documents

- *API Definition*
- *JSON Binding Rules*
- *Scheme Rules*

2.1.4 Data Integrity, Confidentiality, and Non-Repudiation

- *PKI Best Practices*
- *Signature*
- *Encryption*

3 API Encryption Definition

This section introduces the technology used by API encryption, including:

- Data exchange format for the encrypted fields of an API message.
- Mechanism for encrypting and decrypting fields.

3.1 Encryption Data Model

The API uses the customized HTTP header parameter **FSPIOP-Encryption** to represent the encrypted fields of an API message; its value is a JSON object serialization. The data model of this parameter is described in Table 1, Table 2 and Table 3.

Note: If **FSPIOP-Encryption** is present in an API message, then it must also be protected by the API signature. That means **FSPIOP-Encryption** must be included in the JWS Protected Header of the signature.

Name	Cardinality	Type	Description
encryptedFields	1	EncryptedFields	Information about the encrypted fields of an API message

Table 1 – Data model of HTTP Header Field FSPIOP-Encryption

Name	Cardinality	Type	Description
encryptedField	1..*	EncryptedField	Information about the encrypted field of an API message

Table 2 – Data model of complex type EncryptedFields

Name	Cardinality	Type	Description
fieldName	1	String(1..512)	This element identifies the field to be encrypted in the payload of an API message. Because the API payload is a JSON Object serialization string, the field name must be able to identify the exact element path in the JSON Object. A single period ('**.**') character is used to separate the elements in an element path. For example, **payer.personalInfo.dateOfBirth** is a valid value for this element for the API request **POST /quotes**.
encryptedKey	1	String(1..512)	Encrypted Content Encryption Key (CEK) value. Its value is encoded by BASE64URL(JWE Encrypted Key). If there are multiple fields of the API message to be encrypted, we recommend that the same JWE Encrypted Key be used to simplify the implementation; however, this is a decision to be made by each FSP internally based on their implementation.
protectedHeader	1	String(1..1024)	This element identifies the Header Parameters that are applied to JWE to encrypt the specified field. Its value is encoded by BASE64URL(UTF8(JWE Protected Header)). For example, if the JWE Protected Header applied to the encryption is **{"alg":"RSA-OAEP-256","enc":"A256GCM"}**, then the value is **eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkEyNTZHQ00ifQ**.
initializationVector	1	String(1..128)	Initialization Vector value used when encrypting the plaintext. Its value is encoded by BASE64URL(JWE Initialization Vector).
authenticationTag	1	String(1..128)	Authentication Tag value resulting from authenticated encryption of the plaintext with Additional Authenticated Data. Its value is encoded by BASE64URL(JWE Authentication Tag)

Table 3 – Data model of complex type EncryptedField

3.2 Encrypt Fields of API Message

This section describes the encryption process for message fields. The order of the steps is not significant in cases where there are no dependencies between the inputs and outputs of the steps.

1. Determine the algorithm used to determine the CEK value (this is the algorithm recorded in the **alg** (algorithm) Header Parameter of the resulting JWE). Because the CEK should be encrypted with the public key of the API recipient, in the API the available algorithms to protect the CEK can only be **RSA-OAEP-256**.
2. If there are multiple fields to be encrypted in the API message, then perform Steps 3-15 for each field.
3. Generate a random CEK. The FSP can generate the value using either its own application or using the JWE implementation employed.
4. Encrypt the CEK with the algorithm determined by the JWE header parameter **alg**.
5. Compute the encoded key value BASE64URL(JWE Encrypted Key).
6. Generate a random JWE Initialization Vector of the correct size for the content encryption algorithm (if required for the algorithm); otherwise, let the JWE Initialization Vector be the empty octet sequence.
7. Compute the encoded Initialization Vector value BASE64URL(JWE Initialization Vector).
8. If a **zip** parameter was included, compress the plaintext using the specified compression algorithm and let *M* be the octet sequence representing the compressed plain text; otherwise, let *M* be the octet sequence representing the plain text.
9. Create the JSON object or objects containing the desired set of header parameters, which together comprise the JWE Protected Header. Besides the parameter **alg**, the parameter **enc** must be included in the JWE Protected Header. The available values for the parameter **enc** in the API can only be: **A128GC***M*, **A192GC***M*, **A256GC***M*. **A256GC***M* is recommended.
10. Compute the Encoded Protected Header value BASE64URL(UTF8(JWE Protected Header)).
11. Let the Additional Authenticated Data encryption parameter be ASCII(Encoded Protected Header).
12. Encrypt *M* using the CEK, the JWE Initialization Vector, and the Additional Authenticated Data value using the specified content encryption algorithm to create the JWE Cipher text value and the JWE Authentication Tag (which is the Authentication Tag output from the encryption operation).
13. Compute the encoded cipher text value BASE64URL(JWE Cipher Text).
14. Compute the encoded Authentication Tag value BASE64URL(JWE Authentication Tag).
15. Compute the **encryptedField** element (see Table 3) for the HTTP header parameter **FSPIOP-Encryption**.
16. Compute the value for the HTTP Header parameter **FSPIOP-Encryption** as described in Section 2.1. The value for this **FSPIOP-Encryption** is JSON Object Serialization string.

Note: If JWE is used to encrypt some fields of the payload, then the API client should:

1. Encrypt the desired fields.
2. Replace those fields' value with the encoded cipher text in the payload.
3. Sign the payload.

3.3 Decrypt Fields of API Message

If the HTTP Header parameter **FSPIOP-Encryption** (which is also protected by the API signature) is present, then the API message recipient should decrypt the encrypted fields of the API message after the API signature is validated successfully. The message decryption process is the reverse of the encryption process. The order of the steps is not significant in cases where there are no dependencies between the inputs and outputs of the steps. If there are multiple fields being encrypted, then all fields must be decrypted successfully; otherwise it indicates the API message is invalid.

1. Parse the HTTP Header parameter **FSPIOP-Encryption** to get encrypted fields' information, including field name, JWE Protected Header, JWE Encrypted Key, JWE Initialization Vector, and JWE Authentication Tag for each field. If there are multiple fields being encrypted, then perform Steps 2-9 for each encrypted field.
2. Get the cipher text of the encrypted field by parsing the payload with the specified field path. The value of the specified field is already encoded with BASE64URL.
3. Verify that the octet sequence resulting from decoding the encoded JWE Protected Header is a UTF-8-encoded representation of a valid JSON object conforming to JSON Data Interchange Format (defined in RFC 7159[3]); let the JWE Protected Header be this JSON object.
4. Verify that the parameters in the JWE Protected Header understand and can process all fields that are required to support the JWE specification; for example, the algorithm being used.
5. Determine that the algorithm specified by the **alg** (algorithm) Header Parameter matches the algorithm of the public / private key of the API recipient.
6. Decrypt the JWE Encrypted Key with the private key of the API recipient to get the JWE CEK.
7. Let the Additional Authenticated Data encryption parameter be ASCII(Encoded Protected Header).
8. Decrypt the JWE Cipher Text using the CEK, the JWE Initialization Vector, the Additional Authenticated Data value, and the JWE Authentication Tag (which is the Authentication Tag input to the calculation) using the specified content encryption algorithm, returning the decrypted plaintext and validating the JWE Authentication Tag in the manner specified for the algorithm. If the JWE Authentication Tag is incorrect, then reject the input without any decryption.
9. If a **zip** parameter was included, then the API recipient should decompress the decrypted plaintext using the specified compression algorithm.

[3] https://tools.ietf.org/html/rfc7159 - The JavaScript Object Notation (JSON) Data Interchange Format

4 API Encryption/Decryption Examples

This section uses a typical quote process to explain how the API encryption and decryption are implemented using JWE.

As the algorithm of public / private key of the API recipient can only be RSA, the RSA key used for this example is represented in JSON Web Key (JWK, defined in RFC 7517[4]) format below (with line breaks and indentation within values for display purposes only):


```
{
    "kty": "RSA",
    "n": "oahUIoWw0K0usKNuOR6H4wkf4oBUXHTxRvgb48E-
        BVvxkeDNjbC4he8rUWcJoZmds2h7M70imEVhRU5djINXtqllXI4DFqcI1DgjT9Le
        wND8MW2Krf3Spsk_ZkoFnilakGygTwpZ3uesH-
        PFABNIUYpOiN15dsQRkgr0vEhxN92i2asbOenSZeyaxziK72UwxrrKoExv6kc5tw
        XTq4h-QChLOln0_mtUZwfsRaMStPs6mS6XrgxnxbWhojf663tuEQueGC-
        FCMfra36C9knDFGzKsNa7LZK2djYgyD3JR_MB_4NUJW_TqOQtwHYbxevoJArm-
        L5StowjzGy-_bq6Gw",
    "e": "AQAB",
    "d": "kLdtIj6GbDks_ApCSTYQtelcNttlKiOyPzMrXHeI-yk1F7-kpDxY4-
        WY5NWV5KntaEeXS1j82E375xxhWMHXyvjYecPT9fpwR_M9gV8n9Hrh2anTpTD93D
        t62ypW3yDsJzBnTnrYu1iwWRgBKrEYY46qAZIrA2xAwnm2X7uGR1hghkqDp0Vqj3
        kbSCz1XyfCs6_LehBwtxHIyh8Ripy40p24moOAbgxVw3rxT_vlt3UVe4WO3JkJOz
        lpUf-KTVI2Ptgm-dARxTEtE-id-4OJr0h-K-
        VFs3VSndVTIznSxfyrj8ILL6MG_Uv8YAu7VILSB3lOW085-4qE3DzgrTjgyQ",
    "p": "1r52Xk46c-LsfB5P442p7atdPUrxQSy4mti_tZI3Mgf2EuFVbUoDBvaRQ-
        SWxkbk-
        moEzL7JXroSBjSrK3YIQgYdMgyAEPTPjXv_hI2_1eTSPVZfzL0lffNn03IXqWF5M
        DFuoUYE0hzb2vhrlN_rKrbfDIwUbTrjjgieRbwC6Cl0",
    "q":
        "wLb35x7hmQWZsWJmB_vle87ihgZ19S8lBEROLIsZG4ayZVe9Hi9gDVCOBmUDdaD
        YVTSNx_8Fyw1YYa9XGrGnDew00J28cRUoeBB_jKI1oma0Orv1T9aXIWxKwd4gvxF
        ImOWr3QRL9KEBRzk2RatUBnmDZJTIAfwTs0g68UZHvtc",
    "dp": "ZK-
        YwE7diUh0qR1tR7w8WHtolDx3MZ_OTowiFvgfeQ3SiresXjm9gZ5KLhMXvo-uz-
        KUJWDxS5pFQ_M0evdo1dKiRTjVw_x4NyqyXPM5nULPkcpU827rnpZzAJKpdhWAgq
        rXGKAECQH0Xt4taznjnd_zVpAmZZq60WPMBMfKcuE",
    "dq":
        "Dq0gfgJ1DdFGXiLvQEZnuKEN0UUmsJBxkjydc3j4ZYdBiMRAy86x0vHCjywcMlY
        Yg4yoC4YZa9hNVcsjqA3FeiL19rk8g6Qn29Tt0cj8qqyFpz9vNDBUfCAiJVeESOj
        JDZPYHdHY8v1b-o-Z2X5tvLx-TCekf7oxyeKDUqKWjis",
    "qi": "VIMpMYbPf47dT1w_zDUXfPimsSegnMOA1zTaX7aGk_8urY6R8-
        ZW1FxU7AlWAyLWybqq6t16VFd7hQd0y6flUK4SlOydB61gwanOsXGOAOv82cHq0E
        3eL4HrtZkUuKvnPrMnsUUFlfUdybVzxyjz9JF_XyaY14ardLSjf4L_FNY"
}
```


[4] https://tools.ietf.org/html/rfc7517 - JSON Web Key (JWK)

4.1 Encryption Example

The following message text is an example of POST /quotes without encryption sent by Payer FSP to a Payee FSP.

```
POST /quotes HTTP/1.1
FSPIOP-Destination:5678
Accept:application/vnd.interoperability.quotes+json;version=1.0
Content-Length:975
Date:Tue, 23 May 2017 21:12:31 GMT
FSPIOP-Source:1234
Content-Type:application/vnd.interoperability.quotes+json;version=1.0

{
 "payee": {
  "partyIdInfo": { "partyIdType": "MSISDN", "partyIdentifier": "15295558888",
   "fspId": "5678" } },
 "amountType": "RECEIVE",
 "transactionType": { "scenario": "TRANSFER", "initiator": "PAYER",
  "subScenario": "P2P Transfer across MM systems", "initiatorType": "CONSUMER" },
 "note": "this is a sample for POST /quotes",
 "amount": { "amount": "150","currency": "USD" },
 "fees": { "amount": "1.5", "currency": "USD" },
 "extensionList": {
    "extension": [
      { "value": "value1", "key": "key1"},
      { "value": "value2", "key": "key2"},
      { "value": "value3", "key": "key3"}
    ]
 },
 "geoCode": { "latitude": "57.323889", "longitude": "125.520001"
 },
 "expiration": "2017-05-24T08:40:00.000-04:00",
 "payer": {
  "personalInfo": {
   "complexName": { "firstName": "Bill", "middleName": "Ben", "LastName": "Lee"
   }, "dateOfBirth": "1986-02-14" },
  "partyIdInfo": { "partyIdType": "MSISDN",
   "partySubIdOrType": "RegisteredCustomer", "partyIdentifier": "16135551212",
   "fspId": "1234"
  }, "name": "Bill Lee"
 }, "quoteId": "59e331fa-345f-4554-aac8-fcd8833f7d50",
 "transactionId": "36629a51-393a-4e3c-b347-c2cb57e1e1fc"
}
```

In this case, the Payer FSP would like to encrypt two fields of the API message: **payer** and **payee.partyIdInfo.partyIdentifier**.

4.1.1 Encrypt the Required Fields

Because there are two fields to be encrypted, the Payer FSP needs to encrypt the two fields one-by-one.

4.1.1.1 Encrypt "payer"

The Payer FSP performs the following steps to encrypt the field **payer** in the **POST /quotes** API message.

1. Determine the algorithm used to determine the CEK value. In this case, assuming it is **RSA-OAEP-256**.
2. Generate a 256-bit random CEK. In this case, its value is (using JSON Array notation):

```
[191 100 167 60 2 248 21 136 172 39 145 120 102 7 73 31 166 66 114 199 219
157 104 162 7 253 10 105 33 136 57 167]
```

3. Encrypt the CEK with the Payee FSP's public key shown in JSON Web Key format in section 4. In this case, the encrypted value is (using JSON Array notation):

```
[22 210 45 47 153 95 183 79 84 26 194 42 27 152 50 195 163 18 235 121 140
120 224 129 180 120 21 0 46 196 21 114 251 148 127 75 198 42 87 250 186 98
15 136 249 131 224 73 111 108 159 140 107 156 80 30 133 77 86 26 28 13 66 83
248 229 132 77 203 113 229 24 208 155 81 172 9 164 25 126 206 217 25 206 30
218 38 190 128 196 250 233 34 47 86 91 157 140 87 240 29 119 126 136 168 10
87 246 213 23 104 114 215 134 71 87 46 55 131 174 15 193 194 90 194 208 212
15 24 33 143 38 253 125 121 175 220 202 106 95 127 129 192 2 72 137 14 40
147 207 166 239 161 248 159 203 52 223 103 129 54 83 85 199 211 228 56 82 83
135 166 103 42 76 191 146 80 40 192 123 42 18 31 113 25 198 24 58 87 149 47
182 144 86 182 137 253 103 214 34 192 76 254 64 14 114 97 194 28 60 75 164
131 170 210 231 168 205 181 78 79 136 82 115 218 174 5 48 158 162 230 229
164 85 26 135 15 164 206 36 182 131 115 224 207 12 178 234 145 6 181 140 243
41 8 151]
```

4. Compute the encoded key value BASE64URL(JWE Encrypted Key). In this case, its value is:

```
FtItL5lft09UGsIqG5gyw6MS63mMeOCBtHgVAC7EFXL7lH9LxipX-rpiD4j5g-BJb2yfjGucUB6
FTVYaHA1CU_jlhE3LceUY0JtRrAmkGX7O2RnOHtomvoDE-ukiL1ZbnYxX8B13foioClf21Rdoct
eGR1cuN4OuD8HCWsLQ1A8YIY8m_X15r9zKal9_gcACSIkOKJPPpu-h-J_LNN9ngTZTVcfT5DhSU
4emZypMv5JQKMB7KhIfcRnGGDpXlS-2kFa2if1n1iLATP5ADnJhwhw8S6SDqtLnqM21Tk-IUnPa
rgUwnqLm5aRVGocPpM4ktoNz4M8MsuqRBrWM8ykIlw
```

5. Generate a random JWE Initialization Vector of the correct size for the content encryption algorithm. In this case, its value is (using JSON Array notation):

```
[101 98 192 15 167 157 93 152 54 145 173 236 83 4 6 243]
```

6. Compute the encoded Initialization Vector value BASE64URL(JWE Initialization Vector). In this case, its value is:

```
ZWLAD6edXZg2ka3sUwQG8w
```

7. Get the plaintext of the field **payer** of the API message as the payload to be encrypted. In this case, its value is:

```
{
 "personalInfo": {
  "dateOfBirth": "1986-02-14", "complexName": {
  "middleName": "Ben",  "lastName": "Lee", "firstName": "Bill" } },
 "name": "Bill Lee",
 "partyIdInfo": { "fspId": "1234", "partyIdType": "MSISDN",
  "partySubIdOrType": "RegisteredCustomer", "partyIdentifier": "16135551212"
 }
}
```


8. Create the JSON object or objects containing the desired set of header parameters, which together comprise the JWE Protected Header. In this case, its value is:


```
{"alg":"RSA-OAEP-256","enc":"A256GCM"}
```


9. Compute the Encoded Protected Header value BASE64URL(UTF8(JWE Protected Header)). In this case, its value is:


```
eyJhbGciOiJSU0EtT0FFUC0yNTYiLCJlbmMiOiJBMjU2R0NNIn0
```


10. Let the Additional Authenticated Data encryption parameter be ASCII(Base64URL(JWE Protected Header)).
11. Encrypt the plain text using the CEK, the JWE Initialization Vector, and the Additional Authenticated Data value using the specified content encryption algorithm to create the JWE Cipher text value and the JWE Authentication Tag (which is the Authentication Tag output from the encryption operation).
12. Compute the encoded cipher text value BASE64URL(JWE Cipher Text). In this case its value is:


```
BfXbxoyXcWCzL3DwG7B2P5UswlP8MPXerIkKbRR3vDLuN7lfa33puj7VICFeqG1fAlxrXgs_Nvk
ZkE4WlqGNlQ_nBS1xYknxjh7hkPVb-V-Z9ZEvLdcaHlGJrH5oEvR7RIB8TOHgVHP1brlrEptB4-
4ejXXv80cbknRJtDl_mmjaU_Na4irGrWhA3ZhXZM1aM7wtquJLIk-1ZNLadGnGPygl21sEITF8h
fPzbk7Djs45nBc5izWcoskCCNvLDU6PqOEhWe3y6GdsDiqFPB1OeZRq06ZBEfKZzAAJ0u3KZqoC
BAEVHVvt41D3ejVimTVQJs1dVL2HacvuJyVW6YugwFotZbg
```


13. Compute the encoded Authentication Tag value BASE64URL(JWE Authentication Tag). In this case its value is:


```
9GaZEDZD9wmzqVGCI-FDgQ
```


4.1.1.2 Encrypt payee.partyIdInfo.partyIdentifier

1. Determine the algorithm used to determine the CEK value. In this case, assuming it is **RSA-OAEP-256**.
2. Generate a 256-bit random CEK. In this case, the same CEK defined in section 3.1.1.1 is used., Its value is (using JSON Array notation):


```
[191 100 167 60 2 248 21 136 172 39 145 120 102 7 73 31 166 66 114 199 219
157 104 162 7 253 10 105 33 136 57 167]
```


3. Encrypt the CEK with the Payee FSP's public key represented in JSON Web Key format in section 4. In this case, its value is (using JSON Array notation):

```
[149 174 138 153 221 70 241 229 93 27 56 185 185 210 242 238 81 187 207 88
40 43 24 7 245 121 94 73 151 150 249 19 15 158 11 97 80 99 194 60 143 138
168 211 202 210 52 19 128 211 156 179 101 248 95 163 23 166 217 222 14 12
163 206 242 182 170 211 119 22 84 107 3 97 153 207 240 211 82 113 100 254 39
62 224 183 250 176 156 63 198 73 245 187 239 167 136 127 120 130 146 236 29
47 255 116 223 240 39 224 94 165 102 120 242 9 182 84 138 109 205 55 242 20
186 91 140 49 198 244 250 58 123 3 63 22 51 59 5 183 112 17 160 238 34 217
11 109 79 246 174 221 138 118 82 21 15 239 72 185 77 20 178 20 192 89 45 68
140 190 251 233 82 123 33 49 191 135 49 21 25 42 253 171 211 151 7 238 142
206 201 140 206 6 129 23 173 56 153 159 31 39 52 119 102 147 197 213 230 97
113 71 168 184 6 57 183 109 173 233 206 110 112 202 179 74 56 153 184 122
114 234 151 28 15 131 79 192 80 145 130 170 188 82 92 61 121 90 63 148 37
110 20 132 49 131]
```

Note: Although the same CEK is used for the two fields **payer** and **payee.partyIdInfo.partyIdentifier**, the encrypted CEK values may be different from each other because of the use of a random number when encrypting the CEK by JWE implementations (for example, jose4j, nimbus-jose-jwt, and so on). This depends on how the JWE is implemented in each FSP system.

4. Compute the encoded key value BASE64URL(JWE Encrypted Key). In this case, its value is:

```
la6Kmd1G8eVdGzi5udLy7lG7z1goKxgH9XleSZeW-RMPngthUGPCPI-KqNPK0jQTgNOcs2X4X6M
XptneDgyjzvK2qtN3FlRrA2GZz_DTUnFk_ic-4Lf6sJw_xkn1u--niH94gpLsHS__dN_wJ-BepW
Z48gm2VIptzTfyFLpbjDHG9Po6ewM_FjM7BbdwEaDuItkLbU_2rt2KdlIVD-9IuU0UshTAWS1Ej
L776VJ7ITG_hzEVGSr9q9OXB-6OzsmMzgaBF604mZ8fJzR3ZpPF1eZhcUeouAY5t22t6c5ucMqz
SjiZuHpy6pccD4NPwFCRgqq8Ulw9eVo_lCVuFIQxgw
```

5. Generate a random JWE Initialization Vector of the correct size for the content encryption algorithm. In this case, its value is (using JSON Array notation):

```
[86 250 136 87 147 231 201 138 65 75 164 215 147 100 136 195]
```

6. Compute the encoded Initialization Vector value BASE64URL(JWE Initialization Vector). In this case, its value is:

```
VvqIV5PnyYpBS6TXk2SIww
```

7. Get the plain text of the field **payee.partyIdInfo.partyIdentifier** of the API message as the payload to be encrypted. In this case, its value is

```
15295558888
```

8. Create the JSON object or objects containing the desired set of Header Parameters, which together comprise the JWE Protected Header. In this case, its value is:

```
{"alg":"RSA-OAEP-256","enc":"A256GCM"}
```

9. Compute the Encoded Protected Header value BASE64URL(UTF8(JWE Protected Header)). In this case, its value is:

```
eyJhbGciOiJSU0EtT0FFUC0yNTYiLCJlbmMiOiJBMjU2R0NNIn0
```

10. Let the Additional Authenticated Data encryption parameter be ASCII(Encoded Protected Header).

11. Encrypt the plain text using the CEK, the JWE Initialization Vector, and the Additional Authenticated Data value using the specified content encryption algorithm to create the JWE Cipher text value and the JWE Authentication Tag (which is the Authentication Tag output from the encryption operation).
12. Compute the encoded cipher text value BASE64URL(JWE Cipher Text). In this case its value is:

    ```
    WBQN5nLDGK26EiM
    ```

13. Compute the encoded Authentication Tag value BASE64URL(JWE Authentication Tag). In this case its value is:

    ```
    6jQVo7kmZq3jMNXfavxoXQ
    ```

4.1.2 Producing FSPIOP-Encryption

Using the given data model of the header **FSPIOP-Encryption**, get the header's value as shown below (line break and indentation are only for display purpose):


```
{
  "encryptedFields":
  [
    {
      "initializationVector":"ZWLAD6edXZg2ka3sUwQG8w",
      "fieldName":"payer",
      "encryptedKey":"FtItL5lft09UGsIqG5gyw6MS63mMeOCBtHgVAC7EFXL7lH9LxipX-rpiD4j5g-
          BJb2yfjGucUB6FTVYaHA1CU_jlhE3LceUY0JtRrAmkGX7O2RnOHtomvoDE-
          ukiL1ZbnYxX8B13foioClf21RdocteGR1cuN4OuD8HCWsLQ1A8YIY8m_X15r9zKal9_gcAC-
          SIkOKJPPpu-h-J_LNN9ngTZTVcfT5DhSU4emZypMv5JQKMB7KhIfcRnGGDpXlS-
          2kFa2if1n1iLATP5ADnJhwhw8S6SDqtLnqM21Tk-IUnPargUwnqLm5aRVGo-
          cPpM4ktoNz4M8MsuqRBrWM8ykIlw",
      "authenticationTag":"9GaZEDZD9wmzqVGCI-FDgQ",
      "protectedHeader":"eyJhbGciOiJSU0EtT0FFUC0yNTYiLCJlbmMiOiJBMjU2R0NNIn0"
    },
    {
      "initializationVector":"VvqIV5PnyYpBS6TXk2SIww",
      "fieldName":"payee.partyIdInfo.partyIdentifier",
      "encryptedKey":"la6Kmd1G8eVdGzi5udLy7lG7z1goKxgH9XleSZeW-RMPngthUGPCPI-
          KqNPK0jQTgNOcs2X4X6MXptneDgyjzvK2qtN3FlRrA2GZz_DTUnFk_ic-4Lf6sJw_xkn1u--
          niH94gpLsHS__dN_wJ-BepWZ48gm2VIptzTfyFLpbjDHG9Po6ewM_FjM7BbdwEa-
          DuItkLbU_2rt2KdlIVD-9IuU0UshTAWS1EjL776VJ7ITG_hzEVGSr9q9OXB-
          6OzsmMzgaBF604mZ8fJzR3ZpPF1eZhcUeouAY5t22t6c5ucMqzS-
          jiZuHpy6pccD4NPwFCRgqq8Ulw9eVo_lCVuFIQxgw",
      "authenticationTag":"6jQVo7kmZq3jMNXfavxoXQ",
      "protectedHeader":"eyJhbGciOiJSU0EtT0FFUC0yNTYiLCJlbmMiOiJBMjU2R0NNIn0"
    }
  ]
}
```

4.1.3 Re-produce API message with encryption

Using the cipher text of the encrypted field to replace the plain text of the corresponding field of the API message (the string with **light grey** background in the message text below), add parameter **FSPIOP-Encryption** (the string with **light grey** background in the message text below) into the HTTP header of the API message.

Note: The **FSPIOP-Encryption** parameter should be included in the JWS Protected Header for the signature of the API message, and the HTTP body of the API message below should be the JWS Payload for the signature. The signature process is out-of-scope for this document.

```
POST /quotes HTTP/1.1
Accept:application/vnd.interoperability.quotes+json;version=1.0
Content-Type:application/vnd.interoperability.quotes+json;version=1.0
Date:Tue, 23 May 2017 21:12:31 GMT
FSPIOP-Source:1234
FSPIOP-Destination:5678
Content-Length:1068
FSPIOP-Encryption: {"encryptedFields":{
      "initializationVector":"ZWLAD6edXZg2ka3sUwQG8w",
      "fieldName":"payer",
      "encryptedKey":"FtItL5lft09UGsIqG5gyw6MS63mMeOCBtHgVAC7EFXL7lH9LxipX-
rpiD4j5g-BJb2yfjGucUB6FTVYaHA1CU_jlhE3LceUY0JtRrAmkGX7O2RnOHtomvoDE-
ukiL1ZbnYxX8B13foioClf21RdocteGR1cuN4OuD8HCWsLQ1A8YIY8m_X15r9zKal9_gcACSIkOKJPPpu-
h-J_LNN9ngTZTVcfT5DhSU4emZypMv5JQKMB7KhIfcRnGGDpXlS-
2kFa2if1n1iLATP5ADnJhwhw8S6SDqtLnqM21Tk-
IUnPargUwnqLm5aRVGocPpM4ktoNz4M8MsuqRBrWM8ykIlw",
      "authenticationTag":"9GaZEDZD9wmzqVGCI-FDgQ",
      "protectedHeader":"eyJhbGciOiJSU0EtT0FFUC0yNTYiLCJlbmMiOiJBMjU2R0NNIn0"
    }, {
      "initializationVector":"VvqIV5PnyYpBS6TXk2SIww",
      "fieldName":"payee.partyIdInfo.partyIdentifier",
      "encryptedKey":"la6Kmd1G8eVdGzi5udLy7lG7z1goKxgH9XleSZeW-RMPngthUGPCPI-
KqNPK0jQTgNOcs2X4X6MXptneDgyjzvK2qtN3FlRrA2GZz_DTUnFk_ic-4Lf6sJw_xkn1u--
niH94gpLsHS__dN_wJ-
BepWZ48gm2VIptzTfyFlpbjDHG9Po6ewM_FjM7BbdwEaDuItkLbU_2rt2KdlIVD-
9IuU0UshTAWS1EjL776VJ7ITG_hzEVGSr9q9OXB-
6OzsmMzgaBF604mZ8fJzR3ZpPF1eZhcUeouAY5t22t6c5ucMqzSjiZuHpy6pccD4NPwFCRgqq8Ulw9eVo_
lCVuFIQxgw",
      "authenticationTag":"6jQVo7kmZq3jMNXfavxoXQ",
      "protectedHeader":"eyJhbGciOiJSU0EtT0FFUC0yNTYiLCJlbmMiOiJBMjU2R0NNIn0"
    }
  ]
}

{"amount":{"amount":"150","currency":"USD"},"transactionType":{"scenario":"TRANSFE
R","initiator":"PAYER","subScenario":"P2P Transfer across MM
systems","initiatorType":"CONSUMER"},"transactionId":"36629a51-393a-4e3c-b347-
c2cb57e1e1fc","quoteId":"59e331fa-345f-4554-aac8-
fcd8833f7d50","payer":"BfXbcoyXcWCzL3DwG7B2P5UswlP8MPXerIkKbRR3vDLuN7lfa33puj7VICF
eqG1fAlxrXgs_NvkZkE4WlqGNlQ_nBS1xYknxjh7hkPVb-B-
Z9ZEvLdcaHlGJrH5oEvR7RIB8TOHgVHP1brlrEptB4-
4ejXXv80cbknRJtDl_mmjaU_Na4irGrWhA3ZhXZM1aM7wtquJLIk-
1ZNLadGnGPygl21sEITF8hfPzbk7Djs45nBc5izWcoskCCNvLDU6PqOEhWe3y6GdsDiqFPB10eZRq06ZBE
fKZzAAJ0u3KZqoOBAEVHVvt41D3ejVimTVQJs1dVL2HacvuJyVW6ugwFotZbg","expiration":"2017-
05-24T08:40:00.000-
04:00","payee":{"partyIdInfo":{"fspId":"5678","partyIdType":"MSISDN","partyIdentif
ier":"WBQN5nLDGK26EiM"}},"fees":{"amount":"1.5","currency":"USD"},"extensionList":
{"extension":[{"value":"value1","key":"key1"},{"value":"value2","key":"key2"},{"va
lue":"value3","key":"key3"}]},"note":"this is a sample for POST
/quotes","geoCode":{"longitude":"125.520001","latitude":"57.323889"},"amountType":
"RECEIVE"}
```

4.2 Decryption Example

In this example, the Payee FSP receives the POST /quotes API message from Payer FSP. The message is described in Section 3.1.3. If the Payee FSP detects that the HTTP header parameter **FSPIOP-Encryption** is present in the message, then the Payee FSP knows that some fields were encrypted by the Payer FSP. The Payee FSP then performs the following steps to decrypt the encrypted fields.

4.2.1 Parse FSPIOP-Encryption

The Payee FSP verifies that the value of **FSPIOP-Encryption** is a UTF-8-encoded representation of a valid JSON object conforming to RFC 7159. The FSP then parses the HTTP Header parameter **FSPIOP-Encryption** to get encrypted fields information, including field name, JWE Protected Header, JWE Encrypted Key, JWE Initialization Vector, and JWE Authentication Tag for each field.

4.2.2 Decrypt the Encrypted Fields

In this case, the Payee FSP gets two fields **payer** and **payee.partyIdInfo.partyIdentifier** from the HTTP header **FSPIOP-Encryption**. Then the Payee FSP decrypts the two fields one-by-one.

4.2.2.1 Decrypt payer

The Payer FSP performs the following steps to decrypt the field **payer** in the **POST /quotes** API message.

1. Get the encoded BASE64RUL(JWE Protected Header) from the parsed **FSPIOP-Encryption** for the field **payer**. In this case its value is:

   ```
   eyJhbGciOiJSU0EtT0FFUC0yNTYiLCJlbmMiOiJBMjU2R0NNIn0
   ```

2. Decode the encoded JWE Protected Header. In this case its value is:

   ```
   {"alg":"RSA-OAEP-256","enc":"A256GCM"}
   ```

3. Check that the decoded value of JWE Protected Header is a UTF-8-encoded representation of a completely valid JSON object conforming to JSON Data Interchange Format (RFC 7159), and that the parameters in the JWE Protected Header can process all fields that are required to support the JWE specification.
4. Get the encoded BASE64URL(JWE Encrypted Key). In this case its value is:

   ```
   FtItL5lft09UGsIqG5gyw6MS63mMeOCBtHgVAC7EFXL7lH9LxipX-rpiD4j5g-BJb2yfjGucUB6
   FTVYaHA1CU_jlhE3LceUY0JtRrAmkGX7O2RnOHtomvoDE-ukiL1ZbnYxX8B13foioClf21Rdoct
   eGR1cuN4OuD8HCWsLQ1A8YIY8m_X15r9zKal9_gcACSIkOKJPPpu-h-J_LNN9ngTZTVcfT5DhSU
   4emZypMv5JQKMB7KhIfcRnGGDpXlS-2kFa2if1n1iLATP5ADnJhwhw8S6SDqtLnqM21Tk-IUnPa
   rgUwnqLm5aRVGocPpM4ktoNz4M8MsuqRBrWM8ykIlw
   ```

5. Decode the encoded JWE Encrypted Key. In this case its value is as follows (using JSON Array notation):

```
[22 210 45 47 153 95 183 79 84 26 194 42 27 152 50 195 163 18 235 121 140
120 224 129 180 120 21 0 46 196 21 114 251 148 127 75 198 42 87 250 186 98
15 136 249 131 224 73 111 108 159 140 107 156 80 30 133 77 86 26 28 13 66 83
248 229 132 77 203 113 229 24 208 155 81 172 9 164 25 126 206 217 25 206 30
218 38 190 128 196 250 233 34 47 86 91 157 140 87 240 29 119 126 136 168 10
87 246 213 23 104 114 215 134 71 87 46 55 131 174 15 193 194 90 194 208 212
15 24 33 143 38 253 125 121 175 220 202 106 95 127 129 192 2 72 137 14 40
147 207 166 239 161 248 159 203 52 223 103 129 54 83 85 199 211 228 56 82 83
135 166 103 42 76 191 146 80 40 192 123 42 18 31 113 25 198 24 58 87 149 47
182 144 86 182 137 253 103 214 34 192 76 254 64 14 114 97 194 28 60 75 164
131 170 210 231 168 205 181 78 79 136 82 115 218 174 5 48 158 162 230 229
164 85 26 135 15 164 206 36 182 131 115 224 207 12 178 234 145 6 181 140 243
41 8 151]
```

6. Decrypt the JWE Encrypted Key using the specified algorithm **RSA-OAEP-256** with the Payee FSP's private key to get the CEK. In this case the decrypted CEK is (using JSON Array notation):

```
[191 100 167 60 2 248 21 136 172 39 145 120 102 7 73 31 166 66 114 199 219
157 104 162 7 253 10 105 33 136 57 167]
```

7. Get the encoded BASE64URL(JWE Initialization Vector). Its value is:

```
ZWLAD6edXZg2ka3sUwQG8w
```

8. Decode the encoded JWE Initialization Vector. In this case, its value is (using JSON Array notation):

```
[101 98 192 15 167 157 93 152 54 145 173 236 83 4 6 243]
```

9. Get the value of the field **payer** from the API message as the encoded BASE64URL(JWE Cipher Text) to be decrypted. In this case, its value is

```
BfXbxoyXcWCzL3DwG7B2P5UswlP8MPXerIkKbRR3vDLuN7lfa33puj7VICFeqG1fAlxrXgs_Nvk
ZkE4WlqGNlQ_nBS1xYknxjh7hkPVb-V-Z9ZEvLdcaHlGJrH5oEvR7RIB8TOHgVHP1brlrEptB4-
4ejXXv80cbknRJtDl_mmjaU_Na4irGrWhA3ZhXZM1aM7wtquJLIk-1ZNLadGnGPygl21sEITF8h
fPzbk7Djs45nBc5izWcoskCCNvLDU6PqOEhWe3y6GdsDiqFPB1OeZRq06ZBEfKZzAAJ0u3KZqoC
BAEVHVvt41D3ejVimTVQJs1dVL2HacvuJyVW6YugwFotZbg
```

10. Get the encoded Authentication Tag value BASE64URL(JWE Authentication Tag). In this case its value is:

```
9GaZEDZD9wmzqVGCI-FDgQ
```

11. Decrypt the cipher text using the CEK, the JWE Initialization Vector, and the Additional Authenticated Data value using the specified content encryption algorithm to decrypt the JWE Cipher text. In this case, the plain text is

```
{
 "personalInfo": {
  "dateOfBirth": "1986-02-14",
  "complexName": { "middleName": "Ben", "lastName": "Lee",
  "firstName": "Bill" } },
 "name": "Bill Lee",
 "partyIdInfo": {
  "fspId": "1234", "partyIdType": "MSISDN",
  "partySubIdOrType": "RegisteredCustomer", "partyIdentifier":
"16135551212"
 }
}
```


12. Verify that the plaintext is a UTF-8-encoded representation of a completely valid JSON object conforming to RFC 7159, and the content matches the data mode definition for the **payer**.

4.2.2.2 Decrypt payee.partyIdInfo.partyIdentifier

The Payer FSP performs the following steps to decrypt the field **payee.partyIdInfo.partyIdentifier** in the **POST /quotes** API message.

1. Get the encoded BASE64RUL(JWE Protected Header) from the parsed **FSPIOP-Encryption** for the field **payee.partyIdInfo.partyIdentifier**. In this case its value is:

   eyJhbGciOiJSU0EtT0FFUC0yNTYiLCJlbmMiOiJBMjU2R0NNIn0

2. Decode the encoded JWE Protected Header. In this case its value is:

   {"alg":"RSA-OAEP-256","enc":"A256GCM"}

3. Verify that the decoded value of JWE Protected Header is a UTF-8-encoded representation of a completely valid JSON object conforming to RFC 7159, and that the parameters in the JWE Protected Header understand and can process all fields that are required to support the JWE specification.
4. Get the encoded BASE64URL(JWE Encrypted Key). In this case its value is:

   la6Kmd1G8eVdGzi5udLy7lG7z1goKxgH9XleSZeW-RMPngthUGPCPI-KqNPK0jQTgNOcs2X4X6M
XptneDgyjzvK2qtN3FlRrA2GZz_DTUnFk_ic-4Lf6sJw_xkn1u--niH94gpLsHS__dN_wJ-BepW
Z48gm2VIptzTfyFLpbjDHG9Po6ewM_FjM7BbdwEaDuItkLbU_2rt2KdlIVD-9IuU0UshTAWS1Ej
L776VJ7ITG_hzEVGSr9q9OXB-6OzsmMzgaBF604mZ8fJzR3ZpPF1eZhcUeouAY5t22t6c5ucMqz
SjiZuHpy6pccD4NPwFCRgqq8Ulw9eVo_lCVuFIQxgw

5. Decode the encoded JWE Encrypted Key. In this case its value is (using JSON Array notation):

```
[149 174 138 153 221 70 241 229 93 27 56 185 185 210 242 238 81 187 207 88
40 43 24 7 245 121 94 73 151 150 249 19 15 158 11 97 80 99 194 60 143 138
168 211 202 210 52 19 128 211 156 179 101 248 95 163 23 166 217 222 14 12
163 206 242 182 170 211 119 22 84 107 3 97 153 207 240 211 82 113 100 254 39
62 224 183 250 176 156 63 198 73 245 187 239 167 136 127 120 130 146 236 29
47 255 116 223 240 39 224 94 165 102 120 242 9 182 84 138 109 205 55 242 20
186 91 140 49 198 244 250 58 123 3 63 22 51 59 5 183 112 17 160 238 34 217
11 109 79 246 174 221 138 118 82 21 15 239 72 185 77 20 178 20 192 89 45 68
140 190 251 233 82 123 33 49 191 135 49 21 25 42 253 171 211 151 7 238 142
206 201 140 206 6 129 23 173 56 153 159 31 39 52 119 102 147 197 213 230 97
113 71 168 184 6 57 183 109 173 233 206 110 112 202 179 74 56 153 184 122
114 234 151 28 15 131 79 192 80 145 130 170 188 82 92 61 121 90 63 148 37
110 20 132 49 131]
```

6. Decrypt the JWE Encrypted Key using the specified algorithm **RSA-OAEP-256** with the Payee FSP's private key to get the CEK. In this case the decrypted CEK is (using JSON Array notation):

```
[191 100 167 60 2 248 21 136 172 39 145 120 102 7 73 31 166 66 114 199 219
157 104 162 7 253 10 105 33 136 57 167]
```

7. Get the encoded BASE64URL(JWE Initialization Vector). Its value is:

```
VvqIV5PnyYpBS6TXk2SIww
```

8. Decode the encoded JWE Initialization Vector. In this case, its value is (using JSON Array notation):

```
[86 250 136 87 147 231 201 138 65 75 164 215 147 100 136 195]
```

9. Get the value of the field **payee.partyIdInfo.partyIdentifier** from the API message as the encoded BASE64URL(JWE Cipher Text) to be decrypted. In this case, its value is:

```
WBQN5nLDGK26EiM
```

10. Get the encoded Authentication Tag value BASE64URL(JWE Authentication Tag). In this case its value is:

```
6jQVo7kmZq3jMNXfavxoXQ
```

11. Decrypt the cipher text using the CEK, the JWE Initialization Vector, and the Additional Authenticated Data value using the specified content encryption algorithm to decrypt the JWE Cipher text. In this case, the plain text is

```
15295558888
```

12. Verify that the plain text is a valid **partyIdentifier** value.