Table of Contents

Version History

Version	Date	Changes
2.0	2nd May 2025	Added FAQs for Lookup V2.0
1.2	23rd Jan 2025	Added FAQs and formatted the document
	13th Sept 2024	Key-wise explanation of the subscribe payload
	21st Aug 2024	Refined FAQs, removed invalid questions
	16th Jun 2024	Detailed explanation of signing and verification errors
1.1	7th Jun 2024	Refined FAQs, removed invalid questions
	4th Mar 2024	FAQs added based on errors encountered while hitting the subscribe API
	4th Jan 2024	General onboarding process questions added

Onboarding related FAQs

1. **What is to be used as a subscriber_id: http domain or some text id in a particular format?**

   In the **Open Network for Digital Commerce (ONDC)**, the **`subscriber_id`** is a **unique identifier** associated with a network participant (such as a seller app, buyer app, logistics provider) during the onboarding process. It is typically the **domain name** or endpoint that uniquely identifies that participant within the ONDC network.

   The subscriber_id's default value will be the Fully Qualified Domain Name of the subscriber unless specified otherwise as a network policy. (usually a domain format like `ondc.seller.example`)

   - It is **used in API requests** to identify who is making the call, and appears in the `context` object of ONDC payloads.
   - It is **linked with authentication** — used along with keys like `unique_key_id` (UKID) to sign and verify API requests.
   - It is part of the **ONDC Registry's Subscriber object** which stores details of the participant.
   - **Buyer Apps (`bap_id`)** and **Seller Apps (`bpp_id`)** in payloads are both `subscriber_id`s.

```
{
  "context": {
    "domain": "nic2004:52110",
    "bap_id": "buyerapp.example.com",  // subscriber_id of buyer app
    "bpp_id": "ondc.seller.example",   // subscriber_id of seller app
    "unique_key_id": "1234567890",
    "timestamp": "2023-05-15T10:00:00Z"
  }
}
```

2. **What is the difference between subscriber id and subscriber url?**

   The ONDC network utilises two identifiers for subscribers: a subscriber ID and a subscriber URL. While both serve the purpose of identifying participants within the network, they differ in their nature and function.

   Subscriber_id: A unique, identifier assigned to each subscriber within the ONDC network

   Subscriber_url: A publicly accessible URL that points to the subscriber's information and endpoints hosted within the ONDC network. Make sure subscriber_url includes subscriber id.

   Example: Suppose a network participant has the domain **ondc.seller.example**, their subscriber ID becomes "ondc.seller.example" and their subscriber URL can be https://ondc.seller.example/ or any subroute with the domain `ondc.seller.example`.

Field	Meaning	Example
subscriber_id	Unique **identifier** of the participant in the network (usually a domain). Used for identity and authentication.	`sellerapp.example.com`
subscriber_url	The actual **base URL** (endpoint) where the participant's APIs are hosted and can be called.	`https://api.sellerapp.example.com`

3. Subscriber ID not whitelisted by ONDC

```
{
  "message": {
    "ack": {
      "status": "NACK"
    }
  },
  "error": {
    "type": "POLICY-ERROR",
    "code": "144",
    "path": null,
    "message": "Subscriber Id is not whitelisted in our database"
  }
}
```

Get your subscriber_id whitelisted/approved by ONDC. To do that

For Staging: Begin by registering on the ONDC Portal and ensuring your profile is fully completed 100%. Once this is done, you can proceed to request domain whitelisting on portal : Home > Environment Access Request.

For Preprod: Self Declaration is required to be submitted through the portal and then the whitelisting request needs to be raised through the portal itself.

For Prod: After your preprod logs are verified and the QA process is completed, you're ready to go to the production environment. You may then raise a whitelisting request like all the other environments and proceed.

4. Domain Verification Failed

```
{
  "message": {
```

```
    "ack": {
      "status": "NACK"
    }
  },
  "error": {
    "type": "POLICY-ERROR",
    "code": "132",
    "path": null,
    "message": "Domain verification is failed"
  }
}
```

- A domain verification error occurs if the `request_id` in your subscribe payload does not match the `request_id` signed with your private key. If the `request_id` matches, the issue may be due to a mismatch in the signing key pair.
- Please use the same **request_id** in the subscribe utility code, the one that has been used in the subscribe **payload**.
- If you have a custom implementation, kindly use the python, java, php or node utility to understand the implementation in your specific language.

5. Ondc-site-verification file is not found

```
{
  "message": {
    "ack": {
      "status": "NACK"
    }
  },
  "error": {
    "type": "POLICY-ERROR",
    "code": "132",
    "path": null,
    "message": "ondc-site-verification file is not found"
  }
}
```

- Create ondc-site-verification.html and host at domain root level.
  - Here is an example https://<subscriber_id>/ondc-site-verification.html
- Use the correct html formatting provided in the documentation.

- Whitelist all the IPs if you have firewall or IP blocking in your sever:
  34.131.40.9 34.131.211.247 34.131.201.63 34.131.180.63 34.131.78.219
  34.93.10.146 35.200.143.183 34.93.118.120 35.200.183.209 34.93.102.253
  35.200.232.136 34.100.170.176 107.178.231.181/32

6. OCSP Failed

```
{
  "message": {
    "ack": {
      "status": "NACK"
    }
  },
  "error": {
    "type": "POLICY-ERROR",
    "code": "132",
    "path": null,
    "message": "OCSP failed"
  }
}
```

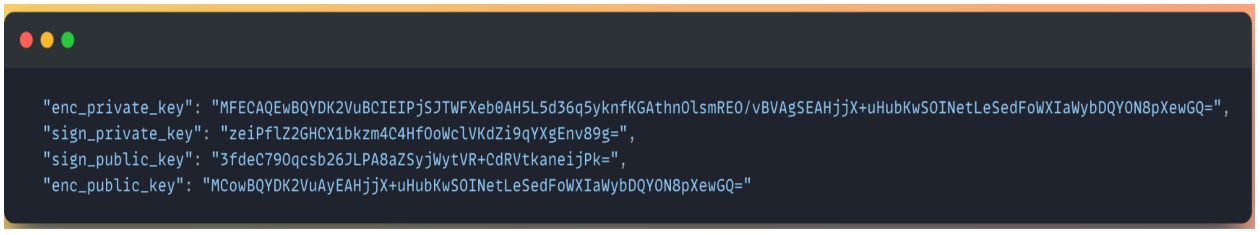
You need to have a valid SSL certificate for your domain. This will be used while performing Online Certificate Status Protocol (OCSP) validation. Please obtain a valid SSL certificate that supports OCSP validation for the purchased domain.

7. How does a subscriber get Registry's public keys to decrypt information on on_subscribe?

Registry's public keys are provided in this document at point no. 6

8. How to generate signing and encryption key pairs?

You can you utilities to generate signing and encryption keys
Key generation utilities



9. Code example for the export format of the public key?

There are fundamentally 2 formats in which Ed25519 & X25519 Keys are exported.

1. Raw (simply base64 encoded of the raw bytes)

2. Pem (Typically used by JCA/JCE here it is bas64 encoded of a X509encoded public key)

If you do an online key pair generation for Ed25519, you will generally get it in raw format. Both these formats are interconvertible. However, since raw is more prevalent, we could use the same for our communication with the registry.

10. **Which algorithm will BAP use for decrypting Challenge String?**
    In general, X25519 Key Agreement algorithm is used to derive a symmetric AES key for encryption and decryption between any two parties.
    We will use it between registry and participants to encrypt and decrypt challenges.

11. **Is subscriber_id unique across the network for a BAP or is it the unique_key_id?**
    The `subscriber_id` and `unique_key_id` need to be unique throughout the network for each Network Participant. Both need to be unique for each onboarded NP.

12. **Should we validate the bpp_id or bap_id in the context against the Authorization header?**
    Yes, bap_id and bpp_id need to be validated. When BPP receives a request, If bap_id in context does not match bap_id in signature, the BPP must return a 401 unauthorised response.

13. **What cipher is used to encrypt a challenge? We need to know it to be able to decrypt the challenge?**
    The encryption would be AES but it is derived using the key agreement mechanism of x25519.

    Every participant registers with the registry 2 public keys. An ed25519 and x25519 and would keep the private key with them

    Registry would use its private and public encryption key to derive the aes
    key for encrypting the challenge.

    Participants will use their own Private key and registry's public key to derive the

    AES key that was used for encryption. And decrypt the challenge using this key.

14. **If subscriber A needs to get the public key of Subscriber B when doing a transaction, then does it need to call lookup API to check whether Subscriber B exists?**

    Yes, Subscriber A needs to call the lookup API to get the public key of Subscriber B and verify if Subscriber B exists in the registry. The lookup API only returns subscriber records that have status = "SUBSCRIBED" or “WHITELISTED”.

    Here's a detailed explanation:

    1. Why Lookup? ONDC relies on public-key cryptography for secure communication. When Subscriber A wants to interact with Subscriber B, it needs Subscriber B's public key to verify signatures and ensure the message is from a trusted source.

2. Lookup API: The ONDC registry provides a lookup API that allows subscribers to fetch the public key of other subscribers using their `subscriber_id`.

3. Active Status Check: The lookup API's response includes a "status" field indicating the subscriber's status. Subscriber A should check if Subscriber B's status is "SUBSCRIBED". This ensures that Subscriber B is active on the network and the communication will be successful.

4. Postman curl Example:

```
curl --location 'https://prod.registry.ondc.org/v2.0/lookup' \
--header 'Content-Type: application/json' \
--header 'Authorization: Signature keyId="example-bap.com|bap1234|ed25519", algorithm="ed25519", created="<timestamp>", expires="<timestamp>", headers="(created)(expires)digest", signature="<signature>"' \
--data '{
  "country": "IND",
  "domain": "ONDC:RET10"
}'
```

15. **Does the lookup API need to be called before calling any APIs?**

The Lookup API is not mandatory to call before every API request. Instead, it serves as a utility that subscribers can use whenever they need to resolve information about another participant—typically to fetch their public keys, network roles, or endpoint details from the ONDC Registry.

This is particularly important for verifying the signatures in data payloads exchanged over the network. For example, you may use the Lookup API to fetch a seller app's signing public key when validating a signed message.

Recommendation for efficiency:

To reduce frequent calls to the Lookup API, participants can cache the public registry dump periodically. This dump includes information about all network participants and their public signing keys. You can then rely on your local cache for most lookups, and use the `v2.0/lookup` API only as a fallback mechanism when the cache is outdated or a new participant needs to be resolved.

16. **Why is the subscriber_url showing as null in the registry after subscribing?**

The subscriber_url appears as null when the participant is only whitelisted. It gets updated in the registry once you complete the onboarding by calling the subscribe API with your subscriber_url. The value provided in that API call will be saved and reflected in the registry records.

Key-wise explanation to subscribe payload

1. ops_no

   ops_no : 1 - Buyer App Registration
   ops_no : 2 - Seller App Registration
   ops_no : 4 - Buyer & Seller App Registration

   Note: ops_no 3 & 5 is deprecated as the feature of Seller On Record (SOR) in registry is obsolete.

2. request_id and unique_key_id

   The `request_id` and `unique_key_id` are distinct random alphanumeric strings or UUIDs generated by the network participant.

3. callback_url

   The callback_url in ONDC is a relative path provided during the subscription process. It tells ONDC where to send verification callbacks like on_subscribe.
   The final callback endpoint URL is constructed as:
   https://<subscriber_url>/<callback_url>/on_subscribe

   If:
   subscriber_id = www.sellerapp.example.com
   callback_url = /api/ondc

   Then the final on_subscribe callback URL will be:
   https://www.sellerapp.example.com/api/ondc/on_subscribe

4. key_pairs

   The network participant must include the `signing_public_key` and `encryption_public_key` in the subscribe payload, which were generated during the key generation process. Additionally, they need to specify the keys' validity period using the `valid_from` and `valid_until` fields.

5. network_participant

   The network participant includes a `domains` object specifying the domains they wish to subscribe to. For instance, if the network participant is whitelisted for `ONDC:RET10` and `ONDC:RET11`, they need to include two objects in the `network_participant` array as shown below:

   ```json
   "network_participant": [
     {
       "subscriber_url": "/ret10",
   ```

```
    "domain": "ONDC:RET10",
    "type": "sellerApp",
    "msn": false,
    "city_code": ["*"]
  },
  {
    "subscriber_url": "/ret11",
    "domain": "ONDC:RET11",
    "type": "sellerApp",
    "msn": false,
    "city_code": ["*"]
  }
]
```

6. subscriber_url

Essentially, this field represents the **relative path** to where your transactional API endpoints are hosted. For example, if your endpoint is subscriber_id/api/v1/search, then the subscriber_url in the subscriber payload should be set to /api/v1.
In the JSON, the subscriber_url is defined for each domain, allowing different relative paths based on domain-specific requirements. For instance, ONDC:RET10 might have a subscriber_url like /ret10, whereas ONDC:RET11 could use /ret11.

Alternatively, you can adopt an API versioning approach — such as /api/v1 — as the subscriber_url. If you're hosting your APIs at the root level, then / would be the value used in the payload.

Signature & Verification FAQs

There are commonly 6 types of error messages:

1. Auth header not found

```
// Auth Header not found
{
  "message": {
        "ack": {
            "status": "NACK"
        }
    },
    "error": {
        "type": "Gateway",
        "code": "10001",
        "message": "Throwable: The auth header not found"
    }
}
```

Reason: Making API requests without setting up authorization header.
Resolution: To resolve this, ensure that you generate the authorization header as per the ONDC protocol. This typically involves signing the request payload using your private key, constructing the header with the appropriate `Authorization` format, and including any required timestamps or identifiers. You can follow the step-by-step instructions provided in the documentation.
Alternatively, you may use the ONDC header generation utility to simplify this process.

2. Authentication failed

```
// Authentication failed
{
   "message": {
        "ack": {
            "status": "NACK"
        }
    },
    "error": {
        "type": "Gateway",
        "code": "10001",
        "message": "Throwable: Authentication failed"
    }
}
```

Reason: The unique key ID assigned at the time of subscription does not match the unique key ID used in the header creation
Additional Possible Reason: The `subscriber_id` in the header may not exist in the registry lookup.

Resolution: Kindly ensure you are using an existing `subscriber_id` within the network. Retrieve the correct unique key ID by calling the lookup API with your subscriber ID. Then, use the correct `ukId` and `subscriber_id` in the header to generate the authorization header. This will resolve the issue.

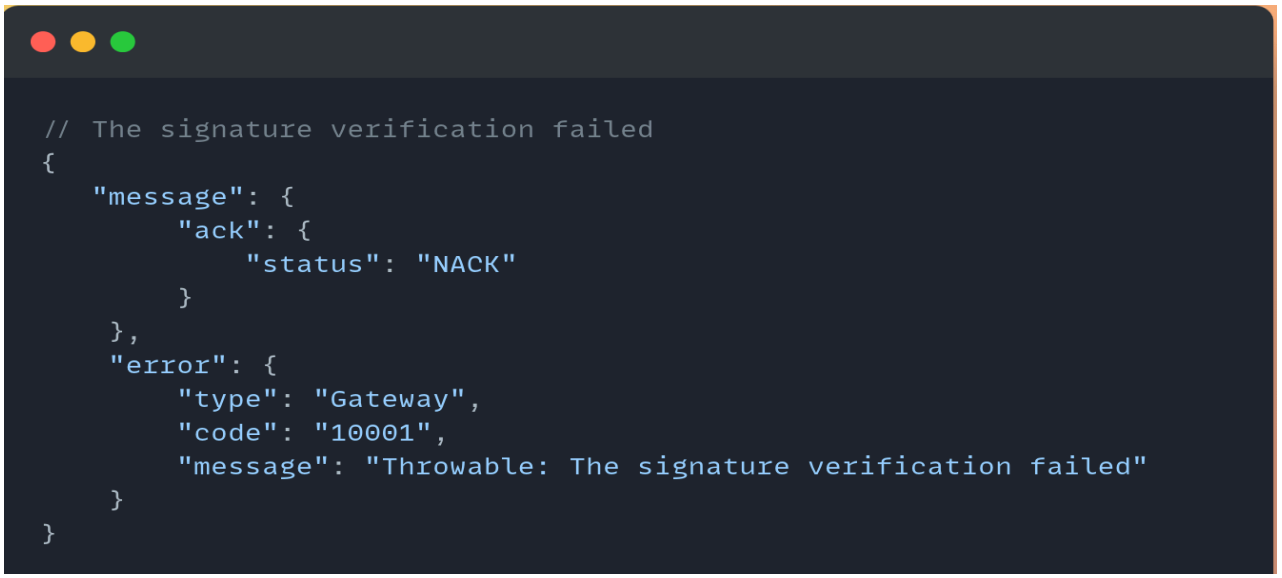
3. Invalid Signature

```
// Invalid Signature
{
   "message": {
        "ack": {
            "status": "NACK"
        }
    },
    "error": {
        "type": "Gateway",
        "code": "10001",
        "message": "Throwable: Invalid Signature"
    }
}
```

Reason: There might be problems when signatures are not encrypted using the same private key, especially since its corresponding public key has been shared with the ONDC network during the subscription process.

Resolution: Use the private key to create a signature. This private key matches the public key that was previously shared with the ONDC network when you subscribed.

4. Signature Verification failed



```
// The signature verification failed
{
   "message": {
        "ack": {
            "status": "NACK"
        }
    },
    "error": {
        "type": "Gateway",
        "code": "10001",
        "message": "Throwable: The signature verification failed"
    }
}
```

Reason: When decrypted signature are not matched with request body payload

Resolution: During the signature generation process, minify the payload. Similarly, before making your API call, minify the request body. This is crucial because tools like VS Code may automatically format your code, introducing extra spaces and commas, which can result in signature verification errors.

5. Signature is forged

Reason: If the signature is not included in the request header adhering to the specified format, it could be due to alterations in the positioning of key-value pairs or the addition of extra spaces.

Resolution: Ensure your Authorization header adheres to the specified format mentioned in this document.

6. Algorithm Mismatched

```
// Algorithm mismatched
{
  "message": {
        "ack": {
            "status": "NACK"
        }
    },
    "error": {
        "type": "Gateway",
        "code": "10001",
        "message": "Throwable: There is mismatch in the algorithm"
    }
}
```

Reason: This issue may arise during the decryption process, possibly because the encryption was not performed as per the ONDC defined method.

Resolution: For creating the Authorization header, use the ONDC utility function exactly as it is provided. However, if you prefer not to use the utility directly, you can employ the same functions and libraries utilised within the ONDC utility to generate headers. In this scenario, ensure that the encryption and decryption algorithms remain consistent. You can also refer to this document.

For computing the digest of the request body, the hashing function will use the **BLAKE-512 (2b) hashing algorithm**. Then to digitally sign the singing string, the subscribers should use the "**ed25519**" **algorithm** for signature.

7. I am facing a Signature verification issue with the Reference apps or Mock servers. How do I resolve this?

Points to consider while generating signatures:

- One of the plausible reasons can be that you aren't sending the same payload as you have used for the generating signature.
    - For example, you have generated the signature with minify json. But when you are making a http request you have or your editor has beautified the payload.

- Secondly, make sure there are no escape characters added in the signature instead of space.
- After generating the signature, changing the context_timestamp to current time while making http request will receive NACK for Signature verification. It is not accepted if you make any change after generating the signature.

- Furthermore, You have added a wrong subscriber_id or unique_key while generating the signature.

- Make sure you put the signature in the **Header** as an **Authorization key** not in Bearer Token.
- While generating a signature through an endpoint as shown in the image below. It returns as json (stringified version) which means addition of escape characters. It is crucial to ensure the signature remains intact and free from any alterations.

```
app.post("/generatesignature", async (req, res) => {
  const authorizationHeader = await createAuthorizationHeader({
    body: JSON.stringify(body),
    signInPrivateKey:
      "your signin private key",
    subscriberId: "your subscriber id",
    subscriberUniqueKeyId: "your unique key id",
  });

  res.json(authorizationHeader);
});
```

FAQs for ONDC Registry Lookup v2.0 API

Q: What is Lookup v2.0 in ONDC?

A: Lookup v2.0 is an ONDC Registry API that allows participants (BAPs, BPPs, etc.) to securely query network details (e.g., seller/buyer endpoints, public keys and unique key ID) using signed requests.

Q: How do I access Lookup v2.0?

A: Use these endpoints:

- Staging: https://staging.registry.ondc.org/v2.0/lookup
- Pre-Prod: https://preprod.registry.ondc.org/v2.0/lookup
- Production: https://prod.registry.ondc.org/v2.0/lookup

Q: What authentication is required?

A: Requests must include:

- Authorization header with a signature (Ed25519 algorithm).
- created and expires timestamps (ISO8601 UTC format).
- digest (Blake-2b hash of the request body).

Example:

```
curl --location 'https://prod.registry.ondc.org/v2.0/lookup' \
--header 'Authorization: Signature keyId="example-bap.com|bap1234|ed25519",
algorithm="ed25519", created="2024-05-04T12:00:00Z", expires="2024-05-04T12:05:00Z",
headers="(created)(expires)digest", signature="<base64_signature>"' \
--data '{"country": "IND", "domain": "ONDC:RET10"}'
```

Q: How to Generate the Signature

A: To generate the signature, follow the standard signing and verification process outlined below:

1. Refer to the detailed Signing and Verification Guide.
2. Alternative Option: Use Provided Utilities

You can skip the manual process and use the ready-to-use utilities provided by us in multiple programming languages.

Q: What parameters are mandatory in the request body?

A: This API doesn't have fixed mandatory or optional parameters.
Instead, the user **must send at least any two keys** from the below list:

- subscriber_id
- country
- ukId
- city
- domain
- type (BAP or BPP)

If fewer than two keys are provided, the API will throw an error.

Q: What's the difference between v1.0 and v2.0?

A:

- In **v1.0**, requests didn't require an authorization header.

- In **v2.0**, **all requests must be signed** and include an **Authorization header**.
  You sign the payload (e.g., lookup) using the same **private key** used during onboarding.This enforces stronger security by validating sender authenticity and payload integrity.

Q: What if my request fails with 401 Unauthorized?

A:

- Verify the signature format and timestamps (created/expires).
- Ensure the digest matches the request body.
- Check if the Ed25519 public key is registered with ONDC.

Q: Where can I find sample code for signing?

A: Refer to the Reference Utility (GitHub) for signing and verification mechanism.

Q: What is the purpose of the ONDC Registry lookup?
A: It allows a participant to verify another participant's metadata—like domain, signing public key, and endpoint—before processing the request.

Q: What data is returned in a lookup response?
A: The response contains:

- subscriber_id
- domain
- signing_public_key
- callback_urls
- city and country codes
- status (SUBSCRIBED, UNSUBSCRIBED, etc.)

Q: When should lookup be performed?
A: Before processing any signed request, the receiving participant should perform a lookup to validate:

1. Whether the sender is a valid ONDC participant
2. That the public key used for signature verification is current

Q:Can lookup responses be cached?

A: Yes, caching **lookup responses** can help **optimize performance**. However, it's important to **refresh the cache periodically**—for example, every few hours—or **in real time** for critical operations such as **order confirmations**, to ensure data remains accurate and up to date.

Additional Resources

- Signing and Verification Process
- Reference Utility - Signing
- Lookup API Swagger

Common Errors

Here are some common errors you might see:

1. Auth header not found

```
{
    "message": {
        "ack": {
            "status": "NACK"
        }
    },
    "error": {
        "code": "1020",
        "message": "The auth header not found"
    }
}
```

Reason: Making API requests without setting up authorization header.
Resolution: To resolve this, ensure that you generate the authorization header as per the ONDC protocol. This typically involves signing the request payload using your private key, constructing the header with the appropriate `Authorization` format, and including any required timestamps or identifiers. You can follow the step-by-step instructions provided in the documentation.
Alternatively, you may use the ONDC header generation utility to simplify this process.

2. Subscriber not found

```
{
    "message": {
        "ack": {
            "status": "NACK"
        }
    },
    "error": {
        "code": "1000",
        "message": "Subscriber not found"
    }
}
```

Reason: Invalid format of the signature, `subscriber_id` not found, presence of backslashes or escape characters in the header, or the `subscriber_id` used while generating the header does not exist in the registry records.
Resolution: Ensure that the header format strictly matches the expected structure (as shown below), without any backslashes, escape characters, or unintended formatting. Additionally, verify that the `subscriber_id` used exists in the registry as a **SUBSCRIBED** one.

Sample format of header:

```
Signature
keyId="buyer-app.ondc.org|207|ed25519",algorithm="ed25519",created="1641287875
```

```
",expires="1641291475",headers="(created)                                    (expires)
digest",signature="fKQWvXhln4UdyZdL87ViXQObdBme0dHnsclD2LvvnHoNxIgcvAwUZOmwAnH
5QKi9Upg5tRaxpoGhCFGHD+d+Bw=="
```

3. There is mismatch in the algorithm

```
{
    "message": {
        "ack": {
            "status": "NACK"
        }
    },
    "error": {
        "code": "1035",
        "message": "The request has expired"
    }
}
```

```
{
    "message": {
        "ack": {
            "status": "NACK"
        }
    },
    "error": {
        "code": "1030",
        "message": "There is mismatch in the algorithm"
    }
}
```

Reason: The error indicates an issue with the highlighted part of the header — possibly due to invalid format, extra characters, or slashes.
Resolution: Check that the header format matches the expected structure (as shown below) and ensure there are no backslashes or extra characters.

Signature keyId="buyer-app.ondc.org|207|ed25519",algorithm="ed25519",created="1641287875",expires="1641291475",headers="(created(expires)digest",signature="fKQWvXhln4UdyZdL87ViXQObdBme0dHnsclD2LvvnHoNxIgcvAwUZOmwAnH5QKi9Upg5tRaxpoGhCFGHD+d+Bw=="

4. The request has expired

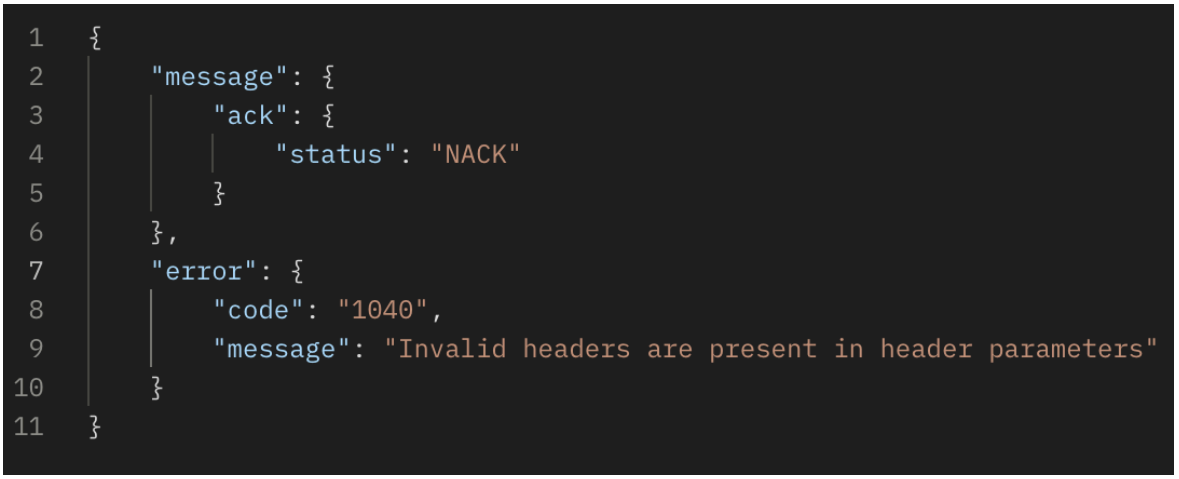
Reason: This error indicates that the `expires` timestamp in the signature header has either passed or is incorrectly formatted. Since the expiry timestamp is part of the signed data, any change to this value without regenerating the signature will cause signature verification to fail. Issues may also arise from extra characters, slashes, or incorrect formatting within the highlighted part of the header.

Resolution: Ensure that the `created` and `expires` timestamps are correctly set relative to the current time and are within the valid window. Also verify that the header format strictly adheres to the expected structure with no additional characters, slashes, or escape sequences. If any timestamp is modified, regenerate the signature to maintain integrity.

Signature keyId="buyer-app.ondc.org|207|ed25519",algorithm="ed25519",created="1641287875",expires="1641291475",headers="(created)(expires)digest",signature="fKQWvXhln4UdyZdL87ViXQObdBme0dHnsclD2LvvnHoNxIgcvAwUZOmwAnH5QKi9Upg5tRaxpoGhCFGHD+d+Bw=="

5. Header parsing failed OR Invalid headers are present in header parameters

```
{
    "message": {
        "ack": {
            "status": "NACK"
        }
    },
    "error": {
        "code": "1100",
        "message": "Header parsing failed"
    }
}
```



```
{
    "message": {
        "ack": {
            "status": "NACK"
        }
    },
    "error": {
        "code": "1040",
        "message": "Invalid headers are present in header parameters"
    }
}
```

Reason: The error indicates an issue with the highlighted part of the header — possibly due to invalid format, extra characters, or slashes.

Resolution: Check the header format carefully and ensure there are no extra characters, slashes, or escape sequences.

Signature keyId="buyer-app.ondc.org|207|ed25519",algorithm="ed25519",created="1641287875",expires="1641291475",headers="(created)(expires)digest",signature="fKQWvXhln4UdyZdL87ViXQObdBme0dHnsclD2LvvnHoNxlgcvAwUZOmwAnH5QKi9Upg5tRaxpoGhCFGHD+d+Bw=="

6. The signature verification failed

```
{
    "message": {
        "ack": {
            "status": "NACK"
        }
    },
    "error": {
        "code": "1045",
        "message": "The signature verification failed"
    }
}
```

Reason: The issue occurs due to a mismatch between the payload and the signing string, often caused by formatting or beautification, even if the payload content is the same.

Resolution: The JSON should be stringified before signing, and the request body must be the exact lookup payload that was signed. Check that the lookup payload and the signing string are exactly the same — without beautification, extra spaces, or formatting changes. Minified JSON is preferred.