**Signing** **UHI APIs in HTTP**

# Introduction

When communicating over HTTP using UHI APIs, the subscribers need to authenticate themselves to perform transactions with other subscribers. Due to the commercial nature of the transactions, every request/callback pair is considered to be a "contract" between two parties. Therefore, it is imperative that all requests and callbacks are digitally signed by the sender and subsequently verified by the receiver.

Furthermore, it is also desirable to ensure that the message was not altered or tampered with during transit.

This document describes a way for network subscribers (EUA/HSPAs) and proxy subscribers (UHI GATEWAYs) to simultaneously add authentication and message integrity to HTTP messages by using digital signatures. How the signatures are generated and the format of those signatures is out of scope of this document. This document specifies the algorithms used in generating the keys, how to construct the signing strings being passed in the headers. Also, it specifies clearly the format of the HTTP headers used for authenticating EUA, HSPAs and UHI GATEWAYs.

# Subscriber authentication

The EUA and HSPA subscriber is expected to send an Authorization header where the "auth-scheme" is "Signature" and the "auth-param" parameters meet the requirements.

The UHI GATEWAY subscriber is expected to send a X-Gateway-Authorization header where the "auth-scheme" is "Signature" and the "auth-param" parameters meet the requirements.

Below is the format of a EUA/HSPA Authorization header in the typical HTTP Signature format:

Authorization::{"headers":"(created) (expires) digest","expires":"1679652060","signature":"EfImhDOjl1SiwwUYKHSmz52u/FYZSggS5B177WaxvQg5OApvCk/OQW/n0nsSdRdm7KM8cXm/T77pdF7jI5G3Bw==","created":"1679652050","keyId":"eua-nha|eua.nha.k1|ed25519","algorithm":"ed25519"}

The UHI GATEWAY will send its signature in the X-Gateway-Authorization header in the exact same format as shown below.

X-Gateway-Authorization::{"headers":"(created) (expires) digest","expires":"1679651256","signature":"iiY0S3XVyxg6OamWTFvldHxkqsTOUDdwNUpgJaYjbP1fRqVFQ73jGxLfEPu5fjIYxhAthyCq3l3KCJAUOKIRAA==","created":"1679651246","keyId":"uhigateway|uhigatewaypubkeyid|ed25519","algorithm":"ed25519"}

# Hashing Algorithm

For computing the digest of the request body, the hashing function will use the BLAKE-512 (2b) hashing algorithm. BLAKE is a cryptographic hash function based on Dan Bernstein's ChaCha stream cipher. For more documentation on the BLAKE-512(2b) algorithm.

Example of hash:

|  |
| --- |
| BLAKE2b-512("The quick brown fox jumps over the lazy dog") =  [43, -21, -56, -128, 89, 51, -80, 12, 75, -88, 17, 125, -16, -111, -83, -19, 102, -12, 49, 57, 111, 2, -127, 88, 22, 15, 102, -23, 127, -83, -20, -29, 46, -47, 75, 95, -19, 65, 77, -125, 4, 49, 73, -12, 71, -78, 40, -63, 114, 112, 101, -65, 64, 56, 4, -87, -15, 13, 7, 10, -83, -3, 95, 84] |

The above hex value encoded in base64 is as below :

|  |
| --- |
| K+vIgFkzsAxLqBF98JGt7Wb0MTlvAoFYFg9m6X+t7OMu0Utf7UFNgwQxSfRHsijBcnBlv0A4BKnxDQcKrf1fVA== |
|  |

# Signing Algorithm

To digitally sign the singing string, the subscribers should use the "ed25519" signature scheme.

**Algorithm**

The Ed25519 algorithm is based on the Elliptic Curve Digital Signature Algorithm (ECDSA), but uses a different curve and a different set of parameters. It uses the twisted Edwards curve known as Curve25519, which is designed to be secure and efficient for use in cryptography.

Ed25519 is designed for high security and performance and is considered to be one of the most secure digital signature algorithms available today. It has several advantages over older algorithms such as RSA and DSA, including smaller key sizes, faster key generation, and faster signature verification.

# UHI Keys Specification

|  |  |  |  |
| --- | --- | --- | --- |
| Keys | Algorithm | Format | Type |
| Public Key | Ed25519 | DER(Distinguished Encoding Rules) | SPKI(Subject Public Key Info) |
| Private Key | Ed25519 | DER(Distinguished Encoding Rules) | PKCS8(Public Key Cryptography Standard #8) |

**Public Key –**

|  |  |  |  |
| --- | --- | --- | --- |
| Keys | Algorithm | Format | Type |
| Public Key | Ed25519 | DER(Distinguished Encoding Rules) | SPKI(Subject Public Key Info) |

DER stands for Distinguished Encoding Rules, which is a format for encoding data structures that is widely used in public-key cryptography. In the context of Ed25519 keys, the DER format is used to represent the public key as a sequence of bytes that can be easily transmitted over the internet. The DER format is commonly used in X.509 certificates and is supported by most cryptographic libraries.

SPKI stands for Subject Public Key Info, which is a format for representing public keys that is used in some security protocols such as SSL/TLS. In the context of Ed25519 keys, the SPKI format is used to represent the public key as a structured data object that includes information about the algorithm and the key itself.

**Private Key –**

|  |  |  |  |
| --- | --- | --- | --- |
| Keys | Algorithm | Format | Type |
| Private Key | Ed25519 | DER(Distinguished Encoding Rules) | PKCS8(Public Key Cryptography Standard #8) |

DER stands for Distinguished Encoding Rules, which is a format for encoding data structures that is widely used in public-key cryptography. In the context of Ed25519 keys, the DER format is used to represent the public key as a sequence of bytes that can be easily transmitted over the internet. The DER format is commonly used in X.509 certificates and is supported by most cryptographic libraries.

PKCS#8 (Public Key Cryptography Standard #8) is a standard that defines a syntax for storing and transferring private key information. It is widely used in the field of cryptography and is supported by many cryptographic libraries.

In the context of Ed25519 keys, the PKCS#8 format is used to represent the private key as a sequence of bytes. The PKCS#8 format provides a way to store the private key information in a secure and standardized way, which can be useful for interoperability between different systems.

## Example Flow (EUA <=> UHI GATEWAY <=> HSPA)

**Step 1 : EUA signs request and calls UHI GATEWAY** Let the below be the request body in this example :

**{**

**"context": {**

**"domain": "nic2004:85111",**

**"country": "IND",**

**"city": "std:080",**

**"action": "search",**

**"core\_version": "0.7.1",**

**"consumer\_id": "eua-nha",**

**"consumer\_uri": "http://uhieuasandbox.abdm.gov.in/api/v1/euaService",**

**"message\_id": "e9a19230-f951-11ec-b135-53aea776f66b",**

**"timestamp": "2022-07-05T15:24:35",**

**"transaction\_id": "e9a19230-f951-11ec-b135-53aea776f66b"**

**},**

**"message": {**

**"intent": {**

**"category": {**

**"descriptor": {**

**"code": "CARDIOLOGY",**

**"name": "CARDIOLOGY"**

**}**

**},**

**"fulfillment": {**

**"type": "Online",**

**"start": {**

**"time": {**

**"timestamp": "2022-07-15T00:00:00"**

**}**

**},**

**"end": {**

**"time": {**

**"timestamp": "2022-07-16T00:00:00"**

**}**

**}**

**},**

**"item": {**

**"descriptor": {**

**"code": "Consultation",**

**"name": "Consultation"**

**}**

**}**

**}**

**}**

**}**

Let EUA’s keys be :

|  |
| --- |
| PrivateKey::MC4CAQAwBQYDK2VwBCIEIGCioWvsJleY53bW5+8G9vWWqsdGA9y1JMoVvLslaFA7  Public key::MCowBQYDK2VwAyEA9AwjtMBySgjkf3cx2pzAYt8pL1qAlhil3WUulCSpFnI= |
|  |

The EUA performs the following steps to create the Authorization header

1. Generate the digest of the request body using the BLAKE-512 hashing function

|  |
| --- |
| W9UFzCRHgPpPoJxDFuGIz9bWqOjpc6f+Tv32bN1X3qSesscMXhUSdMyGsloAgn8+wo95otF2FQDz+YOBVz2hqA== |
|  |

1. Generate the created field. The `created` field expresses when the signature was created. The value MUST be a Unix timestamp integer value. A signature with a `created` timestamp value that is in the future MUST NOT be processed.

(created): 1679652050

1. Generate the expires field. The `expires` field expresses when the signature ceases to be valid. The value MUST be a Unix timestamp integer value. A signature with an `expires` timestamp value that is in the past MUST NOT be processed.

(expires): 1679652060

1. Concatenate the three values, i.e the `created`, `expires` and `digest` in the format as shown below. The below string is the signing string which the EUA is going to use to sign the request

|  |
| --- |
| (created): 1679652050 |
| (expires): 1679652060 |
| digest: |
| BLAKE-512= W9UFzCRHgPpPoJxDFuGIz9bWqOjpc6f+Tv32bN1X3qSesscMXhUSdMyGsloAgn8+wo95otF2FQDz+YOBVz2hqA== |
|  |

1. The EUA will then sign this string using it's registered *signing private key* via the Ed25519 Signature Scheme
2. Finally the EUA will generate a base64 encoded string of the signature and insert it into the **signature** parameter of the **Authorization** header

|  |
| --- |
| EfImhDOjl1SiwwUYKHSmz52u/FYZSggS5B177WaxvQg5OApvCk/OQW/n0nsSdRdm7KM8cXm/T77pdF7jI5G3Bw== |
|  |

1. Finally the Authorization header will look like this. (Let's assume subscriber\_id = example-EUA.com, public\_key\_id = EUA1234)
2. Finally the EUA includes the Authorization header in the request and calls the UHI GATEWAY API

{"headers":"(created) (expires) digest","expires":"1679652060","signature":"EfImhDOjl1SiwwUYKHSmz52u/FYZSggS5B177WaxvQg5OApvCk/OQW/n0nsSdRdm7KM8cXm/T77pdF7jI5G3Bw==","created":"1679652050","keyId":"eua-nha|eua.nha.k1|ed25519","algorithm":"ed25519"}

**Note:**

1. The difference between the `created` and the `expires` field should be equal to the TTL of the request context.
2. Also, the `expires` value should not be more than the expiration time of the key used for signing the request. If the expires value appears to be going beyond the lifespan of the signing key, generate a new key and update it on the registry OR use an existing registered key with an expiry time greater than the `expires` value of the signature

### Step 2 : UHI GATEWAY verifies EUA signature

The UHI GATEWAY performs the following steps to authenticate the EUA and also ensure message

integrity.

1. UHI GATEWAY gets **keyId** from the **Authorization** header

eua-nha|eua.nha.k1|ed25519

1. UHI GATEWAY splits the keyId string into subscriber ID, Unique Key ID and algorithm using the delimiter "|".
2. The keyId uses the format {subscriber id}|{public\_key\_id }|{signing algorithm} . If the signing algorithm extracted from the keyId does not match the value of the algorithm parameter in the Authorization header, then the UHI GATEWAY should return an UHI-1405 unauthorised error.
3. The keyId also contains a **public\_key\_id** which is used when the EUA has uploaded *multiple* public keys to a registry OR when the same domain is being used for implementing multiple types of subscribers
4. The UHI GATEWAY will now look up the registry for the public key of the subscriber by sending the **subscriber\_id** and the **public\_key\_id** via the **lookup/gateway API** or by retrieving a cached copy of the subscriber's public key matching the *subscriber\_id* and *public\_key\_id .*It will receive the public key of the EUA :

MCowBQYDK2VwAyEA9AwjtMBySgjkf3cx2pzAYt8pL1qAlhil3WUulCSpFnI=

1. If no valid key is found, the HSPA must return a NACK response with UHI-1405 Unauthorised response code.
2. UHI GATEWAY will use the EUA's **public key** to verify the signature. If signature is verified, the EUA is considered to be authenticated. If not UHI GATEWAY should return a UHI-1405 error
3. If the signature is not verified, the UHI GATEWAY must return a NACK response with response code UHI-1405. For example, for an invalid signature, the UHI GATEWAY must return the following:

**Response Body:**

{

"message": {

"ack": {

"status": "NACK"

}

},

"error": {

"type": "",

"code": "UHI-1405",

"path": "/RequesterService",

"message": "INVALID SIGNATURE"

}

}

**Step 3 : UHI GATEWAY signs request before forwarding request to HSPA** Let the UHI GATEWAY’s keys be :

|  |
| --- |
| Private Key::MC4CAQAwBQYDK2VwBCIEIOQY8MXyrivtJ2/s0lBg6E5DQQ5GMID+OAXyAlyybdUH |
| Public key::MCowBQYDK2VwAyEAcIx++sNvDok2/3Vh/xjKZ0sLovgsFZz2Aizo5AVb74Q= |
|  |

Before forwarding the request to the HSPA, the UHI GATEWAY performs the following steps to create the **X-Gateway-Authorization** header.

1. Generate the **digest** of the request body using the **BLAKE-512** hashing function

|  |
| --- |
| W9UFzCRHgPpPoJxDFuGIz9bWqOjpc6f+Tv32bN1X3qSesscMXhUSdMyGsloAgn8+wo95otF2FQDz+YOBVz2hqA== |
|  |

1. Generate the `created` field. The `created` field expresses when the signature was created. The value MUST be a Unix timestamp integer value. A signature with a `created` timestamp value that is in the future MUST NOT be processed.

(created): 1679651246

1. Generate the `expires` field. The `expires` field expresses when the signature ceases to be valid. The value MUST be a Unix timestamp integer value. A signature with an `expires` timestamp value that is in the past MUST NOT be processed.

(expires): 1679651256

1. Concatenate the three values, i.e the `created`, `expires` and `digest` in the format as shown below. The below string is the **signing string** which the UHI GATEWAY is going to use to sign the request

|  |
| --- |
| (created): 1679651246 |
| (expires): 1679651256 |
| digest: |
| W9UFzCRHgPpPoJxDFuGIz9bWqOjpc6f+Tv32bN1X3qSesscMXhUSdMyGsloAgn8+wo95otF2FQDz+YOBVz2hqA== |
|  |

1. The UHI GATEWAY will then sign this string using it's registered **signing private key** via the **Ed25519** Signature Scheme
2. Finally the UHI GATEWAY will generate a **base64** encoded string of the signature and insert it into the **signature** parameter of the **X-Gateway-Authorization** header

|  |
| --- |
| iiY0S3XVyxg6OamWTFvldHxkqsTOUDdwNUpgJaYjbP1fRqVFQ73jGxLfEPu5fjIYxhAthyCq3l3KCJAUOKIRAA== |
|  |

1. Finally the **X-Gateway-Authorization** header will look like this. (Let's assume subscriber\_id = uhigateway, public\_key\_id = uhigatewaypubkeyid)
2. {"headers":"(created) (expires) digest","expires":"1679651256","signature":"iiY0S3XVyxg6OamWTFvldHxkqsTOUDdwNUpgJaYjbP1fRqVFQ73jGxLfEPu5fjIYxhAthyCq3l3KCJAUOKIRAA==","created":"1679651246","keyId":"uhigateway|uhigatewaypubkeyid|ed25519","algorithm":"ed25519"}
3. Finally the UHI GATEWAY includes the **X-Gateway-Authorization** header in the request and calls the HSPA **search** API

**Note:**

1. The difference between the `created` and the `expires` field should be equal to the TTL of the request.
2. Also, the `expires` value should not be more than the expiration time of the key used for signing the request. If the expires value appears to be going beyond the lifespan of the signing key, generate a new key and update it on the registry via **subscribe** API OR use an existing registered key with an expiry time greater than the `expires` value of the signature

### Step 4 : HSPA verifies UHI GATEWAY signature

The HSPA performs the following steps to authenticate the EUA and the UHI GATEWAY and also ensure

message integrity.

1. Get **keyId** from the **X-Gateway-Authorization** header

uhigateway|uhigatewaypubkeyid|ed25519

1. Split the **keyID** string using the delimiter `|`. As we know, the subscriber ID is the registered domain name of the subscriber and follows the format{subdomain}.{domain}.{extension}.
2. The **keyID** uses the format {subscriber id}|{public\_key\_id }|{signing algorithm} . If the signing algorithm extracted from the **keyId** does not match the value of the **algorithm** field in the **X-Gateway-Authorization** header, then the HSPA should return an error.
3. The **keyID** also contains a **public\_key\_id** which is used when the UHI GATEWAY has uploaded *multiple* public keys to a registry OR when the same domain is being used for implementing multiple subscribers
4. The UHI GATEWAY will now look up the registry for the public key of the subscriber by sending the **subscriber\_id** and the **public\_key\_id** via the **lookup/gateway API** or by retrieving a cached copy of the subscriber's public key matching the *subscriber\_id* and *public\_key\_id* . It will get the UHI GATEWAY’s public key :

MCowBQYDK2VwAyEAcIx++sNvDok2/3Vh/xjKZ0sLovgsFZz2Aizo5AVb74Q=

1. If no valid key is found, the HSPA must return a NACK response with UHI-1405response code.
2. HSPA will use the UHI GATEWAY's **public key** to verify the signature. If the signature is verified, the UHI GATEWAY is considered to be authenticated.
3. If the signature is not verified, the HSPA must return a NACK response with UHI-1405response cod. For example, for an invalid UHI GATEWAY signature, the HSPA must return the following:

**Headers:**...

**Request Body:**

{

"message": {

"ack": {

"status": "NACK"

}

},

"error": {

"type": "",

"code": "UHI-1405",

"path": "/RequesterService",

"message": "INVALID SIGNATURE"

}

}

### Step 5 : HSPA verifies EUA signature

The HSPA performs the following steps to authenticate the EUA and also ensure message

integrity during transit.

1. HSPA gets **keyId** from the **Authorization** header 2. HSPA splits the **keyID** string using the delimiter `|`.

1. The **keyID** uses the format {subscriber id}|{public\_key\_id }|{signing algorithm} . If the signing algorithm extracted from the **keyId** does not match the value of the **algorithm** field in the **Authorization** header, then the HSPA should return an error.
2. The keyId also contains a **public\_key\_id** which is used when the EUA has uploaded *multiple* public keys to a registry OR when the same domain is being used for implementing multiple types of subscribers
3. The HSPA will now look up the registry for the public key of the EUA by sending the **subscriber\_id** and the **public\_key\_id** via the **lookup/gateway API** or by retrieving a cached copy of the EUA's public key matching the *subscriber\_id* and *public\_key\_id .* The
4. UHI GATEWAY will use the EUA's **public key** to verify the signature. If signature is verified, the EUA is considered to be authenticated. If not UHI GATEWAY should return a UHI-1405 error
5. If the signature is not verified, the UHI GATEWAY must return a NACK response with UHI-1405response code. For example, for an invalid signature, the HSPA must return the following:

**Headers:**...

**Request Body:**

{

"message": {

"ack": {

"status": "NACK"

}

},

"error": {

"type": "",

"code": "UHI-1405",

"path": "/RequesterService",

"message": "INVALID SIGNATURE"

}

}

}

### Step 6 : HSPA signs callback and calls UHI GATEWAY

The HSPA performs the following steps to create the **Authorization** header

{"headers":"(created) (expires) digest","expires":"1679655888","signature":"y8CkJeeIoo1z8wlFZco3tY+sCpIb6/JkFTO5RS+0ZTxF9rv2bGyIC3P91qRCMBNZe4C/jjO0xXGXB1d78RnoDg==","created":"1679655878","keyId":"refhspa|refhspapubkeyid|ed25519","algorithm":"ed25519"}

1. Generate the **digest** of the request body using the **BLAKE-512** hashing function
2. Generate the **created** field. The `created` field expresses when the signature was created. The value MUST be a Unix timestamp integer value. A signature with a `created` timestamp value that is in the future MUST NOT be processed.
3. Generate the **expires** field. The `expires` field expresses when the signature ceases to be valid. The value MUST be a Unix timestamp integer value. A signature with an `expires` timestamp value that is in the past MUST NOT be processed.
4. Concatenate the three values, i.e the `created`, `expires` and `digest` in the format as shown below. The below string is the **signing string** which the HSPA is going to use to sign the request

|  |
| --- |
| (created): 1679655878 |
| (expires): 1679655888 |
| digest: BLAKE-512=W9UFzCRHgPpPoJxDFuGIz9bWqOjpc6f+Tv32bN1X3qSesscMXhUSdMyGsloAgn8+wo95otF2FQDz+YOBVz2hqA== |

1. Sign this string using it's registered *signing private key* via the ed25519 Signature Scheme
2. Generate a base64 encoded string of the signature and insert it into the **signature** parameter of the **Authorization** header
3. The final **Authorization** header will look like this. (Let's assume subscriber\_id = refhspa, public\_key\_id = refhspapubkeyid)

{"headers":"(created) (expires) digest","expires":"1679655888","signature":"y8CkJeeIoo1z8wlFZco3tY+sCpIb6/JkFTO5RS+0ZTxF9rv2bGyIC3P91qRCMBNZe4C/jjO0xXGXB1d78RnoDg==","created":"1679655878","keyId":"refhspa|refhspapubkeyid|ed25519","algorithm":"ed25519"}

1. Finally the HSPA includes the Authorization header in the request and calls the UHI GATEWAY API

### Step 7 : UHI GATEWAY verifies HSPA signature

The UHI GATEWAY performs the following steps to authenticate the HSPA and also ensure message

integrity.

1. UHI GATEWAY gets **keyId** from the **Authorization** header
2. UHI GATEWAY splits the keyId string into subscriber ID, Unique Key ID and algorithm using the delimiter "|".
3. The keyId uses the format {subscriber id}|{public\_key\_id }|{signing algorithm} . If the signing algorithm extracted from the keyId does not match the value of the algorithm parameter in the Authorization header, then the UHI GATEWAY should return an UHI-1405 unauthorised error.
4. The keyId also contains a **public\_key\_id** which is used when the HSPA has uploaded *multiple* public keys to a registry OR when the same domain is being used for implementing multiple types of subscribers
5. The UHI GATEWAY will now look up the registry for the public key of the HSPA by sending the **subscriber\_id** and the **public\_key\_id** via the **lookup/gateway API** or by retrieving a cached copy of the HSPA's public key matching the *subscriber\_id* and *public\_key\_id*
6. UHI GATEWAY will use the HSPA's **public key** to verify the signature. If signature is verified, the HSPA is considered to be authenticated. If not UHI GATEWAY should return a UHI-1405 error
7. If the signature is not verified, the UHI GATEWAY must return a NACK response with UHI-1405 **Unauthorised** response code with. For example, for an invalid signature, the UHI GATEWAY must return the following:

**Response Body:**

{

"message": {

"ack": {

"status": "NACK"

}

},

"error": {

"type": "",

"code": "UHI-1405",

"path": "/RequesterService",

"message": "INVALID SIGNATURE"

}

}

}

### Step 8 : UHI GATEWAY signs callback before calling EUA

Before forwarding the request to the EUA, the UHI GATEWAY performs the following steps to create the

**X-Gateway-Authorization** header

1. Generate the **digest** of the request body using the **BLAKE-512** hashing function
2. Generate the `created` field. The `created` field expresses when the signature was created. The value MUST be a Unix timestamp integer value. A signature with a `created` timestamp value that is in the future MUST NOT be processed.
3. Generate the `expires` field. The `expires` field expresses when the signature ceases to be valid. The value MUST be a Unix timestamp integer value. A signature with an `expires` timestamp value that is in the past MUST NOT be processed.
4. Concatenate the three values, i.e the `created`, `expires` and `digest` in the format as shown below. The below string is the **signing string** which the UHI GATEWAY is going to use to sign the request
5. The UHI GATEWAY will then sign this string using it's registered **signing private key** via the **ed25519** Signature Scheme
6. Finally the UHI GATEWAY will generate a **base64** encoded string of the signature and insert it into the **signature** parameter of the **X-Gateway-Authorization** header
7. Finally the **X-Gateway-Authorization** header will look like this. (Let's assume subscriber\_id = uhigateway, public\_key\_id = uhigatewaypubkeyid)
8. Finally the UHI GATEWAY includes the **X-Gateway-Authorization** header in the request and calls the EUA API

**Note:**

1. The difference between the `created` and the `expires` field should be equal to the TTL of the request.
2. Also, the `expires` value should not be more than the expiration time of the key used for signing the request. If the expires value appears to be going beyond the lifespan of the signing key, generate a new key and update it on the registry via **subscribe** API OR use an existing registered key with an expiry time greater than the `expires` value of the signature

### Step 9 : EUA verifies UHI GATEWAY signature

The EUA performs the following steps to authenticate the EUA and the UHI GATEWAY and also ensure

message integrity.

1. Get **keyId** from the **X-Gateway-Authorization** header
2. Split the **keyID** string using the delimiter `|`.
3. The **keyID** uses the format {subscriber id}|{public\_key\_id}|{signing algorithm} . If the signing algorithm extracted from the **keyId** does not match the value of the **algorithm** field in the **X-Gateway-Authorization** header, then the HSPA should return an error.
4. The **keyID** also contains a **public\_key\_id** which is used when the UHI GATEWAY has uploaded *multiple* public keys to a registry OR when the same domain is being used for implementing multiple subscribers
5. The EUA will now look up the registry for the public key of the subscriber by sending the **subscriber\_id** and the **public\_key\_id** via the **lookup/gateway API** or by retrieving a cached copy of the subscriber's public key matching the *subscriber\_id* and *public\_key\_id* . If no valid key is found, the HSPA must return
6. HSPA will use the UHI GATEWAY's **public key** to verify the signature. If signature is verified, the UHI GATEWAY is considered to be authenticated.
7. If the signature is not verified, the HSPA must return a NACK response with UHI-1405response. For example, for an invalid UHI GATEWAY signature, the EUA must return the similar to following:

**Response Body:**

{

"message": {

"ack": {

"status": "NACK"

}

},

"error": {

"type": "",

"code": "UHI-1405",

"path": "/RequesterService",

"message": "INVALID SIGNATURE"

}

}

### Step 10 : EUA verifies HSPA signature

The EUA performs the following steps to authenticate the EUA and also ensure message

integrity during transit.

1. EUA gets **keyId** from the **Authorization** header 2. EUA splits the **keyID** string using the delimiter `|`.

1. The **keyID** uses the format {subscriber id}|{public\_key\_id}|{signing algorithm} . If the signing algorithm extracted from the **keyId** does not match the value of the **algorithm** field in the **Authorization** header, then the EUA should return an error.
2. The keyId also contains a **public\_key\_id** which is used when the HSPA has uploaded *multiple* public keys to a registry OR when the same domain is being used for implementing multiple types of subscribers
3. The EUA will now look up the registry for the public key of the HSPA by sending the **subscriber\_id** and the **public\_key\_id** via the **lookup/gateway API** or by retrieving a cached copy of the HSPA's public key matching the *subscriber\_id* and *public\_key\_id*
4. EUA will use the HSPA's **public key** to verify the signature. If signature is verified, the HSPA is considered to be authenticated.
5. If the signature is not verified, the EUA must return a NACK response with UHI-1405 **Unauthorised** response code. For example, for an invalid HSPA signature, the EUA must return the following:

**Response Body:**

{

"message": {

"ack": {

"status": "NACK"

}

},

"error": {

"type": "",

"code": "UHI-1405",

"path": "/RequesterService",

"message": "INVALID SIGNATURE"

}

}

**Header Utility to generate Keys**

It's a utility to generate signing key pairs, generating headers and verification of UHI headers.

URL - <https://github.com/NHA-ABDM/UHI/tree/main/header_generator_utility>

**Option 1 –**

Key pair generation - This option is to generate ed25519 public & private key pair.

Output Example –

Private Key::MC4CAQAwBQYDK2VwBCIEIO+JdHA2VllG5p+3DjptC6uO6NMC8h2uXfpBAOeG4iyY

Public key::MCowBQYDK2VwAyEALlZSPoFROXpr9sPgUCoSsfuH8LUKGRgK64MjLWMw3tk=

**Option 2 –**

This option will generate the header based on input provided.

Output Example - {"headers":"(created) (expires) digest","expires":"1678710850","signature":"5z177GF8MAnr1kjy7reKm75kwtHoVvv91nJej79A2XdtO6mw13mY1JVFaeKWUMk77Jgrblk9g97vC2x+0jwECw==","created":"1678710840","keyId":"eua-nha|eua.nha.k1|ed25519","algorithm":"ed25519"}

Graphical user interface, text, application

Description automatically generated

**Option 3 –**

This option will give result of verification of header

Output Example - True or False.