API Request Signing

All HTTP API requests into Kronos (Pegasus) platform must be signed by the client application and validated by the cloud server to ensure they come from authenticated clients. There are 2 important pieces of information - apiKey and secretKey that can be obtained from the cloud portal and embedded into the client application for signing requests. It's recommended that these keys are stored encrypted on the client side.

For example in this document, we use the following keys

- apiKey: 5501f50fdc62aee5d04dbd6a58b68b781ee2aaade8ad1eb24b1e4e77cb282ae2
- secretKey: ARAzUzRzekFwRTNACBQYUx89LlZyImhKFVloHUVMDw8EGRxxSCckFgdFPysAAWJCLDgMdkstZzw3GGVqNHxXcno5Iz54LRBSKy0TaCBwNndkfQNdD38KAA==

The client application needs to process the API request details using 5 steps explained below. The input parameters include such information as the URL, HTTP method, query string, JSON payload, HTTP headers, the security keys, etc. The computation outcome is a signature value which is included in the HTTP header.

Note: This procedure is a customized version of AWS Signature Version 4.

Step 1: Create a Canonical Request for signing

The first step is to build a string representation of your request in a pre-defined format.

```
canonicalRequest =
httpRequestMethod + '\n' +
canonicalURI + '\n' +
canonicalQueryString + '\n' +
hexEncode(hash(payload))
```

httpRequestMethod

Put one of the following methods in a line by itself - GET, POST, PUT, PATCH

```
POST
```

canonicalURI

The canonical URI is the URI-encoded version of the absolute path component of the URI, which is everything in the URI from the HTTP host to the question mark character ("?") that begins the query string parameters (if any).

```
/api/v1/kronos/gateways
```

canonicalQueryString

- Create each line containing a name/value pair in the format name=value. Name field is converted to lowercase and URL-encoded.
- Sort all the lines in ascending order

Example: queryString: lastName=Doe&firstName=Jane&Age=30

```
age=30
firstname=Jane
lastname=Doe
```

hexEncode(hash(payload))

1. The payload (if any) is most likely in JSON format. If there's no payload, use an empty string
2. Hash the payload content with SHA-256
3. Hex-encode the hash value
4. Convert the hex-encoded value to lowercase

Example of an empty payload:

```
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

Combining all of the results above produces the following string

```
POST
/api/v1/kronos/gateways
age=30
firstname=Jane
lastname=Doe
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

Finally, hex-encode the computed canonical request above and lowercase it. This is the result of step 1.

```
5a2d3589ffb15fab720069fbd26fd8e8311a1c7047e5899608faff450df6d7dc
```

Step 2: Create the string to sign

The structure of the string is as follow

```
stringtoSign =
hashedCanonicalRequest + '\n' +
apiKey + '\n' +
requestTimestamp + '\n' +
apiversion
```

hashedCanonicalRequest

This is the result of step 1

```
5a2d3589ffb15fab720069fbd26fd8e8311a1c7047e5899608faff450df6d7dc
```

apiKey

Provided apiKey from Kronos portal

```
5501f50fdc62aee5d04dbd6a58b68b781ee2aaade8ad1eb24b1e4e77cb282ae2
```

requestTimestamp

Current UTC time in ISO-8601 format - YYYY-MM-DDThh:mm:s.sssZ

```
2016-04-12T14:28:36.218Z
```

apiVersion

Current version is 1

```
1
```

Combining all of the results above produces the following string. This is the result of step 2.

```
5a2d3589ffb15fab720069fbd26fd8e8311a1c7047e5899608faff450df6d7dc
5501f50fdc62aee5d04dbd6a58b68b781ee2aaade8ad1eb24b1e4e77cb282ae2
2016-04-12T14:28:36.218Z
1
```

Step 3: Create the signing key

In this step we're going to use the HMAC-SHA256 algorithm a few times to generate the signing key. In the pseudocode, it'll be shown as hmacSha256(key, data). Note the order of the input parameters because if you're using some third-party library for this function, the order of the input parameters might be reversed. The output of this function is in binary, hence note the hex-encode function being used below

```
signingKey = secretKey
signingKey = hexEncode(hmacSha256(apiKey, signingKey))
signingKey = hexEncode(hmacSha256(requestTimestamp,
signingKey))
signingKey = hexEncode(hmacSha256(apiVersion, signingKey))
```

secretKey

Provided secretKey from Kronos portal

apiKey

Provided apiKey from Kronos portal

requestTimestamp

Must be the same value as in step 2

apiVersion

Must be the same value as in step 2

For the example in this guide, here's the output of this step. This is the result of step 3

```
signingKey =
ARAzUzRzekFwRTNACBQYUx89LlZyImhKFVloHUVMDw8EGRxxSCckFgdFPysAAWJCLDgMdkstZzw3GGVqNHxXcno5Iz54LRBSKy0TaCBwNnc
signingKey = 3c6e85f6a719e5b8bd77fde0cbdbe19d947f38451afbc8ef6e49a083d86a9c54
signingKey = 3223bf9bc2d2180046cc40c2e1ed6f9d08261a6c4a394b23c5311e83633a8ef7
signingKey = d0d1518fc5290c22f1444d46d9c08dd03cc33c6fdad8bbcd57be65b1e2b0b493
```

Step 4: Create the final signature

The signature is computed by signing the result from step 2 and step 3

```
signature = hexEncode(hmacSha256(signingKey,
stringToSign)
```

signingKey

Result of step 3

stringToSign

Result of step 2

Result of the current example is

```
28c3ab6cc82294b61e9b2855b428090e474fd1e066c4da63f9715bd2204df553
```

Step 5: Add required headers to request and submit

The following headers are required and will be used by the cloud server to verify the signature

- x-arrow-apikey - provided apiKey
- x-arrow-date - requestTimestamp from step 2
- x-arrow-version - apiVersion from step 2
- x-arrow-signature - final signature from step 4

```
x-arrow-apikey: 5501f50fdc62aee5d04dbd6a58b68b781ee2aaade8ad1eb24b1e4e77cb282ae2
x-arrow-date: 2016-04-12T14:28:36.218Z
x-arrow-version: 1
x-arrow-signature: 28c3ab6cc82294b61e9b2855b428090e474fd1e066c4da63f9715bd2204df553
```