IdentityX 4.1

Authenticating to REST Services

Quick Reference Guide

The IdentityX REST SDK takes care of the security and authentication of the IdentityX REST service endpoints. For customers that wish to construct their own REST calls, or for customers that wish to better understand the authentication process, this guide provides information on Authenticating to the IdentityX REST Services. The ***IdentityX FIDO Integration*** Guide gives information on how to use the REST SDK to communicate with the REST Services.

In order to communicate with the IdentityX REST services, all requests need to have a header containing a Digest message. This is used for authentication, to prevent replay attacks, and also to verify data integrity. The digest contains data like a timestamp, all of the query parameters sent with the request and a unique nonce. The message is calculated with a combination of a cryptographic hash and Hash-based message authentication code (HMAC) algorithms on the contents of the request.

In order to generate the HMAC, each client needs to be in possession of a key. This key is a combination of a Key ID and a shared secret. The shared secret is used for the calculation of the HMAC.

These keys can be generated via the Admin Console.

Table of Contents

1. Attaching a Digest to a Request

The following parts are included in the calculation of the Digest on the request:

Parameter	Description
Resource URL	The URL used to access the REST resource, for example: https://default.fido.identityx-staging.com/rest/v1/registrationChallenges/IVpvdSnQ1l3KAh6wAzOa8w
HTTP Method	The desired action performed on the resource, for example, GET, POST
Query String	The sorted query parameters sent to the resource. These might include information on filters (only showing ACTIVE users for instance), sorting or searching.
Headers	The HTTP Headers included with the request
Request Body	The data sent to the server
Key ID	The ID of the Key which matches the shared secret used to generate the Digest.
Current Date and Time	This is used to verify when the request was sent. If this does not lie within a configurable time window then the request will be rejected.
Unique Nonce	This is a one-time use code to protect against Replay Attacks. This is where a dishonest party re-sends a legitimate message in order to attack the system.

2. Verifying the Digest on the Response

Each response from the server contains a header that holds the message Digest. This can be used to verify the authenticity and the integrity of the response message. It is similar to the Digest calculated on the request and includes the following data:

Parameter	Description
HTTP Status	The HTTP Status response code for the request sent in by the client. Examples: 200 means OK 403 means FORBIDDEN
Headers	Headers sent back with the HTTP response
Response Body	The body of the response sent back to the client
Key ID	This is the Key ID that the client had submitted in the request which is linked to the shared secret used to generate both the request and response digests.
Current Date Time	The current data and time on the server
Nonce Supplied	The one time use code supplied by the client with the request

3. Sample Code

The sample code below will show you how these digests are calculated. These examples can be used to not only understand the digest process but can also be used as a guide on how you can develop an SDK in a language not currently supported or how to talk directly to the REST services without using the provided SDKs.

✓ Constants

To make the code clearer later on, the following constants are defined as follows:

```
// constants used
    String ID_TERMINATOR = "digest_request";
    String AUTHENTICATION_SCHEME = "Digest";
    String ALGORITHM = "HMAC-SHA-256";
    String DEFAULT_ENCODING = "UTF-8";
    String DATE_FORMAT = "yyyyMMdd";
    String TIMESTAMP_FORMAT = "yyyyMMdd'T'HHmmss'Z'";
    String TIME_ZONE = "UTC";
    String AUTHORIZATION_HEADER = "Authorization";
    String DATE_HEADER = "Auth-Date";
```

NOTE: The name of the `AUTHORIZATION_HEADER` is actually configurable and can be changed by setting the static value AuthSettings.*AUTHORIZATION_HEADER*.

✓ Cryptographic Methods Used

Whatever language or environment you use to build a client, you will need to have access to a hash algorithm (SHA256) and a HMAC algorithm (SHA256).

```
    byte[] hash(byte[] textBytes) {
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        md.update(textBytes);
        return md.digest();
    }
    // calculates the HMAC of the data
    byte[] sign(byte[] data, byte[] key, MacAlgorithm algorithm)  {
        Mac mac = Mac.getInstance(algorithm.toString());
        mac.init(new SecretKeySpec(key, algorithm.toString()));
        return mac.doFinal(data);
    }
}
```

✓ Structures Used

Below are some structures used to represent the request, response, and the key objects.

```
    public interface IRequest {
        HttpMethod getMethod();
        URI getResourceUrl();
        QueryString getQueryString();
        HttpHeaders getHeaders();
        InputStream getBody();
    }
    public interface IResponse {
        HttpMethod getMethod();
        URI getResourceUrl();
        QueryString getQueryString();
        HttpHeaders getHeaders();
        InputStream getBody();
    }

    // holds a secret and a secretId; can
    public interface ITokenKey extends IApiKey {
        String getId();
        byte[] applyHMac(byte[] data);
    }
```

✓ Calculating the Digest on a Request

This code shows how a digest would be calculated on a request by the client.

```
   // sample method to set a authentication header on a request
    //
    //
    // params:
    // request - a wrapper for the request to sign
    // apiKey - a wrapper for the shared secret / token
    // nonce - a guid the client generates that makes sure the request
can only be sent once
    setAuthorizationHeader(IRequest request, ITokenKey apiKey, String
nonce) {
```

```
        // build current date and time for UTC in the specific format :
       // DATE_FORMAT = "yyyyMMdd"  TIMESTAMP_FORMAT =
"yyyyMMdd'T'HHmmss'Z'"
        //
        // example (this should be current time though !!!!):
        String dateStamp = "20150622";
        String timestamp = "20150622T142011Z";   //   22/06/2015
14:20:11


        // build a hash of the request in hex format
        // see method description below
        String canonicalRequestHashHex =
buildCanonicalRequestHashHex(request);


        // signed headers string is a sorted list of all the header names
used for the signature of the request separated by ;
        String signedHeadersString = getSignedHeadersString(request);
        String authorizationHeader =
buildAuthorizationHeader(canonicalRequestHashHex, signedHeadersString,
apiKey, dateStamp, timestamp, nonce);


        request.getHeaders().set(AUTHORIZATION_HEADER,
authorizationHeader);
    }




   String buildCanonicalRequestHashHex(IRequest request) {



        URI uri = request.getResourceUrl();


        String method = request.getMethod().toString();


        // make sure the path does not contain multiple / characters like
http://dummy.com//index.html
        String canonicalResourcePath =
canonicalizeResourcePath(uri.getPath());


        // The canonicalized query string is formed by first sorting all
the query string parameters,
        // then URI encoding both the key and value and then joining
them, in order, separating key value pairs with an '&'.
```

```
        String canonicalQueryString = canonicalizeQueryString(request);


        // the canonical headers string consists of all headers sorted by
name and separated by NL (new line) in the format:
        //
        // headerName1:headerValue11,headerValue12 NL
        // headerName2:headerValue21
        //
        // Note: if the content-lenght header has a value of 0 is not
included since some servers will ignore it and thus the signature will
break
        String canonicalHeadersString =
canonicalizeHeadersString(request);


        // signed headers string is a sorted list of all the header names
above separated by ;
        String signedHeadersString = getSignedHeadersString(request);


        // hex the hash of the body of the request
        // see the hash function description below
        String requestPayloadHashHex =
toHex(hash(getRequestPayload(request)));


        // here is what it goes in the hash
        // join everything together
        String canonicalRequest =
                        method + NL +
                        canonicalResourcePath + NL +
                        canonicalQueryString + NL +
                        canonicalHeadersString + NL +
                        signedHeadersString + NL +
                        requestPayloadHashHex;


        // calculate a new hash and hex it
        String canonicalRequestHashHex = toHex(hash(canonicalRequest));


        return canonicalRequestHashHex;


    }



    String buildAuthorizationHeader(String canonicalRequestHashHex,
```

```
signedHeadersString, ITokenKey apiKey, String dateStamp, String
timeStamp, String nonce)  {


        String id = apiKey.getId() + "/" + dateStamp + "/" + nonce + "/"
+ ID_TERMINATOR;
        String stringToSign =
                        ALGORITHM + NL +
                        timeStamp + NL +
                        id + NL +
                        canonicalRequestHashHex;



        byte[] kDate = apiKey.applyHMac((dateStamp +
AUTHENTICATION_SCHEME).getBytes(DEFAULT_ENCODING));


        byte[] kNonce = sign(nonce, kDate, MacAlgorithm.HmacSHA256);


        byte[] kSigning = sign(ID_TERMINATOR, kNonce,
MacAlgorithm.HmacSHA256);


        byte[] signature = sign(toUtf8Bytes(stringToSign), kSigning,
MacAlgorithm.HmacSHA256);


        String signatureHex = toHex(signature);


        String authorizationHeader =
                        AUTHENTICATION_SCHEME + " " +
                        createNameValuePair(DIGEST_ID, id) + ", " +
                        createNameValuePair(DIGEST_SIGNED_HEADERS,
signedHeadersString) + ", " +
                        createNameValuePair(DIGEST_SIGNATURE,
signatureHex);


        return authorizationHeader;
    }


    String createNameValuePair(String name, String value) {
        return name + "=" + value;
    }
```

The possible return codes to this method are outlined in the API reference at http://daoninc.github.io/fido-rest-doco/index.html.

✓ Verifying the Response from the Server

This code shows how a digest would be generated on a response received by the client to verify the response returned by the server.

```
    // you can use this approach to verify responses from the IdentityX
rest services
    // params:
    // response from the server
    // apiKey used for calculating the digest on the request
    // nonce passed with the request
    bool verifyResponse(IResponse response, ITokenKey apiKey, String
nonce) {


        String authHeader =
response.getHeaders().getFirst(AUTHORIZATION_HEADER);
        String dateHeader =
response.getHeaders().getFirst(DATE_HEADER);


        String dateStamp = parseDateHeaderToDateStamp(DATE_FORMAT);
        String timeStamp = parseDateHeaderToTimeStamp(TIMESTAMP_FORMAT)


        String newHeader = buildAuthorizationHeader(response, apiKey,
nonce, dateStamp, timeStamp);


        // check the received header matches the digest on the response
        if (authHeader != newHeader) {
            // something is wrong !!!
        }
    }



    String buildAuthorizationHeader(IResponse response, ITokenKey
apiKey, String nonce, String dateStamp, String timeStamp) {


        String canonicalResponse =
buildCanonicalResponseHashHex(response);


        String id = apiKey.getId() + "/" + dateStamp + "/" + nonce +
"/" + ID_TERMINATOR;
```

```
        String canonicalResponseHashHex =
toHex(hash(canonicalResponse));

        String stringToSign =
                        ALGORITHM + NL +
                        timeStamp + NL +
                        id + NL +
                        canonicalResponseHashHex;

        byte[] kDate = apiKey.applyHMac((dateStamp +
AUTHENTICATION_SCHEME).getBytes(DEFAULT_ENCODING));

        byte[] kNonce = sign(nonce, kDate, MacAlgorithm.HmacSHA256);

        byte[] kSigning = sign(ID_TERMINATOR, kNonce,
MacAlgorithm.HmacSHA256);

        byte[] signature = sign(toUtf8Bytes(stringToSign), kSigning,
MacAlgorithm.HmacSHA256);
        String signatureHex = toHex(signature);

        String signedHeadersString = getSignedHeadersString(response);

        String authorizationHeader =
                        AUTHENTICATION_SCHEME + " " +
                        createNameValuePair(DIGEST_ID, id) + ", " +
                        createNameValuePair(DIGEST_SIGNED_HEADERS,
signedHeadersString) + ", " +
                        createNameValuePair(DIGEST_SIGNATURE,
signatureHex);

        return authorizationHeader;
    }



    String buildCanonicalResponseHashHex(IResponse response) {

        String httpStatusString =
Integer.toString(response.getHttpStatus());
```

```
        // the canonical headers string consists of all headers sorted
by name and separated by NL (new line) in the format:
        //
        // headerName1:headerValue11,headerValue12 NL
        // headerName2:headerValue21
        //
        String canonicalHeadersString =
canonicalizeHeadersString(response);

        // signed headers string is a sorted list of all the header
names above separated by ;
        String signedHeadersString = getSignedHeadersString(response);

        // gets the body of the response
        String responsePayload = getResponsePayload(response);
        String responsePayloadHashHex = toHex(hash(responsePayload));

        String canonicalResponse =
                        httpStatusString + NL +
                        canonicalHeadersString + NL +
                        signedHeadersString + NL +
                        responsePayloadHashHex;

        return canonicalResponse;
    }
```