dideryJS Documentation

Release 0.0.1

Brady Hammond, Michael Mendoza

Aug 27, 2018

Table of Contents

dideryJS helps users interface with distributed or soloed didery servers. The didery project aims at bring about proper key management by providing easy to use key management tools. This documentation is divided into three major parts associated with the different types of functions provided in the dideryJS client library:

- **Help** - Cryptographic and general functions
- **API** - Functions used to interface with a didery api
- **Batch** - Functions used to interface with multiple didery api's at once

CHAPTER 1

Help

1.1 concatenateUint8Arrays

This function concatenates two or more Uint8Arrays. Uint8Arrays are typed arrays representing arrays of 8-bit unsigned integers.

1.1.1 Parameters

concatenateUint8Arrays takes two or more Uint8Arrays.

1.1.2 Return

concatenateUint8Arrays returns a newly concatenated Uint8Array.

1.1.3 Example

```
const didery = require('didery');

let array1 = new Uint8Array([1, 2, 3]);
let array2 = new Uint8Array([4, 5, 6]);
let result = didery.concatenateUint8Arrays(array1, array2);
console.log(result)
// Uint8Array[1, 2, 3, 4, 5, 6]

array1 = new Uint8Array([1, 2, 3]);
array2 = new Uint8Array([4, 5, 6]);
let array3 = new Uint8Array([7, 8, 9]);
result = didery.concatenateUint8Arrays(array1, array2, array3);
console.log(result)
// Uint8Array[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

(continues on next page)

(continued from previous page)

```
array1 = new Uint8Array([a, b, c]);
array2 = new Uint8Array([x, y, z]);
result = didery.concatenateUint8Arrays(array1, array2);
console.log(result)
// Uint8Array[97, 98, 99, 120, 121, 122]
```

1.2 stringToBytes

This function converts a string to a Uint8Array. The Uint8Array is made of the string's unicode character codes. Unicode code points range from 0 to 1114111. The first 128 Unicode code points are a direct match of the ASCII character encoding.

1.2.1 Parameters

stringToBytes takes one string. The provided string will be converted to a Uint8Array.

1.2.2 Return

stringToBytes returns one Uint8Array of character bytes. This byte array is derived from the provided string.

1.2.3 Example

```
const didery = require('didery');

let string = "a";
let bytes = didery.stringToBytes(string);
console.log(bytes);
// Uint8Array[97]

string = "Hello World!";
bytes = didery.stringToBytes(string);
console.log(bytes);
// Uint8Array[72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]

string = "brød";
bytes = didery.stringToBytes(string);
console.log(bytes);
// Uint8Array[98, 114, 248, 100]
```

1.3 parseSignatureHeader

This function parses the signature header for a key inception or rotation event. Signature headers follow the format:

```
Signature: headervalue
```

Where headervalue has the format:

```
tag = "signature"
     or
tag = "signature"; tag = "signature"; ...
```

"tag" is the name of a field in the body of the request whose value is a DID from which the public key for the signature can be obtained. If the same tag appears multiple times then only the last occurrence is returned. Each signature value is a doubly quoted string that contains the actual signature in Base64 url safe format. By default the signatures are EdDSA (Ed25519) which are 88 characters long (with two trailing pad bytes) that represent 64 byte EdDSA signatures. An optional tag name = "kind" with values "EdDSA" or "Ed25519" may be present that specifies the type of signature. All signatures within the header must be of the same kind.

1.3.1 Parameters

parseSignatureHeader takes one string. The given string should be a signature to be parsed.

1.3.2 Return

parseSignatureHeader returns an object with the format:

```
{
  "parsed tag": "parsed value",
  "parsed tag": "parsed value",
  ...
}
```

1.3.3 Example

```
const didery = require('didery');

let signature = "signer=
↪'EPk0ZVCWToPu8RhTDR2WrXtrPbP5hikbrEEew0J6cnFwbvzSAF41148o4VX9ziTf-fJH_
↪vsp1dpq0YoL33OBBw==';";
let result = didery.parseSignatureHeader(signature);
console.log(result);
// {"signer": "EPk0ZVCWToPu8RhTDR2WrXtrPbP5hikbrEEew0J6cnFwbvzSAF41148o4VX9ziTf-fJH_
↪vsp1dpq0YoL33OBBw=="}

signature = "signer='EPk0ZVCWToPu8RhTDR2WrXtrPbP5hikbrEEew0J6cnFwbvzSAF41148o4VX9ziTf-
↪fJH_vsp1dpq0YoL33OBBw==';" +
            "rotation=
↪'QXqlf2WzYEqwxN1RBQRTFbOlBtvc6gZqlp54R47nov8j6fx8kAYdJXPlpFAFyrqfaNXRA5Q6vwzM9vbKGRWlCQ==
↪'";
result = didery.parseSignatureHeader(signature);
console.log(result);
// {
//   "signer": "EPk0ZVCWToPu8RhTDR2WrXtrPbP5hikbrEEew0J6cnFwbvzSAF41148o4VX9ziTf-fJH_
↪vsp1dpq0YoL33OBBw==",
//   "rotation":
↪"QXqlf2WzYEqwxN1RBQRTFbOlBtvc6gZqlp54R47nov8j6fx8kAYdJXPlpFAFyrqfaNXRA5Q6vwzM9vbKGRWlCQ==
↪"
// }
```

1.4 checkConsensus

This function compares entries in an array to verify consensus. A specified percentage of entries must be the same for consensus to be reached. The checkConsensus function compares primitives normally and deep compares objects.

1.4.1 Parameters

checkConsensus takes an array of data entries. Data entries can be objects or primitives, but should all be of the same type. There is also an optional consensus level parameter. This should be a float between 0 and 1. The given float represents the consensus threshold that must be surpassed in order to return true. The default consensus level is 1.

1.4.2 Return

checkConsensus returns a boolean. If consensus is reached the function returns true, otherwise the function returns false.

1.4.3 Example

```
const didery = require('didery');

let data = ["abc", "abc"];
let result = didery.checkConsensus(data);
console.log(result);
// true

data = [{"test": "test"}, {"test": "test"}];
result = didery.checkConsensus(data);
console.log(result);
// true

data = ["abc", "abc", "efg"];
let level = 0.5;
result = didery.checkConsensus(data);
// true

data = [{"test": "test"}, {"testing": "testing"}, {"test": "testing"}];
level = 0.5;
result = didery.checkConsensus(data);
// false
```

1.5 getConsensus

This function compares entries in an array to verify consensus and returns the most frequent entry. A specified percentage of entries must be the same for consensus to be reached. The getConsensus function compares primitives normally and deep compares objects.

1.5.1 Parameters

getConsensus takes an array of data entries. Data entries can be objects or primitives, but should all be of the same type. There is also an optional consensus level parameter. This should be a float between 0 and 1. The given float represents the consensus threshold that must be surpassed in order to return true. The default consensus is 1.

1.5.2 Return

getConsensus returns the most frequent data entry if consensus is reached, otherwise the function throws an error.

1.5.3 Example

```
const didery = require('didery');

let data = ["abc", "abc"];
let result = didery.getConsensus(data);
console.log(result);
// "abc"

data = [{"test": "test"}, {"test": "test"}];
result = didery.getConsensus(data);
console.log(result);
// {"test": "test"}

data = ["abc", "abc", "efg"];
let level = 0.5;
result = didery.getConsensus(data);
// "abc"

data = [{"test": "test"}, {"testing": "testing"}, {"test": "testing"}];
level = 0.5;
result = didery.getConsensus(data);
// Error: Consensus unreached
```

1.6 makeDid

This function creates a DID string from a provided public key. DID's can be created for varying method types. This function is asynchronous.

1.6.1 Parameters

makeDid takes one Uint8Array of a 32 byte verifier key from EdDSA (Ed25519) key pair. There is also an optional parameter for the DID method. This is a string that will be saved in the DID under the DID method. The default DID method is "dad".

1.6.2 Return

makeDid returns a promise that when fulfilled returns a DID string of the format: `did:method:base64_encoded_public_key`.

1.6.3 Example

```
const didery = require('didery');

let vk = new Uint8Array([70, 81, 79, 121, 66, 71, 103, 69, 120, 52, 89, 112, 52, 102,␣
↪78, 51, 54, 68, 117, 70, 109,
106, 87, 49, 107, 55, 113, 75, 79, 86, 111, 101 ]);
didery.makeDid(vk).then(function (response) {
     console.log(response);
});
// "did:dad:RlFPeUJHZ0V4NFlwNGZOMzZEdUZtalcxazdxS09Wb2U="

vk = new Uint8Array([70, 81, 79, 121, 66, 71, 103, 69, 120, 52, 89, 112, 52, 102, 78,␣
↪51, 54, 68, 117, 70, 109, 106,
87, 49, 107, 55, 113, 75, 79, 86, 111, 101 ]);
let method = "rep";
didery.makeDid(vk).then(function (response) {
        console.log(response);
});
// "did:rep:RlFPeUJHZ0V4NFlwNGZOMzZEdUZtalcxazdxS09Wb2U="
```

1.7 extractDidParts

This function parses the out the public key from a DID.

1.7.1 Parameters

extractDidParts takes a DID string. There is also an optional parameter for the DID method. The default did method is "dad". If the specified method does not match the DID method an error will be thrown.

1.7.2 Return

extractDidParts returns the public key string from a DID.

1.7.3 Example

```
const didery = require('didery');

let did = "did:dad:RlFPeUJHZ0V4NFlwNGZOMzZEdUZtalcxazdxS09Wb2U=";
let result = didery.extractDidParts(did);
console.log(result);
// "RlFPeUJHZ0V4NFlwNGZOMzZEdUZtalcxazdxS09Wb2U="

did = "did:rep:RlFPeUJHZ0V4NFlwNGZOMzZEdUZtalcxazdxS09Wb2U=";
let method = "rep"
result = didery.extractDidParts(did, method);
console.log(result);
// "RlFPeUJHZ0V4NFlwNGZOMzZEdUZtalcxazdxS09Wb2U="

did = "did:dad:RlFPeUJHZ0V4NFlwNGZOMzZEdUZtalcxazdxS09Wb2U=";
```

(continues on next page)

(continued from previous page)

```
let method = "rep"
result = didery.extractDidParts(did, method);
console.log(result);
// Error: Invalid DID method
```

1.8 generateKeyPair

This function uses libsodium to generate an ed25519 key pair. This function is asynchronous.

1.8.1 Parameters

generateKeyPair has one optional seed parameter. This seed should be 32 bytes and can be either a Uint8Array or a string.

1.8.2 Return

generateKeyPair returns a promise that when fulfilled returns an array of the format: `[Uint8Array[privateKey], Uint8Array[publicKey]]`.

1.8.3 Example

```
const didery = require('didery');

let seed = "FQOyBGgEx4Yp4fN36DuFmjW1k7qKOVoe";
didery.generateKeyPair(seed).then(function(response) {
     console.log(response);
});
// [Uint8Array[70,81,79,121,66,71,103,69,120,52,89,112,52,102,78,51,54,68,117,70,109,
↪106,87,49,107,55,113,75,79,86,
//              111,101,167,185,202,28,236,26,127,61,230,20,129,200,113,50,88,24,161,
↪11,216,134,159,167,151,183,94,25,
//              189,11,128,151,39,237], Uint8Array[167,185,202,28,236,26,127,61,230,20,
↪129,200,113,50,88,24,161,11,
//              216,134,159,167,151,183,94,25,189,11,128,151,39,237]];

seed = new Uint8Array([167,185,202,28,236,26,127,61,230,20,129,200,113,50,88,24,161,
↪11,216,134,159,167,151,183,94,25,
189,11,128,151,39,237]);
didery.generateKeyPair(seed).then(function(response) {
     console.log(response);
});
// [Uint8Array[167,185,202,28,236,26,127,61,230,20,129,200,113,50,88,24,161,11,216,
↪134,159,167,151,183,94,25,189,11,
//              128,151,39,237,178,213,80,122,33,83,170,168,209,242,204,97,178,93,243,
↪193,162,247,179,79,143,4,132,2,
//              32,133,243,119,209,249,189,67], Uint8Array[178,213,80,122,33,83,170,
↪168,209,242,204,97,178,93,243,193,
//              162,247,179,79,143,4,132,2,32,133,243,119,209,249,189,67]];
```

1.9 signResource

This function uses libsodium to sign a stringified resource with a private key. This function is asynchronous.

1.9.1 Parameters

signResource has two parameters. The first is the string of the resource to be signed. The second is a Uint8Array or an un-encoded key string of the signing private key.

1.9.2 Return

signResource returns a promise that when fulfilled returns a base64 encoded signature string.

1.9.3 Example

```
const didery = require('didery');

let resource = "{\"test\":\"test\"}";
let sk = new Uint8Array([167,185,202,28,236,26,127,61,230,20,129,200,113,50,88,24,161,
↪11,216,134,159,167,151,183,94,
                         25,189,11,128,151,39,237,178,213,80,122,33,83,170,168,209,
↪242,204,97,178,93,243,193,162,247,
                         179,79,143,4,132,2,32,133,243,119,209,249,189,67]);
didery.signResource(resource, sk).then(function (response) {
     console.log(response);
});
// "Sr0U_VLeXIxVgzjvFt0syudWw39ox3Yn2MAJQ5oNiUwIpHY_BwlJzM5R2UE81rxi2d_fkJs-
↪uydWM8uoOW86Ag=="

resource = "{\"test\":\"test\"}";
sk = "kAzBIalx6KT22M0CJyw2RqDibDiR0wwPXl5he6SDOIMph7cj2l2W7zigQ2vneaf1";
didery.signResource(resource, sk).then(function (response) {
     console.log(response);
});
// "vSx_vvioFGre8WQYBuwTbeZZEMVrpa6ehuhaflZIGzwdC4tBZOu3QsWh6M013rkyXV1-
↪61Fh7M2qZN3tpWsCBQ=="
```

1.10 verify

This function uses libsodium to check if a signature for a given message can be verified with a provided verification key. This function is asynchronous.

1.10.1 Parameters

verify has three parameters. The first is a Uint8Array of the signature, the second is a string of the signed resource, and the third is a Uint8Array of the verification key.

1.10.2 Return

verify returns a promise that fulfills and returns true if the signature can be verified. Otherwise, the promise is rejected and throws and error.

1.10.3 Example

```
const didery = require('didery');

let signature = new Uint8Array([188,89,18,248,82,201,46,115,209,235,210,41,149,50,159,
↪180,160,116,132,133,125,134,
                                226,208,176,15,83,159,113,216,145,30,157,63,39,51,235,
↪12,209,233,92,5,64,118,42,141,
                                40,58,154,52,155,184,49,132,74,177,123,242,187,69,247,
↪206,115,8]);
let message = "{\"id\":\"did:dad:Ymx_0Ri3Lnuun-bvG_cA32v0Go3KMRZ79eQ-AUQK4ms=\",\
↪"changed\":" +
              "\"2018-07-04T01:46:56-06:00\",\"signer\":0,\"signers\":" +
              "[\"Ymx_0Ri3Lnuun-bvG_cA32v0Go3KMRZ79eQ-AUQK4ms=\"," +
              "\"8qOLfvSJfuDX2YjKh_BAUwSRD-TnDLhEpn9zP7cwf6Q=\"]}";
let vk = new Uint8Array([98,108,127,209,24,183,46,123,174,159,230,239,27,247,0,223,
↪107,244,26,141,202,49,22,123,245,
                         228,62,1,68,10,226,107]);
didery.verify(signature, message, vk).then(function (response) {
     console.log(response);
}).catch(function (error) {
     console.error(error);
});
// true;

signature = new Uint8Array([188,89,18,248,82,201,46,115,209,235,210,41,149,50,159,180,
↪160,116,132,133,125,134,226,
                            208,176,15,83,159,113,216,145,30,157,63,39,51,235,12,209,
↪233,92,5,64,118,42,141,40,58,
                            154,52,155,184,49,132,74,177,123,242,187,69,247,206,115,
↪8]);
message = "{\"test\":\"test\"]}";
vk = new Uint8Array([98,108,127,209,24,183,46,123,174,159,230,239,27,247,0,223,107,
↪244,26,141,202,49,22,123,245,228,
                     62,1,68,10,226,107]);
didery.verify(signature, message, vk).then(function (response) {
     console.log(response);
}).catch(function (error) {
     console.error(error);
});
// Error: incorrect signature for the given public key
```

1.11 verify64u

This function uses libsodium to check if a signature for a given message can be verified with a provided verification key. This function is asynchronous.

1.11.1 Parameters

verify64u has three parameters. The first is a base64 encoded string of the signature, the second is a string of the signed resource, and the third is a base64 encoded string of the verification key.

1.11.2 Return

verify64u returns a promise that fulfills and returns true if the signature can be verified. Otherwise, the promise is rejected and throws and error.

1.11.3 Example

```
const didery = require('didery');

let signature = "vFkS-
↪FLJLnPR69IplTKftKB0hIV9huLQsA9Tn3HYkR6dPycz6wzR6VwFQHYqjSg6mjSbuDGESrF78rtF985zCA==
↪";
let message = "{\"id\":\"did:dad:Ymx_0Ri3Lnuun-bvG_cA32v0Go3KMRZ79eQ-AUQK4ms=\",\
↪"changed\":" +
              "\"2018-07-04T01:46:56-06:00\",\"signer\":0,\"signers\":"+
              "[\"Ymx_0Ri3Lnuun-bvG_cA32v0Go3KMRZ79eQ-AUQK4ms=\"," +
              "\"8qOLfvSJfuDX2YjKh_BAUwSRD-TnDLhEpn9zP7cwf6Q=\"]}";
let vk = "Ymx_0Ri3Lnuun-bvG_cA32v0Go3KMRZ79eQ-AUQK4ms=";
didery.verify64u(signature, message, vk).then(function (response) {
     console.log(response);
}).catch(function (error) {
     console.error(error);
});
// true;

signature = "vFkS-
↪FLJLnPR69IplTKftKB0hIV9huLQsA9Tn3HYkR6dPycz6wzR6VwFQHYqjSg6mjSbuDGESrF78rtF985zCA==
↪";
message = "{\"test\":\"test\"]}";
vk = "Ymx_0Ri3Lnuun-bvG_cA32v0Go3KMRZ79eQ-AUQK4ms=";
didery.verify64u(signature, message, vk).then(function (response) {
     console.log(response);
}).catch(function (error) {
    console.error(error);
});
// Error: incorrect signature for the given public key
```

1.12 toBase64

This function uses libsodium to convert a Uint8Array to a unicode base64 url-file safe string. This function is asynchronous.

1.12.1 Parameters

toBase64 takes one Uint8Array.

1.12.2 Return

toBase64 returns a promise that when fulfilled returns a base64 encoded string.

1.12.3 Example

```
const didery = require('didery');
let key = new Uint8Array([188,89,18,248,82,201,46,115,209,235,210,41,149,50,159,180,
↪160,116,132,133,125,134,226,208,
                          176,15,83,159,113,216,145,30,157,63,39,51,235,12,209,233,92,
↪5,64,118,42,141,40,58,154,52,
                          155,184,49,132,74,177,123,242,187,69,247,206,115,8]);
didery.toBase64(key).then(function (response) {
      console.log(response);
});
// "vFkS-
↪FLJLnPR69IplTKftKB0hIV9huLQsA9Tn3HYkR6dPycz6wzR6VwFQHYqjSg6mjSbuDGESrF78rtF985zCA=="
```

1.13 fromBase64

This function uses libsodium to convert a unicode base64 url-file safe string to a Uint8Array. This function is asynchronous.

1.13.1 Parameters

fromBase64 takes one base64 encoded string.

1.13.2 Return

fromBase64 returns a promise that when fulfilled returns a Uint8Array.

1.13.3 Example

```
const didery = require('didery');
let key = "vFkS-
↪FLJLnPR69IplTKftKB0hIV9huLQsA9Tn3HYkR6dPycz6wzR6VwFQHYqjSg6mjSbuDGESrF78rtF985zCA=="
didery.fromBase64(key).then(function (response) {
      console.log(response);
});
// Uint8Array[188,89,18,248,82,201,46,115,209,235,210,41,149,50,159,180,160,116,132,
↪133,125,134,226,208,176,15,83,159,
              113,216,145,30,157,63,39,51,235,12,209,233,92,5,64,118,42,141,40,58,154,
↪52,155,184,49,132,74,177,123,
              242,187,69,247,206,115,8];
```

1.14 keyInceptionEvent

This function performs a key inception event. In an inception event two keys pairs are created (either manually or automatically generated). The first key pair represents the current key which can be used to sign data. The second key pair represents the pre-rotated key which will replace the current key on a rotation event. An immutable DID is also created from the current key pair. This is used to prove data provenance and track key history. This function also contains a number of options for managing the newly created key pairs and DID. One can post the key data to one or more didery servers, as well as show or save the private keys and DID.

1.14.1 Parameters

keyInceptionEvent has one optional parameter which is an object containing various options. The possible options are as follows:

Option	Description
currentSeed	A 32 byte Uint8Array or string used as the seed for current key pair generation.
preRotated-Seed	A 32 byte Uint8Array or string used as the seed for pre-rotated key pair generation.
currentKey-Pair	An array with a 64 byte Uint8Array of a private key and 32 byte Uint8Array of a public key. This key pair will be used as the current key pair.
preRotated-KeyPair	An array with a 64 byte Uint8Array of a private key and 32 byte Uint8Array of a public key. This key pair will be used as the pre-rotated key pair.
post	A boolean for whether or not key data should be posted to a didery server. If true at least one url must be provided.
urls	An array of comma separated server URLs strings.
saveCurrent	A boolean for whether or not the current private key should be saved somewhere. If true a storage location must be provided.
savePre-Rotated	A boolean for whether or not the pre-rotated private key should be saved somewhere. If true a storage location must be provided.
saveDid	A boolean for whether or not the DID should be saved somewhere. If true a storage location must be provided.
storageCur-rent	The current private key storage location; Accepted values include “local”, “session” or “download”.
storagePre-Rotated	The pre-rotated private key storage location; Accepted values include “local”, “session” or “down-load”.
storageDid	The DID storage location; Accepted values include “local”, “session” or “download”.
showCurrent	A boolean for whether or not to show the current private key in an alert.
showPre-Rotated	A boolean for whether or not to show the pre-rotated private key in an alert.
showDid	A boolean for whether or not to show the DID in an alert.

1.14.2 Return

keyInceptionEvent returns a promise that when fulfilled returns an array with the current key pair and the pre-rotated key pair: `[[[Uint8Array[current private], [Uint8Array[current public]], [[Uint8Array[pre-rotated private], [Uint8Array[pre-rotated public]]]`.

1.14.3 Example

```
const didery = require('didery');

let options = {};
options.currentSeed = "FQOyBGgEx4Yp4fN36DuFmjW1k7qKOVoe";
options.preRotatedSeed = "FQOyBGgEx4Yp4fN36DuFmjW1k7qKOVoe";
didery.keyInceptionEvent(options).then(function (response) {
     console.log(response);
});
// [[Uint8Array[70,81,79,121,66,71,103,69,120,52,89,112,52,102,78,51,54,68,117,70,109,
↪106,87,49,107,55,113,75,79,86,
// 111,101,167,185,202,28,236,26,127,61,230,20,129,200,113,50,88,24,161,11,216,134,
↪159,167,151,183,94,25,189,11,128,
// 151,39,237], Uint8Array[167,185,202,28,236,26,127,61,230,20,129,200,113,50,88,24,
↪161,11,216,134,159,167,151,183,
// 94,25,189,11,128,151,39,237]], [Uint8Array[70,81,79,121,66,71,103,69,120,52,89,112,
↪52,102,78,51,54,68,117,70,109,
// 106,87,49,107,55,113,75,79,86,111, 101,167,185,202,28,236,26,127,61,230,20,129,200,
↪113,50,88,24,161,11,216,134,
// 159,167,151,183,94,25,189,11,128,151,39,237], Uint8Array[167,185,202,28,236,26,127,
↪61,230,20,129,200,113,50,88,24,
// 161,11,216,134,159,167,151,183,94,25,189,11,128,151,39,237]]]

options = {};
options.currentSeed = "FQOyBGgEx4Yp4fN36DuFmjW1k7qKOVoe";
options.preRotatedSeed = "FQOyBGgEx4Yp4fN36DuFmjW1k7qKOVoe";
options.post = true;
options.urls = ["http://127.0.0.1:8080/"];
options.saveCurrent = true;
options.savePreRotated = true;
options.saveDid = true;
options.storageCurrent = "local";
options.storagePreRotated = "local";
options.storageDid ="local";
options.showCurrent = false;
options.showPreRotated = false;
options.showDid = false;
didery.keyInceptionEvent(options).then(function (response) {
     console.log(response);
});
// [[Uint8Array[70,81,79,121,66,71,103,69,120,52,89,112,52,102,78,51,54,68,117,70,109,
↪106,87,49,107,55,113,75,79,86,
// 111,101,167,185,202,28,236,26,127,61,230,20,129,200,113,50,88,24,161,11,216,134,
↪159,167,151,183,94,25,189,11,128,
// 151,39,237], Uint8Array[167,185,202,28,236,26,127,61,230,20,129,200,113,50,88,24,
↪161,11,216,134,159,167,151,183,
// 94,25,189,11,128,151,39,237]], [Uint8Array[70,81,79,121,66,71,103,69,120,52,89,112,
↪52,102,78,51,54,68,117,70,109,
// 106,87,49,107,55,113,75,79,86,111, 101,167,185,202,28,236,26,127,61,230,20,129,200,
↪113,50,88,24,161,11,216,134,
// 159,167,151,183,94,25,189,11,128,151,39,237], Uint8Array[167,185,202,28,236,26,127,
↪61,230,20,129,200,113,50,88,24,
// 161,11,216,134,159,167,151,183,94,25,189,11,128,151,39,237]]]
```

1.15 keyRotationEvent

This function performs a key rotation event. In a rotation event the previous pre-rotated key becomes the new current key and a new pre-rotated key is created (either manually or automatically generated). After a rotation event the old current key can no longer be used to sign data. This function also contains a number of options for managing the key pairs. One can post the new key data to one or more didery servers, as well as show or save the private keys.

1.15.1 Parameters

keyRotationEvent has three required parameters. The first is a 64 byte Uint8Array of the current private key, the second is a 64 byte Uint8Array of the pre-rotated key, and the third is the DID string associated with the given keys. This function also has one optional parameter which is an object containing various options. The possible options are as follows:

Option	Description
seed	A 32 byte Uint8Array or string used as seed for the new pre-rotated key pair generation. key pair will be used as the current key pair.
preRotated-KeyPair	An array with a 64 byte Uint8Array of a private key and 32 byte Uint8Array of a public key. This key pair will be used as the new pre-rotated key pair.
post	A boolean for whether or not key data should be posted to a didery server.
consensus	A float between 0 and 1 for the consensus threshold used when retrieving key history (key history is only retrieved when posting data).
urls	An array of comma separated server URLs strings.
saveCurrent	A boolean for whether or not the new current private key should be saved somewhere. If true a storage location must be provided.
savePre-Rotated	A boolean for whether or not the bew pre-rotated private key should be saved somewhere. If true a storage location must be provided.
storageCur-rent	The current private key storage location; Accepted values include “local”, “session” or “download”.
storagePre-Rotated	The pre-rotated private key storage location; Accepted values include “local”, “session” or “down-load”.
showCur-rent	A boolean for whether or not to show the new current private key in an alert.
showPre-Rotated	A boolean for whether or not to show the new pre-rotated private key in an alert.

1.15.2 Return

keyRotationEvent returns a promise that when fulfilled returns an array with new pre-rotated key pair: `[Uint8Array[pre-rotated private], Uint8Array[pre-rotated public]]`.

1.15.3 Example

```
const didery = require('didery');

let oldk = new Uint8Array([70,81,79,121,66,71,103,69,120,52,89,112,52,102,78,51,54,68,
↪117,70,109,106,87,49,107,55,113,
                           75,79,86,111,101,167,185,202,28,236,26,127,61,230,20,129,
↪200,113,50,88,24,161,11,216,134,
```

(continues on next page)

(continued from previous page)

```
                          159,167,151,183,94,25,189,11,128,151,39,237]);
let newk = new Uint8Array([188,89,18,248,82,201,46,115,209,235,210,41,149,50,159,180,
↪160,116,132,133,125,134,226,208,
                          176,15,83,159,113,216,145,30,157,63,39,51,235,12,209,233,92,
↪5,64,118,42,141,40,58,154,52,
                          155,184,49,132,74,177,123,242,187,69,247,206,115,8]);
let did = "did:dad:wqfCucOKHMOsGn89w6YUwoHDiHEyWBjCoQvDmMKGwp="
let options = {};
options.seed = "FQOyBGgEx4Yp4fN36DuFmjW1k7qKOVoe";
didery.keyRotationEvent(oldk, newk, did, options).then(function (response) {
    console.log(response);
});
// [Uint8Array[70,81,79,121,66,71,103,69,120,52,89,112,52,102,78,51,54,68,117,70,109,
// 106,87,49,107,55,113,75,79,86,111,  101,167,185,202,28,236,26,127,61,230,20,129,200,
↪113,50,88,24,161,11,216,134,
// 159,167,151,183,94,25,189,11,128,151,39,237], Uint8Array[167,185,202,28,236,26,127,
↪61,230,20,129,200,113,50,88,24,
// 161,11,216,134,159,167,151,183,94,25,189,11,128,151,39,237]]

oldk = new Uint8Array([70,81,79,121,66,71,103,69,120,52,89,112,52,102,78,51,54,68,117,
↪70,109,106,87,49,107,55,113,75,
                       79,86,111,101,167,185,202,28,236,26,127,61,230,20,129,200,113,
↪50,88,24,161,11,216,134,159,167,
                       151,183,94,25,189,11,128,151,39,237]);
newk Uint8Array([188,89,18,248,82,201,46,115,209,235,210,41,149,50,159,180,160,116,
↪132,133,125,134,226,208,176,15,83,
                 159,113,216,145,30,157,63,39,51,235,12,209,233,92,5,64,118,42,141,40,
↪58,154,52,155,184,49,132,74,177,
                 123,242,187,69,247,206,115,8]);
did = "did:dad:wqfCucOKHMOsGn89w6YUwoHDiHEyWBjCoQvDmMKGwp="
options = {};
options.seed = "FQOyBGgEx4Yp4fN36DuFmjW1k7qKOVoe";
options.post = true;
options.consensus = 0.75;
options.urls = ["http://127.0.0.1:8080/"];
options.saveCurrent = true;
options.savePreRotated = true;
options.storageCurrent = "local";
options.storagePreRotated = "local";
options.showCurrent = false;
options.showPreRotated = false;
didery.keyRotationEvent(oldk, newk, did, options).then(function (response) {
    console.log(response);
});
// [Uint8Array[70,81,79,121,66,71,103,69,120,52,89,112,52,102,78,51,54,68,117,70,109,
// 106,87,49,107,55,113,75,79,86,111,  101,167,185,202,28,236,26,127,61,230,20,129,200,
↪113,50,88,24,161,11,216,134,
// 159,167,151,183,94,25,189,11,128,151,39,237], Uint8Array[167,185,202,28,236,26,127,
↪61,230,20,129,200,113,50,88,24,
// 161,11,216,134,159,167,151,183,94,25,189,11,128,151,39,237]]
```

1.16 keyRevocationEvent

This function performs a key revocation event. In a revocation event a key is revoked by rotating to a null key. After a revocation event the current key can no longer be used to sign data. The pre-rotated key also becomes void and can

no longer be rotated to. Revocation does not delete a key history. Data provenance can therefore still be verified using the revoked key's history.

1.16.1 Parameters

keyRevocationEvent has three required parameters. The first is a 64 byte Uint8Array of the current private key, the second is a 64 byte Uint8Array of the pre-rotated key, and the third is the DID string associated with the given keys. This function also has one optional parameter which is an object containing various options. The possible options are as follows:

Option	Description
consensus	A float between 0 and 1 for the consensus threshold used when retrieving key history (key history is only retrieved when posting data).
urls	An array of comma separated server URLs strings.

1.16.2 Return

keyRevocationEvent returns true if the revocation was successful or throws an error if the revocation failed.

1.16.3 Example

```
const didery = require('didery');

let currentKey = new Uint8Array([70,81,79,121,66,71,103,69,120,52,89,112,52,102,78,51,
↪54,68,117,70,109,106,87,49,107,55,113,
                                 75,79,86,111,101,167,185,202,28,236,26,127,61,230,20,129,
↪200,113,50,88,24,161,11,216,134,
                                 159,167,151,183,94,25,189,11,128,151,39,237]);
let preRotatedKey = new Uint8Array([188,89,18,248,82,201,46,115,209,235,210,41,149,50,
↪159,180,160,116,132,133,125,134,226,208,
                                 176,15,83,159,113,216,145,30,157,63,39,51,235,12,209,233,92,
↪5,64,118,42,141,40,58,154,52,
                                 155,184,49,132,74,177,123,242,187,69,247,206,115,8]);
let did = "did:dad:wqfCucOKHMOsGn89w6YUwoHDiHEyWBjCoQvDmMKGwp="
let options = {};
options.consensus = 0.75;
options.urls = ["http://127.0.0.1:8080/"];
didery.keyRevocationEvent(currentKey, preRotatedKey, did, options).then(function␣
↪(response) {
    console.log(response);
});
// true
```

CHAPTER 2

API

2.1 getHistory

This function uses fetch to hit the GET history endpoint of a didery server. The GET history endpoint returns key histories. This functions is asynchronous.

2.1.1 Parameters

getHistory has two optional parameters. First is a string for the server's base URL. The default base URL is the localhost at port 8080. Second is a DID string. If this is supplied, getHistory will retrieve the key history for that DID. The default DID string is an empty string. If no DID string is provided, getHistory will retrieve all key histories.

2.1.2 Return

getHistory returns a promise that when fulfilled returns the server's response to the fetch operation.

2.1.3 Example

```
const didery = require('didery');

let baseURL = "http://myDideryServer.com";
didery.getHistory(baseURL).then(function (response) {
     // Do something with response
}).catch(function (error) {
     console.error(error);
});

baseURL = "http://127.0.0.1:8000";
let did = "did:dad:RlFPeUJHZ0V4NFlwNGZOMzZEdUZtalcxazdxS09Wb2U=";
didery.getHistory(baseURL).then(function (response) {
```

(continues on next page)

(continued from previous page)

```
    // Do something with response
}).catch(function (error) {
    console.error(error);
});
```

2.2 postHistory

This function uses fetch to hit the POST history endpoint of a didery server. The POST history endpoint saves new key histories. Data posted to the servers will be verified against signatures. If any discrepancies are found the operation will fail. This function is asynchronous.

2.2.1 Parameters

postHistory has two required parameters. First is a signature string that will be placed in the required signature header. This string should be of the format:

```
'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";'
```

or:

```
'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪"; kind="ed25519"; ...'
```

Second is an object containing the data to be posted. This data object should be of the format:

```
{
    "id": DID string,
    "changed": datetime string,
    "signer": integer representing the current key index,
    "signers": [current public key, pre-rotated public key]
}
```

There is also an optional base URL parameter. This is a string of the server's base URL. The default base URL is the localhost at port 8080.

2.2.2 Return

postHistory returns a promise that when fulfilled returns the server's response to the fetch operation.

2.2.3 Example

```
const didery = require('didery');

let baseURL = "http://myDideryServer.com";
let signature = 'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";';
```

(continues on next page)

(continued from previous page)

```
let data = {
     "id": "did:dad:Ymx_0Ri3Lnuun-bvG_cA32v0Go3KMRZ79eQ-AUQK4ms=",
     "changed": "2018-07-04T01:46:56-06:00",
     "signer": 0,
     "signers": [
         "Ymx_0Ri3Lnuun-bvG_cA32v0Go3KMRZ79eQ-AUQK4ms=",
         "8qOLfvSJfuDX2YjKh_BAUwSRD-TnDLhEpn9zP7cwf6Q="
     ]
};
didery.postHistory(signature, data, baseURL).then(function (response) {
     // Do something with response
}).catch(function (error) {
    console.error(error);
});
```

2.3 putHistory

This function uses fetch to hit the PUT history endpoint of a didery server. The POST history endpoint updates key histories. The PUT endpoint should only be used for key rotation and revocation events. Data put to the server will be verified against signatures and old key history. If any discrepancies are found the operation will fail. This function is asynchronous.

2.3.1 Parameters

putHistory has three required parameters. First is a signature string that will be placed in the required signature header. This string should be of the format:

```
'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";' +
'rotation="QpQkEmBLmC77ZT30I77sqw-gbkZGQSwC8OpCIA1p6OBdXxvycdW35a5_TgASrraRrLzQMg8bxZ-
↪W-KEXLfTAAg==";'
```

or:

```
'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";' +
'rotation="QpQkEmBLmC77ZT30I77sqw-gbkZGQSwC8OpCIA1p6OBdXxvycdW35a5_TgASrraRrLzQMg8bxZ-
↪W-KEXLfTAAg==";' +
'kind="ed25519"; ...'
```

Second is an object containing the data to be posted. This data object should be of the format:

```
{
    "id": DID string,
    "changed": datetime string,
    "signer": integer representing the current key index,
    "signers": [old public key, ..., current public key, pre-rotated public key]
}
```

Third is a DID string. There is also an optional base URL parameter. This is a string of the server's base URL. The default base URL is the localhost at port 8080.

2.3.2 Return

putHistory returns a promise that when fulfilled returns the server's response to the fetch operation.

2.3.3 Example

```
const didery = require('didery');

let baseURL = "http://myDideryServer.com";
let signature = 'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";' +
                'rotation="h6n48a7RgjHJNETKw82SWaNacYuS04ddc_lZtlijlG071GH0_
↪90T2hyaprcBt2XM7VfKDjp2OnSteNNptFazDQ==";';
let data = {
     "id": "did:dad:Ymx_0Ri3Lnuun-bvG_cA32v0Go3KMRZ79eQ-AUQK4ms=",
     "changed": "2018-07-04T01:46:56-06:00",
     "signer": 1,
     "signers": [
         "Ymx_0Ri3Lnuun-bvG_cA32v0Go3KMRZ79eQ-AUQK4ms=",
         "8qOLfvSJfuDX2YjKh_BAUwSRD-TnDLhEpn9zP7cwf6Q=",
         "kAzBIalx6KT22M0CJyw2RqDibDiR0wwPXl5he6SDOIM="
     ]
};
didery.putHistory(signature, data, baseURL).then(function (response) {
     // Do something with response
}).catch(function (error) {
    console.error(error);
});
```

2.4 deleteHistory

This function uses fetch to hit the DELETE history endpoint of a didery server. The DELETE history endpoint deletes a specific key history. Verification data will be checked against the given signature. If any discrepancies are found the operation will fail. This function is asynchronous. Data should be signed with your current private key unless you have previously revoked your key. In such a case data should be signed with your last pre-rotated key.

2.4.1 Parameters

deleteHistory has three required parameters. First is a signature string that will be placed in the required signature header. This string should be of the format:

```
'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";'
```

or:

```
'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";' +
'kind="ed25519"; ...'
```

Second is an object containing the DID verification data. This data object should be of the format:

```
{
    "id": DID string
}
```

Third is a DID string. There is also an optional base URL parameter. This is a string of the server's base URL. The default base URL is the localhost at port 8080.

2.4.2 Return

deleteHistory returns a promise that when fulfilled returns the server's response to the fetch operation.

2.4.3 Example

```
const didery = require('didery');

let baseURL = "http://myDideryServer.com";
let signature = 'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";';
let data = {
     "id": "did:dad:Ymx_0Ri3Lnuun-bvG_cA32v0Go3KMRZ79eQ-AUQK4ms="
};
didery.deleteHistory(signature, data, baseURL).then(function (response) {
     // Do something with response
}).catch(function (error) {
    console.error(error);
});
```

2.5 getBlobs

This function uses fetch to hit the GET blobs endpoint of a didery server. The GET blobs endpoint returns encrypted private key blobs.This functions is asynchronous.

2.5.1 Parameters

getBlobs has two optional parameters. First is a string for the server's base URL. The default base URL is the localhost at port 8080. Second is a DID string. If this is supplied, getHistory will retrieve the key history for that DID. The default DID string is an empty string. If no DID string is provided, getBlobs will retrieve all encrypted key blobs.

2.5.2 Return

getBlobs returns a promise that when fulfilled returns the server's response to the fetch operation.

2.5.3 Example

```
const didery = require('didery');

let baseURL = "http://myDideryServer.com";
didery.getBlobs(baseURL).then(function (response) {
     // Do something with response
}).catch(function (error) {
     console.error(error);
});

baseURL = "http://127.0.0.1:8000";
let did = "did:dad:RlFPeUJHZ0V4NFlwNGZOMzZEdUZtalcxazdxS09Wb2U=";
didery.getBlobs(baseURL).then(function (response) {
     // Do something with response
}).catch(function (error) {
     console.error(error);
});
```

2.6 postBlobs

This function uses fetch to hit the POST blobs endpoint of a didery server. The POST blobs endpoint saves new encrypted private key blobs. Data posted to the servers will be verified against signatures. If any discrepancies are found the operation will fail. This function is asynchronous.

2.6.1 Parameters

postBlobs has two required parameters. First is a signature string that will be placed in the required signature header. This string should be of the format:

```
'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";'
```

or:

```
'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪"; kind="ed25519"; ...'
```

Second is an object containing the data to be posted. This data object should be of the format:

```
{
    "blob": encrypted blob,
    "changed": datetime string,
    "id": DID string
}
```

There is also an optional base URL parameter. This is a string of the server's base URL. The default base URL is the localhost at port 8080.

2.6.2 Return

postBlobs returns a promise that when fulfilled returns the server's response to the fetch operation.

2.6.3 Example

```
const didery = require('didery');

let baseURL = "http://myDideryServer.com";
let signature = 'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";';
let data = {
     "blob":
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCgo9yjuKHHNJZF
↪4D-7s3CcYmuoWAh6NVtYaf_GWw_2sCrHBAA2mAEsml3thLmu50Dw",
     "changed": "2000-01-01T00:00:00+00:00",
     "id": "did:dad:Qt27fThWoNZsa88VrTkep6H-4HA8tr54sHON1vWl6FE="
};
didery.postBlobs(signature, data, baseURL).then(function (response) {
     // Do something with response
}).catch(function (error) {
    console.error(error);
});
```

2.7 putBlobs

This function uses fetch to hit the PUT blobs endpoint of a didery server. The PUT blobs endpoint updates encrypted key blobs. The PUT endpoint should only be used to update the current key after key rotation and revocation events. Data put to the server will be verified against signatures. If any discrepancies are found the operation will fail. This function is asynchronous.

2.7.1 Parameters

putBlobs has three required parameters. First is a signature string that will be placed in the required signature header. This string should be of the format:

```
'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";'
```

or:

```
'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";'kind="ed25519"; ...'
```

Second is an object containing the data to be posted. This data object should be of the format:

```
{
    "blob":
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCgo9yjuKHHNJZF
↪4D-7s3CcYmuoWAh6NVtYaf_GWw_2sCrHBAA2mAEsml3thLmu50Dw",
    "changed": "2000-01-01T00:00:00+00:00",
    "id": "did:dad:Qt27fThWoNZsa88VrTkep6H-4HA8tr54sHON1vWl6FE="
}
```

Third is a DID string. There is also an optional base URL parameter. This is a string of the server's base URL. The default base URL is the localhost at port 8080.

2.7.2 Return

putBlobs returns a promise that when fulfilled returns the server's response to the fetch operation.

2.7.3 Example

```
const didery = require('didery');

let baseURL = "http://myDideryServer.com";
let signature = 'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";';
let data = {
     "blob":
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCgo9yjuKHHNJZ
↪4D-7s3CcYmuoWAh6NVtYaf_GWw_2sCrHBAA2mAEsml3thLmu50Dw",
     "changed": "2000-01-01T00:00:00+00:00",
     "id": "did:dad:Qt27fThWoNZsa88VrTkep6H-4HA8tr54sHON1vWl6FE="
};
didery.putBlobs(signature, data, baseURL).then(function (response) {
     // Do something with response
}).catch(function (error) {
    console.error(error);
});
```

2.8 getRelays

This function uses fetch to hit the GET relays endpoint of a didery server. The GET relays endpoint returns trusted servers. This functions is asynchronous.

2.8.1 Parameters

getRelays has one optional parameter. It is a string for the server's base URL. The default base URL is the localhost at port 8080.

2.8.2 Return

getRelays returns a promise that when fulfilled returns the server's response to the fetch operation.

2.8.3 Example

```
const didery = require('didery');

let baseURL = "http://myDideryServer.com";
didery.getRelays(baseURL).then(function (response) {
```

(continues on next page)

(continued from previous page)

```
    // Do something with response
}).catch(function (error) {
    console.error(error);
});
```

2.9 postRelays

This function uses fetch to hit the POST relays endpoint of a didery server. The POST relays endpoint saves new trusted servers. This function is asynchronous.

2.9.1 Parameters

postRelays has one required data parameter. It takes an object and should have the following format:

```
{
    "changed": datetime string,
    "host_address": URL string,
    "main": boolean,
    "name": name string,
    "port": port string
}
```

There is also an optional base URL parameter. This is a string of the server's base URL. The default base URL is the localhost at port 8080.

2.9.2 Return

postRelays returns a promise that when fulfilled returns the server's response to the fetch operation.

2.9.3 Example

```
const didery = require('didery');

let baseURL = "http://myDideryServer.com";
let data = {
    "changed": "2000-01-01T00:00:00+00:00",
    "host_address": "127.0.0.1",
    "main": true,
    "name": "alpha",
    "port": "7541"
};
didery.postRelays(data, baseURL).then(function (response) {
    // Do something with response
}).catch(function (error) {
    console.error(error);
});
```

2.10 putRelays

This function uses fetch to hit the PUT relays endpoint of a didery server. The PUT history endpoint updates trusted servers. This function is asynchronous.

2.10.1 Parameters

putRelays has two required parameters. First is a data parameter that takes an object with the following format:

```
{
    "changed": datetime string,
    "host_address": URL string,
    "main": boolean,
    "name": name string,
    "port": port string
}
```

Second is a uid paramter that takes a uid string of the server to be updated. There is also an optional base URL parameter. This is a string of the server's base URL. The default base URL is the localhost at port 8080.

2.10.2 Return

putRelays returns a promise that when fulfilled returns the server's response to the fetch operation.

2.10.3 Example

```
const didery = require('didery');

let baseURL = "http://myDideryServer.com";
let data = {
    "changed": "2000-01-01T00:00:00+00:00",
    "host_address": "127.0.0.1",
    "main": true,
    "name": "alpha",
    "port": "7541"
};
let uid = "1";
didery.putRelays(data, uid, baseURL).then(function (response) {
     // Do something with response
}).catch(function (error) {
    console.error(error);
});
```

2.11 deleteRelays

This function uses fetch to hit the DELETE relays endpoint of a didery server. The DELETE relays endpoint deletes trusted server data. This function is asynchronous.

2.11.1 Parameters

deleteRelays has one required uid parameter. It takes the string of a relay's uid. There is also an optional base URL parameter. This is a string of the server's base URL. The default base URL is the localhost at port 8080.

2.11.2 Return

deleteRelays returns a promise that when fulfilled returns the server's response to the fetch operation.

2.11.3 Example

```
const didery = require('didery');

let baseURL = "http://myDideryServer.com";
let uid = "1";
didery.deleteRelays(uid, baseURL).then(function (response) {
     // Do something with response
}).catch(function (error) {
    console.error(error);
});
```

2.12 getErrors

This function uses fetch to hit the GET errors endpoint of a didery server. The GET errors endpoint returns any errors raised by the server. This functions is asynchronous.

2.12.1 Parameters

getErrors has one optional parameter. It is a string for the server's base URL. The default base URL is the localhost at port 8080.

2.12.2 Return

getErrors returns a promise that when fulfilled returns the server's response to the fetch operation.

2.12.3 Example

```
const didery = require('didery');

let baseURL = "http://myDideryServer.com";
didery.getErrors(baseURL).then(function (response) {
     // Do something with response
}).catch(function (error) {
     console.error(error);
});
```

CHAPTER 3

Batch

3.1 batchGetHistory

This function uses fetch to hit the GET history endpoint of multiple didery servers. The GET history endpoint returns key histories. This functions is asynchronous.

3.1.1 Parameters

batchGetHistory has one required urls parameters. This takes an array of URL strings. There is also an optional did parameter. This takes a DID string, and if supplied, batchGetHistory will retrieve the key history for that DID. The default DID string is an empty string. If no DID string is provided, batchGetHistory will retrieve all key histories.

3.1.2 Return

batchGetHistory returns a promise that when fulfilled returns the servers' responses to the fetch operations.

3.1.3 Example

```
const didery = require('didery');

let urls = [
     "http://myDideryServer.com",
     "http://anotherDideryServer.com",
     "http://oneMoreServer.com"
];
didery.batchGetHistory(urls).then(function (response) {
     // Do something with response
}).catch(function (error) {
     console.error(error);
});
```

(continues on next page)

(continued from previous page)

```
urls = [
     "http://myDideryServer.com",
     "http://anotherDideryServer.com",
     "http://oneMoreServer.com"
];
let did = "did:dad:RlFPeUJHZ0V4NFlwNGZOMzZEdUZtalcxazdxS09Wb2U=";
didery.batchGetHistory(urls, did).then(function (response) {
     // Do something with response
}).catch(function (error) {
     console.error(error);
});
```

3.2 batchPostHistory

This function uses fetch to hit the POST history endpoint of multiple didery servers. The POST history endpoint saves new key histories. Data posted to the server will be verified against signatures. If any discrepancies are found the operation will fail. This function is asynchronous.

3.2.1 Parameters

batchPostHistory has three required parameters. First is a signature string that will be placed in the required signature header. This string should be of the format:

```
'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";'
```

or:

```
'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪"; kind="ed25519"; ...'
```

Second is an object containing the data to be posted. This data object should be of the format:

```
{
    "id": DID string,
    "changed": datetime string,
    "signer": integer representing the current key index,
    "signers": [current public key, pre-rotated public key]
}
```

Third is a urls parameter that takes an array of URL strings.

3.2.2 Return

batchPostHistory returns a promise that when fulfilled returns the servers' responses to the fetch operations.

3.2.3 Example

```
const didery = require('didery');

let urls = [
     "http://myDideryServer.com",
     "http://anotherDideryServer.com",
     "http://oneMoreServer.com"
];
let signature = 'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";';
let data = {
     "id": "did:dad:Ymx_0Ri3Lnuun-bvG_cA32v0Go3KMRZ79eQ-AUQK4ms=",
     "changed": "2018-07-04T01:46:56-06:00",
     "signer": 0,
     "signers": [
         "Ymx_0Ri3Lnuun-bvG_cA32v0Go3KMRZ79eQ-AUQK4ms=",
         "8qOLfvSJfuDX2YjKh_BAUwSRD-TnDLhEpn9zP7cwf6Q="
     ]
};
didery.batchPostHistory(signature, data, urls).then(function (response) {
     // Do something with response
}).catch(function (error) {
    console.error(error);
});
```

3.3 batchPutHistory

This function uses fetch to hit the PUT history endpoint of multiple didery servers. The POST history endpoint updates key histories The PUT endpoint should only be used for key rotation and revocation events. Data put to the servers will be verified against signatures and old key history. If any discrepancies are found the operation will fail. This function is asynchronous.

3.3.1 Parameters

batchPutHistory has four required parameters. First is a signature string that will be placed in the required signature header. This string should be of the format:

```
'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";' +
'rotation="QpQkEmBLmC77ZT30I77sqw-gbkZGQSwC8OpCIA1p6OBdXxvycdW35a5_TgASrraRrLzQMg8bxZ-
↪W-KEXLfTAAg==";'
```

or:

```
'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";' +
'rotation="QpQkEmBLmC77ZT30I77sqw-gbkZGQSwC8OpCIA1p6OBdXxvycdW35a5_TgASrraRrLzQMg8bxZ-
↪W-KEXLfTAAg==";' +
'kind="ed25519"; ...'
```

Second is an object containing the data to be posted. This data object should be of the format:

```
{
    "id": DID string,
    "changed": datetime string,
    "signer": integer representing the current key index,
    "signers": [old public key, ..., current public key, pre-rotated public key]
}
```

Third is a did parameter that takes a DID string. Fourth is a urls parameter that takes an array of URL strings.

3.3.2 Return

batchPutHistory returns a promise that when fulfilled returns the servers' responses to the fetch operations.

3.3.3 Example

```
const didery = require('didery');

let urls = [
     "http://myDideryServer.com",
     "http://anotherDideryServer.com",
     "http://oneMoreServer.com"
];
let signature = 'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";' +
                'rotation="h6n48a7RgjHJNETKw82SWaNacYuS04ddc_lZtlijlG071GH0_
↪90T2hyaprcBt2XM7VfKDjp2OnSteNNptFazDQ==";';
let data = {
     "id": "did:dad:Ymx_0Ri3Lnuun-bvG_cA32v0Go3KMRZ79eQ-AUQK4ms=",
     "changed": "2018-07-04T01:46:56-06:00",
     "signer": 1,
     "signers": [
         "Ymx_0Ri3Lnuun-bvG_cA32v0Go3KMRZ79eQ-AUQK4ms=",
         "8qOLfvSJfuDX2YjKh_BAUwSRD-TnDLhEpn9zP7cwf6Q=",
         "kAzBIalx6KT22M0CJyw2RqDibDiR0wwPXl5he6SDOIM="
     ]
};
didery.batchPutHistory(signature, data, urls).then(function (response) {
     // Do something with response
}).catch(function (error) {
    console.error(error);
});
```

3.4 batchDeleteHistory

This function uses fetch to hit the DELETE history endpoint of multiple didery servers. The DELETE history endpoint deletes a specific key history. Verification data will be checked against the given signature. If any discrepancies are found the operation will fail. This function is asynchronous. Data should be signed with your current private key unless you have previously revoked your key. In such a case data should be signed with your last pre-rotated key.

3.4.1 Parameters

batchDeleteHistory has four required parameters. First is a signature string that will be placed in the required signature header. This string should be of the format:

```
'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";'
```

or:

```
'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";' +
'kind="ed25519"; ...'
```

Second is an object containing the DID verification data. This data object should be of the format:

```
{
    "id": DID string
}
```

Third is a did parameter that takes a DID string. Fourth is a urls parameter that takes an array of URL strings.

3.4.2 Return

batchDeleteHistory returns a promise that when fulfilled returns the servers' responses to the fetch operations.

3.4.3 Example

```
const didery = require('didery');

let urls = [
     "http://myDideryServer.com",
     "http://anotherDideryServer.com",
     "http://oneMoreServer.com"
];
let signature = 'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";' +
                'rotation="h6n48a7RgjHJNETKw82SWaNacYuS04ddc_lZtlijlG071GH0_
↪90T2hyaprcBt2XM7VfKDjp2OnSteNNptFazDQ==";';
let data = {
     "id": "did:dad:Ymx_0Ri3Lnuun-bvG_cA32v0Go3KMRZ79eQ-AUQK4ms="
};
didery.batchDeleteHistory(signature, data, urls).then(function (response) {
     // Do something with response
}).catch(function (error) {
    console.error(error);
});
```

3.5 batchGetBlobs

This function uses fetch to hit the GET blobs endpoint of multiple didery servers. The GET blobs endpoint returns encrypted private key blobs.This functions is asynchronous.

3.5.1 Parameters

batchGetBlobs has one required urls parameters. This takes an array of URL strings. There is also an optional did parameter. This takes a DID string, and if supplied, batchGetBlobs will retrieve the encrypted blobs for that DID. The default DID string is an empty string. If no DID string is provided, batchGetBlobs will retrieve all encrypted blobs.

3.5.2 Return

batchGetBlobs returns a promise that when fulfilled returns the servers' responses to the fetch operations.

3.5.3 Example

```
const didery = require('didery');

let urls = [
     "http://myDideryServer.com",
     "http://anotherDideryServer.com",
     "http://oneMoreServer.com"
];
didery.batchGetBlobs(urls).then(function (response) {
     // Do something with response
}).catch(function (error) {
     console.error(error);
});

urls = [
     "http://myDideryServer.com",
     "http://anotherDideryServer.com",
     "http://oneMoreServer.com"
];
let did = "did:dad:RlFPeUJHZ0V4NFlwNGZOMzZEdUZtalcxazdxS09Wb2U=";
didery.batchGetBlobs(urls).then(function (response) {
     // Do something with response
}).catch(function (error) {
     console.error(error);
});
```

3.6 batchPostBlobs

This function uses fetch to hit the POST blobs endpoint of multiple didery servers. The POST blobs endpoint saves new encrypted private key blobs. Data posted to the server will be verified against signatures. If any discrepancies are found the operation will fail. This function is asynchronous.

3.6.1 Parameters

batchPostBlobs has three required parameters. First is a signature string that will be placed in the required signature header. This string should be of the format:

```
'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";'
```

or:

```
'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪"; kind="ed25519"; ...'
```

Second is an object containing the data to be posted. This data object should be of the format:

```
{
    "blob": encrypted blob,
    "changed": datetime string,
    "id": DID string
}
```

Third is a urls parameter that takes an array of URL strings.

3.6.2 Return

batchPostBlobs returns a promise that when fulfilled returns the servers' responses to the fetch operations.

3.6.3 Example

```
const didery = require('didery');

let urls = [
     "http://myDideryServer.com",
     "http://anotherDideryServer.com",
     "http://oneMoreServer.com"
];
let signature = 'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";';
let data = {
     "blob":
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCgo9yjuKHHNJZF
↪4D-7s3CcYmuoWAh6NVtYaf_GWw_2sCrHBAA2mAEsml3thLmu50Dw",
     "changed": "2000-01-01T00:00:00+00:00",
     "id": "did:dad:Qt27fThWoNZsa88VrTkep6H-4HA8tr54sHON1vWl6FE="
};
didery.batchPostBlobs(signature, data, urls).then(function (response) {
     // Do something with response
}).catch(function (error) {
    console.error(error);
});
```

3.7 batchPutBlobs

This function uses fetch to hit the PUT history endpoint of multiple didery servers. The PUT blobs endpoint updates encrypted key blobs. The PUT endpoint should only be used to update the current key after key rotation and revocation events. Data put to the servers will be verified against signatures. If any discrepancies are found the operation will fail. This function is asynchronous.

3.7.1 Parameters

putBlobs has four required parameters. First is a signature string that will be placed in the required signature header. This string should be of the format:

```
'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";'
```

or:

```
'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";'kind="ed25519"; ...'
```

Second is an object containing the data to be posted. This data object should be of the format:

```
{
    "blob":
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCgo9yjuKHHNJZ
↪4D-7s3CcYmuoWAh6NVtYaf_GWw_2sCrHBAA2mAEsml3thLmu50Dw",
    "changed": "2000-01-01T00:00:00+00:00",
    "id": "did:dad:Qt27fThWoNZsa88VrTkep6H-4HA8tr54sHON1vWl6FE="
}
```

Third is a did parameter that takes a DID string. Fourth is a urls parameter that takes an array of URL strings.

3.7.2 Return

batchPutBlobs returns a promise that when fulfilled returns the servers' responses to the fetch operations.

3.7.3 Example

```
const didery = require('didery');

let urls = [
     "http://myDideryServer.com",
     "http://anotherDideryServer.com",
     "http://oneMoreServer.com"
];
let signature = 'signer=
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCg==
↪";';
let data = {
     "blob":
↪"AeYbsHot0pmdWAcgTo5sD8iAuSQAfnH5U6wiIGpVNJQQoYKBYrPPxAoIc1i5SHCIDS8KFFgf8i0tDq8XGizaCgo9yjuKHHNJZ
↪4D-7s3CcYmuoWAh6NVtYaf_GWw_2sCrHBAA2mAEsml3thLmu50Dw",
```

(continues on next page)

(continued from previous page)

```
    "changed": "2000-01-01T00:00:00+00:00",
    "id": "did:dad:Qt27fThWoNZsa88VrTkep6H-4HA8tr54sHON1vWl6FE="
};
didery.batchPutBlobs(signature, data, urls).then(function (response) {
    // Do something with response
}).catch(function (error) {
    console.error(error);
});
```

3.8 batchGetRelays

This function uses fetch to hit the GET relays endpoint of multiple didery servers. The GET relays endpoint returns trusted servers. This functions is asynchronous.

3.8.1 Parameters

getRelays has one required urls parameters. This takes an array of URL strings.

3.8.2 Return

getRelays returns a promise that when fulfilled returns the servers' responses to the fetch operations.

3.8.3 Example

```
const didery = require('didery');

let urls = [
    "http://myDideryServer.com",
    "http://anotherDideryServer.com",
    "http://oneMoreServer.com"
];
didery.batchGetRelays(urls).then(function (response) {
    // Do something with response
}).catch(function (error) {
    console.error(error);
});
```

3.9 batchPostRelays

This function uses fetch to hit the POST relays endpoint of multiple didery servers. The POST relays endpoint saves new trusted servers. This function is asynchronous.

3.9.1 Parameters

batchPostRelays has two required parameter. First is a data parameter takes an object and should have the following format:

```
{
    "changed": datetime string,
    "host_address": URL string,
    "main": boolean,
    "name": name string,
    "port": port string
}
```

Second is a urls parameter that takes an array of URL strings.

3.9.2 Return

batchPostRelays returns a promise that when fulfilled returns the servers' responses to the fetch operations.

3.9.3 Example

```
const didery = require('didery');

let urls = [
     "http://myDideryServer.com",
     "http://anotherDideryServer.com",
     "http://oneMoreServer.com"
];
let data = {
    "changed": "2000-01-01T00:00:00+00:00",
    "host_address": "127.0.0.1",
    "main": true,
    "name": "alpha",
    "port": "7541"
};
didery.batchPostRelays(data, urls).then(function (response) {
     // Do something with response
}).catch(function (error) {
    console.error(error);
});
```

3.10 batchPutRelays

This function uses fetch to hit the PUT relays endpoint of multiple didery servers. The PUT history endpoint updates trusted servers. This function is asynchronous.

3.10.1 Parameters

batchPutRelays has three required parameters. First is a data parameter that takes an object with the following format:

```
{
    "changed": datetime string,
    "host_address": URL string,
    "main": boolean,
    "name": name string,
```

(continues on next page)

(continued from previous page)

```
    "port": port string
}
```

Second is a uid paramter that takes a uid string of the server to be updated. Third is a urls parameter that takes an array of URL strings.

3.10.2 Return

batchPutRelays returns a promise that when fulfilled returns the servers' responses to the fetch operations.

3.10.3 Example

```
const didery = require('didery');

let urls = [
     "http://myDideryServer.com",
     "http://anotherDideryServer.com",
     "http://oneMoreServer.com"
];
let data = {
    "changed": "2000-01-01T00:00:00+00:00",
    "host_address": "127.0.0.1",
    "main": true,
    "name": "alpha",
    "port": "7541"
};
let uid = "1";
didery.batchPutRelays(data, uid, urls).then(function (response) {
     // Do something with response
}).catch(function (error) {
    console.error(error);
});
```

3.11 batchDeleteRelays

This function uses fetch to hit the DELETE relays endpoint of multiple didery servers. The DELETE relays endpoint deletes trusted server data. This function is asynchronous.

3.11.1 Parameters

batchDeleteRelays has two required parameters. Second is a urls parameter that takes an array of URL strings.

3.11.2 Return

batchDeleteRelays returns a promise that when fulfilled returns the servers' responses to the fetch operations.

3.11.3 Example

```
const didery = require('didery');

let urls = [
     "http://myDideryServer.com",
     "http://anotherDideryServer.com",
     "http://oneMoreServer.com"
];
let uid = "1";
didery.batchDeleteRelays(uid, urls).then(function (response) {
     // Do something with response
}).catch(function (error) {
    console.error(error);
});
```

3.12 batchGetErrors

This function uses fetch to hit the GET errors endpoint of multiple didery servers. The GET errors endpoint returns any errors raised by the server. This functions is asynchronous.

3.12.1 Parameters

batchGetErrors has one required urls parameters. This takes an array of URL strings.

3.12.2 Return

batchGetErrors returns a promise that when fulfilled returns the servers' responses to the fetch operations.

3.12.3 Example

```
const didery = require('didery');

let urls = [
     "http://myDideryServer.com",
     "http://anotherDideryServer.com",
     "http://oneMoreServer.com"
];
didery.batchGetErrors(baseURL).then(function (response) {
     // Do something with response
}).catch(function (error) {
     console.error(error);
});
```