Symmetric Searchable Encryption component

Verification manual version 0.2.0

Component name: SSE – Symmetric Searchable Encryption

Component deployment name: sse

Changelog

Table 1 Document Change History

Version	Date	Pages	Author	Modification
0.1	04/03/21	3	Hai-Van Dang	Initial document
0.2	9/3/2021	3	Hai-Van Dang	Update the examples
0.3	22/3/2021	4	Hai-Van Dang	Add notes when running tests with SGX

1. Table of Contents

1. Terminology

Terminology/ Abbreviation	Explanation
SSE server	Server which stores encrypted data
SSE Trusted Authority (SSE TA)	Server which stores metadata necessary for upload/ search encrypted data
SSE client	An application which utilizes SSE javascript APIs, i.e. sse.js, to upload/ search over encrypted data

2. Automatic tests with Jest

This section describes how to write and run automatic tests with Jest [1,2]

1. (if not installing yet) Install *nodejs* and *npm*

```
sudo apt-get install -y nodejs
curl -fsSL https://deb.nodesource.com/setup_current.x | sudo -E bash -
```

2. Prepare environment and write test script

- Go to SSE client source code folder
- (if not creating the folder yet) Create a folder Tests

```
mkdir Tests
```

- Go into the folder Tests

```
cd Tests
```

- Initialize the project

```
npm init -y
```

It will generate *package.json* file. Open up **package.json** and configure the script if you wish:

```
{
  "name": "Testing_SSE",
  "version": "1.0.0",
  "description": "Testing SSE functionalities with Jest",
  "main": "index.js",
  "scripts": {
    "test": "jest"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "jest": "^25.5.4",
    "node-fetch": "^2.6.0"
  },
  "dependencies": {
    "file-saver": "^2.0.2",
    "node-fetch-npm": "^2.0.4"
  }
}
```

- (if not installing yet) Install Jest

```
sudo npm i jest --save-dev
```

- (if not creating yet) Create __tests__ folder

```
mkdir __tests__
```

- (if not creating yet) Create the test script file, for example *testscript.js*. Write test cases in `testscript.js`.

3. Run the test

Assuming that SSE Server and SSE TA are running.

- Edit the source code `sse.js` of SSE client to run with `nodejs` and `jest`:

Uncomment the following code section

```
////////CONFIGURATION FOR AUTOMATIC TESTS WITH JEST- Start ///////////////
const $ = require('./jquery-3.4.1.min.js') // for jest automatic testing
const sjcl = require('./sjcl.js') // for jest automatic testing
module.exports =
[uploadData,search,updateData,deleteData,uploadKeyG,encryptUploadBlob,downloadDecr
yptBlob]; // for jest automatic testing
////////CONFIGURATION FOR AUTOMATIC TESTS WITH JEST- End ///////////////
```

Fill parameter values of *sseConfig* variable in *sse.js*. Example:

```
var sseConfig={
        'base_url_ta' : 'http://51.145.96.96:8000', //{url} URL of SSE TA. Example:
http://127.0.0.1:8000
        'base_url_sse_server' : 'http://51.145.96.96:8080',//{url} URL of SSE Server
        'salt' : 'ZWdhYndlZmc=', //{base64, 8 bytes} Salt value which is used for key
generation from a passphrase, if needed
        'iv' : 'bGd3YmFnd2c=', //{base64, 8 bytes} Initial vector value which is used for
encryption
        'iter' : 10000, //{number} Number of iteration for generating key from passphrase, if
needed. Example: 1000,10000
```

```
        'ks' : 256, // {number} key size. Example: 128, 256
        'ts' : 64, // {number} tag size. Example: 64
        'mode' : 'ccm', // {string} Encryption mode. Example: ccm
        'adata':'', // {string, ''} This is not supported to be configurable yet
        'adata_len' : 0, //{number, 0} This is not supported to be configurable yet
        'hash_length' : 256, // {number} length of hash value
        'chunk_size' : 32768,// {number} Size of 1 slice/ chunk for encryption (in uint8
items). The value 32768 (=1024^32) is selected by running experiments to avoid the memory
crash. This can be selected differently, but needs to run experiments to predict any memory
crash in the browser.
        'no_chunks_per_upload' : 30, // {number} Number of chunks to be packed in 1 upload.
The value 30 is selected by running experiments to avoid the memory crash. This can be
selected differently, but needs to run experiments to predict any memory crash in the
browser.
        'salt_ta' : 'ZRVja4LFrFY=', // {base64, 8 bytes} salt value which is used for key
generation from a passphrase, if needed
        'iv_ta' : 'YXp5bWJscWU=', // {base64, 8 bytes} Initial vector value which is used for
encryption
        'iter_ta' : 10000, //{number} Number of iteration for generating key from passphrase,
if needed. Example: 1000,10000
        'ks_ta' : 128, //{number} keysize. If SGX is enabled at TA, it must be set to 128 bits
to be compatible with AES encryption in SGX enclave
        'ts_ta' : 64, // {number, 64} tag size. It is not supported to be configurable yet.
        'mode_ta' : 'ccm', // {string} Encryption mode. Example: ccm
        'adata_ta':'', // {string, ''} This is not supported to be configurable yet
        'adata_len_ta' : 0, //{number, 0} This is not supported to be configurable yet
        'sgx_enable': true, // {boolean} true if SGX is enabled at SSE TA, false otherwise
        'base_url_cp_abe' : 'https://asclepios-cpabe.euprojects.net', //{url} URL of CP-ABE
server
        'debug' : true // {boolean} true if debug, false otherwise
}
```

- Edit the keys `(key_KeyG1, key_Kenc1, key_KeyG2, key_Kenc2)` in `testscript.js` if you use different values for at least one of `(salt,iv,iter,ks,ts,salt_ta,iv_ta,iter_ta,ks_ta,ts_ta)` in `sseConfig` than using the above example, or you change the values of the passwords (`KeyG1, Kenc1, KeyG2, Kenc2) in testscript.js.`

```
const KeyG1="123"; //password
const Kenc1 = "abc"; // password
const KeyG2="456"; //password
const Kenc2 = "def";//password

const key_KeyG1="fb77d1464189bb07f7f1d6d524b9eaaf"; // Key (hex string) which is generated
from the password in KeyG1
const key_Kenc1 = "ed0f78e4cfd589337faf5b6d5e07637081426bcf32b5a2fab9a4b7517147bf2c"; // Key
(hex string) which is generated from the password in Kenc1

const key_KeyG2="9751c56747f3867e76d39968d3de9e31"; // Key (hex string) which is generated
from the password in KeyG2
const key_Kenc2 = "d89f8ad6988ba7aba42ddbdf2453f2241d3edafc96757f750911df1c8e986179"; // Key
(hex string) which is generated from the password in Kenc2
```

- Go to Tests folder

```
cd Tests
```

- Run the test:

```
npm test
```

3. Notes when running tests with SGX

When enabling SGX, the process of uploading verification keys to SSE Trusted Authority component require longer time because of SGX enclave creation. It may happen that the test

verification happens before the finishing of the uploading verification key process; hence, it leads to failed test cases. Therefore, it is better to manually upload the verification keys before running the tests automatically. After uploading the keys manually, you can run the other test cases except the following ones (which needs to be disabled).

```
/*
describe("upload shared key to TA",() => {
	//input is a password. The key will be generated from the password.
	test("upload shared key to TA should return true", () => {
		var key = computeKey(KeyG1,true);
		var result = uploadKeyG(key,keyid1);
		expect(result).toEqual(true);
	});
	// input is a key
	test("upload shared key to TA should return true", () => {
		var result = uploadKeyG(key_KeyG2,keyid2);
		expect(result).toEqual(true);
	});
});
*/
```

References

1. Jest javascript testing framework, https://jestjs.io/
2. Jest tutorial, https://www.valentinog.com/blog/jest/