Symmetric Searchable Encryption component

Configuration 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	26/2/2021	6	Hai-Van Dang	Initial release
0.2	9/3/2021	5	Hai-Van Dang	Add repositories Update technical notes

1. Table of Contents

1. Terminology

Terminology/ Abbreviation	Explanation
SSE server	Server which stores encrypted data
SSE Trusted Authority (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
sse.js	The provided javascript APIs
SSE database	Database of SSE server
TA database	Database of TA server
teep-deployer server	The server which provides functionalities to create/ install

	SGX enclave.
teep-deployer client	The client which invokes Rest APIs provided by teep-deployer server. It also contains implementation of the application which will be installed into a SGX enclave.

2. Repositories

This section lists the source code and document repositories of SSE.

1. The paper which explains the SSE components, and the interactions among the components. However, the current implementation does not match with the paper completely.

- Dang, H. V., Ullah, A., Bakas, A., & Michalas, A. (2020, October). Attribute-Based Symmetric Searchable Encryption. In *International Conference on Applied Cryptography and Network Security* (pp. 318-336). Springer, Cham.

2. SSE manual: https://gitlab.com/asclepios-project/ssemanual

- SSE_API_specification: It describes SSE client APIs, their inputs and outputs
- SSE_deployment_manual: It describes how to deploy components of SSE scheme, which include Trusted Authority, SSE Server, and a demo SSE client
- SSE_technical_notes: It is this document
- SSE_verification: It describes how to run automatic tests using Jest [5].

2. Source code:

- SSE client (javascript library and a demo client): https://gitlab.com/asclepios-project/sseclient
  where
  - sse/static/js/sse.js: the main javacript library implementation
  - sse/static/js/main.js: an example which uses sse.js for SSE functionalities
  - Tests/__tests__/testscript.js: a test script which is used to run the automatic tests with jest
- SSE server: https://gitlab.com/asclepios-project/symmetric-searchable-encryption-server
- SSE TA: https://gitlab.com/asclepios-project/sseta
  where
  - teepclient [4]: implementation of SGX-based functionalities. This is the teep-deployer client.
  - sjcl-0.2.1 [3]: the customized sjcl-python library
- teep-deployer server : https://gitlab.com/asclepios-project/teep-deployer/-/tree/teep-sse

3. Others

- (For benchmarking) How to generate synthetic data for SSE benchmarking: https://gitlab.com/asclepios-project/json-generator
- docker-compose file to deploy components of SSE:
  - https://gitlab.com/asclepios-project/sse-deployment
  - https://gitlab.com/asclepios-project/asclepios-sse-docker-compose
- ADT file to deploy the components of SSE with MiCADO:

- https://gitlab.com/asclepios-project/sse-deployment
- https://gitlab.com/asclepios-project/micado-adts/-/tree/asclepios/ADT/sleep

3. Technical notes

This section describes a few technical notes during the development. Among them, the notes (b) and (c) aimed at solving the incompatibility between different cryptographic libraries at the client side (SSE client) and the server side (SSE TA).

a. Re-compile SLCL javascript

In the SSE client implementation, we rely on SJCL library for cryptographic functions (encryption, decryption, hashing, etc.). The SJCL source code can be found at https://github.com/bitwiseshiftleft/sjcl. However, we have compiled only necessary sub-libraries in https://github.com/bitwiseshiftleft/sjcl/tree/master/core for our need instead of using the ready-to-use compiled js at https://cdnjs.com/libraries/sjcl. The reason we compiled ourselves is the ready-to-use compiled js does not contain functions for progressive encryption and decryption.

In order to compile sjcl js, we download sjcl from https://github.com/bitwiseshiftleft/sjcl, and add a few sub-libraries (marked in red color) into config.mk as below, then invoke *make* command line to build it:

```
SOURCES= core/sjcl.js core/aes.js core/bitArray.js core/codecString.js core/codecHex.js core/codecBase32.js core/codecBase64.js core/sha256.js core/ccm.js core/ocb2.js core/gcm.js core/hmac.js core/pbkdf2.js core/random.js core/convenience.js core/exports.js core/ocb2progressive.js core/codecBytes.js core/sha1.js
COMPRESS= core_closure.js
```

The compiled sjcl.js is copied into sse/static/js folder.

b. Encryption using SJCL at SSE client and decryption at SSE TA without SGX using sjcl-python package

In the TA implementation without SGX, we rely on sjcl-python package (https://github.com/berlincode/sjcl) for cryptographic functions (encryption, decryption, hashing). However, the *encrypt* function in sjcl-python does not allow to provide SALT and IV as parameters (https://github.com/berlincode/sjcl/blob/master/sjcl/sjcl.py#L156), which is incompatible with the encryption function in SJCL javascript. In order to make the encryption at SSE client and TA compatible, we have modified sjcl-python package so that the *encrypt* function accepts SALT and IV values as parameters. Therefore, instead of using the ready-to-use python package sjcl (https://pypi.org/project/sjcl/), we use the modified version which locates at *sjcl-0.2.1/sjcl.*
Apart from that, we have also modified sjcl-python package to allow encryption/ decryption with a password or a key. In case of using a password, a key is generated from the password inside the encryption/ decryption function.
The modified encryption/ decryption functions [2,3] have become:

```
def encrypt(self, plaintext, passphrase, salt = "", iv="", mode="ccm", count=10000, dkLen=16, iskey=False)

def decrypt(self, data, passphrase, iskey=False)
```

Parameters:

passphrase = a password or a key
salt = salt value in UTF8 format
iv = iv value in UTF8 format
mode = cipher mode
count = number of iterations for generating a key from a password
dkLen = number of bytes of key size = keysize/8
iskey=false, passphrase is a password. Otherwise, passphrase is a key.

c. Encryption using SJCL at SSE client and decryption at SSE TA with SGX enabled using mbedtls library

In the TA implementation with SGX, we rely on mbedtl library (https://github.com/ARMmbed/mbedtls) for cryptographic functions (encryption, decryption, hashing). There is no direct instruction on how to make encryption using SJCL to be compatible with decryption using mbedtls. The current implementation is based on the hints at https://stackoverflow.com/questions/23074176/crypto-is-sjcl-javascript-encryption-compatible-with-openssl. The main idea is that the ciphertext message is composed of the ciphertext content (ct) and the tag content (tag). Therefore, at first, the ciphertext content is split from the ciphertext message, then it is fetched into the decryption function of mbedtls [1].

```
size_t msg_len = size - TAG_LEN; //the ciphertext string contains
ciphertext content plus tag. Therefore, the length of ciphertext is equal
to the size of the string subtracted TAG_LEN

ret = mbedtls_ccm_auth_decrypt( &m_aescontext, msg_len,

                        m_operating_iv, AES_IV_SIZE,

                        NULL, 0,

                        (const unsigned char*)input_buf, output_data,

                        input_buf + msg_len, TAG_LEN );
```

d. Re-compile teep-deployer client [4]

The compilation requires the machine is SGX supported. Please follow instructions at https://gitlab.com/asclepios-project/teep-deployer.

References

1. Using mbedtls to decrypt data which is encrypted by SJCL library, https://github.com/UoW-CPC/Asclepios-TrustedAuthority/blob/0.4/teepclient/common/aes_ccm.cpp#L105
2. Customized encryption function of sjcl-python package, https://github.com/UoW-CPC/Asclepios-TrustedAuthority/blob/0.4/sjcl-0.2.1/sjcl/sjcl.py#L160
3. Customized decryption function of sjcl-python package, https://github.com/UoW-CPC/Asclepios-TrustedAuthority/blob/0.4/sjcl-0.2.1/sjcl/sjcl.py#L87
4. Teep-deployer client, https://gitlab.com/asclepios-project/sseta/-/tree/develop-sgx/teepclient
5. Jest javascript testing frameworkd, https://jestjs.io/