Symmetric Searchable Encryption component

Deployment manual version 0.5.0

Component name: SSE – Symmetric Searchable Encryption

Component deployment name: sse

Changelog

Table 1 Document Change History

Version	Date	Pages	Author	Modification
0.1	5/3/2020	9	Hai-Van Dang	Initial release: deployment with docker in cloud or locally
0.2	30/3/2020	9	James Bowden	Deployment with docker compose
0.3	11/08/2020	12	Hai-Van Dang	Minio deployment Deployment with MySQL Deployment with database-as-a-service
0.4	15/12/2020	12	Hai-Van Dang	Minio deployment with SSL support Providing technical notes on re-compiling SJCL javascript, and modifying sjcl python package
0.5	2/3/2021	13	Hai-Van Dang	Refactor the document Support enabling SGX at SSE TA Provide technical notes about implementation of decryption using mbedtls library in a SGX enclave Explain and instruct on the configuration settings

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
SSE database	Database of SSE server
TA database	Database of TA server

2. Introduction

This manual instructs how to deploy the three main Symmetric Searchable Encryption components (SSE server, SSE Trusted Authority, and a demo SSE client) and the two additional components (MinIO server [3] and teep-server [4]). SSE client [7] is a web application which uses SSE services such as uploading encrypted data, search/ update/ delete over the stored encrypted data. SSE server [5] stores the encrypted data which is uploaded by the SSE client. It also provides the services such as search/ update/ delete over the stored data. SSE Trusted Authority (SSE TA) [6] stores the metadata which is used for generating the search/ update/ delete token (at the SSE client), and verifying the tokens on behalf of the SSE server. Apart from that, MinIO server stores encrypted binary large objects (blob), and teep-server provides SGX [2] related services such as creating, installing SGX enclaves. These two components are optional. MinIO is only necessary when a client application wish to support encrypting blob data and search/ update/ delete over its encrypted metadata. Meanwhile, teep-server is needed when we wish to enable SGX at SSE TA to increase the security level.

Figure 1 illustrates these components and their connections, where sse-client meaning SSE client, ta meaning SSE TA, sse meaning SSE server, minio meaning MinIO server, ta-db meaning the database of SSE TA, and sse-db meaning the database of SSE server.

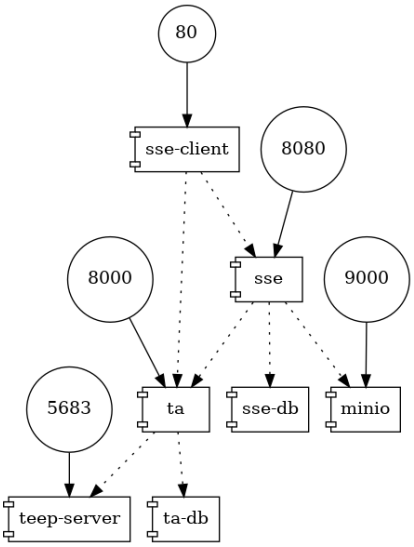


Figure 1 SSE main components and two additional components

For further explanation on the components SSE Server, SSE TA, and SSE client, please refer to the paper [8].

3. Deployment with docker-compose

It is possible to deploy all the SSE components on a single host using docker-compose. The following section describes how to do that.

1. (If using SGX at SSE TA) Get source code of teep-server

```
git clone https://gitlab.com/asclepios-project/teep-deployer
cd teep-deployer
git checkout teep-sse
cd ..
```

Requirement for SGX enabling: the machine needs to be able to run SGX applications. In cloud environment, at the time of this writing document, Microsoft Azure supports Intel SGX enabled virtual machines [1].

2. Get the docker-compose definition from github

```
git clone https://gitlab.com/asclepios-project/sse-deployment
```

The docker-compose file involves the definitions of the SSE Server, SSE TA, SSE client, MinIO Server, and TEEP deployer server. If you do not need to use SGX for SSE TA, and/or use MinIO Server, you can customize the docker-compose file by removing teep-server and/or minio definitions.

3. Configure the local environment

```
cd sse-deployment
```

If using SGX at SSE TA:

```
cp template_sgx.env .env
```

If not using SGX at SSE TA:

```
cp template_nonsgx.env .env
```

Edit the .env file to modify the configurations. At minimum, you need to provide modify the values ending with _HERE such as DATABASE_USER_HERE, DATABASE_USER_PASSWORD_HERE in the template. You can also modify other configurations by following the meaning and examples of configuration values in section 8. Please note that a few configurations need synchronization between components.

4. Build the docker images

```
docker-compose build
```

5. Run the SSE service

```
docker-compose up
```

Once it has successfully booted, each component is accessible on the host as follows:

- SSE Client - http://localhost:80
- SSE Server - http://localhost:8080
- SSE TA - http://localhost:8000
- MinIO server – http://localhost:9000

6. Create the default data bucket at MinIO server

Go to http://localhost:9000
Login using the MINIO_ACCESS_KEY and MINIO_SECRET_KEY defined in the .env file
Select the following icon to create a bucket named as “asclepios”[1].

4. Deployment using MiCADO

Please find examples of ADT which can be used to deploy the application with MiCADO at https://gitlab.com/asclepios-project/sse-deployment/-/tree/main and/or https://gitlab.com/asclepios-project/micado-adts/-/tree/asclepios/ADT/sleep.

[1] This name has been define in the .env file. If you want to use another name, please also change the name in the .env file.

5. Deployment in development mode

The following paragraphs instruct how to run the three applications, i.e. SSE server, TA, SSE client in development mode with SQLite instead of Postgres/ MySQL, and without docker containers. It also describes how to deploy MinIO server with/ without SSL, and as a storage server or a gateway to Amazon S3 storage.

5.1. SSE server/ SSE TA/ SSE client without docker containers

1. Re-configure database to use SQLite

In case of SSE TA:

```
cp template_settings_sqlite.py TA/settings.py
```

In case of SSE Server:

```
cp template_settings_sqlite.py SSEServer/settings.py
```

In case of SSE client:

```
cp template_settings_sqlite.py SSEclient/settings.py
```

2. Initialize database in SQLite

This needs to run once when database needs to be initialized (e.g. for the 1st run of the application, or after deleting the previous database, i.e. db.sqlite3 file).

```
python3 manage.py migrate
```

3. (In case of SSE TA and SSE Server) Define values of the environment variables, which are defined in [10] (for SSE TA) and [9] (for SSE Server).

Table 2 Environment variables at SSE Server

```
export DJANGO_LOGLEVEL=DEBUG \
DJANGO_DEBUG=true \
TA_SERVER=http://127.0.0.1:8000 \
ALLOWED_HOSTS=* \
MINIO_ACCESS_KEY=MINIO_ACCESS_KEY_HERE \
MINIO_SECRET_KEY=MINIO_SECRET_KEY_HERE \
MINIO_BUCKET_NAME=asclepios \
MINIO_URL=127.0.0.1:9000 \
MINIO_SSL_SECURE=false \
MINIO_EXPIRE_GET=1 \
MINIO_EXPIRE_PUT=1
```

Table 3 Environment variables at SSE TA

```
export DJANGO_LOGLEVEL=DEBUG \
DJANGO_DEBUG=True \
ALLOWED_HOSTS=* \
HASH_LENGTH=256 \
IV=TA_IV_VALUE_IN_UTF8_HERE
```

4. (In case of SSE client) Fill constant values of sseConfig variable in sse.js [11].

5. Run SSE server in development mode

In case of SSE TA:

```
python3 manage.py runserver 0.0.0.0:8000
```

In case of SSE Server:

```
python3 manage.py runserver 0.0.0.0:8080
```

In case of SSE client:

```
python3 manage.py runserver 0.0.0.0:80
```

5.2. MinIO deployment

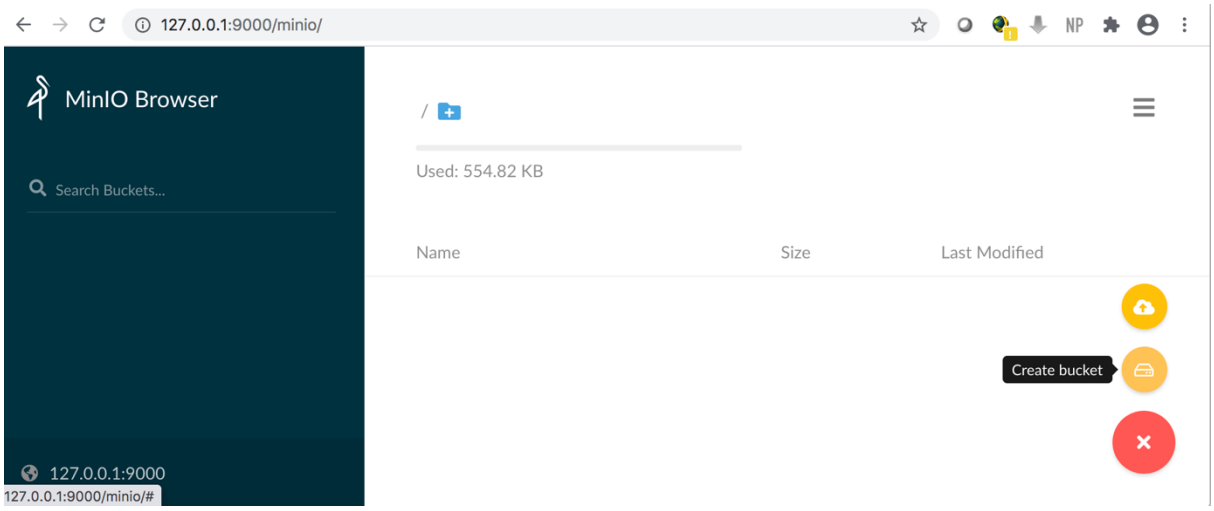
A. As docker container without SSL

1. Deploy as a docker container:

```
docker run -p 9000:9000 --name minio \
        -e 'MINIO_ACCESS_KEY=choose_access_key_here' \
        -e 'MINIO_SECRET_KEY=choose_secret_key_here' \
        -v folder_in_host_machine:/data \
        minio/minio server /data
```

If deployed successfully, you can access Minio server at http://127.0.0.1:9000 with the chosen access key and secret key.

2. Create a bucket:



B. As a gateway to Amazon S3 without SSL

Assuming that you've created an Amazon S3 bucket. You can run Minio as a gateway to the created Amazon S3 with the following command.

```
docker run -p 9000:9000 --name minio-s3 \
        -e 'MINIO_ACCESS_KEY=access_key_from_aws' \
        -e 'MINIO_SECRET_KEY=access_secret_from_aws' \
        minio/minio gateway s3
```

C. As docker container with SSL using self-signed certificate

1. Generate self-signed certificate using OpenSSL

Example:

```
openssl genrsa -out private.key 2048

openssl req -new -x509 -days 3650 -key private.key -out public.crt -subj
"/C=US/ST=state/L=location/O=organization/CN=<domain.com>"
```

2. Create a folder to store data in the host machine

```
mkdir data
```

3. Create a folder to store private.key and public.crt in host machine

```
mkdir config

mkdir config/certs
```

4. Copy private.key and public.crt to config/certs

Run minio as a docker container with SSL enabled

```
docker run -p 9000:9000 --name minio \
```

```
        -e 'MINIO_ACCESS_KEY=choose_access_key_here ' \
        -e 'MINIO_SECRET_KEY=choose_secret_key_here ' \
        -v data:/data \
        -v config:/root/.minio \
        minio/minio server /data
```

If deployed successfully, you can access Minio server at https://127.0.0.1:9000 with the chosen access key and secret key.

6. Deployment SSEServer/ TA with MySQL/ Postgres

Database engine can be changed easily with the existing template files.

In order to change to MySQL, copy the template files then re-build the docker image for TA and/ or SSEServer.

```
cp template_Dockerfile_mysql Dockerfile
cp template_requirements_mysql.txt requirements.txt
cp template_entrypoint_mysql.sh entrypoint.sh
cp template_settings_mysql.py TA/settings.py
```

In order to change to Postgres, copy the template files then re-build the docker image for TA and/ or SSEServer.

```
cp template_Dockerfile_postgres Dockerfile
cp template_requirements_postgres.txt requirements.txt
cp template_entrypoint_postgres.sh entrypoint.sh
cp template_settings_postgres.py TA/settings.py
```

7. Deployment SSEServer/ TA with external database-as-a-service

In order to use database-as-a-service, for instance Amazon Relational Database Service, at first you need to create a database at the CSP, for i.e. AWS. Then you change the environment variables DB_HOST and DB_PORT correspondingly.

Example:
DB_HOST=….rds.amazonaws.com
DB_PORT=3306

8. Configurations explanation and example

Table 4 Configurations at SSE Client

Parameters	Value	Meaning	Example	Configurable	Need compatibility with
SSE_CLIENT_ALLOWED_HOSTS	boolean	Log level of Django	True/ False	Y	
SSE_CLIENT_DJANGO_DEBUG	boolean	Debug level of Django	True/ False	Y	
SSE_CLIENT_TA_URL	URL	URL of SSE TA	http://127.0.0.1:8000	Y	
SSE_CLIENT_SSE_URL	URL	URL of SSE Server	http://127.0.0.1:8080	Y	
SSE_CLIENT_SALT	base64 string, 8 bytes	Salt value which is used for key generation from a passphrase, if needed	ZRVja4LFrFY=	Y	
SSE_CLIENT_IV	base64, 8 bytes	Initial vector value which is used for encrypting data	ZRVja4LFrFY=	Y	
SSE_CLIENT_ITER	Number	Number of iteration for generating key from passphrase, if needed	10000	Y	
SSE_CLIENT_KS	Number	Key size	128 or 256	Y	
SSE_CLIENT_TS	Number	Tag size	64	Y	
SSE_CLIENT_MODE	string	Cipher mode	ccm	Possibly, but not tested for other mode like “gcm” yet	
SSE_CLIENT_ADATA	String	Authenticated data	"	Y	
SSE_CLIENT_ADATA_LEN	Number	Length of authenticated data	0	Y	
SSE_CLIENT_HASH_LEN	Number	Output length of hash function	256	Y	
SSE_CLIENT_CHUNK_SIZE	Number	Size of 1 slice/ chunk for encryption (in uint8 items) when encrypting blob data	32768	Y, but needs experimenting to avoid memory crash in browser	
SSE_CLIENT_NO_CHUNKS_PER_UPLOAD	Number	Number of chunks to be packed in 1 upload when encrypting and uploading blob data	30	Y, but needs experimenting to avoid memory crash in browser	
SSE_CLIENT_SALT_TA	base64 string, 8 bytes	Salt value which is used for key generation from a passphrase, if needed	ZRVja4LFrFY=	Y	
SSE_CLIENT_IV_TA	base64 string, 8 bytes	Initial vector value which is used for encryption	ZRVja4LFrFY=	Y	TA_IV[2] (in case SGX is not enabled at SSE TA) or iv[3] (in case SGX is enabled) at SSE TA
SSE_CLIENT_ITER_TA	number	Number of iteration for generating key from passphrase, if needed	10000	Y	
SSE_CLIENT_KS_TA	number	Key size	128, 256	Y if SGX is not enabled. N if SGX is enabled (its value is 128 then)	TA_KS at SSE TA
SSE_CLIENT_TS_TA	number	Tag size	64	N	
SSE_CLIENT_MODE_TA	string	Cipher mode	ccm	Possibly if SGX is not enabled (“ccm” or “gcm”), but not tested yet. N if SGX is enabled.	TA_MODE at SSE TA
SSE_CLIENT_ADATA_T	String	Authenticated data	"	N	

[2] The TA_IV value is in UTF-8 string format. Meanwhile, SSE_CLIENT_IV_TA is in base64 format. You can use this page to https://onlineutf8tools.com/convert-utf8-to-base64 to convert between them

[3] In case using SGX, if you want to change value SSE_CLIENT_IV_TA, you need to change the value of *iv* variable at https://github.com/UoW-CPC/Asclepios-TrustedAuthority/blob/0.4/teepclient/common/aes_ccm.cpp#L10 at SSE TA, and re-compile the code in https://github.com/UoW-CPC/Asclepios-TrustedAuthority/tree/0.4/teepclient to update the application image https://github.com/UoW-CPC/Asclepios-TrustedAuthority/blob/develop-sgx/teepclient/enclave_a/enclave_a.signed

A					
SSE_CLIENT_ADATA_LEN_TA	Number	Length of authenticated data	0	N	
SSE_CLIENT_SGX_ENABLE	boolean	true if SGX is enabled at SSE TA, false otherwise	true/ false	Y	TA_SGX at SSE TA

Table 5 Configurations at SSE TA

Parameters	Value	Meaning	Example	Configurable	Need compatibility with
TA_DJANGO_LOGLEVEL	boolean	Log level of Django	True/ False	Y	
TA_DJANGO_DEBUG	boolean	Debug level of Django	True/ False	Y	
TA_ALLOWED_HOSTS	String	URLs to be allowed to access to SSE TA	*	Y	
TA_DB_NAME	string	Name of database	tadb	Y	
TA_DB_USER	string	Database user name	tauser	Y	
TA_DB_PASSWORD	string	Database user's password	tapwd	Y	
TA_DB_HOST	string	Database hostname	ta-db	Y	
TA_DB_PORT	Number	Database port	5432 if using postgres, 3306 if using mysql	Y	
TA_HASH_LENGTH	Number	Output length of hash function	256	Y	
TA_IV (in case SGX is not enabled)	Utf-8 string	Initial vector value which is used for encryption	abcdefgh[4]	Y	SSE_CLIENT_IV_TA at SSE client
iv defined in teepclient/common/aecs_ccm.cpp (in case SGX is enabled)	Array of bytes	Initial vector value which is used for encryption	{0x61,0x7a,0x79,0x6d,0x62,0x6c,0x71,0x65}[5]	Y	SSE_CLIENT_IV_TA at SSE client
TA_MODE	string	Cipher mode	ccm	Y if SGX is not enabled (ex. "gcm","ccm"). N if SGX is enabled	SSE_CLIENT_MODE_TA at SSE client
TA_KS	number	Key size	128, 256	Y if SGX is not enabled. N if SGX is enabled (in such case, TA_KS=128)	SSE_CLIENT_KS_TA at SSE client
TA_TEEP_SERVER	URL	URL of TEEP deployer server	coap://127.0.0.1:5683/teep	Y	
TA_SGX	number	Indicate if SGX is enable	1 if SGX is enabled, 0 otherwise	Y	SSE_CLIENT_SGX_ENABLE at SSE client

Table 6 Configurations at SSE Server

Parameters	Value	Meaning	Example	Configurable	Need compatibility with
SSE_SERVER_DJANGO_LOGLEVEL	boolean	Log level of Django	True/ False	Y	
SSE_SERVER_DJANGO_DEBUG	boolean	Debug level of Django	True/ False	Y	

[4] This value is in UTF-8 string format. Meanwhile, SSE_CLIENT_IV_TA is in base64 format. You can use this page to https://onlineutf8tools.com/convert-utf8-to-base64 to convert between them

[5] If you want to change this value, you need to re-compile the source code in https://github.com/UoW-CPC/Asclepios-TrustedAuthority/tree/0.4/teepclient to update the application image https://github.com/UoW-CPC/Asclepios-TrustedAuthority/blob/develop-sgx/teepclient/enclave_a/enclave_a.signed . Please follow instructions at [4] to re-compile the code.

SSE_SERVER_TA_SERVER	URL	URL of SSE TA	http://127.0.0.1:8000	Y	
SSE_SERVER_DB_NAME	String	Name of database	ssedb	Y	
SSE_SERVER_DB_USER	String	Database user name	sseuser	Y	
SSE_SERVER_DB_PASSWORD	String	Database user's password	ssepwd	Y	
SSE_SERVER_DB_HOST	String	Database hostname	sse-db	Y	
SSE_SERVER_DB_PORT	Number	Database port	5432 if using postgres, 3306 if using mysql	Y	
SSE_SERVER_ALLOWED_HOSTS	String	URLs to be allowed to access to SSE Server	*	Y	
MINIO_ACCESS_KEY	String	Access key to MinIO server	minio	Y	
MINIO_SECRET_KEY	String	Secret key to access MinIO server	miniopwd	Y	
MINIO_BUCKET_NAME	String	Name of data bucket created at MinIO server	mydata	Y	Bucket has been/ will be created at MinIO server
MINIO_URL	URL	URL of MinIO server	127.0.0.1:9000[6]	Y	
MINIO_SSL_SECURE	Boolean	Indication if communication to MinIO server is protected SSL or not	True/ False	Y	SSL configuration at MinIO Server [5.2]
MINIO_EXPIRE_GET	Number	Limit number of usage of a pre-signed GET URL to retrieve data from MinIO server	1	Y	
MINIO_EXPIRE_PUT	number	Limit number of usage of a pre-signed GET URL to update/ submit data to MinIO Server	1	Y	

9. 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
```

[6] Please do not include http into the value of MINIO_URL

```
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 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 ciphertex message, then it is fetched into the decryption function of mbedtls.

```
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 );
```

References

1. Azure confidential computing, https://docs.microsoft.com/en-gb/azure/confidential-computing/confidential-computing-enclaves
2. Costan, V., & Devadas, S. (2016). Intel SGX Explained. IACR Cryptol. ePrint Arch., 2016(86), 1-118.
3. MinIO Server, https://min.io/
4. Teep-deployer implementation, https://gitlab.com/asclepios-project/teep-deployer
5. SSE Server implementation, https://gitlab.com/asclepios-project/symmetric-searchable-encryption-server
6. SSE Trusted Authority implementation, https://gitlab.com/asclepios-project/sseta
7. SSE Client implementation, https://gitlab.com/asclepios-project/sseclient
8. 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.
9. Environment variables defined for SSE Server, https://gitlab.com/asclepios-project/symmetric-searchable-encryption-server/-/blob/develop/api/resources.py#L29
10. Environment variables defined for SSE TA, https://gitlab.com/asclepios-project/sseta/-/blob/develop-sgx/parameters.py
11. Configurations for SSE client, https://gitlab.com/asclepios-project/sseclient/-/blob/develop-sgx/sse/static/js/sse.js#L25