Symmetric Searchable Encryption component

Deployment manual version 0.1.0
Component name: SSE – Symmetric Searchable Encryption
Component deployment name: sse

Changelog

2020-03-05: Version 0.1.0

Initial release

Table of Contents

Terminology

Terminology/ Abbreviation	Explanation
End-user	User who uploads/ searches data
SSE server	Server which stores encrypted data
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

Introduction

This manual instructs how to deploy Symmetric Searchbale Encryption components, i.e. SSE server, Trusted Authority, and an example SSE client, i.e. a web application, which uses SSE services, in docker containers. They are assumingly deployed in three different machines, where port 80 is reserved for each application (i.e. SSE server, TA, SSE client), and port 5432 is reserved for each own database server (i.e. SSE database server, TA database server). If deployed in the same machine, further port configurations need to be done in order to make sure each application runs in different port, and there is no port conflict of the two databases, i.e. SSE database and TA database.

For a simpler deployment during development, i.e. three applications are run locally in development environment, please refer to the section **Local deployment in development mode**.

SSE Server

SSE Server stores encrypted data, and support to search over encrypted data.

1. Configure database name, database user and password in deploy.env file

```
DJANGO_LOGLEVEL=[log level of SSE server application]

TA_SERVER=[URL of Restful API URL provided by TA to perform search function]

DB_NAME=[Name of database]

DB_USER=[Database username]

DB_PASSWORD=[Database user password]

DB_HOST=[Name of postgres container, i.e. sse-postgres]

DB_PORT=[Port of postgres database, i.e. 5432]

ALLOWED_HOSTS=[List of IP addresses, which are allowed to access services provided by SSE server]
```

where
- Name of postgres container is the container name which will be defined in step (3), i.e. sse-postgres
- DJANGO_LOGLEVEL: This log level describes the severity of the messages that the logger will handle. Python defines the following log levels:

    o DEBUG: Low level system information for debugging purposes

    o INFO: General system information

    o WARNING: Information describing a minor problem that has occurred.

o ERROR: Information describing a major problem that has occurred.

o CRITICAL: Information describing a critical problem that has occurred.

Example:

```
DJANGO_LOGLEVEL=DEBUG
TA_SERVER=http://[IP address of TA server]/api/v1/search/
DB_NAME=db1
DB_USER=user1
DB_PASSWORD=pwd1
DB_HOST=sse-postgres
DB_PORT=5432
ALLOWED_HOSTS=['*']
```

2. Define a docker network, where the two docker containers, i.e. database and SSE application, reside in.

```
docker network create sse-network
```

3. Initialize Postgres SQL database as a docker container

```
sudo docker run -d -p 5432:5432 --name sse-postgres --net sse-network -e
PGDATA:/var/lib/postgresql/data/ -v [path to folder which persistenly
stores data]:/var/lib/postgresql/data/ -e POSTGRES_USER=[SSE database
username] -e POSTGRES_PASSWORD=[SSE database password] -e POSTGRES_DB=[SSE
database name] postgres
```

where:
- 5432 is the listening port of Postgres SQL server
- sse-network is the name of docker network which is created in step (2)
- sse-postgres is a chosen name of the container
- [path to folder which persistently stores data] is a chosen folder to store data. This helps to preserve data even if docker container is shut down.
- [SSE database username] is a chosen username
- [SSE database password] is a chosen password
- [SSE database name] is a chosen database name

Example:

```
sudo docker run -d -p 5432:5432 --name sse-postgres --net sse-network -e
PGDATA:/var/lib/postgresql/data/ -v
/home/ubuntu/data/data/ssedata:/var/lib/postgresql/data/ -e
POSTGRES_USER=user1 -e POSTGRES_PASSWORD=pwd1 -e POSTGRES_DB=db1 postgres
```

4. Build and deploy SSE Server as a docker container

```
sudo docker build -t sse [folder contains SSE server source code]
```

```
sudo docker run -d -it -p 80:8080 --name sse-server --net sse-network --
env-file [folder contains SSE server source code]/deploy.env sse
```

where:
- 80: listening port on the host machine
- 8080: listening port on the SSE container, which is configured in "entrypont.sh" of the source code
- sse-network is the name of docker network which is created in step (2)
- "deploy.env" file contains configuration details of the server

All the logs will be written to standard output, which can be seen through the command

```
docker logs sse-server
```

where

- sse-server is the name of the container defined above in step (3).

In order to check if you deploy SSE server successfully, please access to http://[IP address of SSE server]/api/v1/, which should return the list of supported APIs like below

```json
{
    "ciphertext": {
        "list_endpoint": "/api/v1/ciphertext/",
        "schema": "/api/v1/ciphertext/schema/"
    },
    "map": {
        "list_endpoint": "/api/v1/map/",
        "schema": "/api/v1/map/schema/"
    },
    "search": {
        "list_endpoint": "/api/v1/search/",
        "schema": "/api/v1/search/schema/"
    }
}
```

Trusted Authority (TA)

Trusted Authority stores metadata, which are needed for uploading/ searching data.

1. Configure database name, database user and password in deploy.env file

```
DJANGO_LOGLEVEL=[log level of SSE server application]

TA_SERVER=[URL of Restful API URL provided by TA to perform search
function]

DB_NAME=[Name of TA database]

DB_USER=[TA database username]
```

```
DB_PASSWORD=[TA database user password]

DB_HOST=[Name of postgres container, i.e. ta-postgres]

DB_PORT=[Port of postgres database, i.e. 5432]

ALLOWED_HOSTS=[List of IP addresses, which are allowed to access services
provided by TA server]
```

where
- Name of postgres container is the container name which will be defined in step (3), i.e. ta-postgres
- DJANGO_LOGLEVEL: This log level describes the severity of the messages that the logger will handle. Python defines the following log levels:

    o DEBUG: Low level system information for debugging purposes

    o INFO: General system information

    o WARNING: Information describing a minor problem that has occurred.

    o ERROR: Information describing a major problem that has occurred.

    o CRITICAL: Information describing a critical problem that has occurred.

Example:

```
DJANGO_LOGLEVEL=DEBUG
DB_NAME=tadb1
DB_USER=tauser1
DB_PASSWORD=tapwd1
DB_HOST=ta-postgres
DB_PORT=5432
ALLOWED_HOSTS=['*']
```

2. Define a docker network, where the two docker containers, i.e. database and TA application, reside in.

```
docker network create ta-network
```

3. Initialize Postgres SQL database as a docker container

```
sudo docker run -d -p 5432:5432 --name ta-postgres --net ta-network -e
PGDATA:/var/lib/postgresql/data/ -v [path to folder which persistenly
stores data]:/var/lib/postgresql/data/ -e POSTGRES_USER=[TA database
username] -e POSTGRES_PASSWORD=[TA database password] -e POSTGRES_DB=[TA
database name] postgres
```

where:
- 5432 is the listening port of Postgres SQL server
- sse-network is the name of docker network which is created in step (2)
- sse-postgres is a chosen name of the container

- [path to folder which persistenly stores data] is a chosen folder to store data. This helps to preserve data even if docker container is shut down.
- [TA database username] is the chosen username in deloy.env file
- [TA database user password] is the chosen password in deploy.env file
- [TA database name] is a chosen database name in deploy.env file

Example:

```
sudo docker run -d -p 5432:5432 --name ta-postgres --net ta-network -e
PGDATA:/var/lib/postgresql/data/ -v
/home/ubuntu/data/ta:/var/lib/postgresql/data/ -e POSTGRES_USER=ta1 -e
POSTGRES_PASSWORD=tapwd1 -e POSTGRES_DB=tadb1 postgres
```

4. Build and deploy TA server as a docker container

```
sudo docker build -t ta [folder contains TA source code]

sudo docker run -d -it -p 80:8000 --name ta-server --net ta-network --env-
file [folder contains TA source code]/deploy.env ta
```

where:
- ta-network is the name of docker network which is created in step (1)
- 80: listening port on the host machine
- 8000: listening port on the TA container, which is configured in "entrypont.sh" of the source code
- "deploy.env" file contains configuration details of the server

All the logs will be written to standard output, which can be seen through the command

```
docker logs ta-server
```

where

- ta-server is the name of the container defined above in step (3).

In order to test if TA server is successfully deployed, you can try to access http://[IP address of TA]/api/v1/, which return the list of supported APIs like below

```
{
    "fileno": {
        "list_endpoint": "/api/v1/fileno/",
        "schema": "/api/v1/fileno/schema/"
    },
    "longrequest": {
        "list_endpoint": "/api/v1/longrequest/",
        "schema": "/api/v1/longrequest/schema/"
    },
    "search": {
        "list_endpoint": "/api/v1/search/",
        "schema": "/api/v1/search/schema/"
    },
```

```
    "searchno": {
        "list_endpoint": "/api/v1/searchno/",
        "schema": "/api/v1/searchno/schema/"
    }
}
```

SSE Client (example application)

This web application is an example application, which shows how to utilize the provided JS library, i.e. sse.js, to upload/ searching data.

1. Configure URL of TA and SSE, salt value and iv value (which are needed for encryption) in deploy.env file.

2. Build and deploy SSE Client application as a docker container

```
sudo docker build -t sse-client [folder contains web application source code]

sudo docker run -d -it -p 80:80 --name sseclient --env-file [folder contains web application source code]/deploy.env sse-client

# Fill contanst values (url_ta_ip,etc.) in sse.js with values from environment variables, which are defined in deploy.env

sudo docker exec -it sseclient bash -c 'sed -i -e "s|ta_url|$ta_url|" sse/static/js/sse.js'

sudo docker exec -it sseclient bash -c 'sed -i -e "s|sse_url|$sse_url|" sse/static/js/sse.js'

sudo docker exec -it sseclient bash -c 'sed -i -e "s|salt_value|$salt_value|" sse/static/js/sse.js'

sudo docker exec -it sseclient bash -c 'sed -i -e "s|iv_value|$iv_value|" sse/static/js/sse.js'
```

where:
- 80: listening port on the host machine, and on the SSE client container (which is configured in "entrypont.sh" of the source code)
- "deploy.env" contains configurations for URL of TA server (ta_url), URL of SSE srever (sse_url), salt value and iv value. The salt and iv values are used for encrypting/ decrypting, which are shared between users and TA.

Local deployment in development mode

The following paragraphs instruct how to run the three applications, i.e. SSE server, TA, SSE client, in development mode. In this scenario, the applications will not be run in docker containers. Furthermore, SQLite is used as database instead of Postgres SQL.

SSE Server

1. Re-configure database to use SQLite instead of Postgres SQL in "SQLServer/settings.py" in the source code

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3', # for local test
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'), # for local test
        #'ENGINE': 'django.db.backends.postgresql_psycopg2',
        #'NAME': os.environ['DB_NAME'],# database name
        #'USER': os.environ['DB_USER'], # user name
        #'PASSWORD': os.environ['DB_PASSWORD'], # user password
        #'HOST': os.environ['DB_HOST'], # postgres server
        #'PORT': os.environ['DB_PORT'],  # postgres port
    }
}
```

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

```
python3 manage.py migrate
```

3. Create environment variables, i.e. DJANGO_LOGLEVEL, TA_SERVER url, ALLOWED_HOSTS, which SSE server needs to access

```
source local.sh
```

where local.sh is provided in the source code
4. Run SSE server in development mode

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

TA Server

1. Re-configure database to use SQLite instead of Postgres SQL in "TA/settings.py" in the source code

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3', # for local test
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'), # for local test
        #'ENGINE': 'django.db.backends.postgresql_psycopg2',
        #'NAME': os.environ['DB_NAME'],# database name
        #'USER': os.environ['DB_USER'], # user name
        #'PASSWORD': os.environ['DB_PASSWORD'], # user password
```

```
        #'HOST': os.environ['DB_HOST'], # postgres server
        #'PORT': os.environ['DB_PORT'],  # postgres port
    }
}
```

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

```
python3 manage.py migrate
```

3. Run SSE server in development mode

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

SSE client

Run SSE client in development mode

```
# Fill contanst values (url_ta_ip,etc.) in sse.js with values from
environment variables, which are defined in deploy.env

sed -i -e "s|ta_url|$ta_url|" sse/static/js/sse.js

sed -i -e "s|sse_url|$sse_url|" sse/static/js/sse.js

sed -i -e "s|salt_value|$salt_value|" sse/static/js/sse.js

sed -i -e "s|iv_value|$iv_value|" sse/static/js/sse.js

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