

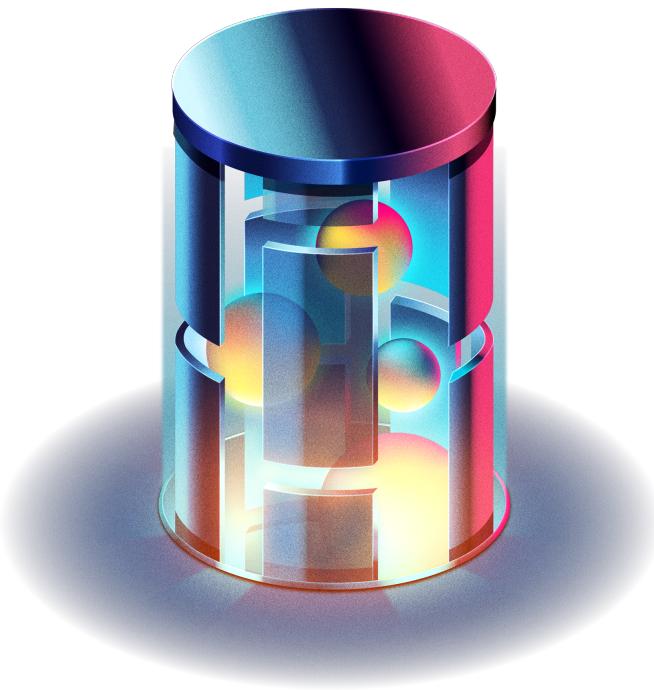


Tutorial: Executing a blockchain-based confidential application in minutes

IEEE ICBC, 1-5 May 2023

Aimen Djari, Anthony Simonet-Boulogne and Ambre Toulemonde

05/01/2023



Contents

| | |
|--|-----------|
| I. Context | 3 |
| II. Hands-on | 4 |
| 1. Initialize | 4 |
| 1.1. Set Development Environment | 4 |
| 1.2. Fill app.py | 5 |
| 1.3. Fill Dockerfile | 6 |
| 1.4. Fill sconify.sh | 7 |
| 2. Create & Deploy App | 8 |
| 2.1. Initialize app | 8 |
| 2.2. Sconify app | 8 |
| 2.3. Deploy app | 8 |
| 3. Create & Deploy Dataset | 9 |
| 3.1. Initialize Dataset | 9 |
| 3.2. Encrypt dataset | 10 |
| 3.3. Publish encrypted file | 10 |
| 3.4. Deploy Dataset | 10 |
| 4. Create Requester Secret | 11 |
| 5. Execute | 11 |
| 6. Get Result | 12 |
| 7. Have fun | 12 |
| III. Bonus | 13 |
| 1. Result Encryption | 13 |
| 2. Pass arguments instead of secrets | 13 |

I. Context

In this tutorial, you will learn how to build and run a Confidential Computing application with the Scone TEE framework using a confidential dataset and requester secrets.

Dependencies

- Npm (9.2.0)
- Node.js (19.2.0)
- Unzip (6.00)

Install the iExec SDK cli

```
$ npm i -g iexec
```

Github

- Repository address: <https://github.com/icbc-tutorial-tee/dataset>
- Username: icbc-tutorial-tee
- Password: icbctutorialtee

II. Hands-on

1. Initialize

For this tutorial, we will initialize our development environment by (*i*) creating a folder tree, (*ii*) configuring our environment, (*iii*) initializing your wallet's storage space¹ and (*iv*) creating the files you'll need for our confidential application execution.

1.1. Set Development Environment

1. Let us create a new folder where to develop our confidential application.

```
$ mkdir ICBC-TEE-TUTORIAL  
$ cd ICBC-TEE-TUTORIAL/
```

2. Initialize the iExec project and create your wallet (you'll need to provide a password to unlock your wallet, **do not forget it!**)

```
$ iexec init
```

3. Create the source code directory

```
$ mkdir src
```

4. Set the proper configuration for our demo

```
$ sed -i '' 's|"bellecour": {}|"bellecour": { "sms": { "scone":  
"https://sms.scone-debug.v8-bellecour.iex.ec" } }|g' chain.json
```

5. Initialize your wallet's storage space

```
$ iexec storage init --tee-framework scone
```

¹iExec enables running apps producing output files, you will need a place for storing your apps outputs.

1.2. Fill app.py

For this tutorial, we will execute a confidential app written in Python, the source code is the following (you can also find the file in the github repository provided in Section I):

```
import os, sys
import json
import xlrd

iexec_out = os.environ['IEXEC_OUT']
iexec_in = os.environ['IEXEC_IN']
dataset_filename = os.environ['IEXEC_DATASET_FILENAME']

try:
    secret_email = os.environ["IEXEC_REQUESTER_SECRET_1"]
except Exception:
    exit(11)

text = 'This email is safe! \n'

# Check the confidential file exists and open it
try:
    # Open the Workbook and Worksheet
    workbook = xlrd.open_workbook(iexec_in + '/' + dataset_filename)
    worksheet = workbook.sheet_by_index(0)

    # Iterate the rows
    for i in range(1, worksheet.nrows):
        sheet_email = worksheet.cell_value(i,2)

        if sheet_email.lower() == secret_email.lower():
            text = "This email is compromised! \n"
            break
except OSError:
    exit(3)

# Append some results in /iexec_out/
with open(iexec_out + '/result.txt', 'w+') as fout:
    fout.write(text)
# Declare everything is computed
with open(iexec_out + '/computed.json', 'w+') as f:
    json.dump({ "deterministic-output-path" : iexec_out + '/result.txt' }, f)
```

1.3. Fill Dockerfile

iExec leverages Docker containers to ensure the execution of your application on a decentralized infrastructure. The Dockerfile we will need to containerize our application is the following (you can also find the file in the github repository provided in Section I):

```
FROM python:3.7.3-alpine3.10
### install python dependencies if you have some
RUN pip3 install xlrd==1.2.0
COPY ./src /app
ENTRYPOINT ["python3", "/app/app.py"]
```

1.4. Fill sconify.sh

To build Confidential Computing (TEE) application, a developer would usually need to use the Intel SGX SDK. With iExec, you don't need to manipulate it. Instead iExec supports high-level frameworks, known as TEE frameworks, such as **SCONE**. The advantage of using SCONE is the ability to make the application Intel SGX-enabled without changing the source code. The only thing we are going to do is rebuilding the app using the Trusted-Execution-Environment tooling provided by SCONE as we will see in sub-section 2.2. To do so, we will use the following bash script (you can also find the file in the github repository provided in Section I):

```
#!/bin/bash
# Declare the app entrypoint
ENTRYPOINT="python3 /app/app.py"
# Declare image related variables
IMG_NAME=tee-scone-hello-world
IMG_FROM=${IMG_NAME}:temp-non-tee
IMG_TO=${IMG_NAME}:1.0.0-debug
# build the regular non-TEE image
docker build . -t ${IMG_FROM}
# Run the sconifier to build the TEE image based on the non-TEE image
docker run -it \
    -v /var/run/docker.sock:/var/run/docker.sock \
    docker-icbc.westeurope.cloudapp.azure.com/iexec-sconify-image:5.7.5-v8 \
        \
        sconify_iexec \
        --name=${IMG_NAME} \
        --from=${IMG_FROM} \
        --to=${IMG_TO} \
        --binary-fs \
        --fs-dir=/app \
        --host-path=/etc/hosts \
        --host-path=/etc/resolv.conf \
        --binary=/usr/local/bin/python3.7 \
        --heap=1G \
        --dlopen=1 \
        --no-color \
        --verbose \
        --command=${ENTRYPOINT} \
        && echo -e "\n-----\n" \
        && echo "successfully built TEE docker image => ${IMG_TO}" \
        && echo "application mrenclave.fingerprint is $(docker run \
            --rm -e SCONE_HASH=1 ${IMG_TO})"
```

2. Create & Deploy App

In this section, we will show you how to create and deploy the application you'll be executing in the TEE.

2.1. Initialize app

TEE applications require some information (in `iexec.json` file) to be initialized by executing the following command:

```
$ iexec app init --tee
```

Note that these information need to be filled for deployment (see following sections for more details).

2.2. Sconify app

As said before, we will use SCONE TEE framework to make our application SGX-enabled. For that, we will use the `sconify.sh` file to wrap the sconification process.

```
$ chmod +x sconify.sh  
$ ./sconify.sh
```

2.3. Deploy app

Before being able to execute our app, we'll need to push it somewhere public such as the chosen worker will be able to download it, check its integrity (checksum and fingerprint verification) and execute it.

1. Push docker image (do not forget to replace `<name>` with your actual name)

```
$ docker tag tee-scone-hello-world:1.0.0-debug \  
docker-icbc.westeurope.cloudapp.azure.com/tee-app:1.0.0-<name>-debug  
$ docker push \  
docker-icbc.westeurope.cloudapp.azure.com/tee-app:1.0.0-<name>-debug
```

2. Update `iexec.json`

- Replace multiaddr value with
`docker-icbc.westeurope.cloudapp.azure.com/tee-app:1.0.0-<name>-debug` (do not forget to replace `<name>` with your actual name)
- Replace checksum value with the output given by the following command (do not forget to replace `<name>` with your actual name):

```
$ docker pull \
  docker-icbc.westeurope.cloudapp.azure.com/tee-app:1.0.0-<name>-debug
  | grep "Digest: sha256:" | sed 's/.*sha256:/0x/'
```

- Replace fingerprint with the output given by the following command (**do not forget to replace <name> with your actual name**):

```
$ docker run -it --rm -e SCONE_HASH=1 \
  docker-icbc.westeurope.cloudapp.azure.com/tee-app:1.0.0-<name>-debug
```

3. Deploy app

At this stage, your application is ready to be deployed on the iExec platform hence on the blockchain (the app will have its own address then) by running the following command:

```
$ iexec app deploy
```

Note that each time you change your app file (*src/app.py*), you have to sconify it and re-deploy it (*i.e.* you have to process the steps 2.2 and 2.3 with the new app file).

3. Create & Deploy Dataset

iExec datasets are an on-chain representation (as a smart contract) of a file that can be consumed by a Dapp. You can encrypt datasets on your machine, set a price for their usage (in RLC) and set restrictions about who/which Dapps can use them.

In this section, we show you how to create and deploy a file as a dataset so your application can use it. We use an Excel file containing names and email addresses, you can find it here:

<https://github.com/icbc-tutorial-tee/dataset/blob/main/emails.xlsx>

3.1. Initialize Dataset

You can initialize the dataset with the following:

```
$ iexec dataset init --encrypted
```

This command will create the `datasets/encrypted`, `datasets/original` and `.secrets/datasets` folders. A new `datasets` section will be added to the `iexec.json` file as well.

You will have to put your dataset file in `datasets/original`. If you modify your `emails.xlsx` and want to use our repository, rename it “`emails-<your-name>.xlsx`”.

3.2. Encrypt dataset

You can encrypt the dataset with the following:

```
$ iexec dataset encrypt
```

This command encrypts the dataset with standard AES encryption (AES-256).

3.3. Publish encrypted file

As you can see, the command generated the file

`datasets/encrypted/emails-<your-name>.xlsx.enc`. That file is the encrypted version of your dataset, you should push it somewhere accessible because the worker will download it during the execution process. You will have to enter this file's URI in the `iexec.json` file (`multiaddr` attribute) when you will deploy your dataset. Make sure that the URI is a DIRECT download link (not a link to a web page for example).

Note: You can use Github for example to publish the file but you should add `/raw/` to the URI like this: <https://github.com/icbc-tutorial-tee/dataset/raw/main/emails-<your-name>.xlsx.enc>

3.4. Deploy Dataset

1. Replace checksum in `iexec.json` with
-

```
$ sha256sum datasets/encrypted/emails-<your-name>.xlsx.enc
```

2. Deploy the dataset (the dataset will have its own address then)
-

```
$ iexec dataset deploy
```

3. Push the secret to decrypt the dataset (key)
-

```
$ iexec dataset push-secret
```

4. Check if the secret is present (optional)
-

```
$ iexec dataset check-secret
```

Note that each time you change your dataset file, you have to encrypt it, push it and deploy it (i.e. you have to process the steps 3.2, 3.3, 3.4 with the new dataset file).

4. Create Requester Secret

It is possible to securely consume requester-provided secrets in the application. The requester secrets are pushed in the SMS and only exposed to authorized apps inside enclaves.

In this section, we show you how to create and push any requester secret which we use as input variables in our example. In our case, we use one secret: `my_email` that will contain the email you want to test.

You can push the requester secret in the SMS with the following:

```
$ iexec requester push-secret my_email
```

If you need to check if the secret is present in the SMS, you can run the following:

```
$ iexec requester check-secret <my_secret_name>
```

5. Execute

Now, you can run your app with the following command. An app execution is called a deal, it is an agreement between all parties (requester and providers) in the iExec network. A deal is created when requester and providers' are agreed and is recorded in the blockchain. The following command will launch the app execution on the iExec platform (it will also output the dealid you will need for the end of the tutorial):

```
$ iexec app run --tag tee,scone --workerpool  
debug-v8-bellecour.main.pools.iexec.eth --secret 1=my-email --dataset  
<dataset_address> --skip-preflight-check
```

This command takes the secret `my_email` as input. Do not forget to replace `<dataset_address>` with your dataset address that is given in step 3.4 (in case you did not keep it when you deployed your dataset, you can find this address in `deployed.json`).

As opposed to the deal, a task within iExec is an instance where computing power is required (a deal can lead to more than one task). You can get the taskid with the following command (replace `<dealid>` with the output of the previous command):

```
$ iexec deal show <dealid>
```

Note that the application execution may take some time. If you need to debug your application, you can get debug information of a task (replace `<taskid>` with the output of the previous command):

```
$ iexec task debug <taskid>
```

and the app execution logs:

```
$ iexec task debug <taskid> --logs
```

You can also look at your task in the blockchain explorer:

<https://explorer.iex.ec/bellecour>

6. Get Result

The result is uploaded by the worker in public storage and you can get it with the following (replace <taskid> with the output of the previous command):

```
$ iexec task show <taskid> --download my-result && unzip my-result.zip \  
-d my-result && cat my-result/result.txt
```

7. Have fun

In this tutorial, we showed you how to check if a secret email can be found in an encrypted dataset. The excel file (`emails.xlsx`) we provided you contains more information than just email addresses, you can now modify `app.py` to include other columns and personalize your app.

III. Bonus

1. Result Encryption

As you can see in step 6, the result is uploaded in public storage and anyone can download it and get the result. It is possible to keep your result private by encrypting it with the following instructions. Before executing your app, you have to generate and deploy result encryption keys as follows:

- Generate result encryption key

```
$ iexec result generate-encryption-keypair
```

- Push result encryption key

```
$ iexec result push-encryption-key --tee-framework scone
```

- Check if the result encryption key is present (optional)

```
$ iexec result check-encryption-key --tee-framework scone
```

Then, you can execute your app with the following:

```
$ iexec app run --tag tee,scone --workerpool
debug-v8-bellecour.main.pools.iexec.eth --secret 1=my-email --dataset
<dataset_address> --skip-preflight-check --encrypt-result
```

You can get the taskid with the following command:

```
$ iexec deal show <dealid>
```

2. Pass arguments instead of secrets

1. Read argument in app.py

```
arg = sys.argv[1..n]
```

2. Add argument in command line

```
$ iexec app run --tag tee,scone --workerpool
debug-v8-bellecour.main.pools.iexec.eth --secret 1=my-email
--args arg1 arg2 argn --dataset <dataset_address>
--skip-preflight-check --encrypt-result
```
