# Full disk encryption Recipe

## 1.Register a key

The key to encrypt the image disk is distributed by the KBS (Key broker Service), besides, which will bind a unique keyid with the key. The keyid is the key's identifier in the KBS. To run the KBS in the host machine refer to steps mentioned in the following link - <https://github.com/intel/trustauthority-kbs.git>

Once the KBS up and running use below curl request to retrieve key and key-id from KBS.

1. Fetch the bearer token from KBS.

curl --insecure --location 'https://127.0.0.1:9443/kbs/v1/token' --header 'Accept: application/jwt' --header 'Content-Type: application/json' --data '{ "username": "username", "password": "passowrd" }'

Specify the username and password given in the KBS ENV file.

Also modify the KBS url based on your configuration.

1. Create a key transfer policy for the TDX workload.
2. curl --insecure --location 'https://127.0.0.1:9443/kbs/v1/key-transfer-policies' --header 'Accept: application/json' --header 'Content-type: application/json' --header 'Authorization: Bearer <token generated in step 1>' --data '{ "attestation\_type": "TDX", "tdx": { "attributes": { "mrsignerseam": [ "000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000" ], "mrseam": [ "9790d89a10210ec6968a773cee2ca05b5aa97309f36727a968527be4606fc19e6f73acce350946c9d46a9bf7a63f8430" ], "mrtd": [ "f2dd2696f69b950645832bdc095ffd11247eeff687eeacdb57a58d2ddb9a9f94fea40c961e19460c00ffa31420ecbc18" ], "seamsvn": 4, "enforce\_tcb\_upto\_date": false } } }'

Change the attribute values from your TD machine.

1. Create a key.

curl --insecure --location 'https://127.0.0.1:9443/kbs/v1/keys' --header 'Accept: application/json' --header 'Content-type: application/json' --header 'Authorization: Bearer token generated in step 1' --data '{"key\_information": { "algorithm": "RSA", "key\_length": 3072 },

"transfer\_policy\_id" : "take the policy ID from step3"

}'

1. Retrieve the key.

curl --insecure --location 'https://https://127.0.0.1:9443/<key transfer policy link from step 4> ' --header 'Accept: application/json' --header 'Content-type: application/json' --header 'Attestion-type: "TDX"' --data '{ "attestation\_token":" token from TD"}'

1. The response of step 5 will have wrapped\_key and wrapped\_swk, these are in encrypted format and use the below script to extract key suppose to use for encrypting the image.

## 2. Build the fde-agent.

## The fde-agent is placed in the tdx-tools repo, go to FDE\_DIR=attestaion/full-disk-encryption.

## The fde-agent is responsible for decrypting a guest image and mounting it as the rootfs. The fde-agent depends on dynamic libraries libtdx-attest and libtdx-attest-dev in DCAP 1.16. The DCAP 1.16 can be downloaded from https://download.01.org/intel-sgx/sgx-dcap/1.16/. Install the libraries and build the fde-agent using below the command.

## make clean -C ${FDE\_DIR}/attestation/full-disk-encryption

## make -C ${FDE\_DIR}/attestation/full-disk-encryption

## 3. Encrypted image preparation

There are several ways to create FDE image. This repo follows installing fde-agent and initramfs-tools in the encrypted image, append the option `cryptdevice` in the kernel command line and then updating the Grub config.

cd ${FDE\_DIR}/attestation/full-disk-enryption/tools/image

sudo ./fde-image.sh -k $KEY -i $KEY\_ID

The KEY=key and KEY\_ID=keyid are retrieved in step 1. This process will create the encrypted image.

## 4.OVMF variable enrollment.

Variables like KBS URL, KBS Certificate and Key-ID need to be enrolled into the OVMF. These variables are used from OVMF by fde-agent to help the remote attestation and retrieve key from KBS. The retrieved key will be used for decrypting the root file system in system boot time. The python script “tdx-tools/attestation/full-diskencryption/tools/image/enroll\_vars.py” helps enroll these variables into OVMF.

$ cd attestation/full-disk-encryption/tools/image

# Enroll user data

$ cat userdata.txt

{

"keyid":"sth"

}

#sth is the key-id retrieved during the step1.5

$ NAME="KBSUserData"

$ GUID="732284dd-70c4-472a-aa45-1ffda02caf74"

$ DATA="userdata.txt"

$ python3 tools/image/enroll\_vars.py -i OVMF.fd -o OVMF.fd -n $NAME -g $GUID -d $DATA

cat url.txt

https://192.168.6.4:9443/kbs/v1/keys/562bd3ec-0cdb-4b85-89eb-1d5fd7d76f83/transfer

$ # Enroll KBS URL

$ NAME="KBSURL"

$ GUID="0d9b4a60-e0bf-4a66-b9b1-db1b98f87770"

$ DATA="url.txt"

$ python3 tools/image/enroll\_vars.py -i OVMF.fd -o OVMF.fd -n $NAME -g $GUID -d $DATA

$ # Enroll KBS Certificate

$ NAME="KBSCert"

$ GUID="d2bf05a0-f7f8-41b6-b0ff-ad1a31c34d37"

$ DATA="cert.cer

$ python3 tools/image/enroll\_vars.py -i OVMF.fd -o OVMF.fd -n $NAME -g $GUID -d $DATA

#cert.cer - use the certificate used for bring up KBS service.

5. Boot TD VM

Launch a TD guest from the encrypted guest image. The script “tdxtools/start-qemu.sh” can launch it. Please use the encrypted guest image and OVMF mentioned in above steps for launching the VM.

Export below variables based on the imaged created in the system.

$ export OVMF\_PATH=/path/to/OVMF

$ export IMAGE\_PATH=/path/to/image

$ cd tdx-tools/

./start-qemu.sh -b grub -q vsock -o ${OVMF\_PATH} -i ${IMAGE\_PATH}

Verify the encryption status by running the command in the tdvm guest.

$ blkid

/dev/vda1: UUID="429dbd8c-fce8-45b0-8518-50af94f5ebba" LABEL="rootfs-enc" TYPE="crypto\_LUKS" PARTLABEL="rootfs" PARTUUID="5aba073d-1779-49f7-b2a7- d2a9d5aeefd9"

The TYPE of encrypted partition should be `crypto\_LUKS`