STANDARD OPERATING PROCEDURE
Do not Photocopy

Document Information Classification: Highly Restricted

Title:	**TRE Key Management**
Effective Date:	**07 Jun 2019**
Reference Number:	**SOP-09-09**
Version Number:	**1.2**
Owner:	**TRE Infrastructure and Security Management Process Owner,**
Review Date:	**26 Nov 2020**

Table of Contents

1. Purpose

A policy on the use, protection and lifetime of cryptographic keys should be developed and implemented through their whole lifecycle. This document provides the procedure for the management of TRE encryption keys.

2. Scope

All keys that are used in TRE encryption processes. This includes encryption of data volumes and encryption of communication between TRE core services.

3. Responsibilities

The TRE Administrators are responsible for:

- ensuring that encryption keys are securely managed and backed up.

4. Procedure

Data volumes attached to project instances must be encrypted at the block level. The Linux Unified Key Setup (LUKS) utility ‘cryptsetup’ is used to do this. The Ansible configuration management tool is used in ad-hoc command mode and executed from the Ansible host, to invoke LUKS key creation on project instances via a deployment script or command line.

LUKS key creation scripts performs the following steps:

- Create a restricted access key folder
- Use an entropy generation tool to create a keyfile in the key folder
- Restrict access to the keyfile to the root account only
- Format the data volume as an encrypted container
- Create the volume filesystem
- Create a mount point for the volume
- Populate crypttab to automate mounting of the volume at boot

```
#!/usr/bin/env bash
ansible tre001 -m file -a 'name=keys path=/root/keys mode=700 owner=root group=root state=directory'
ansible tre001 -m shell -a 'dd if=/dev/urandom of=/root/keys/crypto_keyfile.bin bs=512 count=10'
ansible tre001 -m file -a 'name=/root/keys/crypto_keyfile.bin mode=400 owner=root group=root'
ansible tre001 -m shell -a 'cryptsetup -q -y luksFormat /dev/vdb --key-file /root/keys/crypto_keyfile.bin'
ansible tre001 -m shell -a 'cryptsetup luksOpen /dev/vdb cryptoblock --key-file /root/keys/crypto_keyfile.bin'
ansible tre001 -m shell -a 'mkfs.ext4 /dev/mapper/cryptoblock'
ansible tre001 -m mount -a 'src=/dev/mapper/cryptoblock path=/home state=mounted fstype=ext4'
ansible tre001 -m lineinfile -a 'dest=/etc/crypttab line="cryptoblock /dev/vdb /root/keys/crypto_keyfile.bin luks,timeout=20"'
ansible tre001 -m shell -a ‘cryptsetup -v luksHeaderBackup /dev/vdb --header-backup-file cryptoblock-backup’
```

4.1. Keyfile and LUKS-header backup

It is critical that the LUKS keyfile and the header (metadata describing the LUKS encrypted container) are retrieved and securely stored. Both the keyfile and header must be duplicated to prevent corrupt or accident deletion. The primary store for these files is a KeePass database called KeyStore.kdbx. This database is replicated across two devices, an administration workstation in the Secure Data Room in Vaughan House and a USB device stored in the key safe.

Entries are created within the KeyStore database on the administration workstation. Each entry references the TRE VM instance. At least every month, the database is duplicated to the key safe, therefore online and offline versions are preserved.

```
$ sftp root@<instance>:/root/keys/crypto_keyfile.bin
$ sftp root@<instance>:/root/keys/cryptoblock-backup
```

When the header backup file transfer is complete and the file has been uploaded to the KeyStore database, the cryptoblock-backup file must be erased. The commands are:

```
$ ssh <instance>:/root/keys
$ shred -n 5 -uzxv /root/keys/cryptoblock-backup
```

It is important that the cryptoblock-backup file be erased, as this contains the unlocked and enabled key slot, as this allows recovery of the encrypted volume, it must be appropriately protected.

4.2. File level backups

Duplicity is used to backup file level data for data volumes that hold imported and derived data-sets within project instances. Duplicity encrypts backups with the GnuPG encryption utility. Public/Private keypairs are used to encrypt and sign backups. Each backup is encrypted against three GPG keys, a project instance private key, a TRE admin public key and a TRE admin offline public key.

The off-line TRE admin key has never been installed on a networked machine and is considered a safe key (un-tampered with) and therefore is guaranteed to not be compromised.

4.3. GnuPG

A script is called at deployment to setup the GPG keyring. The script performs the following tasks:

- Transfer TRE master keys to the instance
- Distribute the key input text file used to create an instance keyring
- Update the project instance key name
- Remove the keyring input text file
- Import the TRE master public keys to the instance keyring

```
#!/usr/bin/env bash
```

```
ansible tre022 -m copy -a "src=/root/ansible/dist/gpg-dist/tre-adminm-public-key.gpg.asc dest=/root/tre-adminm-public-key.gpg.asc mode=0600 owner=root group=root"
ansible tre022 -m copy -a "src=/root/ansible/dist/gpg-dist/tre-admin-public-key.gpg.asc dest=/root/tre-admin-public-key.gpg.asc mode=0600 owner=root group=root"
ansible tre001 -m copy -a "src=/root/ansible/dist/base/gpg-gen-file.txt.dist dest=/root/gpg-gen-file.txt mode=600 owner=root group=root"
ansible tre001 -m replace -a "dest=/root/gpg-gen-file.txt regexp='tre000' replace='tre001'"
ansible tre001 -m shell -a "gpg --gen-key --batch gpg-gen-file.txt"
ansible tre001 -m file -a "name=/root/gpg-gen-file.txt state=absent"
ansible tre001 -m shell -a "gpg --import tre-adminm-public-key.gpg.asc"
ansible tre001 -m shell -a "gpg --import tre-admin-public-key.gpg.asc"
```

Once the keyring has been created, the instance key passphrase must be changed by connecting to host via SSH and editing the keyring.

```
$ ssh <instance>
```

List the private key

```
$ gpg -K

/root/.gnupg/secring.gpg
------------------------
sec   4096R/EF4DE7A6 2018-02-21
uid              tre000
ssb   4096R/181DEAA0 2018-02-21

$ gpg --edit EF4DE7A6 passwd
```

Edit the passphrase by supplying the temporary default passphrase to enable modification to the existing key. Generate a new entry within the KeyStore database and open the passphrase generator. Select a character length over 16 characters with an upper, lower alpha-numerical case sequence. Copy the passphrase to the key change dialog box and enter it twice when prompted.

With the passhprase changed, the private key can now be exported and attached to the entry in the KeyStore database. We only require the secret key, however, retrieving the ownertrust and keyring directory helps preserve meta data.

Export GPG keys and store in KeyStore database

```
$ gpg2 --export-ownertrust > otrust-tre000.txt
$ gpg2 -a --export-secret-key > secret-key-tre000.asc
$ tar cjvf  /root/keys/gpgbackup-tbz2 /root/.gnupg
$ scp secret-key-tre000.asc
$ scp otrust-tre000.txt
$ scp /root/keys/gpgbackup-tbz2 /root/.gnupg
```

4.4. Key Database

Passphrase access to the KeyStore database is securely recorded in the TREv2 passphrase database.

Locations of database instances:

Primary online location – administration workstation in Secure Data Room
Secondary off-line location - USB Device within the Key Safe (VH 2.007)

5. Cross-referenced ISMS Documents

Number	Type	Title
<NO DATA>	<NO DATA>	<NO DATA>

6. Appendices

None