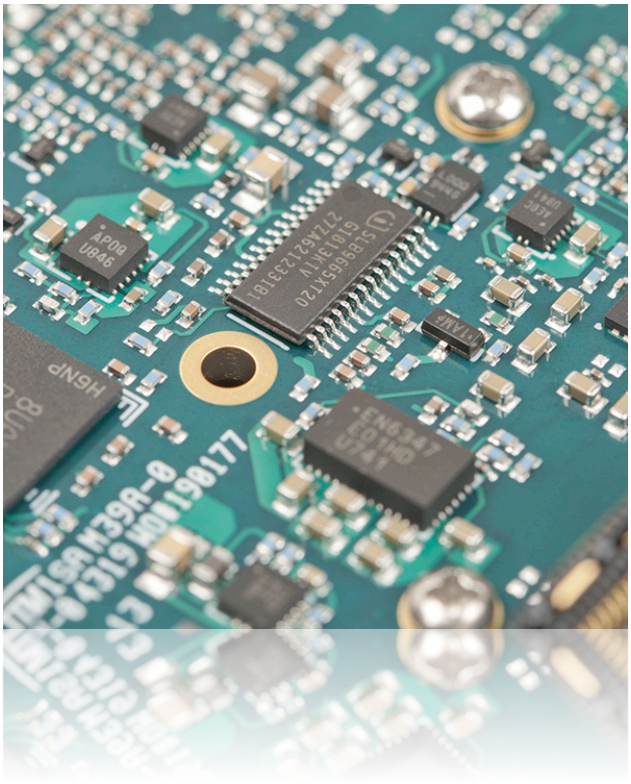
TPM Implementation

Trusted Platform Module

User's Manual

TPM Implementation Rev. A



RTD Embedded Technologies, Inc.
AS9100 and ISO 9001 Certified

RTD Embedded Technologies, Inc.
103 Innovation Boulevard
State College, PA 16803 USA
Telephone: 814-234-8087
Fax: 814-234-5218

www.rtd.com

sales@rtd.com
techsupport@rtd.com

PC/104

DESIGNED AND MADE IN THE UNITED STATES

AS9100 - ISO 9001 CERTIFIED

Revision History

Rev A	Initial Release

Advanced Analog I/O, Advanced Digital I/O, aAIO, aDIO, a2DIO, Autonomous SmartCal, "Catch the Express", cpuModule, dspFramework, dspModule, eBuild, expressMate, ExpressPlatform, "MIL Value for COTS prices", multiPort, PlatformBus, and PC/104EZ are trademarks, and "Accessing the Analog World", dataModule, IDAN, HiDAN, HiDANplus, RTD, the RTD logo, and StackNET are registered trademarks of RTD Embedded Technologies, Inc. (formerly Real Time Devices, Inc.). PS/2 is a trademark of International Business Machines Inc. PCI, PCI Express, and PCIe are trademarks of PCI-SIG. PC/104, PC/104-Plus, PCI-104, PCIe/104, PCI/104-Express and 104 are trademarks of the PC/104 Consortium. All other trademarks appearing in this document are the property of their respective owners.

Failure to follow the instructions found in this manual may result in damage to the product described in this manual, or other components of the system. The procedure set forth in this manual shall only be performed by persons qualified to service electronic equipment. Contents and specifications within this manual are given without warranty, and are subject to change without notice. RTD Embedded Technologies, Inc. shall not be liable for errors or omissions in this manual, or for any loss, damage, or injury in connection with the use of this manual.

Copyright © 2021 by RTD Embedded Technologies, Inc. All rights reserved.

Table of Contents

1 Introduction **5**

1.1 Description 5

1.2 Known Security Weaknesses 5

1.3 References 5

1.4 RTD CPU Modules with TPM 6

1.5 Contact Information 6

1.5.1 Sales Support 6

1.5.2 Technical Support 6

2 RTD BIOS Navigation **7**

2.1 Entering the BIOS 7

2.2 Enter the Setup Utility, On Chief River CPUs enter the SCU 7

2.3 Go to the Security Tab, Chief River CPUs will show a TPM 1.2 device 8

2.4 Set the TPM to Available 8

2.5 Enable the TPM 9

2.6 Clear the TPM 9

2.7 Set up an Administration Password, Optional but Recommended 10

2.8 Go to the Boot Tab, Optional for Linux Operating Systems 10

2.9 Press F10 and Select Save 11

3 Windows Implementation **12**

3.1 Compatible Versions 12

3.2 Software Implementation 12

3.3 Migration between TPM 1.2 and 2.0 with Windows 12

4 Linux Implementation **13**

4.1 Linux Complications 13

4.2 TPM 1.2 Installation Instructions 13

4.3 TPM 2.0 Instructions 19

4.4 Migration between TPM 1.2 and 2.0 with Linux 28

4.5 Reading Windows Bitlocker Encrypted Partitions in Linux 28

5 Conclusion **31**

5.1 Windows, Security over Flexibility 31

5.2 Linux, Flexibility over Security 31

6 Disclaimer and Limitation of Liability **32**

6.1 Content Disclaimer 32

6.2 Referrals and Links to Third Party Sites 32

1 Introduction

1.1 Description

Please read this entire guide before attempting to configure your TPM and system.

What is a TPM?

TPM stands for "Trusted Platform Module". The TPM is a chip on your mother board, either directly soldered or on a removable module, that provides tamper resistant storage for keys that can be used to encrypt file systems. This device protects the data on your drives from being read should your device or drive be stolen. An un-encrypted drive can be removed from your system and connected to another, bypassing the need for a user account and password. With a TPM encrypted drive the only way to access the data is by logging in as a user with access to the directories containing your sensitive information. As such, it is highly recommended to save a copy of any drive encryption keys either on paper or a USB drive in the event your device experiences a hardware failure. Without this key your data will be lost with almost zero options for recovery. For best results, this key should never be saved locally, even temporarily, as the key may leave a lasting trace on flash or magnetic drive.

1.2 Known Security Weaknesses

The TPM chip has a handful of known security weaknesses:

- Removeable TPM modules can have monitoring hardware inserted between them and the motherboard and the eSPI or LPC bus data can be decoded, yielding any keys requested from the TPM. This is a comparatively cheap and simple method of acquiring the encryption key. RTD CPU modules have surface mount soldered TPM chips which makes this method of attack more difficult.

- Entities with access to electron microscopes and high-power lasers can break into the TPM chip and read its contents. In 2010, Christopher Tarnovsky used an electron microscope to read the keys from the bus, much like the above method but requiring million-dollar equipment. This crack was performed on a TPM 1.2 chip, but TPM 2.0 chips are still vulnerable.

- TPM 1.2 chips were only capable of using SHA1 hashing and RSA keys which can be cracked with brute force by super computers. TPM 2.0 chips addressed this issue by using SHA256 hashing and AES keys which are considered nearly impossible to crack with brute force. Often AES-256-GCM-Sha256 key security is described with superlatives like it would take every computer on the planet tens of billions of years or a computer with energy consumption equal to the Sun's output would take 5 years. These statements convey the security provided by AES-256-GCM-Sha256 encryption, which is the method that will be used by the TPM 2.0 installation instructions shown below for both Windows and Linux.

- In 2015 the United States CIA claimed to have determined TPM keys by using differential power analysis. A technique that converts changes in current flow in and out the TPM chip into the bits being transmitted on the LPC or eSPI bus. While exact details of the method were not shared, it likely does not require expensive equipment like an electron microscope.

- In 2017 a new algorithm, named Return of Coppersmith's Attack (ROCA), was discovered that allowed the prediction of private RSA keys from the public half of the key. This is not a problem for TPM 2.0 chips using AES encryptions, but older TPM 1.2 chips and USB smart cards were limited to RSA keys and were therefore vulnerable to ROCA. The currently accepted fix to this attack is to use RSA key lengths that have been shown to be less susceptible to ROCA, which include 1952-bit, 3072-bit, and 3936-bit. However, it is still ideal to upgrade to TPM 2.0 AES encryption hardware wherever possible.

- TPM chips are still vulnerable to software attacks, malware, and viruses. Once the system has booted, the encrypted drives are unlocked with the keys from the TPM, allowing any software attacks full access to important data and even the keys within the TPM. As such, it is still highly recommended to maintain firewalls, anti-virus, and any other internet security methods.

1.3 References

The following is a list of references used throughout this design. It is expected that the audience of this document have a working understanding of these references.

Title	Description	Source
TPM Summary and Operation	An in-depth white paper on how the TPM works.	https://trustedcomputinggroup.org/resource/trusted-platform-module-tpm-summary/

1.4 RTD CPU Modules with TPM

The following RTD CPU modules are available with TPM chips:

Table 1: RTD CPUs with TPM

CPU Series	Description	TPM Version	More Information and Purchasing
Chief River	Intel Core i7 based CPU modules	1.2	https://www.rtd.com/PC104/CM/processor_cr.htm
Bay Trail	Intel Atom E3800 based CPU modules	2.0	https://www.rtd.com/PC104/CM/processor_bt.htm

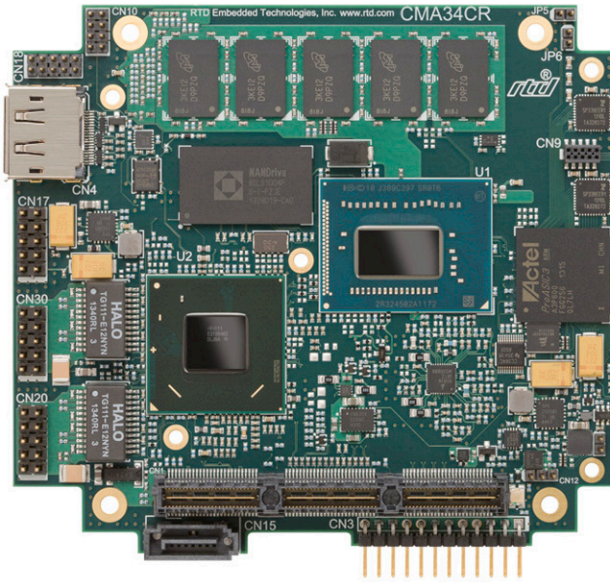


Figure 1: Chief River CPU Module

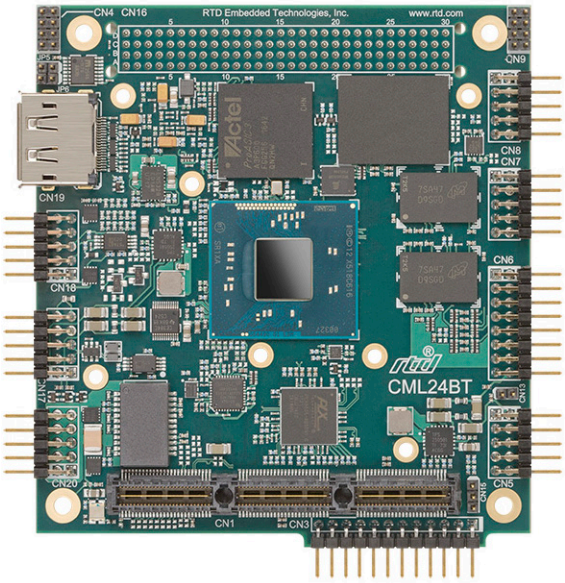


Figure 2: Bay Trail CPU Module

1.5 Contact Information

1.5.1 Sales Support

For sales inquiries, you can contact RTD Embedded Technologies sales via the following methods:

Phone: 1-814-234-8087 Monday through Friday, 8:00am to 5:00pm (EST).
E-Mail: sales@rtd.com

1.5.2 Technical Support

If you are having problems with your system, please try the steps in the Troubleshooting section of this manual.

For help with this product, or any other product made by RTD, you can contact RTD Embedded Technologies technical support via the following methods:

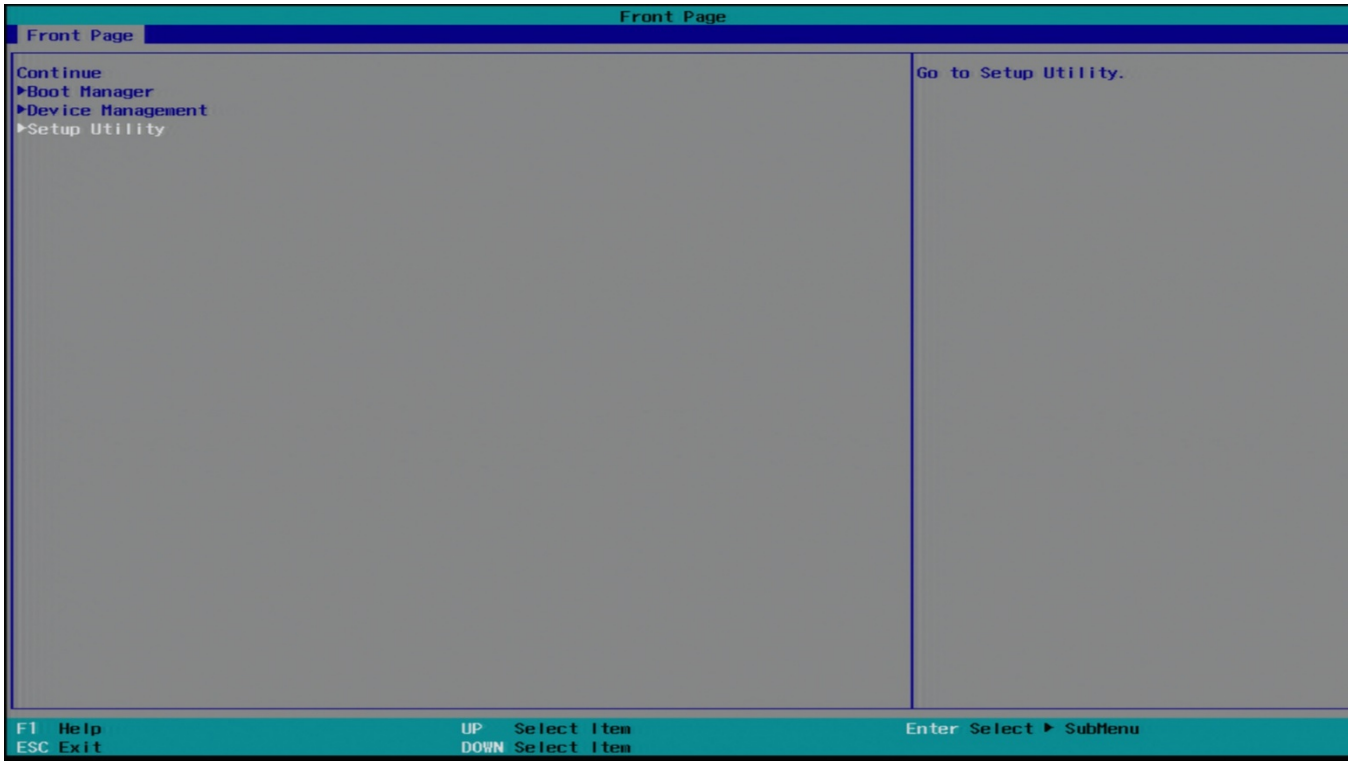
Phone: 1-814-234-8087 Monday through Friday, 8:00am to 5:00pm (EST).
E-Mail: techsupport@rtd.com

2 RTD BIOS Navigation

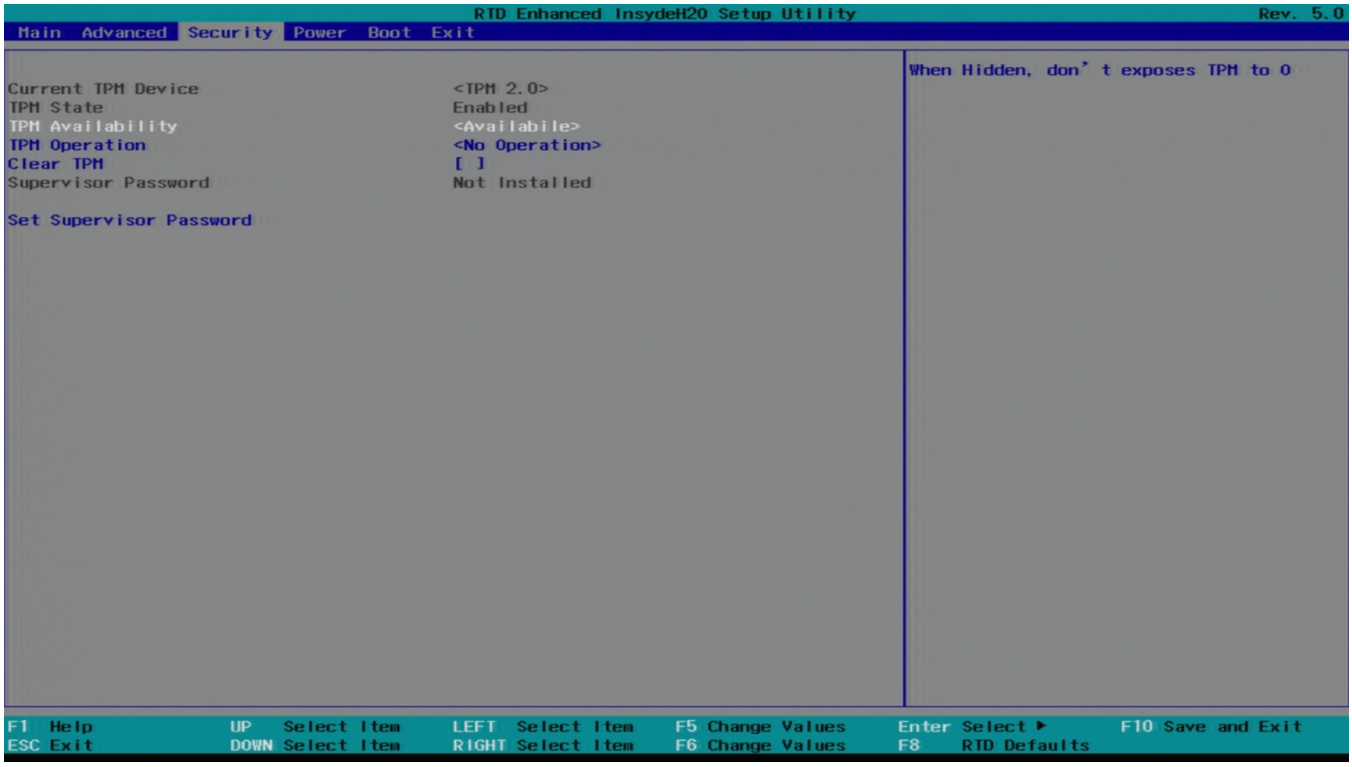
2.1 Entering the BIOS

Pressing the <Esc> key during boot will bring up the BIOS main menu. The time window for hitting escape can be very short if the default Quick Boot option is enabled. Hitting escape repeatedly during boot is recommended. The images below are from a Baytrail CPU module.

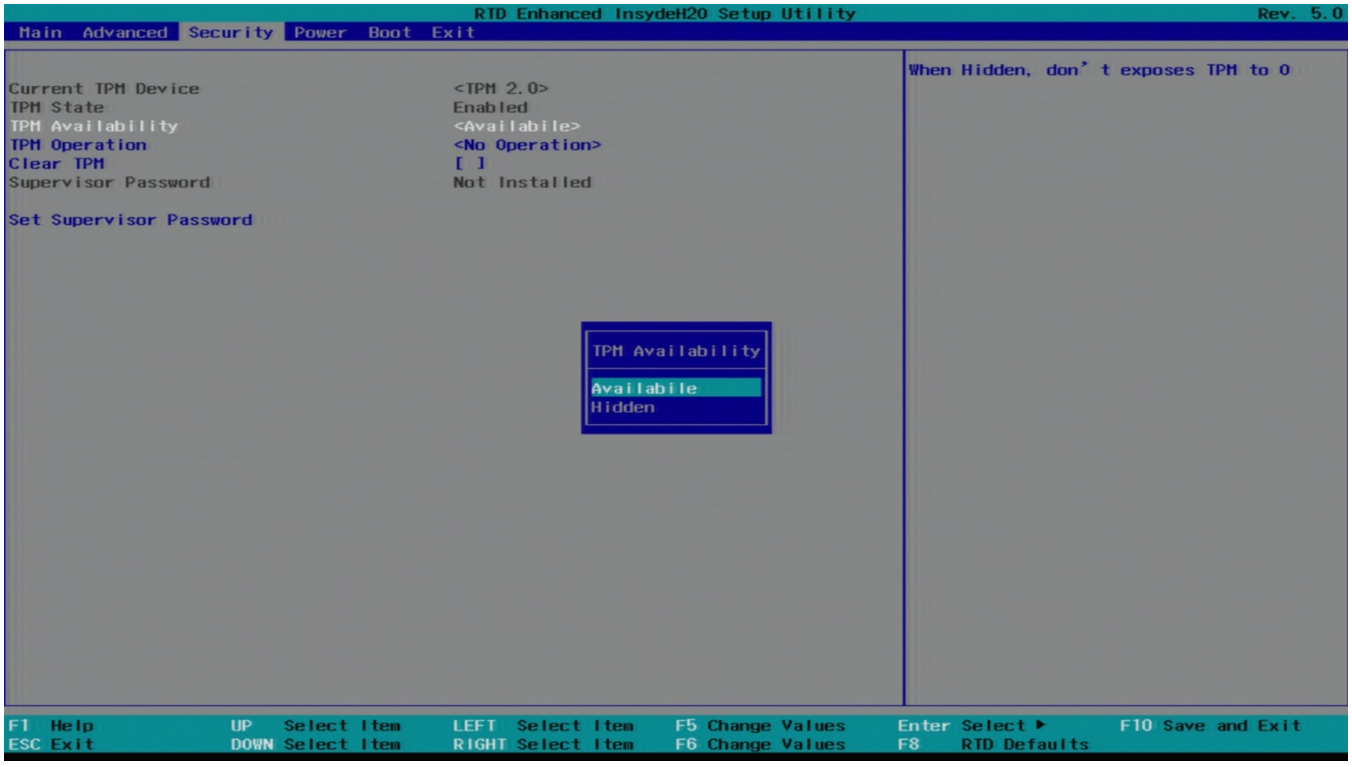
2.2 Enter the Setup Utility, On Chief River CPUs enter the SCU



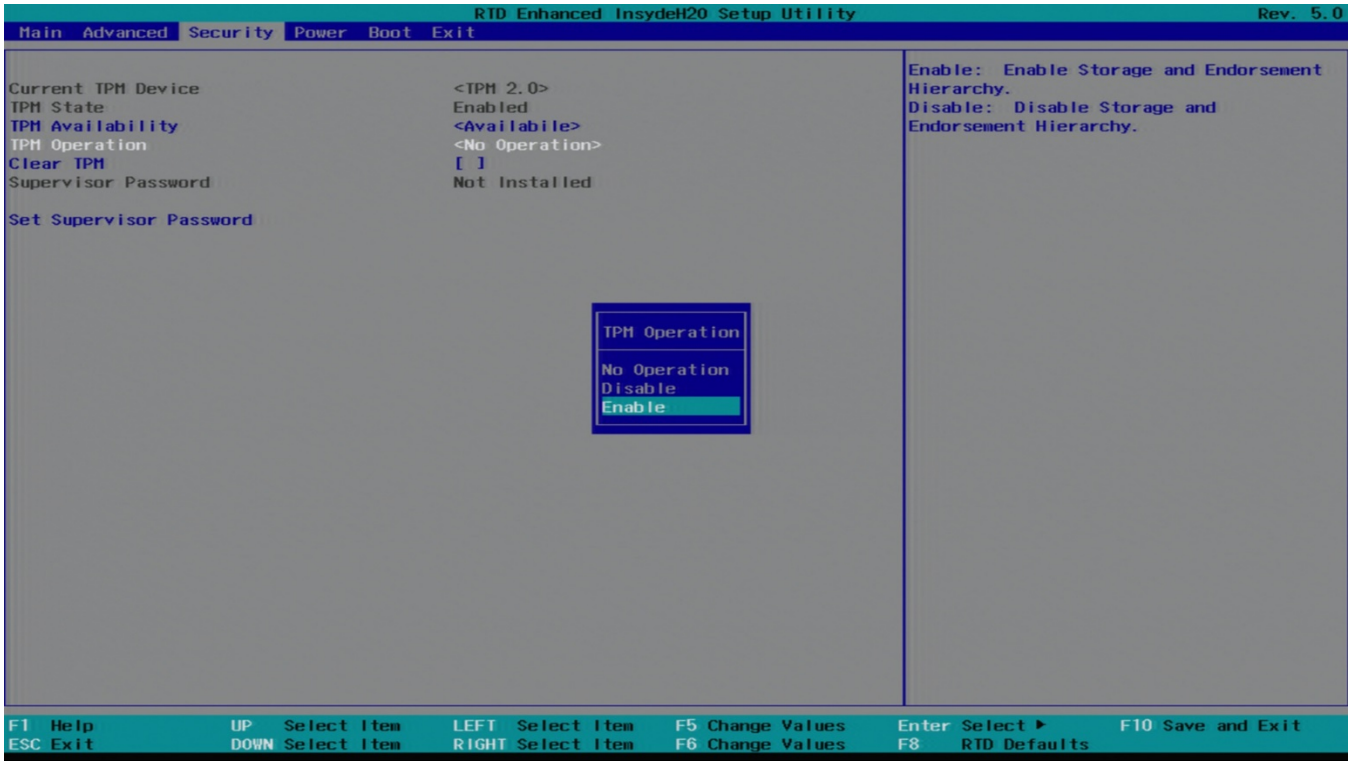
2.3 Go to the Security Tab, Chief River CPUs will show a TPM 1.2 device



2.4 Set the TPM to Available

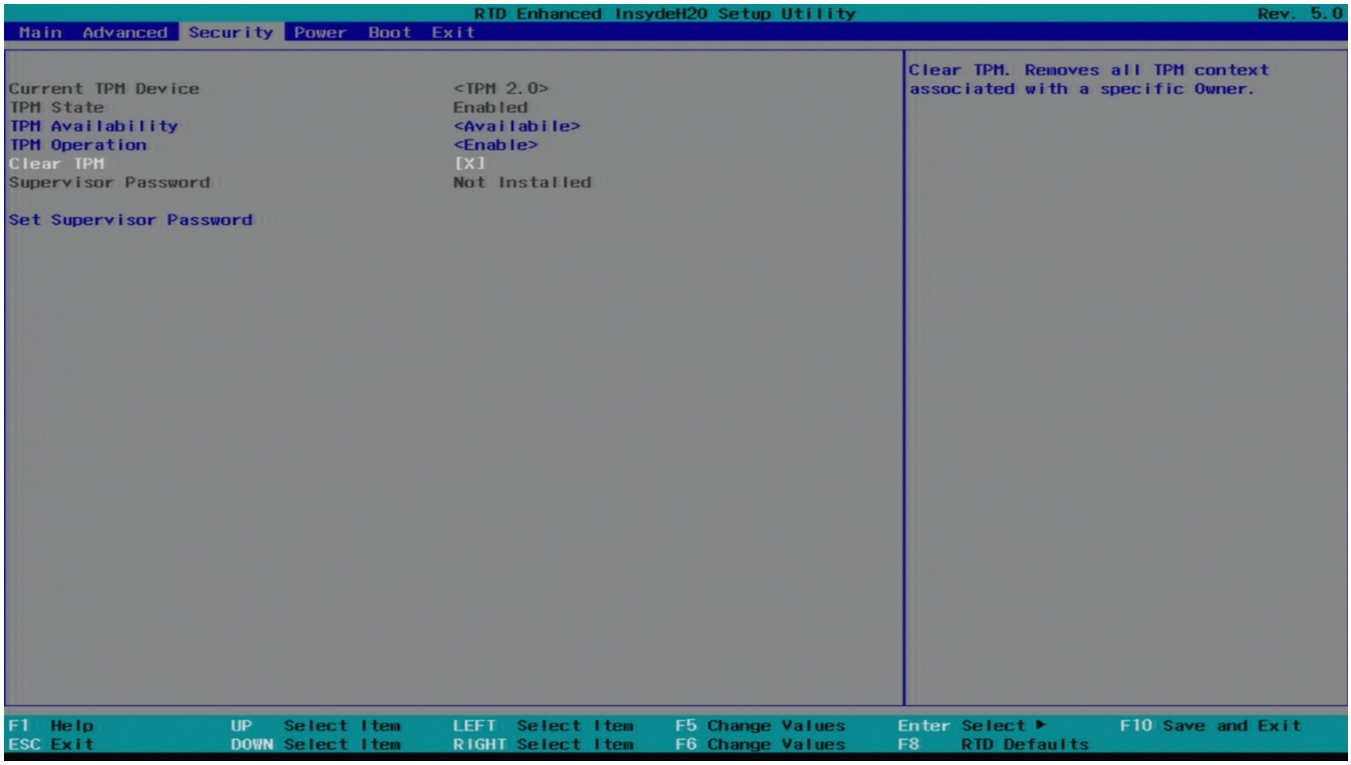


2.5 Enable the TPM

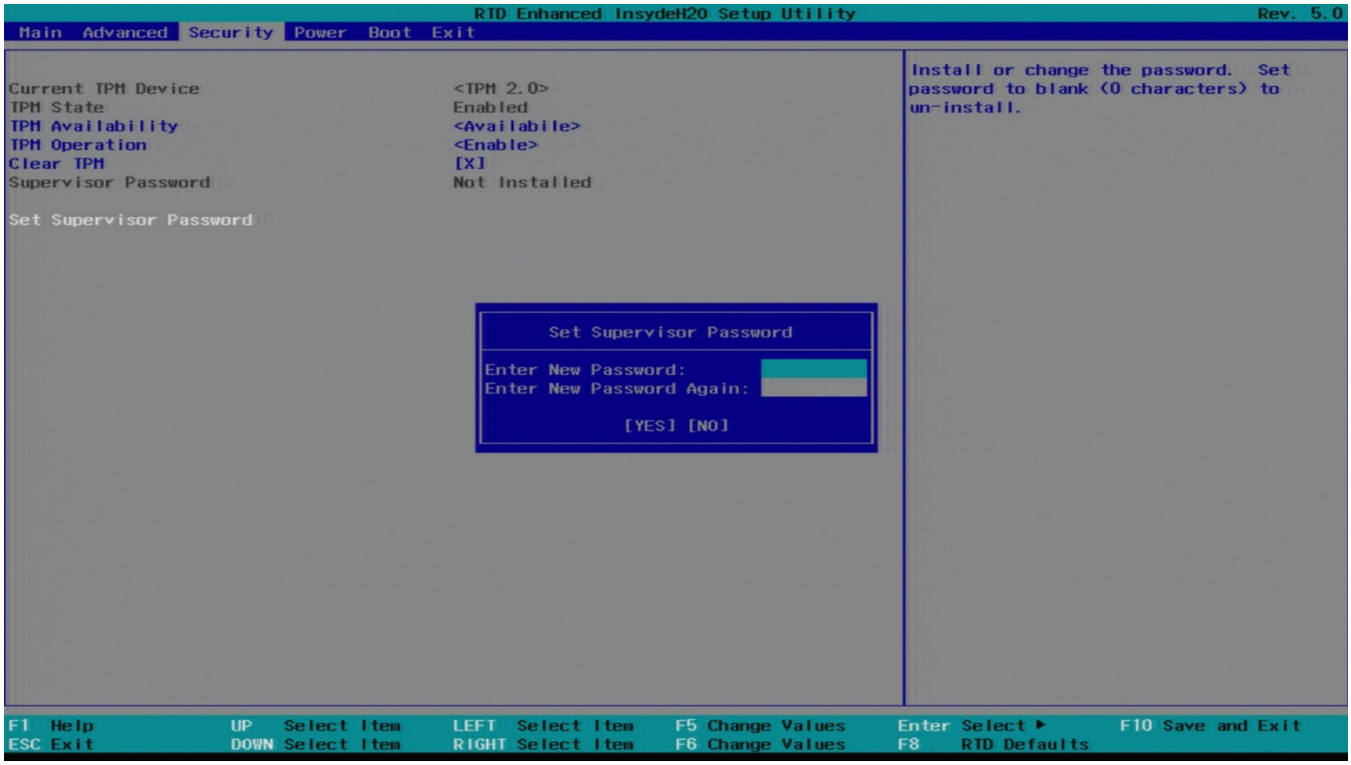


2.6 Clear the TPM

Do not perform this step if your TPM contains keys that are not backed up or if you have previously set up the TPM for the current installation of your operating system. Clearing the TPM will delete any keys inside and may prevent previously installed Windows operating systems from booting and Linux operating systems from mounting any TPM encrypted partitions.

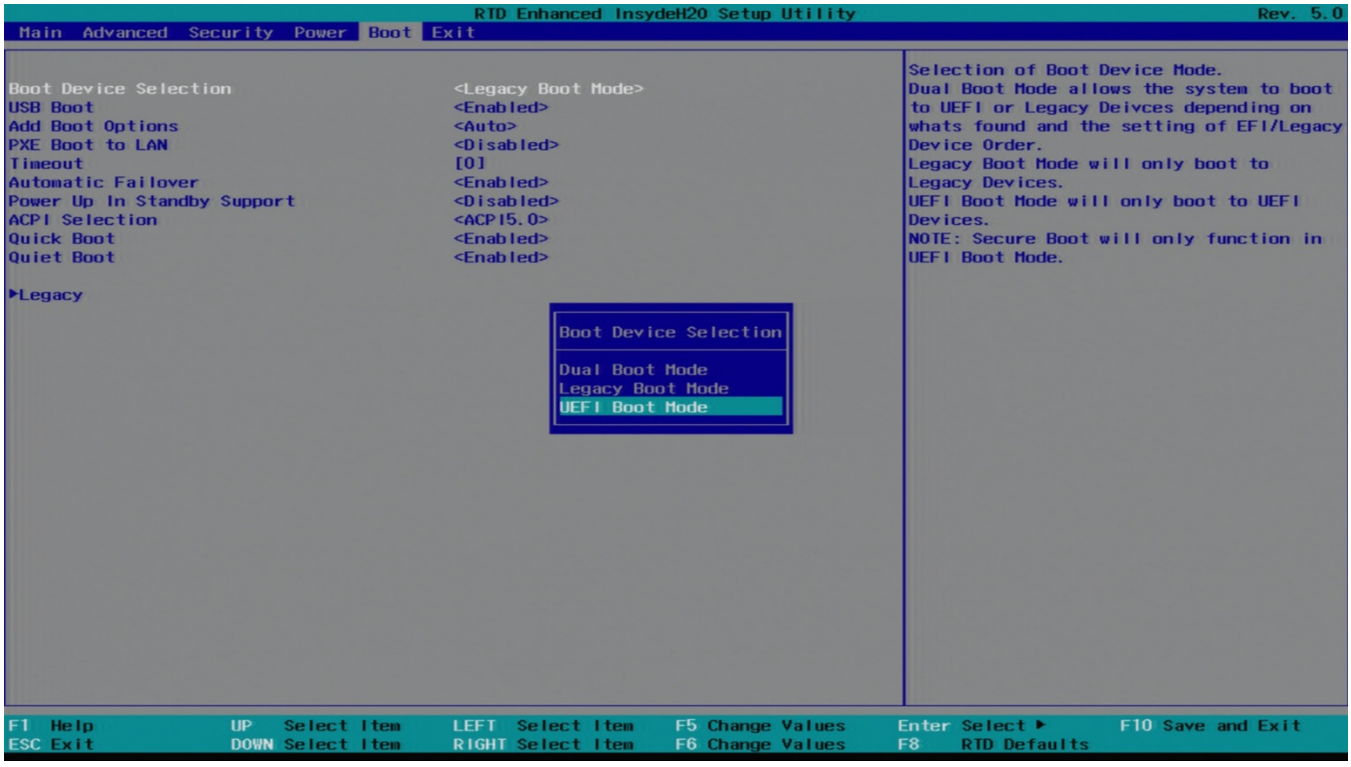


2.7 Set up an Administration Password, Optional but Recommended



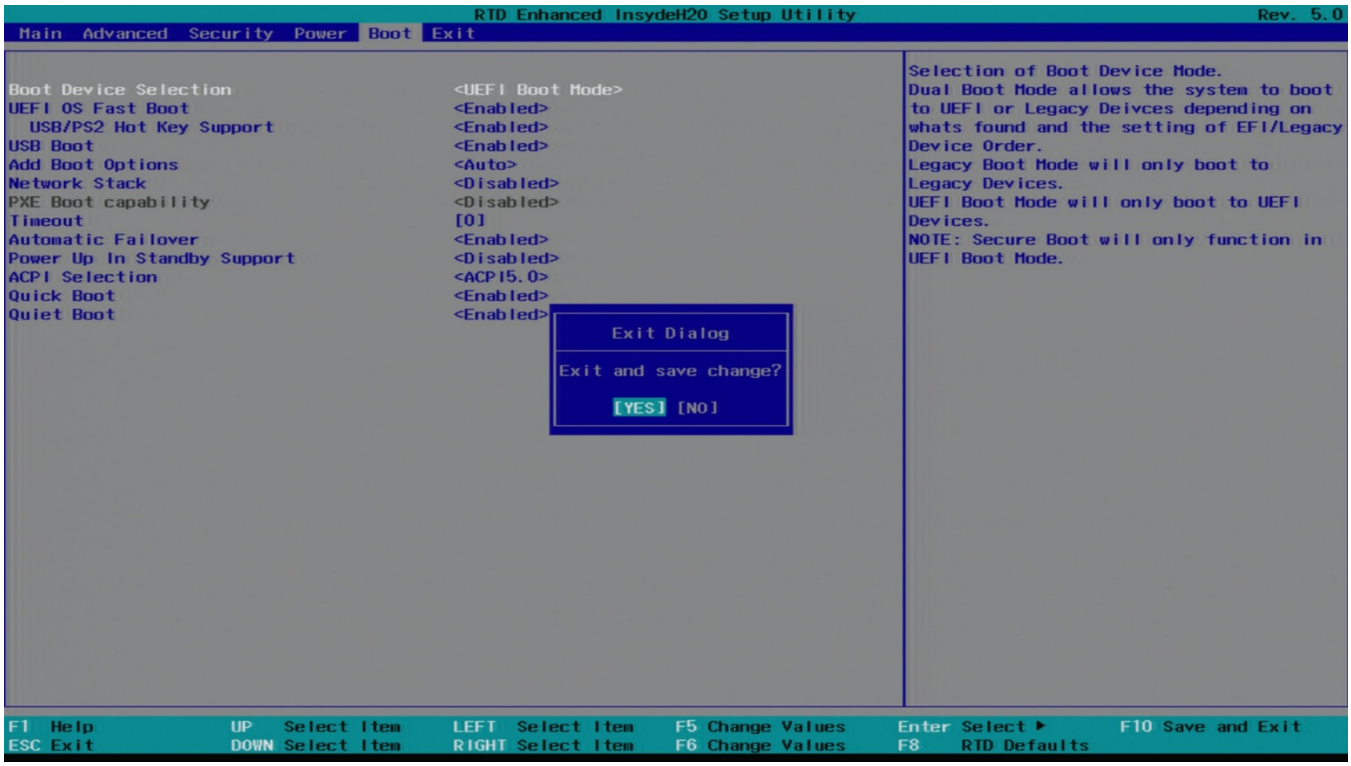
2.8 Go to the Boot Tab, Optional for Linux Operating Systems

Changing the Boot Device Selection to UEFI mode is only required for Windows Bitlocker. Linux Operating systems can use TPM based encryption libraries in either the UEFI or Legacy boot modes. Despite being optional, UEFI is faster and more secure than Legacy and therefore highly recommended. If you have already installed a Windows Operating system in Legacy or Dual Boot mode, you will need to reinstall Windows in UEFI mode to gain access to Bitlocker functions.



2.9 Press F10 and Select Save

Pressing F10 on your keyboard will bring up a menu that allows you to save any changes made in the BIOS before exiting.



3 Windows Implementation

3.1 Compatible Versions

Not all windows versions come with nor support Bitlocker. Those that do, support both TPM 1.2 and TPM 2.0 chips. Versions of Windows prior to 7 do not have TPM / Bitlocker support.

Table 2: Compatible Versions of Windows

Versions	Editions
Windows 7	Enterprise or Ultimate
Windows 8	Professional or Enterprise
Windows 10	Professional, Enterprise, or Education

3.2 Software Implementation

Recommendations before proceeding to Bitlocker setup:

- Encrypting a partition with Bitlocker can take a very long time depending on the size and used space of the drive. It is recommended that you encrypt your partitions before installing any other software to minimize the initial encryption time.
- Have an external drive ready to store the encryption key. A USB thumb drive is ideal. It is highly recommended to keep a copy of the encryption key on an external drive. This key will allow you to recover your drive's data in the event your computer has a hardware failure or the TPM gets cleared in the BIOS. This external drive should be stored somewhere safe to ensure your data remains secure. Never save this key on a local drive, especially an unencrypted partition. If attackers find this file locally then they will be able to easily access your encrypted files.

The following instructions were written for Windows 8 and 10 running on RTD Chief River (i7) or Bay Trail (Atom) CPU modules. Installation on other hardware platforms may be different.

Step 1: Enable and clear the TPM in the bios, Set UEFI Boot Mode

Pressing <Esc> during boot will take you into the BIOS. On RTD CPU modules the TPM settings are in SCU under the security tab. Here you can enable/disable the TPM, clear the TPM, and set a supervisor password. Setting a supervisor password will prevent users without the password from changing TPM settings. A supervisor password is recommended on multiuser machines to ensure other users do not clear the TPM. Under the Boot tab you must change the Boot Mode to UEFI Boot Mode, Windows will not allow Bitlocker encryption on Legacy installations of Windows. After completing all the above steps, press F10 and select save to exit the BIOS and update the settings. More in-depth instructions with pictures of the BIOS setup procedure is available in Section 2.

Step 2: Turn on Bitlocker
Open file explorer and go to This PC, found in the left navigation window. Right click on the drive you wish to encrypt and select turn on Bitlocker. Running Bitlocker will require administrator privileges. Bitlocker will give up to 3 unlocking options for your drive.

- Password: This method will require you to enter a password to access the encrypted partition. If the boot partition is encrypted, then the password will have to be entered at startup.
- Smart card: This method will require a smart card. Smart cards are USB-based devices that function similarly to a TPM chip. These come in many brands and form factors.
- Automatic: This method will use the TPM chip on your motherboard. If this option is not visible, then make sure Step 1 (above) was completed. If this option is still not visible, you may need to encrypt your boot partition with Bitlocker first and then restart your system.

Bitlocker will give you 3 options to back up this key; the Microsoft cloud, print the key, or save a file containing the recovery key to a USB drive. After saving your recovery key you can begin encrypting partitions. Selecting "Encrypt entire drive" will ensure that any new files you create will also be encrypted. If you do not encrypt the entire partition, then new files will not be encrypted. A boot partition will start encryption after a restart while all other partitions will start encryption in the background immediately. Encryption can be very time consuming for large partitions.

3.3 Migration between TPM 1.2 and 2.0 with Windows

Windows and Bitlocker can recognize and support both versions of TPM chips (1.2 and 2.0). As such it is possible to still access your encrypted files should you upgrade or downgrade your system's TPM version. To do this you will need the recovery key to unlock the encrypted partition. Once the partition is unlocked you can enable Bitlocker and encrypt the partition again. This will generate a new key for the partition, and you will no longer need the old key at this point. Windows installations are built specifically for the processor architecture of the system. Moving a Windows installation to a different system must take this into account by having a compatible processor or doing a full operating system re-install and Bitlocker encryption on the boot partition.

4 Linux Implementation

4.1 Linux Complications

Unlike Windows, Linux (As of June 2020) does not have a simple way of encrypting your entire drive. Each stage of the decryption process takes several libraries, meaning that these parts of your file system must not be entirely encrypted. As a result, it is best to create another partition and keep all your sensitive information on it, while leaving your operating system on an un-encrypted partition. Symbolic and/or hard links can help facilitate boot partition access to the secure partition after it has been mounted at start-up. There are also differences between the TPM 1.2 and TPM 2.0 libraries, making cross compatible encrypted partitions more difficult to set up compared to Windows. The cryptfs library used to encrypt partitions on TPM 2.0 systems uses 32-byte alphanumeric keys while TPM 1.2 chips only have 20-byte registers. There are 2 methods available to make the encrypted partition compatible between TPM 1.2 and TPM 2.0:

1. Use a pair of TPM 1.2 PCR registers to hold the 32-byte key which keeps the entire key secure at the cost of using 2 PCR registers in the TPM.
2. Store only 20 bytes of the key in the TPM and keep the other 12 bytes in the automatic mounting script, saving a PCR register. This may be useful if you are using other applications that save keys in the TPM and space is at a premium. Both methods will be shown in the TPM 1.2 set-up instructions to follow.

Both the following TPM 1.2 and TPM 2.0 set-up instructions are meant to be entered into the terminal of a Debian-based Linux distribution and have been tested in Ubuntu 18.04 kernel versions 4.13, 4.15, 5.0 and Ubuntu 20.04 kernel versions 5.3, and 5.4. Terminal commands are likely to be similar for other non-Debian Linux distributions, but the commands may need modified to work with your operating system. Possible differences may include changes to file locations, library package names, package manager software, text editor software, etc.

4.2 TPM 1.2 Installation Instructions

TPM 1.2 currently has native support in most Linux distributions and requires only a few libraries; trousers, tpm-tools, and cryptsetup.

Step 1: Enable and clear the TPM in the bios, Set UEFI Mode

Pressing <Esc> during boot will take you into the BIOS. On RTD CPU modules the TPM settings are in the SCU under the security tab. Here you can enable/disable the TPM, clear the TPM, and set a BIOS supervisor password. Setting a supervisor password will prevent users without the password from changing TPM settings. A supervisor password is recommended to ensure other users do not clear the TPM. It is recommended that you use the UEFI boot mode which is found under the Boot tab. The UEFI boot mode adds support for more than 4 partitions, partitions larger than 2TB, faster boot times, and more robust fault management. After completing all the above steps, press F10 and select save to exit the BIOS and update the settings. More in-depth instructions with pictures of the BIOS setup procedure is available in **Section 2**.

Step 2: Installing necessary libraries

```
sudo apt update
# Refresh package information
sudo apt -y install trousers
# trousers is a software stack manager for the TPM chip.
sudo apt -y install tpm-tools
# tpm-tools provides a series of terminal commands for the user to access
# the TPM chip.
sudo apt -y install cryptsetup
# cryptsetup is a tool that facilitates drive encryption using the
# Linux Unified Key Setup.
sudo apt -y install pwgen
#pwgen will assist the user in creating random keys.
```

Step 3: Make sure the trousers daemon is running after installation

```
sudo /etc/init.d/trousers start
# Make sure the trousers daemon is running since it was just
# installed.
systemctl status -i --nopager trousers.service
# Confirm the daemon is active and running, if not a restart might be
# needed.
```

Step 4: Take ownership of the TPM chip with your user account

```
tpm_takeownership -z -y
# This tpm-tools command will make your account the owner of the TPM
# chip. Only your user account and root should be able to run tpm-
# tools now.
```

Step 5: Figure out the NVRAM addresses mapped to your TPM chip

On the RTD Bay Trail and Chief River CPU modules, the TPM is mapped to the hex address 0x10000001. If you are using different computer hardware your TPM will likely be mapped elsewhere. To figure out the PCR (Platform Configuration Register) mapping on your device you can run the '***tpm_nvinfo -n***' command to list all the mapped NVRAM sections found by the tpm-tools library. The first PCR will already contain the ownership privileges from the 'tpm_takeownership -z -y' terminal command from **Section 4.2 Step 4** above. TPM 1.2 PCR registers are 20 bytes long (0x14 in hex) so the next available PCR register would be at offset 0x10000015. If you have already mapped a key to offset 0x10000015 during a prior install this section may be listed even after clearing the TPM chip and it will be unnecessary to perform **Section 4.2 Step 6**. If 0x10000015 has already been allocated and you have not previously allocated it for TPM key storage, then another piece of software may be using it. Choose the next free PCR slot from the table below and enter it in **Section 4.2 Step 6** below.

TPM 1.2 PCR Slot Addresses:

Slot 2: 0x10000015	Slot 3: 0x10000029	Slot 4: 0x1000003D	Slot 5: 0x10000051	Slot 6: 0x10000065
Slot 7: 0x10000079	Slot 8: 0x1000008D	Slot 9: 0x100000A1	Slot 10: 0x100000B5	Slot 11: 0x100000C9

To calculate a slot address you can calculate (20 * PCRslot) + 268435437 and then convert it to hex here.

Step 6: Allocate the NVRAM section at 0x10000015

If you are planning to store only the latter 20 bytes of the key replace <key size> with 20.
If you are planning to store the entire 32-byte key in the TPM, replace <key size> with 32.
Replace the '0x10000015' of the key slot address variable if another PCR slot was chosen from the table above.

```
keysize='<key size>'
keyslotaddress='0x10000015'
tpm_nvdefine -i $keyslotaddress -s $keysize -p 'AUTHREAD|AUTHWRITE' -y -z
# -s defines the size of the key we are going to use. Normally 20-
# byte keys are used by tpm 1.2 however using a 32-byte key will
# allow the drive to be compatible with TPM 2.0 chips as well. If you
# have no intention of ever using a TPM 2.0 device you may use a 20-
# byte key. If tpm_nvinfo -n already showed a mapped key at
# 0x10000015 you do not need to run this command. If there is a
# conflicting key mapped in this space tpm_nvdefine will return an
# error.
```

Keep in mind that variables like keysize and keyslotaddress will be lost if you exit the terminal or restart your computer. These lines will need entered again as they are used in later commands of the set-up instructions.

Step 7: Pick and encrypt your partition

Replace <selected partition> with the drive device file path found with lsblk. Usually /dev/sd<character><number>.

```
lsblk | grep -e part
# This give a more user-friendly output of all the available
# partitions. Usually /dev/sd<character><number> is the location of
# the drive you intend to encrypt.
selectedpartition='<selected partition>'
sudo umount $selectedpartition
# Replace <selected partition> with /dev/sd<character><number> found
# by the previous lsblk command. Make sure the partition is not
# mounted so formatting and/or encryption can take place.
cd /dev/shm
mkdir key
cd key
# /dev/shm is mapped only to NVRAM so creating a key here will not
# leave a trace unlike on magnetic or solid-state hard drives.
```

Execute only 1 of the following code segments below based on your chosen key length and if you have a recovery key from a previous installation:

Table 3: Key Set-up Options

Options	Creating a New Key	Use a Recovery Key
Storing only the latter 20 bytes of the key in the TPM	Code Segment 1	Code Segment 2
Storing all 32 bytes of the key in the TPM	Code Segment 3	Code Segment 4

Code Segment 1: Generate a new 20-byte key.

```
pwgen -s 20 1 >> temp.key
# Fill the key with 20 random alphanumeric characters.
knownkeyheader='000000000000'
# Pick your own 12 alphanumeric ASCII characters for the known header.
```

Code Segment 2: Replace <path to backup key> with the location of the key file for the encrypted partition.

```
backupkeypath='<path to backup key>'
cat $backupkeypath | tr -d ' \n\t\r' | cut -c13-32 > temp.key
# Copy a backup key and remove all spaces, new lines, and tabs.
# Parse out the 13th through 32nd character, 20 in total.
knownkeyheader=`cat $backupkeypath | tr -d ' \n\t\r' | cut -c1-12`
# Cut out the first 12 characters and store in the key header variable.
# Using back ticks will place the output string into a variable.
```

Code Segment 3: Generate a new 32-byte key compatible with TPM 2.0 systems.

```
pwgen -s 32 1 > temp.key
# Fill the key with 32 random characters.
```

Code Segment 4: Replace <path to backup key> with the location of the key file for the encrypted partition.

```
cat <path to backup key> | tr -d ' \n\t\r' > temp.key
# Copy a backup key and remove all spaces, new lines, and tabs.
```

The following commands will store the key in the TPM. Always execute these despite your key set-up option chosen above. Replace <key slot address> with the address found with ***'tpm_nvinfo -n'***. 0x10000015 on RTD CPU modules if you have not used any TPM PCR registers for other applications. Replace <key size> with 20 or 32, the number of bytes you want to store in the TPM.

```
tpm_nvwrite -i $keyslotaddress -s $keysize -f temp.key -z
# This will save the key in the TPM chip.
rm -f temp.key
# Delete the key.
tpm_nvread -i $keyslotaddress -z -f temp.key
# Read the key back from the tpm chip and make sure it exists and is
# the same.
```

If you are storing only the latter 20 bytes of the key in the TPM then you must also run the following:

```
echo -n "$knownkeyheader" > tempheader.key
# Create a separate key file with the first 12 bytes of the key
cat tempheader.key temp.key | tr -d ' \n\r\t' > tempwhole.key
# Concatenate the header with the 20 bytes from the TPM chip.
# Also removes and new lines, tabs, or spaces.
# Unfortunately, you cannot output directly to temp.key so we must output
# to another file and copy back to temp.key
cat tempwhole.key > temp.key
rm -f tempheader.key
rm -f tempwhole.key
# Delete the header key files when done.
```

The following command will format and encrypt the partition. Skip this command if the partition already contains data you do not wish to lose. This command does warn and prompt you if you try to run it.

```
sudo cryptsetup -v luksFormat $selectedpartition secure --key-file
temp.key
# The two above lines are one command.
# This will name and encrypt the selected partition, secure is the
# name but you can change it. Do not run this command on already
# encrypted drives as it will delete all data on the partition.
```

Always run this command to decrypt the partition. This will create a new device file for the decrypted partition. In Ubuntu 18.04 this device file can be found at /dev/dm-<number>. If you have only 1 encrypted partition, then <number> will be 0.

```
sudo cryptsetup -v luksOpen $selectedpartition secure --key-file temp.key
# This will decrypt the drive and allow it to be mounted and/or
# formatted later. This will also make a new device file for the
# decrypted partition <secure device name>. On ubuntu this new device
# file was always /dev/dm-0 but if you have multiple encrypted
# partitions this is likely to be different.
```

The following command will format and encrypt the partition. Skip this command if the partition already contains data you do not wish to lose. Replace <secure device name> below with the path to the decrypted partition, usually /dev/dm-<number>.

```
securedevicename='<secure device name>'
sudo mkfs.ext4 $securedevicename
# Formats the partition to an ext4 file system.
# This command is optional if the partition you are encrypting
# already has a file system.
```

Step 8: Back up your key and then delete it

Replace <USB drive path> with the path to where you want to save your backup key file. Do NOT save this key on a local un-encrypted drive or partition!!!!

```
sudo nano temp.key
# This command will allow you to view the key if you intend to write
# it down for safe keeping. Rather than store it in a text file.
USBpath='<USB drive path>'
tpm_nvread -i $keyslotaddress -z -f $USBpath
# This command will allow you to save your key from the TPM to an
# external device.
rm -f temp.key
cd ..
rm -rf key
# Delete key and folder made in NVRAM.
```

Step 9: Create the script that mounts the encrypted partition automatically

```
# Change /securemount below if you would like to change the script
# installation directory. This directory MUST be on your boot partition as
# it must be unencrypted and available to mount TPM encrypted partitions.
sudo mkdir /securemount
cd /securemount
# Create and move into a new folder for the scripts.
desiredmountpoint='/secure'
# You may change the mount point for the encrypted partition.
```

Run the following code if you stored all 32 bytes of your key in the TPM:

```
echo '#!/bin/sh' > mount.sh
echo 'cd /dev/shm' >> mount.sh
echo 'mkdir secmount' >> mount.sh
echo 'cd secmount' >> mount.sh
echo "tpm_nvread -i $keyslotaddress -z -f temp.key" >> mount.sh
echo "cryptsetup -v luksOpen $selectedpartition secure --key-file
temp.key" >> mount.sh
# The two above lines are one command.
echo "mount $securedevicename $desiredmountpoint" >> mount.sh
echo 'rm temp.key' >> mount.sh
echo 'cd ..' >> mount.sh
echo 'rm -rf secmount' >> mount.sh
```

This script moves to NVRAM, reads the key from the TPM chip in NVRAM, unlocks the encrypted partition with the key, mounts the decrypted partition, and then deletes the key from NVRAM. The echo commands will build the script using saved variable values.

Run the following code if you stored only 20 bytes of your key in the TPM:

```
echo '#!/bin/sh' > mount.sh
echo 'cd /dev/shm' >> mount.sh
echo 'mkdir secmount' >> mount.sh
echo 'cd secmount' >> mount.sh
echo "tpm_nvread -i $keyslotaddress -z -f temppcr.key" >> mount.sh
echo "echo -n '$knownkeyheader' > tempheader.key" >> mount.sh
echo "cat tempheader.key temppcr.key | tr -d ' \n\r\t' > temp.key" >>
mount.sh
# The two above lines are one command.
echo 'rm tempheader.key' >> mount.sh
echo 'rm temppcr.key' >> mount.sh
echo "cryptsetup -v luksOpen $selectedpartition secure --key-file
temp.key" >> mount.sh
# The two above lines are one command.
echo "mount $securedevicename $desiredmountpoint" >> mount.sh
echo 'rm temp.key' >> mount.sh
echo 'cd ..' >> mount.sh
echo 'rm -rf secmount' >> mount.sh
```

This script moves to NVRAM, reads the key from the TPM chip in NVRAM, appends the key to the known key header, unlocks the encrypted partition with the key, mounts the decrypted partition, and then deletes the key from NVRAM.

Step 10: Create the script that locks and mounts the encrypted partition automatically

This script unmounts and locks the encrypted partition.

```
echo '#!/bin/sh' > unmount.sh
echo "umount $desiredmountpoint" >> unmount.sh
echo 'cryptsetup luksClose secure' >> unmount.sh
```

Step 11: Creating the service file for init.d

```
echo '[Unit]' > secmount.service
echo 'Description=Mounts the encrypted partition' >> secmount.service
echo 'After=trousers.service' >> secmount.service
echo >> secmount.service
echo '[Install]' >> secmount.service
echo 'WantedBy=multi-user.target' >> secmount.service
echo >> secmount.service
echo '[Service]' >> secmount.service
echo 'Type=forking' >> secmount.service
echo 'KillMode=process' >> secmount.service
echo "ExecStart=/securemount/mount.sh" >> secmount.service
echo "ExecStop=/securemount/unmount.sh" >> secmount.service
echo 'Type=oneshot' >> secmount.service
echo 'RemainAfterExit=yes' >> secmount.service
echo 'User=root' >> secmount.service
```

Step 12: Installing the service file

```
sudo chmod 770 /securemount/mount.sh
# Make sure the mount script is marked as executable.
sudo chmod 770 /securemount/unmount.sh
# Make sure the unmount script is marked as executable.
sudo systemctl enable /securemount/secmount.service
# Use systemctl to add the service to init.d
sudo service secmount start
# Start the service
systemctl status -l --no-pager secmount
# Check the service ran once with no errors. It should say Active (exited)
if it ran once.
```

Now restart your machine and check if the encrypted partition is mounted automatically.

4.3 TPM 2.0 Instructions

TPM 2.0 currently does not have native support in most Linux distributions. Fortunately, a series of libraries from GitHub and IBM are available to facilitate TPM 2.0. These libraries also depend on a significant number of native Linux libraries.

Step 1: Enable and clear the TPM in the bios, Set UEFI Boot Mode

Pressing <Esc> during boot will take you into the BIOS. On RTD CPU modules the TPM settings are in the SCU under the security tab. Here you can enable/disable the TPM, clear the TPM, and set a BIOS supervisor password. Setting a supervisor password will prevent users without the password from changing TPM settings. A supervisor password is recommended to ensure other users do not clear the TPM. It is recommended that you use the UEFI boot mode which is found under the Boot tab. The UEFI boot mode adds support for more than 4 partitions, partitions larger than 2TB, faster boot times, and more robust fault management. After completing all the above steps, press F10 and select save to exit the BIOS and update the settings. More in-depth instructions with pictures of the BIOS setup procedure is available in **Section 2**.

Step 2: Installing necessary libraries

```
sudo apt update
# Make sure the repository information is up to date.
sudo apt -y install git
# Required to fetch and clone the Github libraries
sudo apt -y install autoconf
sudo apt -y install autoconf-archive
sudo apt -y install automake
sudo apt -y install pkg-config
sudo apt -y install build-essential
sudo apt -y install libtool
sudo apt -y install gcc
sudo apt -y install g++
sudo apt -y install doxygen
sudo apt -y install libgcrypt20-dev
sudo apt -y install libgcrypt20
sudo apt -y install libssl-dev
sudo apt -y install libjson-c-dev
sudo apt -y install libcurl4-gnutls-dev
# libsapi0 is no longer available in Ubuntu 20.04
# This will cause some TPM-tools functionality to be lost,
# however the automatic drive decryption and mounting still works
sudo apt -y install libsapi0
sudo apt -y install libsapi-dev
sudo apt -y install libsapi-utils
sudo apt -y install gnu-efi
# libglib2.0 has been renamed in Ubuntu 20.04
# Please use the following command for Ubuntu 20.04
sudo apt -y install glib2.0
# Please use the following command for Ubuntu 18.04 and 16.04
sudo apt -y install libglib2.0-dev
sudo apt -y install libglib2.0
sudo apt -y install pandoc
sudo apt -y install python-yaml
sudo apt -y install python
sudo apt -y install python-dev
sudo apt -y install xxd
sudo apt -y install cmake
sudo apt -y install libcmocka0
sudo apt -y install libcmocka-dev
sudo apt -y install uthash-dev
sudo apt -y install procps
sudo apt -y install iproute2
sudo apt -y install libini-config-dev
sudo apt -y install m4
sudo apt -y install libdbus-glib-1-dev
sudo apt -y install expect
sudo apt -y install uuid
sudo apt -y install uuid-dev
# These are the dependencies of the TPM 2.0 libraries.
sudo apt -y install wget
# Required to fetch the IBM TPM simulator.
sudo apt -y install cryptsetup
# cryptsetup is a tool that facilitates drive encryption using the
# Linux Unified Key Setup.
sudo apt -y autoremove
# Clean up after libraries are installed
```

Step 3: Create a directory to build all the GitHub libraries

You will be able to add other users to a group later that allows them to use the TPM. You may modify this path to suit your operating system and needs. Keep in mind that variables like useraccount and installdir will be lost if you exit the terminal or restart your computer. These lines will need entered again as they are used in later commands of the set-up instructions.

```
useraccount=`whoami`
installdir="/home/$useraccount/Documents/software/tpm2"
mkdir -p $installdir
cd $installdir
# You may change this directory path.
```

Step 4: Get and compile the TPM 2.0 software stack library

```
git clone https://github.com/tpm2-software/tpm2-tss.git
# Download the most recent stable build tpm2-tss library from Github.
cd tpm2-tss
./bootstrap
# Bootstrap script sets up config files based on your kernel version and
# Linux distribution.
udevrulesdir="/"
./configure --with-udevrulesdir="$udevrulesdir" --enable-doxygen-doc --
with-udevrulesprefix=70- --enable-unit  --prefix=/usr
# This is all one line. Configure script populates config files and checks
# that all library dependencies have been met.
make doxygen-doc
# tpm2-tss uses Doxygen for its manual and documentation pages.
make -j$(nproc) check
# Compile the tpm2-tss library and uses some scripts to check
functionality.
sudo make install
# Installs the library. Makes some links to C header and object files.
ls $udevrulesdir | grep tpm
# If this command returns no file names then the rules file was not
created automatically. This occurs because # rules.d folder is sometimes
owned by root but ./configure must run as a normal user.
```

If the rules.d folder does not contain a file with TPM in its name, then the previous ls command will return no text. If this is the case then you will need to run the following code to create the correct rules.d privilege file. This file will determine the access permissions of the TPM device files found in /dev.

```
cd $udevrulesdir
# All one line
sudo echo '============================================================='
> 70-tpm-udev.rules
# All one line
sudo echo 'KERNEL=="tpm[0-9]", MODE="0660", OWNER="tss"' >> 70-tpm-
udev.rules
# All one line
sudo echo 'KERNEL=="tpmrm[0-9]", MODE="0660", OWNER="tss", GROUP="tss"' >>
70-tpm-udev.rules
# All one line
sudo echo '============================================================='
>> 70-tpm-udev.rules
```

The following commands will refresh and reload rules.d to include the new permissions for the TPM. You should be run even if you did not have to create the file yourself using the commands above.

```
sudo udevadm control --reload-rules && sudo udevadm trigger
sudo ldconfig
# tpm2-tss added some udev/rules.d files while installing tpm2-tss. These
# commands will refresh udev/rules.d to include these new rules files.
cd ..
# Move back to head build directory
```

Step 5: Create a user account for the tpm2-tss and set up permissions

```
sudo useradd --system --user-group tss
# Create a user account and group that will control who has access to the
# TPM chip.
sudo passwd tss
# Set up a password for the tss account
sudo chown tss:tss /dev/tpm0
sudo chown tss:tss /dev/tpmrm0
# Change the owner of the TPM device files to the tss user account
sudo chmod 660 /dev/tpm0
sudo chmod 660 /dev/tpmrm0
# Change read/write access of the TPM device files
ls -la | grep /dev/tpm
# Check that owner and group has read/write and other does not
```

The user account ***tss*** will be the owner of the TPM chip. Users in the ***tss*** group will be able to use TPM related commands. The '***ls -la***' command will allow you to check the owner has changed to ***tss*** and the permissions are '***crw-rw----***'.

Step 6: Get and compile the TCTI library UEFI patch

```
git clone https://github.com/tpm2-software/tpm2-tcti-uefi.git
# Download the most recent stable build tpm2-tcti library from Github.
cd tpm2-tcti-uefi
./bootstrap
# Bootstrap script sets up config files based on your kernel version and
# Linux distribution.
./configure
# Configure script populates config files and checks all library
# dependencies have been met.
make
# Compile the tpm2-tcti-uefi library
sudo make install
# Installs the library. Makes some links to C header and object files.
```

It is not necessary to acquire and build the TCTI library if you are not using the UEFI boot mode on your system.

Step 7: Get and compile the TPM 2.0 daemon library

```
git clone https://github.com/tpm2-software/tpm2-abrmd.git
# Download the most recent stable build tpm2-abrmd library from Github.
cd tpm2-abrmd
sudo ./bootstrap
# Bootstrap script sets up config files based on your kernel version and
# Linux distribution. Run as root because /etc is usually owned by root.
sudo ./configure --with-dbuspolicydir=/etc/dbus-1/system.d --with-
systemdsystemunitdir=/lib/systemd/system --enable-unit  --prefix=/usr
# This is all one line. Configure script populates config files and checks
# that all library dependencies have been met.
sudo make check
# Compile the tpm2-abrmd library and uses some scripts to check
functionality.
sudo make install
# Installs the library. Makes some links to C header and object files.
sudo udevadm control --reload-rules && sudo udevadm trigger
sudo ldconfig
# tpm2-abrmd added some udev/rules.d files while installing tpm2-tss.
# These commands will refresh udev/rules.d to include these new rules
# files.
cd ..
sudo pkill -HUP dbus-daemon
sudo systemctl daemon-reload
sudo ldconfig
# Kill the dbus daemon and reload it since tpm2-abrmd made some changes to
it.
sudo service tpm2-abrmd start
sudo systemctl enable tpm2-abrmd
systemctl status -l --no-pager tpm2-abrmd.service
# Start the tpm2 daemon and check that the daemon is running without
errors.
```

Step 8: Get and install the TPM 2.0 simulator by IBM

The IBM TPM simulator will allow you to test the functionality of your TPM libraries during the install process.

```
mkdir ibmtpm2
cd ibmtpm2
wget https://downloads.sourceforge.net/project/ibmswtpm2/ibmtpm1563.tar.gz
# Create a directory to extract and compile the IBM library in.
tar -zxvf ibmtpm1563.tar.gz
# Extract the library from the .tar ball.
rm ibmtpm1563.tar.gz
# You may remove the .tar ball from your system after extracting it.
cd src
make -j$(nproc)
# Compile the IBM TPM simulator library
sudo make install
# Install the IBM TPM simulator. Makes some links to C files.
cd ..
cd ..
```

Step 9: Get and compile the TPM 2.0 tools library

```
git clone https://github.com/tpm2-software/tpm2-tools.git
# Download the most recent stable build tpm2-tools library from Github.
cd tpm2-tools
./bootstrap
# Bootstrap script sets up config files based on your kernel version and
# Linux distribution.
./configure --enable-unit=abrmd
# Configure script populates config files and checks that all library
# dependencies have been met. The enable unit option adds additional
# scripts for functionality testing.
make -j$(nproc) check
# Compile the tpm2-tools library and uses some scripts to check
# functionality. There are currently 6 test scripts that fail in Ubuntu
# 18.04 kernel 5.3 but they do not cause issues with the functions used in
# this paper. More than 6 test failures could indicate your installation
# failed. In Ubuntu 20.04 kernel 5.4 libsapi is no longer available so a
# large number of test scripts now fail. TPM libraries still function as
# intended however.
sudo make install
# Installs the library. Makes some links to C header and object files.
cd ..
```

Step 10: Get and install the TPM 2.0 encryption/decryption library

```
git clone https://github.com/jiazhang0/cryptfs-tpm2
# Download the most recent stable build cryptfs-tpm2 library from Github.
cd cryptfs-tpm2
make
# Compile the cryptfs-tpm2 library.
sudo make install
# Install the cryptfs-tpm2 simulator. Makes some links to C header and
object files.
sudo ldconfig
cd ..
```

Step 11: Clear TPM chip and generate a new key

(-p <path to key file>) denotes an optional argument that will pull a key from a file. Replace **<path to key file>** with the path to the key file. Generally, this option should be used when setting up automatic mounting of an already encrypted partition. The key file must contain exactly 32 alphanumeric characters and must not contain any special characters, white space characters, carriage returns, or new lines. When the optional argument is not included when running the cryptfs-tpm2 command then the cryptfs library will use the random number generator in the TPM 2.0 chip to generate a new 32-byte key. The key is then stored in the TPM chip for safe keeping.

```
cd tpm2-tools/tools
# Move into this directory to use tpm2-tools commands.
sudo -u tss tpm2_clear -v
# This tpm2-tools command will clear the TPM and grant ownership.
# This command must be run as the tss user so it is the owner.
# Otherwise the TPM might be owned by your user account or root and the
# software stack library will not function as intended.
sudo cryptfs-tpm2 -v seal all -P auto (-p <path to key file>)
# This command may exit with error code 11 because the TPM is owned by tss
# but the cryptfs-tpm2 must be run as root. Despite the error it should
# state that it succeeded to unseal the key.
mkdir -p /dev/shm/keysetup
cd /dev/shm/keysetup
# Move to NVRAM so the key file leaves no traces on disk.
sudo cryptfs-tpm2 -v unseal passphrase -P auto -o passphrase.key
# Pulls the key out of the TPM into a file in NVRAM
```

Step 12: Choose the partition to encrypt

Replace <character><number> to match the partition device file of the partition you a plan to encrypt.

```
lsblk | grep -e part
# Displays a list of partitions. Usually /dev/sd<character><number>
selectedpartition='/dev/sd<character><number>'
# Replace <character><number> with those that match your partition.
# Do NOT try to unmount your boot partition!
sudo umount $selectedpartition
# Make sure the partition is not mounted already.
securename='secure'
# You may change the name from secure.
sudo cryptsetup -y -v luksFormat $selectedpartition $securename --key-file
passphrase.key
# The above two lines are one command.
# This command will encrypt the chosen partition using the key in NVRAM
# and name it. Do not run this command if the partition is already
# encrypted.
sudo cryptsetup -v luksOpen $selectedpartition $securename --key-file
passphrase.key
# The above two lines are one command.
# This command will unlock the partition with the key in NVRAM.
securedevice='/dev/dm-0'
# <secure device> must be replaced with /dev/dm-0. This
# device name may change depending on your Linux distribution and how many
# encrypted drives you have mounted.
sudo mkfs.ext4 $securedevice
# This command will format the partition to an ext4 file system.
# Do not run this command if the partition already has a file system.
sudo rm passphrase.key
cd ..
sudo rm -rf keysetup
```

Step 13: Step up the mount point of the encrypted partition

```
desiredmountpoint='/secure'
# You may change the above path stored in desiredmountpoint to select a
# different mount point.
sudo mkdir -p $desiredmountpoint
# This command will create the directories needed for the mount point.
sudo mount $securedevice $desiredmountpoint
# This command will mount the secure device to the desired mount point.
```

Step 14: Set up a new group to control access to the encrypted partition

```
sudo groupadd secure
# Create a new group named secure.
sudo usermod -a -G secure $useraccount
# Adds the user <account> to the secure group.
sudo chown -R "$useraccount:secure" $desiredmountpoint
# Changes ownership of the encrypted partition to account and members of
# the secure group.
sudo chmod -R g+rwx $desiredmountpoint
sudo chmod -R o-rwx $desiredmountpoint
# Changes read, write, execute privileges for the encrypted partition.
```

Step 15: Make the automatic mounting script

```
cd $installdir
mkdir tpm2-luks
cd tpm2-luks
# Move back to build directory and create new folder for automation
# scripts.
echo '#!/bin/sh' > mount.sh
echo 'cd /dev/shm' >> mount.sh
echo 'mkdir secmount' >> mount.sh
echo 'cd secmount' >> mount.sh
echo 'cryptfs-tpm2 unseal passphrase -P auto -o passphrase.key' >>
mount.sh
# The above command is all one line.
echo "cryptsetup luksOpen $selectedpartition $securename --key-file
passphrase.key" >> mount.sh
# The above command is all one line.
echo "mount $securedevice $desiredmountpoint" >> mount.sh
echo 'rm passphrase' >> mount.sh
echo 'cd ..' >> mount.sh
echo 'rm -rf secmount' >> mount.sh
# The echo commands will build the script using the variables created.
```

The automatic mounting script above will move into NVRAM and pull the key from the TPM chip. The script will then use the key to unlock and mount the encrypted partition.

Step 16: Make the automatic unmounting script

```
echo '#!/bin/sh' > unmount.sh
echo "umount $desiredmountpoint" >> unmount.sh
echo "cryptsetup luksClose $securename" >> unmount.sh
```

The automatic unmounting script above will unmount the secure device and then closes the encrypted partition.

Step 17: Make the init.d service file

```
echo '[Unit]' > secmount.service
echo 'Description=Mounts the encrypted partition' >> secmount.service
echo 'After=tpm2-abrmd.service' >> secmount.service
# Makes sure tpm daemon (abrmd) is loaded before trying to use it
echo >> secmount.service
echo '[Install]' >> secmount.service
echo 'WantedBy=multi-user.target' >> secmount.service
echo >> secmount.service
echo '[Service]' >> secmount.service
echo 'Type=forking' >> secmount.service
echo 'KillMode=process' >> secmount.service
echo 'ExecStart=/securemount/mount.sh' >> secmount.service
# Run mount script at start up
echo 'ExecStop=/securemount/unmount.sh' >> secmount.service
# Run unmount script at shutdown
echo 'Type=oneshot' >> secmount.service
# Run mount/unmount scripts only once
echo 'RemainAfterExit=yes' >> secmount.service
echo 'User=root' >> secmount.service
# User account to run scripts in
```

Step 18: Installing the automation scripts

```
sudo mkdir -p /securemount
# Create the directory for the automation scripts.
# This directory must be owned by root.
sudo cp *.* /securemount
# Copy this automation scripts into /securemount.
# They must be owned by root.
sudo chmod 770 /securemount/mount.sh
sudo chmod 770 /securemount/unmount.sh
# Mark both shell scripts as executable.
sudo systemctl enable /securemount/secmount.service
# Install the new service file. Makes a symbolic link to secmount.service
sudo service secmount start
# Starts the service, it may throw errors because the partition is already
mounted.
systemctl status -l --no-pager secmount
# Displays the status of the service.
```

The systemctl status command should have an output that looks like the following:

```
● secmount.service - Mounts the encrypted partition
   Loaded: loaded (/secmount/secmount.service; enabled; vendor preset: enabled)
   Active: active (exited) since Thu 2017-10-05 22:04:27 SGT; 1h 37min ago
  Process: 1097 ExecStart=/secmount/mount.sh (code=exited, status=0/SUCCESS)
 Main PID: 1097 (code=exited, status=0/SUCCESS)
   CGroup: /system.slice/secmount.service
```

The output should not mention any errors or execution failures. If errors occur the cause is likely related to permissions. Use 'ls -la /secmount' to confirm that the scripts and secmount.service are marked as executable and are owned by root. If the status looks correct then restart your computer and make sure the encrypted partition is automatically mounted at start up.

4.4 Migration between TPM 1.2 and 2.0 with Linux

If you used either of the methods above for TPM 1.2 or 2.0, it is possible to access your encrypted files should you want to migrate to a different version of TPM. To do this you will need the key to unlock the encrypted partition. Because both above methods use compatible 32-byte keys, the partition will not need re-encrypted. Linux distributions will usually tolerate a processor architecture change without an operating system re-install, however some system stability issues may occur. If both processors have the same version of TPM chip, then it is possible to have the key in both devices' TPM chips to facilitate processor or drive swaps without losing access to encrypted files. Since Linux cannot encrypt the boot partition, you can easily re-install your operating system on the boot partition without losing the files on the encrypted partition. This allows for the processor swap to also change TPM versions from both 1.2 to 2.0 and 2.0 to 1.2.

4.5 Reading Windows Bitlocker Encrypted Partitions in Linux

Linux does provide a library that can decrypt Bitlocker partition, named Dislocker. You must still have the recovery key to the Windows Bitlocker partition. Once mounted the partition will be read only.

Step 1: Install Dislocker decryption library.

```
sudo apt -y install dislocker
# Install the dislocker library.
```

Step 2: Set up mount point folders.

```
desiredmountpoint='/bitlocker'
decryptpoint=''
# You may change the above variables to suit your needs.
sudo mkdir -p $desiredmountpoint
sudo mkdir -p $decryptpoint
# Create 2 directories. One will be the final mount point and the other
# will be where the decrypted device is mounted.
```

Step 3: Identify the Bitlocker partition.

```
lsblk | grep -e part
# Displays a list of partitions. Usually /dev/sd<character><number>
selectedpartition='/dev/sd<character><number>'
# Replace <character><number> with those that match your partition.
sudo umount $selectedpartition
# Make sure the partition is not mounted already.
# Do NOT try to unmount your boot partition!
```

Step 4: Use Dislocker to decrypt the partition

There are 3 methods of providing the encryption key to Dislocker:

- Manually enter the passphrase.
- Manually enter the recovery key.
- Parse the recovery key from the back up key file created by Bitlocker.

Bitlocker recovery keys are 48 characters separated by 7 '-' characters which makes them time consuming to type. Below is two code blocks using Dislocker to decrypt and mount a read-only Windows Bitlocker partition.

Use this command if you are manually typing a passphrase. Replace <passphrase> with the passphrase.

```
sudo dislocker -r -V $selectedpartition -u<passphrase> -- $decryptpoint
# <password> is the password to unlock the encrypted partition and must
# come immediately after the -u option (no spaces). The -- between the
# password is not a typo.
```

Use this command if you are manually typing a recovery key. Replace <key> with the passphrase.

```
sudo dislocker -r -V $selectedpartition -p<key> -- $decryptpoint
# <recovery key> is the password to unlock the encrypted partition and
# must come immediately after the -p option (no spaces). The -- between
# the password is not a typo.
```

Use this command if you are parsing a recovery key from a text file. Replace <path to key file> with the path to the location of the recovery key text file.

```
recoverykeypath='<path to key file>'
echo '#!/bin/sh' > temp.sh
# Set up variables in new script instance.
echo "keypath='$recoverykeypath'" >> temp.sh
echo "partition='$selectedpartition'" >> temp.sh
echo "recoverykey=''" >> temp.sh
echo 'while read -r line' >> temp.sh
echo 'do' >> temp.sh
# The following commands build the line filter. $iskey will be true if
# the line contains 7 - characters.
echo -n "\t" >> temp.sh
echo -n 'iskey=$(echo -n "$line" | tr -d " \t" | fold -w 1' >> temp.sh
echo -n ' | sort | uniq -c | awk ' >> temp.sh
echo -n "'" >> temp.sh
echo -n '$1==7' >> temp.sh
echo -n "'" >> temp.sh
echo -n ' | grep "-" | tr -d " ")' >> temp.sh
echo "/tif [ ! -z \"$iskey\" ]" >> temp.sh
# If $iskey is an empty string then no lines contained 7 - characters.
echo "/tthen" >> temp.sh
echo "\t\tif [ -z \"$recoverykey\" ]" >> temp.sh
# If $recoverykey is empty then this is the first valid key in the file.
echo "\t\tthen" >> temp.sh
echo "\t\t\trecoverykey=$(echo \"$line\" | tr -d \" \t\r\")" >> temp.sh
# Save key into $recoverykey and remove whitespace characters.
echo "\t\t\techo 'A valid key was found!'" >> temp.sh
echo "\t\t\techo \"$recoverykey\"" >> temp.sh
echo "\t\t\techo" >> temp.sh
echo "\t\tfi" >> temp.sh
echo "\tfi" >> temp.sh
echo "done < $keypath" >> temp.sh
echo 'sudo dislocker -r -V $partition "-p$recoverykey" -- $decryptpoint'
>> temp.sh
# The above command is all one line.
sudo chmod +x temp.sh
# Make sure the temp script is marked as executable.
./temp.sh
rm -f temp.sh
```

The above script searches a text file line by line while checking for 7 - (dash) characters. The key to this script is the line that starts with $iskey, a chain of piped commands. This command counts the number of occurrences of every character and filters for 7 occurrences of the dash character. If $iskey is an empty strip then 7 dashes didn't occur in that line of the text file, meaning no key was on that line.

Step 5: Mount the decrypted device

```
sudo mount -r -o loop $decryptpoint/dislocker-file $desiredmountpoint
```

Now your Windows Bitlocker partition is accessible by navigating to the desired mount point folder.

5 Conclusion

5.1 Windows, Security over Flexibility

Windows Bitlocker offers the best security possible while using a TPM chip. Boot partition encryption and user-friendly set up are the primary features being offered by Windows Bitlocker. The downside to Windows Bitlocker is the lack of cross platform support. Windows requires a full re-install and re-encryption should you ever need to, or want to, upgrade your hardware. You will need the partition recovery key(s) to facilitate this transition, so it is important to store them in a safe and known location. A UEFI installation of Windows will not boot on hardware with a different processor architecture. By using two partitions you can simply overwrite your old Windows installation while keeping your other files safe on the other partition. This multi-partition method is highly recommended for users of older or obsolete CPUs as finding the same architecture device may be expensive, difficult, or impossible.

5.2 Linux, Flexibility over Security

The Linux methods of TPM usage offer flexibility and compatibility at the expense of set-up complexity and ease of use. The biggest downside of Linux TPM usage is the inability to encrypt the boot partition. The GRUB Boot Loader does not yet have TPM support to handle partition decryption before the operating system is loaded. While this downside has some work arounds to maintain security, such as using multiple partitions to encrypt important parts of the system, but it requires significant forethought and knowledge compared to Windows Bitlocker. On the other hand, Linux TPM usage provides the flexibility of being able to swap processors of various architectures and does not require partitions to be re-encrypted should the operating system change.

6 Disclaimer and Limitation of Liability

6.1 Content Disclaimer

RTD Embedded Technologies hereby disclaims all (and have not authorized anyone to make any) warranties express or implied, including, but not limited to, any warranties of non-infringement of third party rights with respect to open source software, title, merchantability and fitness for a particular purpose. The parties are not relying and have not relied on any representations or warranties whatsoever regarding RTD Embedded Technologies. RTD Embedded Technologies makes no warranty regarding any third-party software.

6.2 Referrals and Links to Third Party Sites

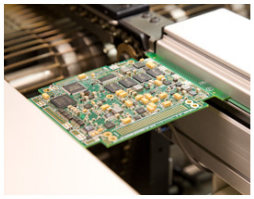
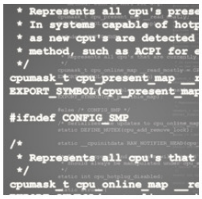
This document may contain links to other web sites (linked sites). The linked sites are not under the control of RTD Embedded Technologies. RTD Embedded Technologies is not responsible for any contents linked or referred to from this document. EFACTS is not responsible for webcasting or any other form of transmission received from any linked site. RTD Embedded Technologies is providing this document to you as a convenience, and the inclusion of any link or third-party software does not imply endorsement by RTD Embedded Technologies or any association with its operators. RTD Embedded Technologies is not liable for any damage that occurs from the use of the information presented in this document.

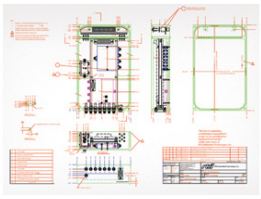
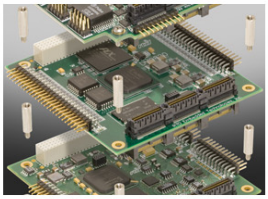
RTD Embedded Technologies encourages you to review the privacy statements of websites you choose to link to from this document so that you can understand how those websites collect, use, and share information. RTD Embedded Technologies is not responsible for the privacy statements on websites outside of this document and rtd.com.

RTD Embedded Technologies, Inc.
103 Innovation Boulevard
State College, PA 16803 USA
Telephone: 814-234-8087
Fax: 814-234-5218

www.rtd.com

sales@rtd.com
techsupport@rtd.com

PC/104

DESIGNED AND MADE IN THE UNITED STATES™

AS9100 · ISO 9001 CERTIFIED

Copyright 2021 by RTD Embedded Technologies, Inc. All rights reserved.