MSE | MASTER OF SCIENCE IN ENGINEERING

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL

MSE Project Report

A Root of Trust for Measurement

Mitigating the Lying Endpoint Problem of TNC

Sansar Choinyambuu schoinya@hsr.ch

Advisor: Prof. Dr. Andreas Steffen

June 15, 2011

Hochschule Rapperswil

Oberseestrasse 10

CH-8640 Rapperswil

Table of Contents

Abstract

We got familiar with the “lying endpoint” problem during the implementation of the Posture Broker Protocol [21, 19] for strongSwan [15]. It was caused by the lacking trust in the communicating endpoint within the Network Endpoint Assessment (NEA) architecture [20]. Whether it is a NEA client or a server, it is essential to be sure about the trustworthiness of the corresponding node. That is to say, whether the communicating node behaves “good” or “bad”, we are certain that it reports its current state accurately and we obtain it in uncompromised state in a secure way. We may take the Posture Collector as an example which intentionally misrepresents the posture of the NEA client node and thus leads to an inappropriate access recommendation.

This is the general problem which could be transferred onto the field of everyday technology usage, if we consider the amount of trust we have in technology. How can the user be sure that the software that is running on his/her computer is genuine and thus can be trusted? How can the client be sure that the sensitive information he’s giving to the server is handled in trustworthy way? The answers and solutions to these questions and challenges are provided by the broad context of “Trusted Computing”.

Substantial assistance can be provided with the Trusted Platform Module (TPM), a hardware component for measuring and verifying trust. Using the available solutions and projects for Static and Dynamic “Root of Trust for Measurements”, we would like to explore and demonstrate the possibility of gaining verifiable trust on launched environments and on executable code.

1. Introduction

The term “Trusted Computing” has reshaped the simple context of “Trust” in the domain of IT Security. The Trusted Computing Group (TCG) describes "Technical Trust" as: "an entity can be trusted if it always behaves in the expected manner for the intended purpose” [18]. The ability to verify the trustworthiness of an entity is another important concept introduced by Trusted Computing.

The question of “What is Trust based on?” leads us to the “Root of Trust” notion. The TCG’s Root of Trust, the Trusted Platform Module (TPM) is a microcontroller security chip that is permanently attached to the mother board of most enterprise level computers that are shipping today. The TPM is used during the boot process to establish a trust level and gather the measurements on the launching environment for reporting purposes. The measurements are saved in the TPM’s Platform Configuration Registers (PCR) in a manner that is computationally infeasible to forge. (A detailed introduction on this operation follows in 2.1)

In the “Static Root of Trust for Measurement” (SRTM) by as defined by the TCG, a fixed or immutable piece of trusted code in the BIOS is loaded at the start of the entire booting chain, and every piece of code in the chain is measured by the predecessor code before it is executed. Thus any compromised code in the chain cannot escape from being measured and therefore from being detected. The SRTM however has obvious scalability shortcomings if we intend to measure every component in the trust chain; in addition, it gives only the load time guarantee but not the run time guarantee for the launched environment.

“Dynamic Root of Trust for Measurement” (DRTM) is the underlying trust mechanism for Intel’s Trusted Execution Technology (TXT) and AMD’s Secure Virtual Machine (SVM) technology. It introduces platform level enhancements in order to provide run time protection and guarantee. In contrast with the SRTM, the DRTM has the advantage that the launch of the measured environment can occur at any time without resorting to a platform reset.

In this paper, we are going explore the extent to which available solutions can provide “Trust”, both with Static and Dynamic Root of Trust for Measurements.

1.1 Objectives

The goal of the present project is to explore the possibilities of guarantying verifiable load time and run time genuineness of the executable code.

1.2 Structure of the report

The entire idea behind “Trusted Computing” in the field of measuring the launched environment and consequently gaining trust in it is based either on the Static or Dynamic “Root of Trust for Measurement” concept. Therefore the present paper is also roughly divided into corresponding two parts. In each part we introduce the concept firstly, followed by the detailed explanation based on available solutions and projects.

Finally, the experience we collected as well as the challenges we faced and the outlook into the future are discussed in conclusion.

In addition to this, Acronyms, Table of Figures, and Table of Source code References, Bibliography, Weekly status reports and Project plan follow the main chapters. The Contents of the CD which is reached out with the project report, can be seen in the Appendix as well as patch to an identity.c tool from PrivacyCA.

2. Terms and Technologies

In this section, the commonly used terminologies are explained and an introduction is given to the "Trusted Computing" technologies which are closely related to the topic of the paper.

2.1 Trusted Platform Module (TPM)

In the late 90's IBM started shipping the motherboard with an embedded public key smart card chip. The concept was to make public key hardware tokens available at very low cost, by embedding them and eliminating the need for separate smart cards and readers. The other vendors looked at similar solutions and the need for a single common standard has led to the TPM Specifications and the standardized TPM chip which is delivered with most of the enterprise level systems shipping today.

The Trusted Computing Group (TCG), an alliance initiated by AMD, Hewlett-Packard, IBM and Intel is now a consortium of more than 100 companies that forged open standards for Trusted Computing. The current version of the TPM specification is 1.2 Revision 103, published on July 9, 2007 [16]. On the other hand, there is a lot of criticism, and controversy over Trusted Computing has been raised by the very same specifications from TCG, in that critics claim [22] that the specification takes control of the operation of the computer away from the computer user, and that it will provide an easy way for companies to force computer users into moving away from competitive software and to build Digital Rights Managements into the computing platform.

The TPM has been designed to protect security by ensuring the following: [1]

- Private keys cannot be stolen or given away.
- The addition of malicious code is always detected.
- Malicious code is prevented from using the private keys.
- Encryption keys are not easily available to a physical thief.

The TCG chip accomplishes these goals with three main groups of functions, as follows:

- Public key authentication functions
- Integrity measurement functions
- Attestation functions

The *public key authentication* functions provide for on-chip key pair generation using a hardware random number generator, along with public key signature, verification, encryption, and decryption. By generating the private keys in the chip, and encrypting them anytime they are transferred outside the chip, the TPM guarantees that malicious software cannot access the keys at all. Even the owner of the keys cannot give the private keys away to phishing or pharming attacks, as the keys are never visible outside the chip unencrypted. Malicious code could use the private keys on the TPM, so some way needs to be provided to ensure that malicious code cannot use the keys either.

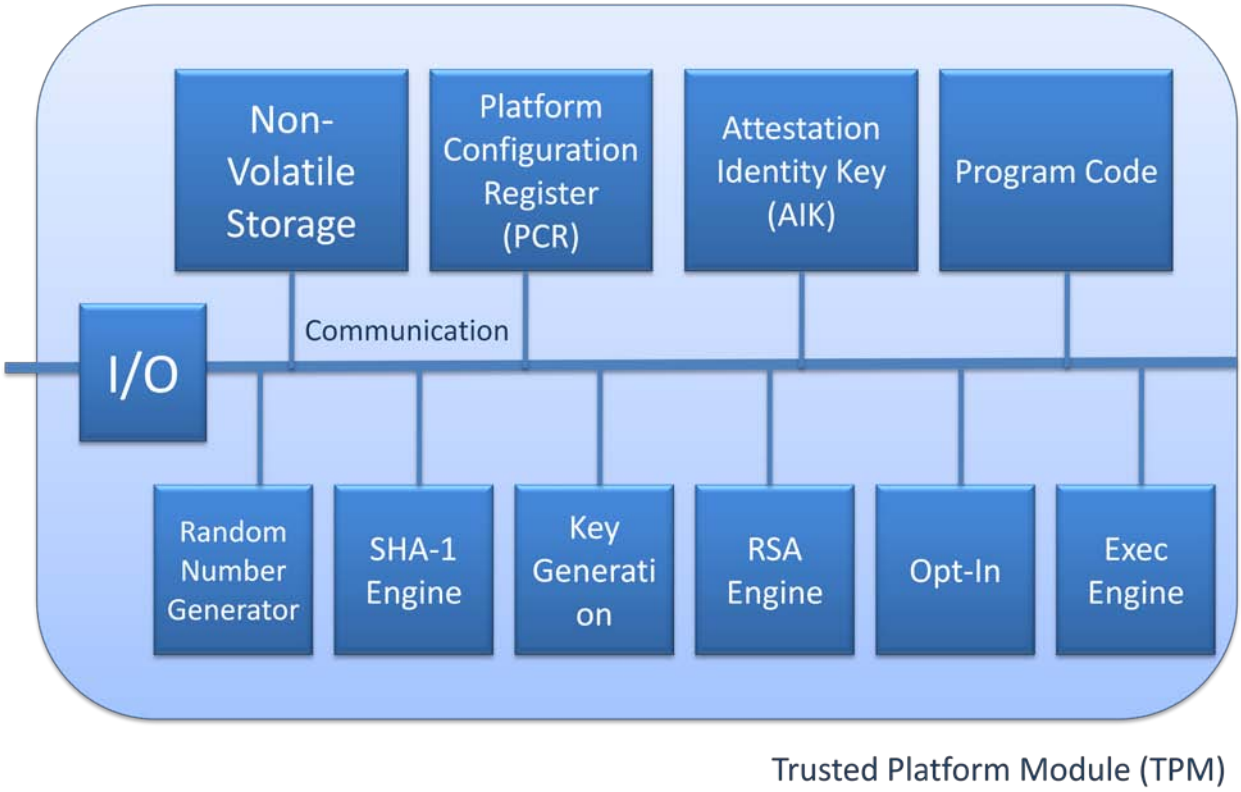

Figure 1.Trusted Platform Module component architecture

The *integrity measurement* functions provide the capability to protect private keys from access by malicious code. In a trusted boot, the chip stores in the Platform Configuration Registers (PCRs) hashes of configuration information throughout the boot sequence. Once booted, data (such as private keys) can be "sealed" under a PCR. The sealed data can be unsealed only if the PCR has the same value as at the time of sealing. Thus, if an attempt is made to boot an alternative system, or a virus has "backdoored" the operating system, the PCR value will not match and the unseal will fail, thus protecting the data from access by the malicious code.

The *attestation* functions keep a list of all the software measurements committed to the PCRs, and can then sign them with a private key known only by the TPM. Thus, a trusted client can prove to a third party that its software has or has not been compromised.

2.1.1 PCRs

The TPM contains 24 Platform Configuration Registers according to TPM Specification 1.2 [[16] , that allow a secure storage and reporting of integrity measurement which is essentially the SHA-1 digest of the unit that has been measured.

Each PCR is capable of holding 20 bytes and the measurement value doesn't simply overwrite the old PCR value, the updating of the saved value on PCR is a rather complex operation called "extending". It concatenates the 20 bytes of data already held in the PCR with 20 bytes of new data calculated by hashing the new unit being measured, and these 40 bytes of data are then hashed again using the SHA-1 algorithm and the result is written to the PCR.

$$PCR_{n+1} = SHA1\ (\ PCR_n + \ SHA1(Component)\ )$$

This way, an unlimited number of measurements can be stored in a single PCR and the order of the measurement is preserved thanks to the irreversible feature of SHA-1 hashing algorithm.

The following usages are foreseen for each of 24 PCRs according to the TPM Specification [16] [23].

PCR Index	Usage
0	Core BIOS, POST BIOS, Embedded Option ROMS
1	Platform and Motherboard Configuration and Data
2	Option ROM Code
3	Option ROM Configuration and Data
4	IPL code
5	IPL configuration data
6	State transition (sleep, hibernate, and so on)
7	Reserved for OEM
8	Not assigned
9	Not assigned

10	Not assigned
11	Not assigned
12	Not assigned
13	Not assigned
14	Not assigned
15	Not assigned
16	Used for debugging
17	Dynamic CRTM
18	Platform defined
19	Used by a trusted operating system
20	Used by a trusted operating system
21	Used by a trusted operating system
22	Used by a trusted operating system
23	Application support

Table 1.The Standard Usage of PCRs [[1] [[16], [[23]

One of the TPM operations which is of significant importance to the Attestation feature is the TPM Quote. This operation is used to authoritatively verify the contents of a TPM's PCRs. During attesting, a composite hash of a selected set of PCRs is computed. This composite hash can be compared with the one computed while attesting.

The TPM quote operation returns signed data and a signature. The data that is signed contains the list of selected PCRs, the composite hash for the selected PCRs, and a nonce provided as input which is used to prevent replay attacks. Given the signature, the evaluating entity replaces the nonce in the signed data generated at provisioning time, and checks to see if the signature is valid for the data. If so, this check ensures the selected PCRs contain values that match the ones measured during provisioning [29].

2.1.2 Root of Trust

The foundation of all operations the TPM is capable of, rests on the "Root of Trust". The concept of technical "trust" doesn't differ in aspect substantially from the way people gain trust in each other. Namely, the trust between the people is based on the confidence we have in person. This very confidence is provided by the Roots of Trust in case of the Trusted Computing.

In the Trusted Platform Architecture defined by TCG there are three different Roots of Trust:

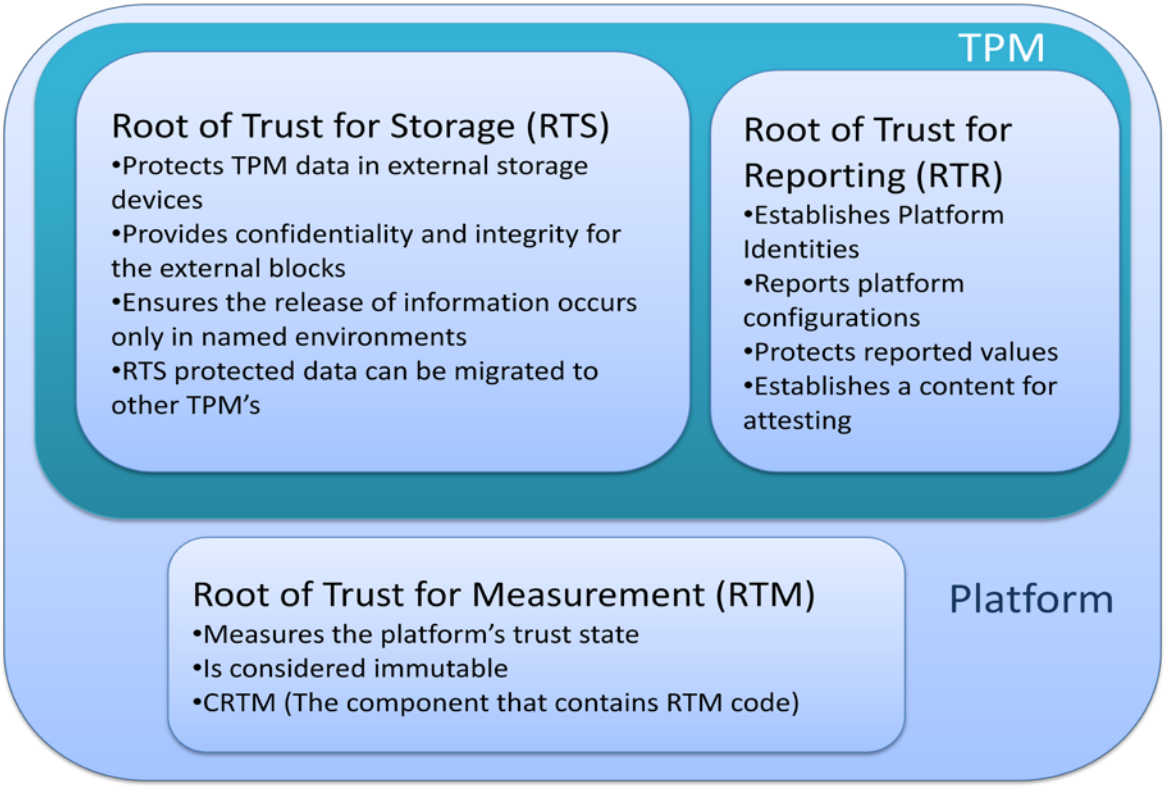

Figure 2.Root of Trust defined by TCG

The Root of Trust for Storage is a trusted entity that provides confidentiality and integrity protection for stored information without interference leakage. It uses Platform Configuration Registers (PCR) and RSA encryption to protect data and ensures that data can only be accessed if platform is in known state (Sealing).

Root of Trust for Reporting is a component trusted to report to external parties information about the platform state accurately and correctly and in an unforgeable way using PCRs and RSA signatures. Root of Trust for Reporting and Root of Trust for Storage are the basic building blocks of TPM security design. None of the TPM service can function without their sole and mutual utilization.

The Root of Trust for Measurement defined by the TCG relies on the piece of RTM code which is neither stored on TPM nor encrypted or sealed using it. Thus, it is depicted out of the TPM range in Figure 2. Core Root of Trust (CRTM) code is considered immutable and is envisioned as a Root of Trust for Measurement on given environment.

2.2 Trusted Boot

The booting process (the abbreviation of the longer term Bootstrapping) has the responsibility of initializing the main hardware components (the task of the BIOS), and afterwards starting the operating system of the platform (usually carried out by the Boot loader).

One of the security goals of Trusted Computing by TCG was to establish trust in a booted/launched environment. The Chapters which follow will explain how this trust is obtained using “Static Root of Trust for Measurement” and “Dynamic Root of Trust for Measurement” respectively. Therefore, the general introduction to the trusted boot and few detailed insights are given here.

Before any program executable can be trusted, the environment in which all the software is running should be trusted. That means the state of the booted environment must be verifiable and only when its uncompromised state is measured, “trust” would be established. There should be also methods to report the measured state to third parties in a reliable and verifiable way using “Root of Trust for Report”, so that the certain actions could be taken depending on the state measurement of launched environment.

The new hardware and software enhancements (TPM and TSS (TCG Software Stack, which were made available thanks to TCG efforts, introduce a significant advancement for the purpose of measuring the integrity of a launched environment. Nevertheless, just the availability of them doesn’t guarantee the protected environment. There is a variety of middle-ware and high level applications developed under open source licenses, which make use of the platform level secure service enhancements such as TPM.

Being normally implemented as a passive hardware chip, the TPM is not able to influence the program flow of the CPU. It receives commands and data from the CPU, executes the commands sent and returns the results. Hence, an additional active software component has to take an appropriate administrative action in case of any inconsistency during the booting which in turn can be measured and detected with the help of the TPM.

The Trusted Computing Group defines “Trusted boot” or “Authenticated boot”, as the booting process, in which every entity in the booting sequence is measured before it is executed by its predecessor. On the contrary there is a term “Secure boot”, which interrupts and stops the Booting

process actively if an undesired state of the platform is detected [24]. Despite the fact that the term “Trusted boot” is usually used in the linked context of “Static Root of Trust for Measurement” defined by TCG, both the SRTM and DRTM way of trusted boot should be understood under this general term in this paper.

As aforementioned in the TPM introduction, there are controversies raised over the topic of “Trusted Computing stealing” the control over the hardware from the users. Therefore the active component has to provide the users the possibility to configure as for what should be done in case of the detection of undesired compromise of the launched environment.

3. Static Root of Trust for Measurement

3.2 Introduction

As the name “Static Root of Trust for Measurement” suggests, the entire trust begins with the static, immutable piece of code, which is called Core Root of Trust for Measurement (CRTM). On ordinary computing platforms the BIOS is the first component to be executed. Therefore the Trusted Platform needs an additional entity, which would measure the BIOS itself and act as a CRTM. This entity is a fundamental Trusted Building Block (TBB) that remains unchanged during the lifetime of the platform.

The CRTM consist of the CPU instructions that are normally stored within a chip on the motherboard. Alternatively, the CRTM can also be an integrated part of the BIOS itself.

For the developers there are two ways to integrate the CRTM with the BIOS as foreseen by the TCG. Firstly, the whole CRTM code is contained in the BIOS boot block, so that the rest of the BIOS can be updated as usual. The drawback of this approach is that the CRTM has to measure the rest of the BIOS for integrity. The second solution from the TCG proposes not to strictly define the position of CRTM within the BIOS, so that the whole BIOS becomes a part of TBB. In both cases the vendors have the responsibility to provide a trustworthy CRTM (BIOS in second approach) – one of the important TBB's from TCG [24].

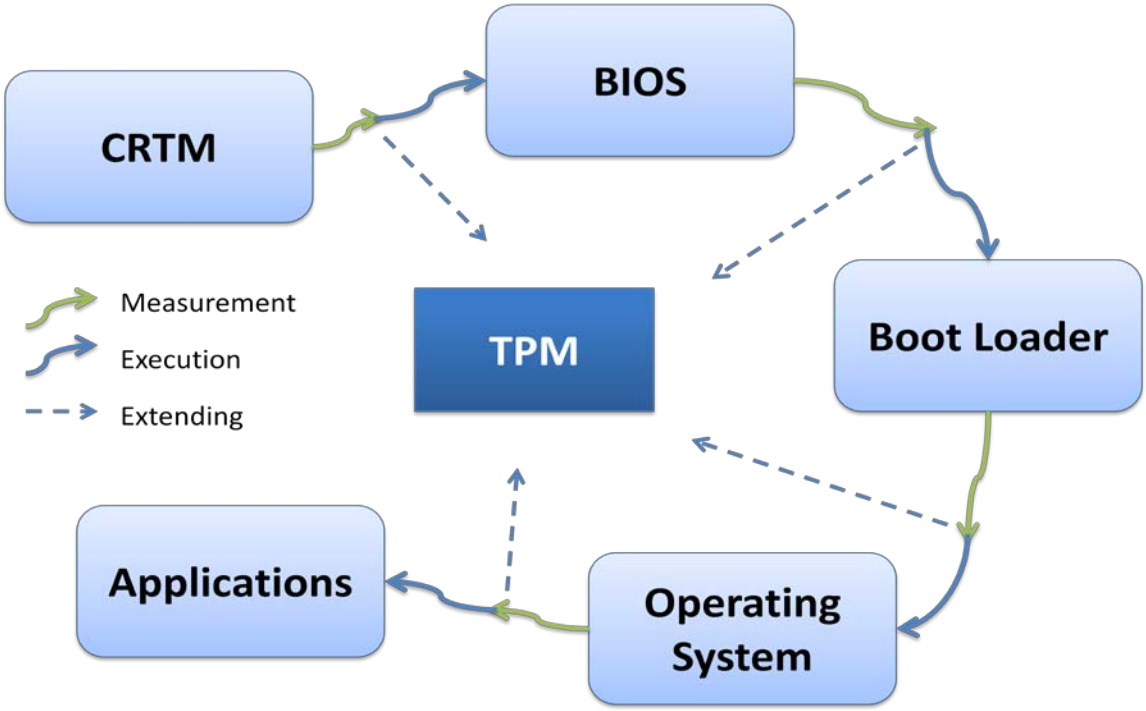

Figure 3.Chain of Trust

The trustworthy CRTM should reliably measure the integrity of next piece of code that follows in the boot sequence. The result of the measurement is extended into the PCR (the usage of certain PCR is listed in 2.1.1) before the control is transferred to the successor code in the sequence. If each component in the sequence in turn measures the next before transferring control, a chain of trust if established. If this measurement chain continues through the entire boot sequence, the resultant PCR values reflect the measurement of all files used [1]. An Example of a booting process which builds such a Chain of Trust is illustrated in the Figure 3.

Even if one of the components in the chain gets compromised, it cannot avoid being measured before its execution during the next reboot. During the execution, the compromised software is technically not able to take the extended measurement back from the PCR.

Since the control is transferred to the malicious software, it is free to fabricate all the subsequent measurements. However, it is technically not possible for the compromised code to generate the fake measurements, which once extended to PCR would equal to the value it would have had after an uncompromised boot. This property is provided by the cryptographic strength of SHA-1 hashing algorithm, which makes it computationally infeasible for the malicious code to calculate an extension value that would “fix” the PCR values.

In the following sections, the establishment of Chain of Trust s explained based on the certain open source implementations.

3.2 Extended Boot Loaders

Normal boot loaders can be extended with the features that make them capable of granting the integrity of the boot process. The idea behind the extension is to utilize “Root of Trust for Measurement” functionality offered by TCG specifications to carry the chain of trust forward which is started by the CRTM. This is done by measuring all critical components during the boot process, i.e., stage2 of GRUB, the OS kernel or OS modules, together with their parameters.

There exist several open source boot loader implementations such as TrustedGRUB and GRUB-IMA that enhance the GNU GRUB. The TrustedGRUB offers some supplementary features in addition to the GRUB-IMA functionalities. Moreover, the GRUB-IMA has massive shortcomings in terms of speed due to the fact that they use SHA-1 hashing operations in hardware offered by TPM. On the contrary TrustedGRUB has implemented the SHA-1 measurements within the software in release 1.1.5, so they've overcome the speed difficulties

On account of these reasons and also since it was the boot loader that is used for testing the SRTM for this semester project, the next section introduces the TrustedGRUB. For detailed information on both of the boot loaders and some summaries please refer to [9][17].

3.2.1 TrustedGRUB

TrustedGRUB is an enhancement of the open-source bootloader GNU GRUB, developed by Sirrix AG together with the Chair for System Security at Ruhr-University Bochum, Germany [9].

After the BIOS measures the boot loader located in the Master Boot Record, it transfers control to the loader, in this case to the TrustedGRUB. Then the chain of trust is carried forth by TrustedGRUB, by measuring the integrity of the Operating System it is configured to load and extending the result into a PCR. The extended values of the PCRs provide the evidence to attest the booted system configurations.

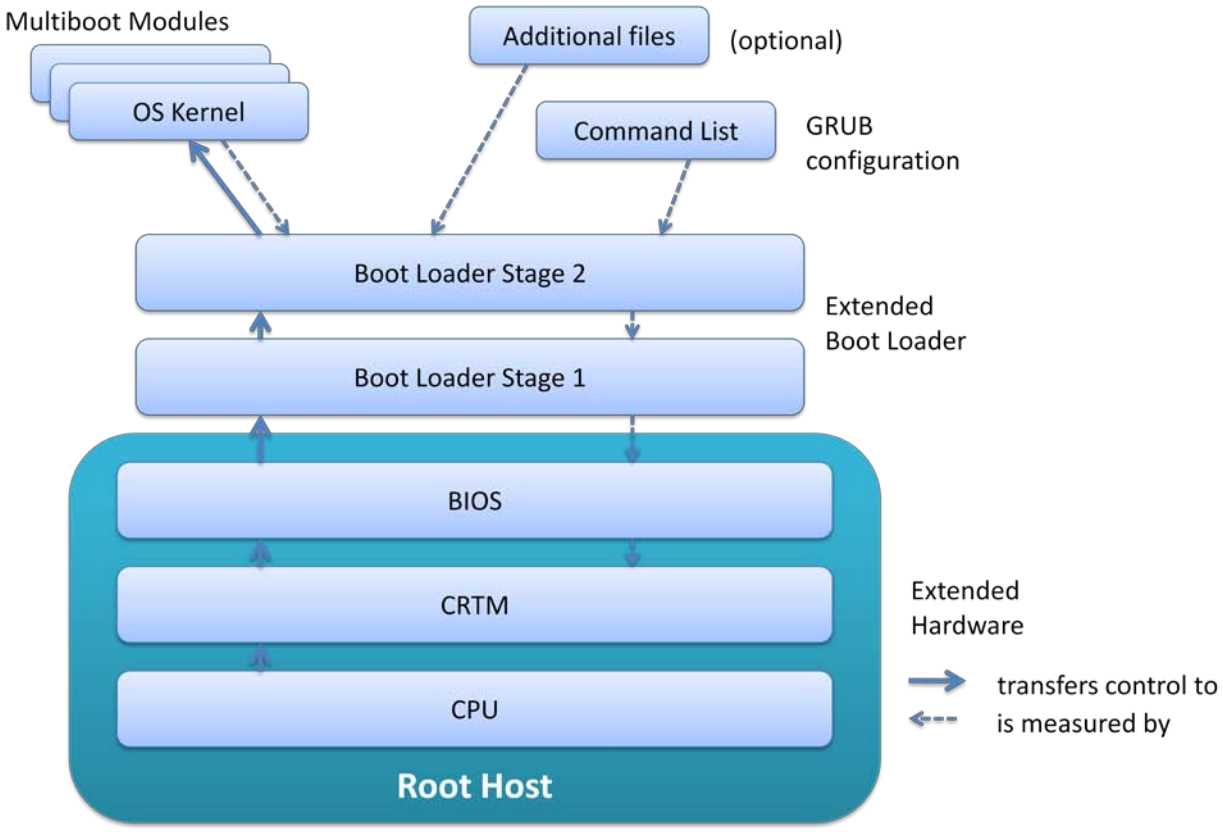

Figure 4.Chain of Trust, extended Boot Loader is in use (TrustedGRUB)

Furthermore, TrustedGRUB offers the important feature to verify the integrity of arbitrary file after the loading of Operating System. With the help of this functionality, the users can continue the chain of trust with the necessary components to be integrity measured. This is realized by providing a "checkfile"-option, where TrustedGRUB will load and verify the given files by comparing the SHA1-results with precalculated values stored in the checkfile. All files verified are additionally extended into a PCR as well. This feature is explained also in following Configuration section.

Additionally, a TPM can be used to encrypt data in such a way that the data is "sealed" to a certain platform configuration, i.e., it can only be decrypted if the booted configuration has not changed in the meantime. Therefore, it becomes impossible to bypass security policies by booting another operating system or by extracting the hard disk [9].

Moreover, if a module is configured with parameter "with-password-dialog", Trusted GRUB prompts for a password during the boot and replaces this parameter with the "password=password_you_have_entered" string. Thereupon, pre-boot authentication is possible during the next reboot.

3.2.1.1 Configuration

The following is the format of the boot entry in the TrustedGRUB configuration file /boot/grub/menu.lst:

```
title Ubuntu 10.10, kernel 2.6.38-020368-generic
      uuid 154e4bc5-aa94-4e1d-9f1c-e6c5cda3ed38
kernel /boot/vmlinuz-2.6.38-020638-generic root=UUID=154e4bc5-aa94-
      4e1d-9f1c-e6c5cda3ed38 ro quiet splash
initrd /boot/initrd.img-2.6.38-020638-generic
checkfile /boot/grub/checkfile
```

The checkfile option takes the path to the file, which contains the list of file files, which a user wants to verify the integrity of:

```
2647eeae7290c5a58dacb87347ba074de7e47bac (hd0,1)/etc/passwd
a97fbdba48d4a6340baff683941079dde56044e0 (hd0,1)/etc/shadow
6fc01c858d17593a309b91d5fe5859c545409861 (hd0,0)/home/Example.txt
```

Each line presents a file to be measured and starts with a 40 bytes SHA-1 hash value of the succeeding file (the value can be created either by sha1sum under GNU/Linux or the executable called create_sha1 that comes with the TrustedGRUB distribution), followed by a single space and an

absolute path to the file. A comment has to be added here that the checkfile itself should not exceed a 8096 bytes limit.

The integrity of all files listed in this checkfile is verified during system boot by comparing the referenced hash values to the newly-computed values. If some of these do not match, a warning is displayed offering the option of either continuing with the booting of a potentially contaminated system or aborting the boot process completely. This is an initial step towards secure boot as defined in section 2.2. All files verified by the checkfile option are extended into the PCR 13 [17].

3.3 AIK, AIK Credential and PrivacyCA

Every TPM has a 2048 bit unalterable RSA key generated in it during manufacture time, called Endorsement Key (EK). There are lots of privacy concerns raised over the fact that any platform can be uniquely identified and tracked down if an EK is used for digital signature operation. This explains the reasoning of concepts behind Attestation Identity Key (AIK).

Allegedly an AIK is an alias for the platform-unique key, the EK. At creation time, it gets tied to a TPM identity, which is in turn tied to an EK. In this way, the AIK can be proven to be created by a genuine TPM without exposing any part of the EK itself. It provides a mechanism to ensure that a verifier is communicating with the genuine TPM without revealing the identity of it.

Like the EK, an AIK is a 2048 bit asymmetric RSA key which is used only to sign the information generated inside the TPM (e.g. PCR values). The TPM is able to create an unlimited number of AIKs.

When an AIK is used to sign the data, the verifier needs assurance that the key is indeed TPM protected. The platform must prove to the verifier that the TPM owns the AIK and the AIK is tied to valid endorsement, platform and conformance credentials. Therefore the AIK needs a vouching in the form of a certificate which confirms the authenticity of it.

One approach to obtain such a confirmation relies on a Trusted Certificate Authority called Privacy CA (PCA), which issues AIK credentials [14]. The Privacy CA is trusted not to reveal sensitive information. It is also trusted not to misrepresent the trust properties of platforms for which AIK credentials are issued. For instance, a Privacy CA is trusted not to issue AIK credentials when the verification of the endorsement credential fails.

Within the scope of this project, we tested the Attestation IMV/IMC pair from TNC@FHH (an introduction follows in a subsequent section) using the AIK key and credential generation tool ***identity.c*** from Privacy CA [14]. We faced certain difficulties to generate the AIK itself, as the identity.c code was implemented to get the TPM secret (i.e. the owner password) in a pop-up window. The secrets of both TPM and SRK were set to the well known password of 20 bytes of zeroes. Therefore identity.c was modified partly, so that it gets the owner password from a defined string of

20 bytes of zeroes. The corresponding patch for identity.c is attached to the present paper in appendix.

The Process of creating an AIK and obtaining certificate for it is depicted in the following figure:

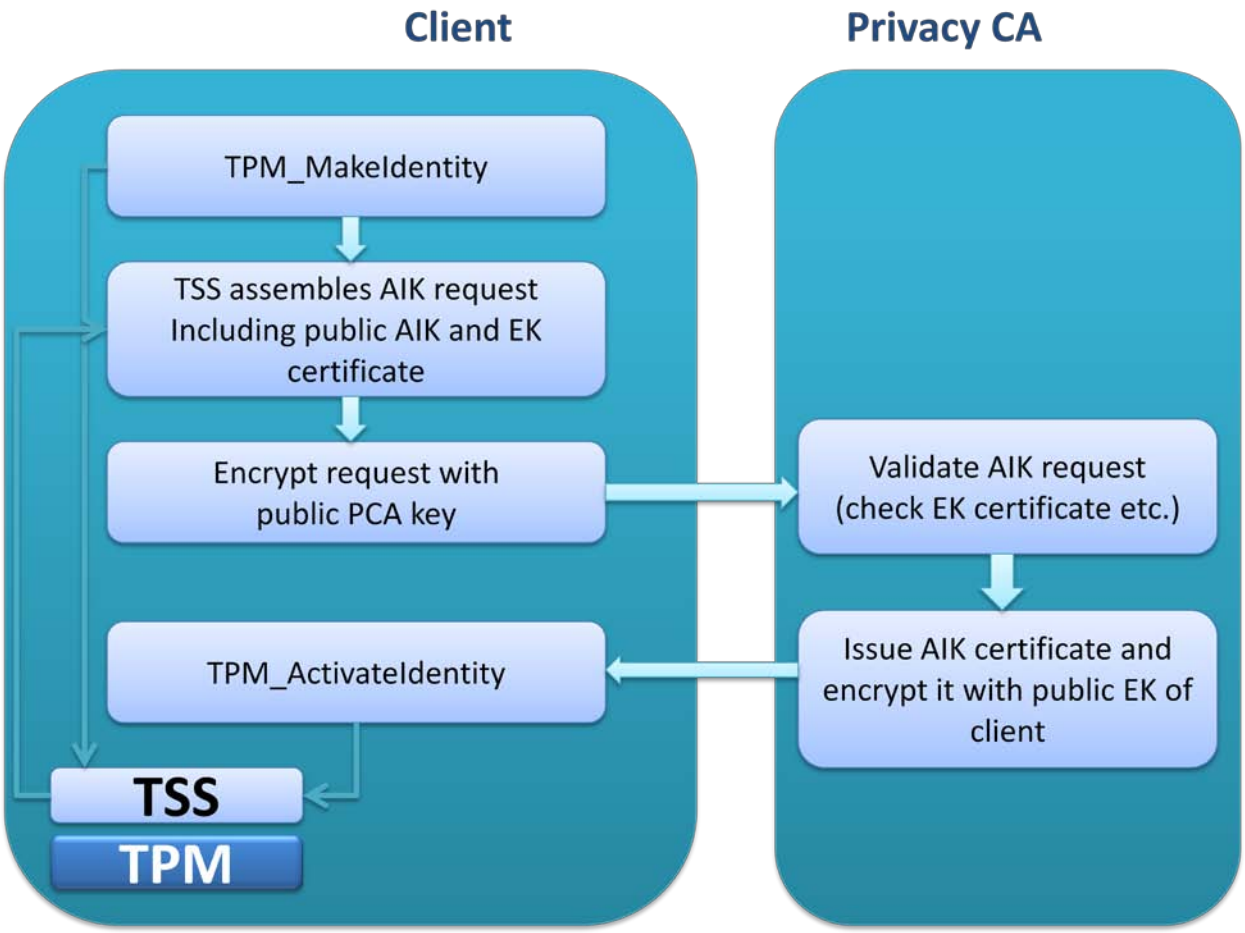

Figure 5.AIK generation and obtaining the AIK certificate using Privacy CA

Another approach to obtain AIK credentials is the TCG-specified 2Direct Anonymous Attestation" (DAA) protocol. It is included in the TPM 1.2 [16] specification; in order to overcome the privacy concerns introduced by a bad-natured Privacy CA. DAA is not introduced here in detail, for more information please refer to the TPM Specifications.

3.4 Attestation IMV/IMC pair from TNC@FHH

TNC@FHH is the open source implementation of the Trusted Network Connect (TNC) framework from TCG, developed by the Trust@FHH research group at Fachhochschule Hannover, University of Applied Sciences and Arts, Lower Saxony, Germany [11]. In the distribution package as of version 0.7.0, among several IMV/IMC pair implementations, an Attestation IMV and IMC pair is included

which implements a simple binary attestation protocol. This IMV/IMC pair is of special interest to our project. Considering the combination of features offered by TrustedGRUB we could attest the state of the platform (integrity measurements of components contained in chain of trust, additionally optional arbitrary file verifications).

First of all, our TPM-capable host has to generate an AIK and obtain the certificate from PrivacyCA using the identity.c tool. The PCR values that reflect the integrity of the client platform have to be signed by the AIK, which are then checked by the verifier in turn.

```
# compiling  identity.c
gcc -o identity identity.c -lcurl -ltspi
# Obtaining AIK and certificate
identity /home/user/PrivacyCA/AIK_KeyBlob /home/user/PrivacyCA/AIK_Cert.der
# Getting SHA-1 fingerprint from AIK certificate
openssl x509 -sha1 -in AIK_Cert.der -noout -fingerprint
```

The integrity of the Client is verified based on its TPM PCR values. We configure the TrustedGRUB in a way that it measures the integrity of components that are building the chain of trust. In addition to that, shared object files used by Attestation IMV/IMC are measured as well using the “checkfile” option of TrustedGRUB.

On the Verifier side we configure the list of PCRs we would like to quote along with its expected values. According to the TrustedGRUB documentation:

- PCR 4 contains MBR information and stage1
- PCR 8 contains bootloader information stage2 part1
- PCR 9 contains bootloader information stage2 part2
- PCR 12 contains all command line arguments from menu.lst and those entered in the shell
- PCR 13 contains all files checked via the checkfile-routine
- PCR 14 contains all files which are actually loaded (e.g., Linux kernel, initrd, modules...)

In case of all the above PCRs are listed for the quotation, we use the capabilities of TrustedGRUB to the full extent and can verify integrity of every component on the Client which has been measured. However, the reference hash values of the PCRs have to be taken from the clean secure state of the entities that are measured. Only when this is guaranteed, we benefit from the further measurements and attestations and subsequently detect the inconsistencies.

Furthermore, the fingerprint of the client AIK certificate has to be known to the verifier, due to the fact that the current TNC@FHH Attestation IMV is not capable of verifying CA certificate trust paths. The following figure illustrates the steps involved in the handshake between NEA Client and NE Server using the Attestation IMV and IMC, respectively.

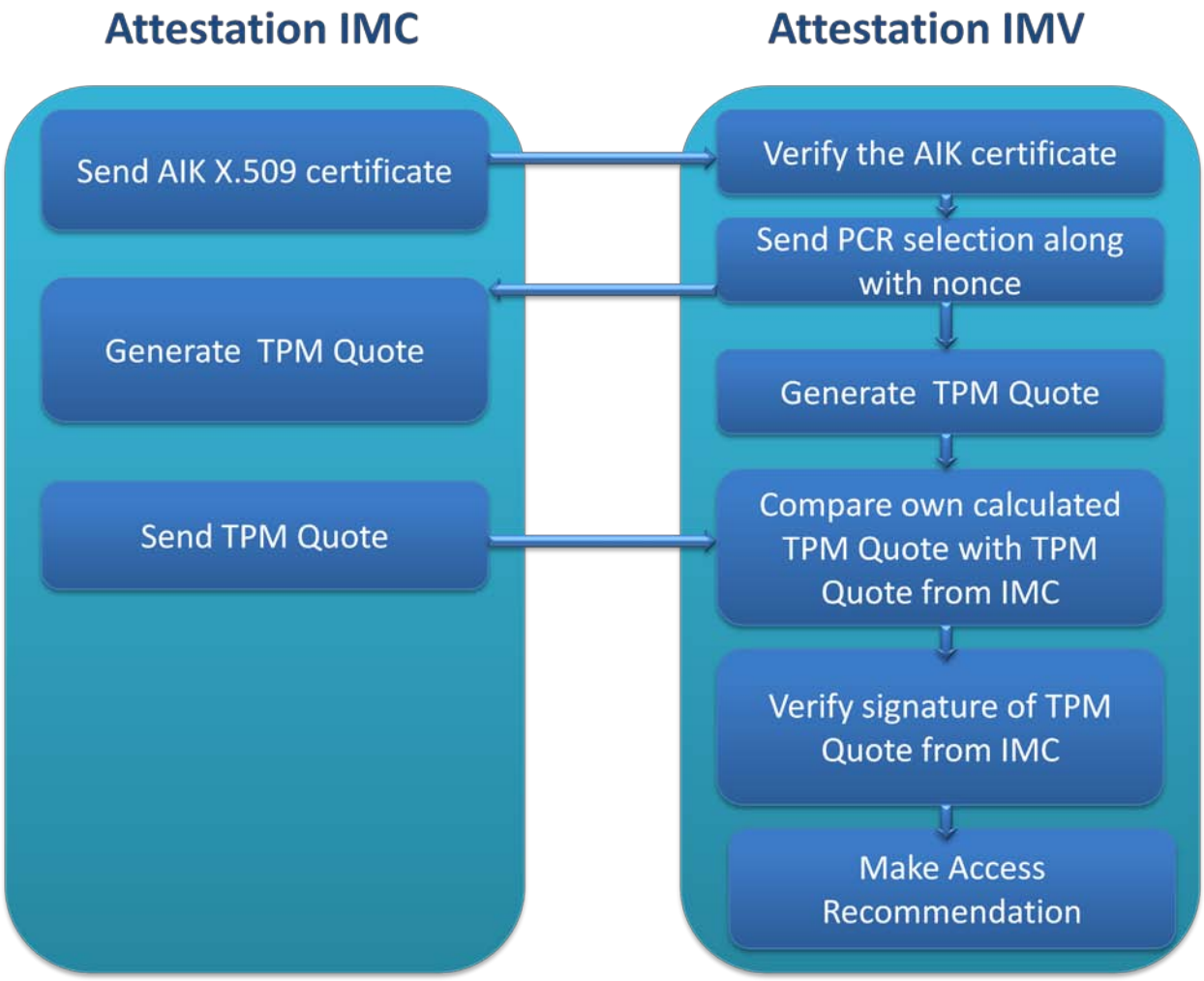

Figure 6.Handshake between Attestation IMV and IMC

3.4.1 Configurations

Our testing environment consisted of a TPM capable host called “carol” which acted as a NEA Client (Access Requestor) accompanied by a UML Guest which played the role of a NEA Server (Verifier). The UML guest called “moon” from the strongSwan testing framework was used together with its certificates. On both systems, the ***strongSwan*** [15] software was installed, with the TNC related plugins enabled and configured. The ownership of the TPM on the host was taken with the well known secret (20 bytes of zeroes) and the ***Trousers*** with ***tpm_tools*** [8] package were installed.

The following are the configurations on the Client side:

```
# /boot/grub/menu.lst          TrustedGRUB configuration file is same as in section 3.2.1.1

#/boot/grub/Checkfile
# The list of files to be integrity verified by TrustedGRUB
c908f8f5222ba77c3c8da20bae27ad3214c2343b (hd0,0)/usr/local/lib/libattestationimc.so.0.7.0
6fc01c858d17593a309b91d5fe5859c545409861 (hd0,0)/usr/local/lib/libnaaeap.so.0.7.0
0441648593e47d1b3e60f303edbee06b1c6fc070 (hd0,0)/usr/local/lib/libtncs.so.0.7.0
57657cf8b0f77d1c138507de46e7e304be57ef1d (hd0,0)/usr/local/lib/libimunit.so.0.7.0
```

```
#/etc/tnc/attestationimc.file
# Path to a x509 certificate for the AIK to use
x509_certificate_file  /home/user/PrivacyCA/AIK_Cert.der
# Path to AIK blob
aik_key_file  /home/user/PrivacyCA/AIK_KeyBlob
```

```
#/etc/tnc_config
# IMV-Configuration file of TNC@FHH-TNC-Server
# path to IMV libraries depends on where the TNC@FHH package was installed
IMV "Attestation" /usr/local/lib/libattestationimv.so.0.7.0
```

```
# /etc/strongswan.conf - strongSwan configuration file
charon {
  load=aes des sha1 sha2 md5 random x509 revocation pubkey pkcs1 pgp pem fips-prf gmp xcbc
hmac attr kernel-netlink resolve socket-default stroke updown eap-identity eap-md5 eap-tls eap-
ttls eap-tnc tnccs-20 tnc-imv tnc-imc
  multiple_authentication= no
  plugins {
    eap-tnc {
      protocol = tnccs-2.0
    }
  }
}
```

```
# /etc/ipsec.conf - strongSwan IPsec configuration file
        config setup
        plutostart=no
        charondebug="tnc 3"
```

```
conn %default
        ikelifetime=60m
        keylife=20m
        rekeymargin=3m
        keyingtries=1
        keyexchange=ikev2
conn home
        left=192.168.0.20
        leftid=carol@strongswan.org
        leftauth=eap
        leftfirewall=yes
        right=192.168.0.1
        rightid=@moon.strongswan.org
        rightsubnet=10.1.0.0/16
        #rightauth=pubkey
        #aaa_identity="C=CH, O=Linux strongSwan, CN=aaa.strongswan.org"
        auto=start
```

Likewise the Server side is configured as follows:

```
# /etc/tnc/attestationimv.policy

# PCR values of reference system
# pcrX=<20 byte SHA-1 hash as 40 characters HEX string>
pcr13=d23456a8901234567890d234567890f234567890
# Known AIKs, identified by fingerprint of X.509 AIK credential
aik=DC:30:E6:EA:F1:97:5D:90:E6:AE:D0:A3:C8:62:5C:61:93:9B:96:4B
```

```
#/etc/tnc_config
# IMC-Configuration file of TNC@FHH-TNC-Client
# path to IMC libraries depends on where the TNC@FHH package was installed
IMC "Attestation" /usr/local/lib/libattestationimc.so.0.7.0
```

```
# /etc/strongswan.conf - strongSwan configuration file
charon {
  multiple_authentication=no
  plugins {
    eap-ttls {
      phase2_method = md5
      phase2_piggyback = yes
      phase2_tnc = yes
    }
    eap-tnc {
      protocol = tnccs-2.0
    }
  }
}
```

```
# /etc/ipsec.conf - strongSwan IPsec configuration file
config setup
        strictcrlpolicy=no
        plutostart=no
        charondebug="tnc 3"
conn %default
        ikelifetime=60m
        keylife=20m
        rekeymargin=3m
        keyingtries=1
        keyexchange=ikev2
conn rw-access
        rightgroups=allow
        leftsubnet=10.1.0.0/28
        also=rw-eap
        auto=add
conn rw-isolate
        rightgroups=isolate
        leftsubnet=10.1.0.16/28
        also=rw-eap
        auto=add
conn rw-eap
        left=192.168.0.1
        leftcert=moonCert.pem
        leftid=@moon.strongswan.org
        leftauth=eap-ttls
        leftfirewall=yes
        rightauth=eap-ttls
        rightid=*@strongswan.org
        rightsendcert=never
        right=%any
```

3.4.2 Test Results

Several test scenarios were configured in order to examine the functionality of the Attestation IMV and IMC pair. As aforementioned, the testing environment consisted of TPM capable host PC that is running Ubuntu 10.10 with 2.6.38 64 bit kernel and has identity “carol” as corresponds to a strongSwan UML testing environment as well as an UML guest environment which has the strongSwan identity “moon”. The Figure below illustrates the connection between the entities in the testing environment.

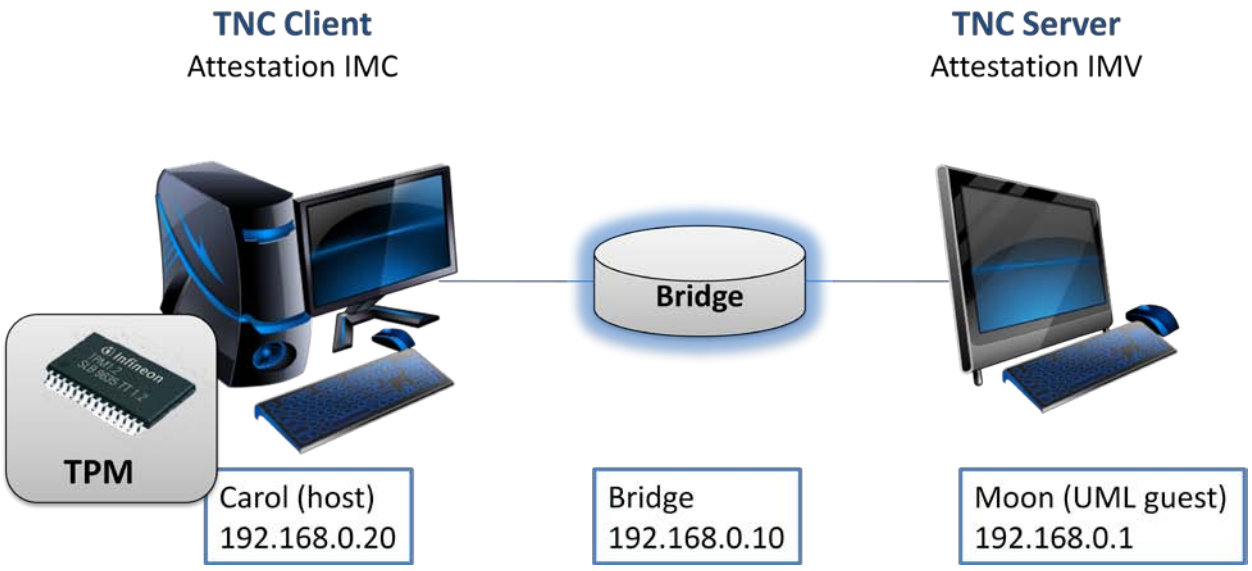

Figure 7.Testing environment for Attestation IMV/IMC

The following scenarios were tested in scope of the project:

- Successful Attestation
- Incorrect PCR value configured
- Unknown AIK certificate configured
- Invalid path to AIK certificate configured
- Invalid path to AIK key blob configured

All the scenarios delivered expected results, which in turn influences the access recommendation made by the TNC server. The strongSwan log files for each of the above scenarios are contained on the CD which is attached to the present report.

4. Dynamic Root of Trust for Measurement

4.1 Introduction

According to the definition of "Static Root of Trust for Measurement" in the TCG 1.1b specification, the platform is assumed to be in a secure state and starts in an immutable environment when it is booted (CRTM). This trust is maintained by measuring the next item in the boot chain and storing the measurement inside the TPM before executing that next item. Then any program at any point in the chain can gain confidence in its own integri ty, and the integrity of the components it relies upon, by comparing the stored values with the ones it was expecting.

The concept is problematic for several reasons:

Scalability and Inclusivity*:* The number of components which build the boot chain is indeed very large. Each component's Trusted Computing Base (TCB) and hence security, depends on many layers of code which executed earlier in the chain. Many modern systems for example Windows and Linux have an ill-defined TCB and therefore require all executable content to be measured (executables, libraries and shell scripts etc) [1]. Those components which assemble chain of trust (including TCB) are subject to frequent patching and updating with the variety of configuration possibilities. Also there are good reasons why some of the elements in the chain may be executed in different orders, which gives rise to different measurement values in PCRs [25]. Due to all these facts the maintaining of expected values for integrity measurements is a substantially troublesome task.

Time of measurement (Time-of-check-time-of-use TOCTOU problem): The machine can be booted securely into a known good state, but in the time between the measurement and the execution of the code an attacker have every chance exploit the vulnerabilities of the component. It is essential to be aware that the Static Root of Trust for Measurement only gives load-time guarantee not run-time guarantee; that is it gives assurance of what program is loaded, not necessarily what is running [1].

These shortcomings were identified by TCG and as a part of the TCG 1.2 Specifications they defined a new mechanism for authenticated boot: Dynamic Root of Trust for Measurement or DRTM [1]. The major chip vendors AMD and Intel implemented technologies that allowed the DRTM; i.e. the late launch of a measured environment. Although AMD's Secure Virtual Machine (SVM) and Intel's Trusted Execution Technology (TXT) (former code name LT – LaGrande Technology) are more than just DRTM implementation, DRTM is the core concept on which they are built.

The DRTM implementations introduce a new CPU instruction family (SKINIT for AMD and SENTER for Intel) which creates a controlled and attested execution environment. A fixed piece of code is loaded from a trustworthy source which is able to measure and launch a nominated (white-listed) piece of software (recording the measurement as before). This is then a very short chain of trust, and

by having it launched by a piece of inherently trustworthy code (untainted by anything else which has happened since the platform was rebooted), the platform can jump directly into a trustworthy state — a 'measured launched environment' [25]. That explains the "dynamic" part of the DRTM, since the launching of protected environment can happen at any time without the need of rebooting.

Intel's DRTM implementation "Trusted Execution Technology" was studied intensively during the present semester project. Therefore it is introduced in detail in the next section, followed by the introductions to applications and projects which make use of TXT. Please refer to the external sources for more detailed information on AMD's SVM technology and the comparison thereof to TXT.

4.2 Intel Trusted Execution Technology (TXT)

Intel's technology for safer computing, Intel Trusted Execution Technology (Intel TXT), defines platform-level enhancements that provide the building blocks for creating trusted platforms [3]. Initially delivered to market with the Intel VPro technology based client platforms in 2007, Intel TXT has been extended to mobile platforms as well. Intel TXT is specifically designed to harden platforms from the emerging threats of hypervisor attacks, malicious root kit installations, or other software based attacks. It increases protection by allowing greater control of launch stack through Measured Launch Environment (MLE) and enabling isolation in the boot process. More precisely, it extends the Virtual Machine Extensions (VMX) environment of Intel Virtualization Technology (Intel VT), permitting verifiably secure installation, launch and use of a hypervisor or operating system (OS) [4].

A key aspect of the protection which is provided by TXT is the provision of an isolated execution environment and associated sections of memory where operations can be conducted on sensitive data, invisible to the rest of the system. Likewise, Intel TXT provides for a sealed portion of storage where sensitive data such as encryption keys can be kept, helping to shield them from being compromised during an attack by malicious code. To make sure that code is, in fact, executing in this protected environment, attestation mechanisms verify that the system has correctly invoked Intel TXT. These capabilities complement other key features of Intel vPro processor technology, including Intel AMT and Intel VT.

Intel AMT enhances the security and central remote management of business PCs by providing a firmware-based out-of band communication channel through which a management console can reach the PC even when it is powered off or the operating system (OS) is non-functional or missing. A management engine within the PC chipset stores authentication information in non-volatile memory that it uses to pass information across the same physical network interface used by the host OS, but with its own logical identity and IP address. This mechanism allows system administrators to dramatically extend their management reach, including the ability to remotely discover hardware and software, power machines up and down, and deploy security patches and other software, regardless

of system state. Using Intel AMT, support organizations can also isolate PCs from the rest of the network if they become compromised by malware.

Intel VT allows simpler and more robust virtualization than software-only solutions by means of a new hardware layer that provides a hardware assist to virtualization. This layer reduces the complexity of the virtual machine monitor (VMM) and eliminates compute-intensive software translations in the virtualization software by enabling a new, higher privilege mode for VMM operation. This innovation directly benefits Intel TXT by reducing the overhead associated with system virtualization and allowing the guest operating system (OS) and applications to run in their intended mode. Intel VT is fully supported by leading providers of virtual machine monitor software. Intel VT offers software vendors reduced costs and risk, improved reliability and availability, enhanced security, and simpler VMM development [27].

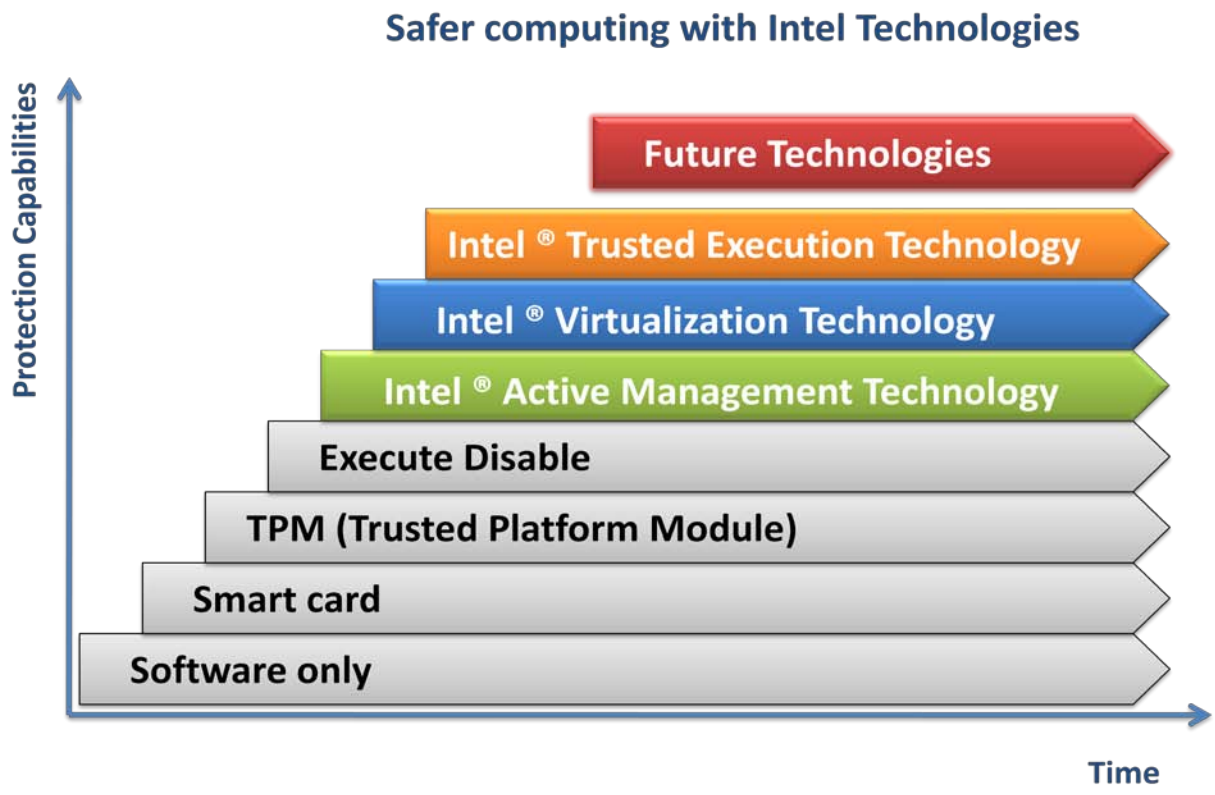

Figure 8.Safer computing with Intel Technologies [26]

4.2.1 Components

The key role of the TXT is to create a safe, isolated execution space for the software within a larger system. The software which is executed in this space is provided with the interaction protection

from unauthorized processes. Multiple instances of such execution space may exist on a single system and each has dedicated resources that are managed by processor, chipset and OS kernel. TXT architecture provides this capability with the help of number of system components' features.

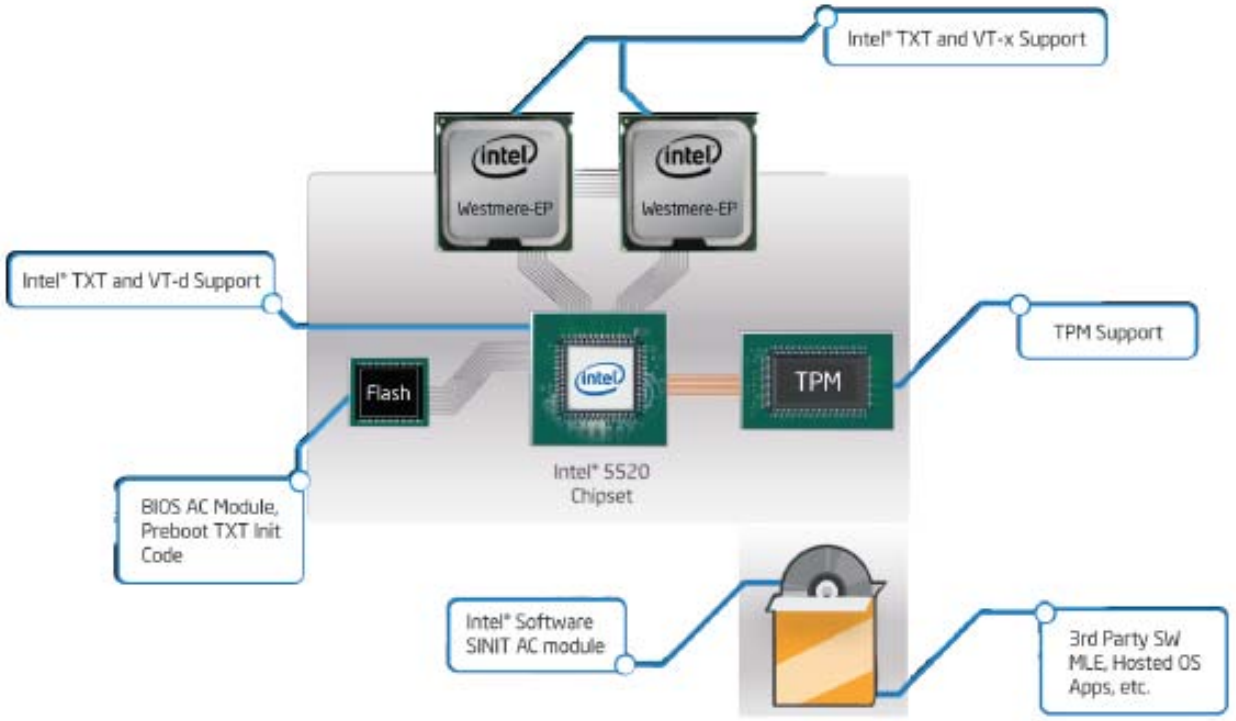

Figure 9.Intel TXT components [4]

These components include the innovations developed within Intel, as well as the third party resources:

- Trusted extensions integrated into the Intel chipsets and processors

Along with the standard partitions, extensions made to the processor allows one or more protected partitions, which provides hardened access to memory and other system resources to isolate the execution on the protected partition. Memory protection policy is enforced by the extensions made to chipset, which also provides protected channels to graphics hardware and input/output devices on behalf of the protected partition, protecting the data throughout the system. Furthermore, an interface to communicate with the TPM is provided by the chipset extensions.

- Authenticated Code Modules (ACM)

Intel has been releasing a special code module called ACM, which supports the establishment of measured environment. It is loaded into internal RAM (memory space called Authenticated Code Execution Area ACEA) within the processor. The ACM is first authenticated with the digital signature contained in its header. The processor calculates the hash of ACM and uses the result to validate the

signature. Only when the ACM is passed the authentication would it be executed. Since the authenticated code module is held within the internal RAM of the processor, execution of the module can occur in isolation with respect to the contents of external memory or activities on the external processor bus. ACMs are chipset and processor specific, and its header contains the information on which chipset and processor it supports.

- Launch Control Policy (LCP) tools

As a policy engine, LCP operates on the policy data structures that are rooted in and protected by the TPM. TPM contains the platform supplier policy from vendors and platform owner policy. These policies specify what values represent the "known good" measurements of the software component. Policy engine rules dictate that the platform owner's policy overrides the stored manufacturer policy. This allows a manufacturer to point to a MLE that is installed during production and at the same time provides opportunity for the platform owner to update and override it with their own choice of MLE. The details on LCP can be found in the document Intel TXT Software Development Guide [3].

- TPM 1.2

Cryptographic keys and the measurements made by Intel TXT are stored on TPM. The stored measurements in turn provide the evidence for attestation to demonstrate the successful invocation of the TXT environment. The storage spaces accessible within a TPM device are grouped by a locality attribute and are a separate set of address ranges from the Intel TXT Public and Private spaces [3].

The following localities are defined:

Locality 0 : Non-trusted and legacy TPM operation

Locality 1 : An environment for use by the Trusted Operating System

Locality 2 : Trusted OS

Locality 3 : Authenticated Code Module

Locality 4 : Intel TXT hardware use only.

Intel TXT DRTM is the GETSEC[SENTER] instruction and the system insures the instruction has special messages to communicate to the TPM. These special messages take advantage of TPM localities 3 and 4 to protect the messages and inform the TPM that GETSEC[SENTER] is sending the messages. Immediately after the MLE receives the control following the GETSEC[SENTER] instruction the following actions are taken on the PCRs. First of all PCRs 17-23 are reset to 0, followed by the certain measurements extended to the PCR 17 and PCR 18. In case there are additional elements are defined for measurement through LCP, those measurements are extended into PCR 19. Also TPM's Non volatile RAM provides repository for Launch Control Policies.

- Intel TXT and VT enabled BIOS

In order the TXT capabilities are available to use at all, the security options within BIOS for Intel TXT and Intel VT should be enabled. TPM is mostly configured opt-out in BIOS; therefore it also has to be enabled.

- Hypervisor or OS environment

The environment which is measured and securely launched through Intel TXT is called Measured Launch Environment (MLE). It can be arbitrary type of code but the usual case is to launch Hypervisor (Virtual Machine Monitor VMM), OS environment (Linux kernel etc) or an application that requires isolated execution.

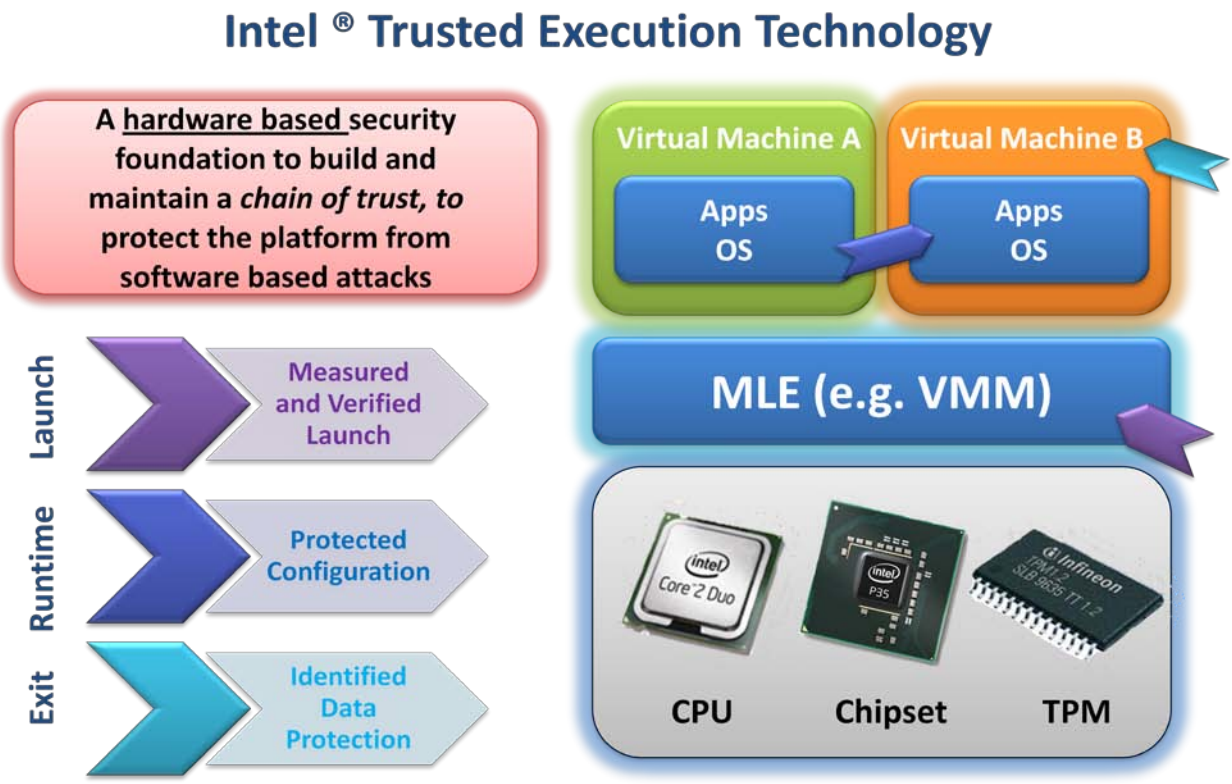

Figure 10.Intel Trusted Execution Technology [26]

4.2.2 SENTER sequence

When launching an MLE, the environment must load two code modules: MLE and ACM into memory. The launch process must successfully validate the digital signature of ACM before continuing. With the AC module and MLE in memory, the launching environment can invoke the GETSEC[SENTER] instruction provided by SMX.

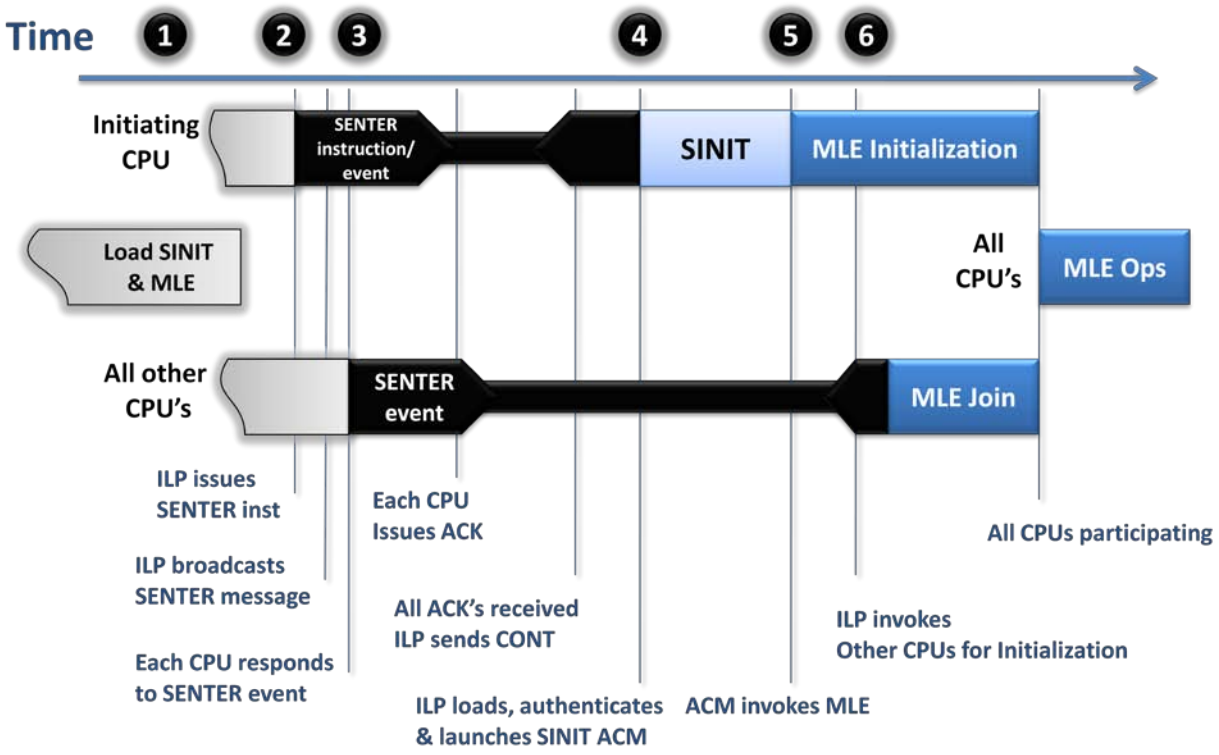

Figure 11.GETSEC[SENTER] sequence [2]

GETSEC[SENTER] broadcasts messages to the chipset and other physical or logical processors in the platform. In response, other logical processors perform basic cleanup, signal readiness to proceed, and wait for messages to join the environment created by the MLE. As this sequence requires synchronization, there is an initiating logical processor (ILP) and responding logical processor(s) (RLP(s)). After all logical processors signal their readiness to join and are in the wait state, the initiating logical processor loads, authenticates, and executes the AC module. The AC module tests for various chipset and processor configurations and ensures the platform has an acceptable configuration. It then measures and launches the MLE.

The MLE initialization routine completes system configuration changes (including redirecting INITs, SMIs, interrupts, etc.); it then issues a new SMX instruction that wakes up the responding logical processors (RLPs) and brings them into the measured environment. At this point, all logical processors and the chipset are correctly configured. At some later point, it is possible for the MLE to exit and then be launched again, without issuing a system reset [3].

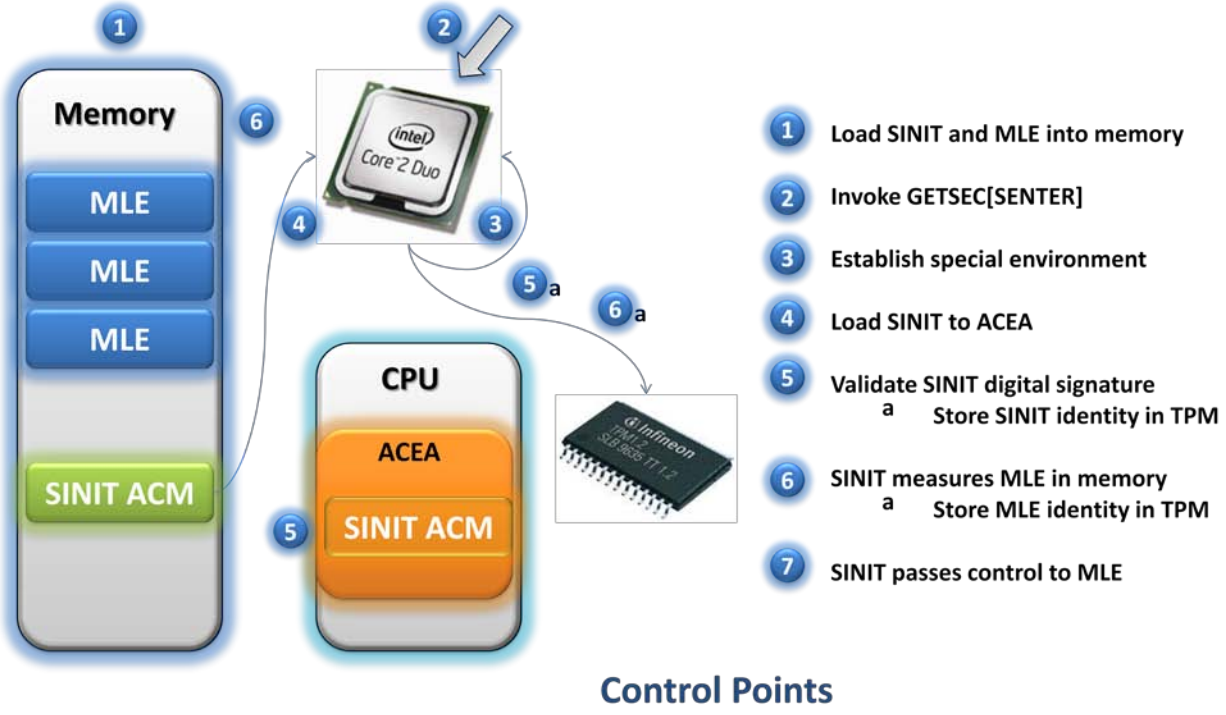

Figure 12.Launch sequence of MLE [26]

4.2.3 SEXIT sequence

The late launch premise is that the protected environment can successfully terminate without requiring a reboot of the platform. The operand for this command is GETSET [SEXIT]. The GETSEC [SEXIT] command works in the reverse of GETSEC [SENTER]. Just as GETSEC [SENTER] needs all processors to rendezvous and have a consistent view of the protections, GETSEC [SEXIT] needs all processors to rendezvous and release all protections. Having one processor still believe that memory has protection results in the exposure of protected information.

The MLE controls the issuing of GETSEC [SEXIT] and the MLE can block issuing of the command. The MLE is the only entity on the platform that can issue GETSEC [SEXIT]. Once the MLE issues the GETSEC [SEXIT] commands, the following sequence occurs. Firstly ILP validates GETSEC [SEXIT] command came from an executing MLE. After that ILP issues the SEXIT-ACK, each of the RLP responds to the message such that the chipset knows that all processors responded. After the rendezvous, the GETSEC [SENTER] starts and executes an authenticated code module. The GETSEC [SEXIT], being already in a protected environment, needs no authenticated module.

After all protected environments are shutdown; the ILP sends the SEXIT-CONTINUE message to each RLP. The RLP accepts the command, removes all protections in the platform, and moves to execute the next instruction. The next instruction was designated as the instruction to execute upon receipt of the SEXIT-ACK.

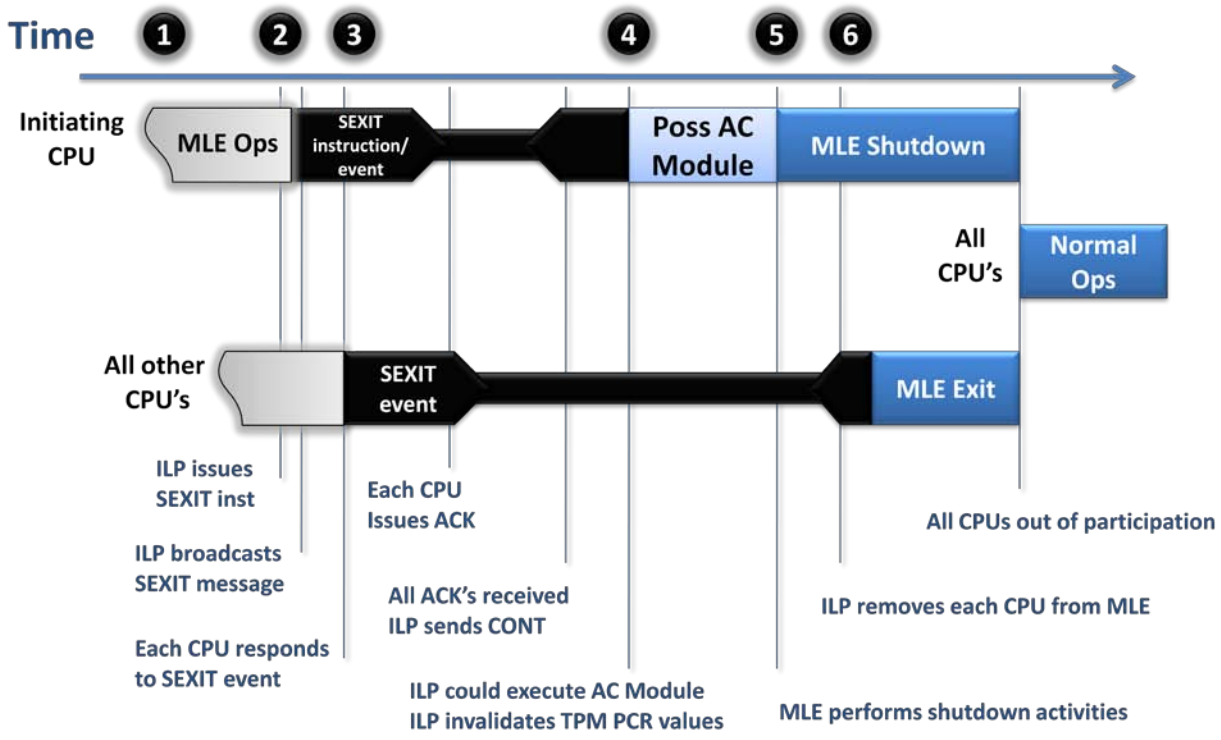

Figure 13.GETSEC[SEXIT] sequence [2]

The MLE shutdown can occur as part of most of the GETSEC leaf functions when recovery from the error would not result in a reliable state. Reliable state would include the inability to complete the GETSEC[SENTER] process [2].

4.3 tboot

By using the hardware-based root of trust that Intel TXT provides, many of these shortcomings from Static Root of Trust for Measurement can be mitigated. Specifically: many pre-launch components can be removed from the trust chain, DMA protection is provided to all launched components, a large number of platform configuration checks are performed and values locked, protection is provided for any data in the event of an improper shutdown, and there is support for policy-based execution/verification [28].

Trusted Boot (tboot) is an open source, pre- kernel/VMM module that uses Intel TXT to perform a measured and verified launch of an OS kernel/VMM. The project is developed by Intel and the most recent version at the time of this writing was released under the label tboot v.20101005. Tboot is hosted on Sourceforge under [12], where AC modules for various chipset/processor configurations can also be found with the instruction on how to find the appropriate ACM for the particular platform.

Currently tboot supports launching Xen (an open source VMM hypervisor) and Linux Kernels. According to the documentation, tboot works with 32bit, 32bit PAE, and 64bit (x86_64) Linux kernels. During the present project, we were able to test tboot successfully with Linux kernel 2.6.38, both 32 and 64 bit versions on the Ubuntu 10.10 Maverick distribution. The prerequisites for successfully running the tboot are identical with those of Intel TXT, namely enabling the TPM, Intel TXT and Intel VT in the BIOS. The Linux kernel has to be built with the kernel option CONFIG_INTEL_TXT (which should be the case for the kernels distributed with major distribution releases) enabled, in order to support the Intel TXT capabilities (disabling DMA protection put in place by TXT etc.). The option can be found under Security Top Level menu and is called “Enable Intel ® Trusted Execution Technology (TXT)”, which could possibly be marked as experimental.

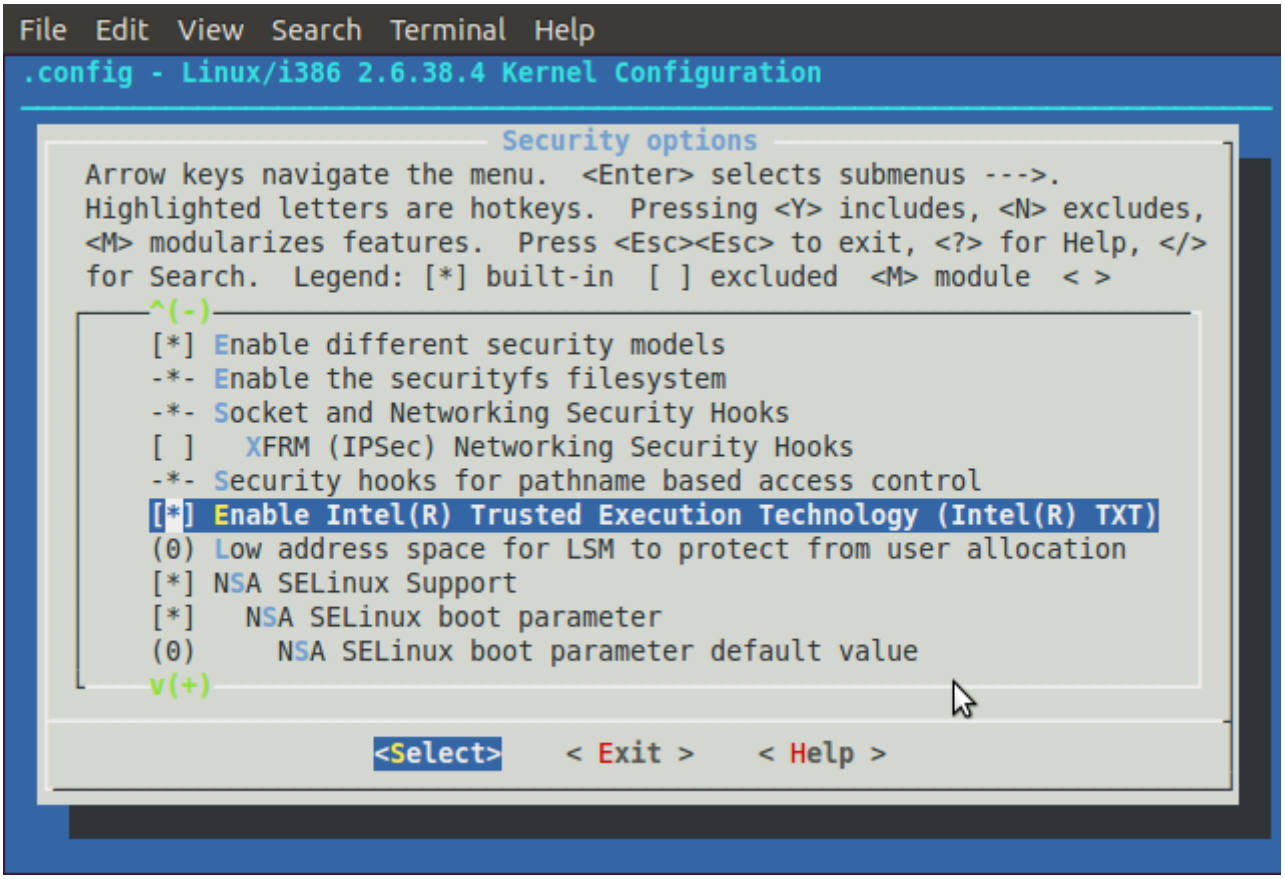

Figure 14.Kernel option to enable Intel TXT

After the installation process, tboot is an executable that can be launched by the bootloader as the kernel. We tested the tboot with both GRUB legacy and GRUB v.2 boot loaders successfully. The following is the part of the simple configuration files, with tboot launching options:

menu.lst configurations file for GRUB legacy bootloader:

```
# /boot/grub/menu.lst

title Ubuntu 10.10, kernel 2.6.38 w/ Intel(R) Trusted Execution Technology
kernel /boot/tboot.gz logging=serial,vga,memory
module /boot/vmlinuz-2.6.38-020638-generic root=UUID=154e4bc5-aa94-4e1d-9f1c-
e6c5cda3ed38 ro quiet splash
module /boot/initrd.img-2.6.38-020638-generic
module /boot/i5_i7_DUAL_SINIT_18.BIN
```

I5_i7_DUAL_SINIT_18.BIN is the file which Intel TXT refers to Authenticated Code Module (ACM), as already mentioned it is chipset and processor specific. In this case we have ACM for Core i5 and Core i7 Intel processors. tboot.gz is the binary, provided by tboot which is going be launched by bootloader in place of kernel. tboot will output various information about its progress to the terminal, serial port, and/or an in-memory log; the output locations can be configured with a command line switch. In the above configuration we had chosen to output the log on all three channels.

grub.cfg configurations file for GRUB v.2 bootloader:

```
# /boot/grub/grub.cfg

menuentry 'Ubuntu, with Linux 2.6.38.4 32 bit with tboot' --class ubuntu --class gnu-linux --class
gnu --class os {
        recordfail
        insmod part_msdos
        insmod ext2
        set root='(hd0,msdos5)'
        search --no-floppy --fs-uuid --set fda690a0-99f0-4247-b21e-fb717d317faf
        multiboot       /boot/tboot.gz tboot.gz logging=serial,vga,memory
        module          /boot/vmlinuz-2.6.38.4 vmlinuz-2.6.38.4 root=UUID=fda690a0-99f0-
4247-b21e-fb717d317faf ro   crashkernel=384M-2G:64M,2G-:128M quiet splash
        module          /boot/initrd.img-2.6.38.4 initrd.img-2.6.38.4
        module          /boot/i5_i7_DUAL_SINIT_18.BIN i5_i7_DUAL_SINIT_18.BIN
        module /home/user /tboot-20101005/lcptools/list.data list.data
}
```

The difference to be noticed above is the repetition of module names as well as an additional line for the Launch Control Policy data configuration. The tboot distribution contains LCP tools and the simple instructions on how to use them. We will come to this point later in this section.

Log outputs of successful booting process using the above configuration file (with and without LCP) can be found in the CD attached to this paper.

4.3.1 Working sequence [28]

First of all tboot performs all of the work necessary to determine if the platform supports Intel TXT and, if so, executes the GETSEC[SENTER] processor instruction that initiates the DRTM. If tboot determines that the system does not support Intel TXT or is not configured correctly (e.g. the SINIT AC Module was incorrect), it will directly launch the kernel with no changes to any state.

The GETSEC[SENTER] instruction will return control to tboot and tboot then verifies certain aspects of the environment (e.g. TPM NV lock, e820 table does not have invalid entries, etc.).

It will wake the Responding Logical Processors from the special sleep state the GETSEC[SENTER] instruction had put them in and place them into a wait-for Start Inter Processor Interrupt (SIPI) state. Because the processors will not respond to an INIT or SIPI when in the TXT environment, it is necessary to create a small VT-x guest for the APs. When they run in this guest, they will simply wait for the INIT-SIPI-SIPI sequence, which will cause VMEXITs, and then disable VT and jump to the SIPI vector. This approach seemed like a better choice than having to insert special code into the kernel's MP wakeup sequence.

Tboot then applies an (optional) user-defined launch policy to verify the kernel and initrd. This policy is rooted in TPM NV and is described in the tboot project. The tboot project also contains code for tools to create and provision the policy. Policies are completely under user control and if not present then any kernel will be launched. Policy action is flexible and can include halting on failures or simply logging them and continuing. For more information on LCP please refer to documentation released by Intel [3].

Tboot reserves its own location and certain other TXT-related regions in memory. As part of its launch, tboot DMA protects all of RAM (using the VT-d PMRs). Thus, the kernel must be booted with 'intel_iommu=on' option in order to remove this blanket protection later when it’s necessary. Tboot will populate a shared page with some data about itself and pass this to the Linux kernel as it transfers control. The location of the shared page is passed via the boot_params struct as a physical address. The kernel will look for the tboot shared page address and, if it exists, map it.

As one of the checks/protections provided by TXT, it makes a copy of the VT-d Direct Memory Access Remappings in a DMA-protected region of memory and verifies them for correctness. The VT-d code will detect if the kernel was launched with tboot and use this copy instead of the one in the Advanced Configuration and Power Interface (ACPI) table, which describes the computer’s hardware configuration and power management capabilities.

At this point, tboot and TXT are out of the picture until a shutdown. In order to put a system into any of the sleep states after a TXT launch, TXT must first be exited. This is to prevent attacks that attempt to crash the system to gain control on reboot and steal data left in memory.The kernel will

perform all of its sleep preparation and populate the shared page with the ACPI data needed to put the platform in the desired sleep state. Then the kernel jumps into tboot via the vector specified in the shared page.

Tboot will clean up the environment and disable TXT, and then use the kernel-provided ACPI information to actually place the platform into the desired sleep state.

4.4 Flicker

Flicker is an infrastructure for a Secure Execution Architecture (SEA) that executes security-sensitive code of an application while trusting only the mandatory Trusted Code Base (TCB) and TPM. It has been developed by Jonathan McCune et al at Carnegie Mellon University, Pennsylvania, USA. For the present project we tested flicker version 0.2 with a Intel platform capable of TXT. Shortly before the present document was drafted, the new version 0.5 of Flicker was released publicly, and the noticeable change was that it supports both Linux OS with non-PAE 32 bit kernel and Windows.

The original plan for the semester project was to develop a simple application and execute it with the Flicker, but since we didn't succeed in running Flicker v.0.2 with its Hello World application on Linux 64 bit 2.6.38 kernel, we could not reach the initial aim. Thus, Flicker and its operation are introduced in this section, without any additional experience reports. However the 64 bit compiled version of the Flicker v0.2 as well as patches for each modified source files can be found on the CD, which is reached out with this paper.

Flicker executes the application in isolation from all other software and hardware on the system (including OS). The isolation is provided with the hardware enhancements in commodity platforms from Intel and AMD, namely AMD's Virtual Machine Technology and Intel's Trusted Execution Technology [6]. The prerequisites to run Flicker are the same as for tboot when the Intel hardware arrangement is used.

The architecture of modern computing platforms follows the layered model where the applications form the highest layer and the hardware the fundamentals. Hence, the Trusted Computing Base (TCB) of each application contains many layers of code, including the system firmware, firmware of various peripheral devices the bootloader, OS kernel and application's own code. The trend towards feature-rich platforms that support diversity of hardware configurations, increases the size and the complexity of each layer tremendously. Thus enormous time and computing resources would be wasted if the task of securing an application is extended to trusting its complete TCB, especially when the additional effort is not necessarily indispensable. When isolation from the external sources is provided, the TCB of an arbitrary application can be minimized. In the

Figure 13, the TCB minimization obtained by Flicker is illustrated. It is foreseen that only security-sensitive part of an application is executed in isolated environment.

Flicker provides strong isolation guarantees while adding less than 250 lines of code to the application's TCB, compared to the thousands of lines of code for today's common VMM's or OS's. Consequently, Flicker circumvents entire layers of legacy system software and eliminates reliance on their integrity for security protection [7].

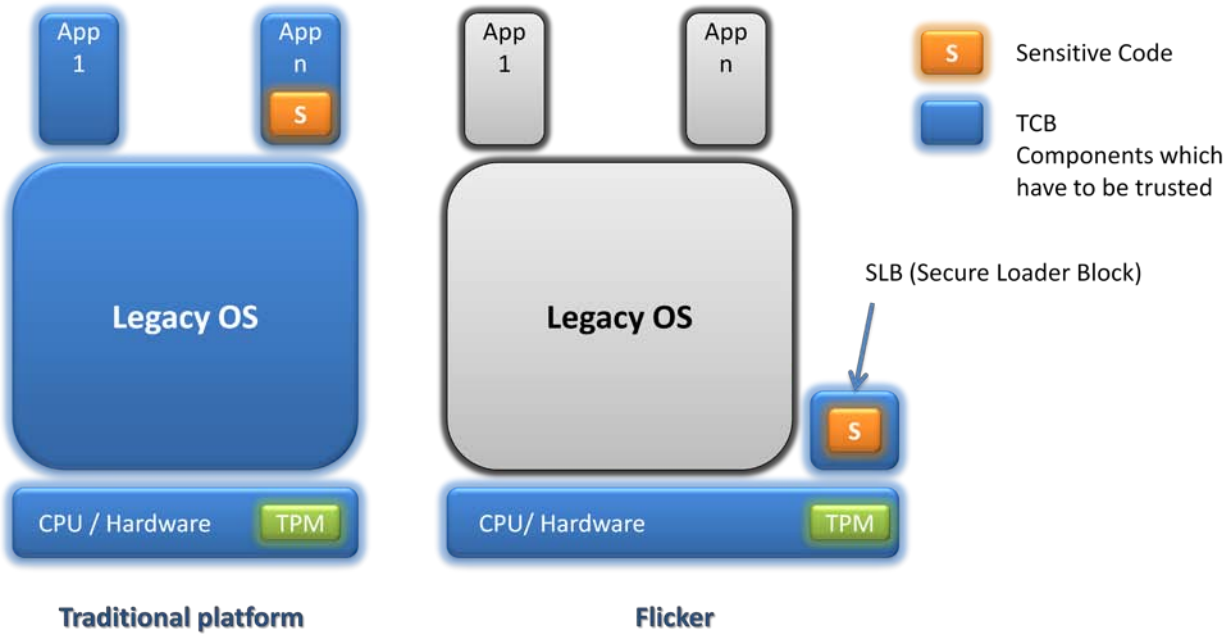

Figure 15.TCB of an application: Traditional platform vs. Flicker [5]

None of the software which was executing before Flicker begins can observe or intervene with the Flicker process and all traces of Flicker code execution can be eliminated before regular execution resumes. The use of Flicker as well as exact code executed with its input and output can be attested to an external party.

Within the scope of the Flicker project, numbers of research papers have been released. The detailed information on how to alter the Flicker architecture and modify the hardware in order to improve performance issues on SVM and TXT architecture [5]. The Flicker developers also have provided small code modules to extract sensitive operations and relative code from an application; or to protect the existing execution environment from malicious PALs.

4.4.1 Working sequence [7]

The rough execution sequence of the Flicker pursues the following operations: Flicker first pauses the current execution environment and executes small piece of code using SENTER instruction and

then resumes operation of the previous execution environment. The security sensitive code selected for the Flicker protection is called Piece of Application Logic (PAL). The Figure.14 illustrates this Flicker execution sequence.

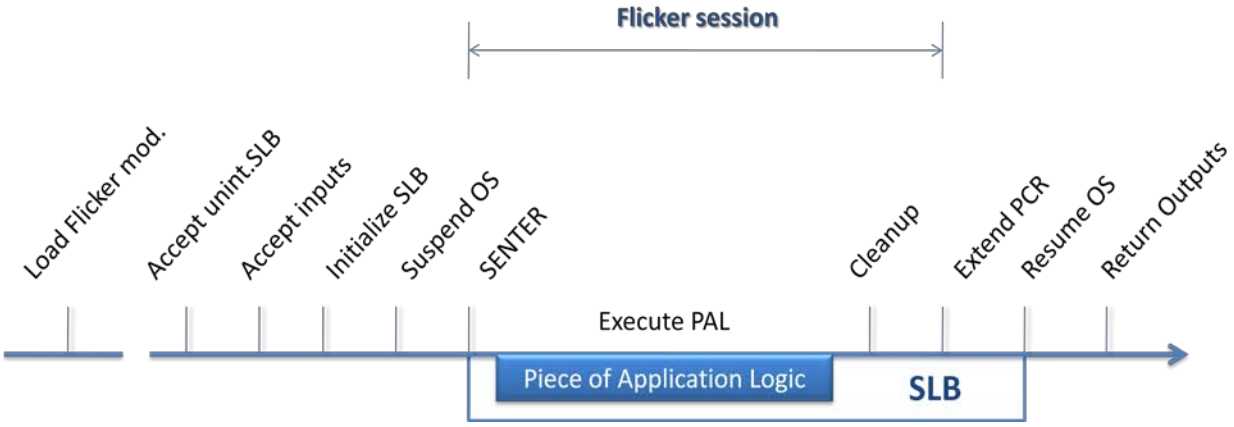

Figure 16.Timeline of Flicker execution sequence [7]

First of all Flicker's implementation for Linux OS's: sysfs kernel module called flicker-module is loaded. Sysfs is a virtual file system that exposes the kernel state. The applications interact with the flicker-module via four file system entries; ***control*** for signal the state change of Flicker session, ***slb*** to allocate memory for PAL, ***inputs*** to write the input to the PAL, ***output*** to export the outputs from PAL.

Then flicker-module has to initialize several entries (among others the starting address of PAL) before the PAL is executed. After that the all CPU's except the ILP has to receive INIT Inter Processor Interrupt so that they correspond correctly during the execution of SENTER. However it is not possible if the other AP's are executing processes. Flicker uses the CPU hotplug support available in Linux kernels from starting from version 2.6.19, to deschedule all AP's. At this point the OS state needs to be saved; in particular Flicker saves information about the Linux Kernel's page table, so that the OS can be restored after the PAL execution.

The SENTER instructions enable the hardware protections made available by Intel TXT, and once the environment has been prepared PAL executes its application-specific logic. During the PAL execution, the output parameters are written. The PAL's exit triggers the cleanup where it erases any sensitive data left in memory by the PAL. To signal the completion of PAL execution, a well known value is extended to PCR 17, so that the remote party can distinguish between the values generated by PAL (trusted) and those produced after the OS is resumed (untrusted).

The flicker-module restores the execution state saved during Suspend OS phase and fully restores control to the Linux kernel by re-enabling interrupts.

5. Conclusion

There are a number of techniques to obtain verifiable trust on executing environment on a platform. The approach specified by TCG, “Static Root of Trust for Measurement” is well defined and found a number of applications so far. However the obvious shortcomings of the SRTM method have been hindering the common use of it and triggered the need for better solutions. The answer to the problem from the vendors Intel and AMD are made available through the implementation of “Dynamic Root of Trust for Measurement” which is based on enhancements to the existing hardware and a new CPU instruction set.

In this semester project, both the SRTM and DRTM approaches were explored closely as reported in the corresponding sections of the present paper. We faced a number of problems and challenges during the testing of applications of the concepts, as can be seen in the Weekly Status Reports. In the completion phase of the project, the compiling and running of Flicker on the Linux Kernel version 2.6.38 (32 bit) has turned out to be a rather challenging task, which could not be finished despite the invested time and effort. As aforementioned, a new version 0.5 of Flicker was released shortly before the documentation draft, which allegedly supports non-PAE 32 bit Linux kernels, independent of its version. Therefore an immediate task in the future would be to test the new version of Flicker and investigate on the implementation of an example PAL, preferably an application related with the TNC architecture so that the “Lying endpoint” problem can be mitigated.

Acronyms

AC module – Authenticated Code Module

AIK – Attestation Identity Key

DMA – Direct Memory Access

DRTM – Dynamic Root of Trust for Measurement

ILP – Initiating Logical Processor

IMC – Integrity Measurement Collector

IMV – Integrity Measurement Verifier

LCP – Launch Control Policy

LT – LaGrande Technology (former name of TXT)

MLE – Measured Launched Environment

NEA – Network Endpoint Assessment

PAL – Piece of Application Logic

PB – Posture Broker

PCR – Platform Configuration Register

SIPI – Start Inter Processor Interrupt

RLP – Responding Logical Processor

SRK – Storage Root Key

SRTM – Static Root of Trust for Measurement

TBB – Trusted Building Block

TCB – Trusted Computing Base

TCG – Trusted Computing Group

TNC – Trusted Network Connect

TPM – Trusted Platform Module

TSS – Trusted Computing Group Software Stack

TXT – Trusted eXecution Technology

UML –User Mode Linux

Glossary

Attestation (of the platform) – Process of proving to an outside verifier that the platform is in a certain state by providing the measurement of its current state and a proof that the measurements genuinely reflect the present state of the system.

Integrity – The term is used to denote the pristine state of a component within a trusted platform.

Integrity Measurements - The process of obtaining metrics of platform characteristics that affect the integrity (trustworthiness) of a platform, and putting digests of those metrics in shielded locations (called Platform Configuration Registers: PCRs).

Chain of Trust – Result of the sequential loading process in which the involved components always measure the integrity of its successor entity in a chain before transferring the control to it.

Privacy CA - An entity, typically a Trusted Third Party (TTP) that blinds a verifier to a platform's EK. An entity (typically well known and recognized) trusted by both the Owner and the Verifier, that will issue AIK Credentials. A Verifier may also adopt the role of a Privacy CA. In that case the roles are co-located but are logically distinct.

Trusted Building Block -The parts of the Root of Trust that do not have shielded locations or protected capabilities. Normally includes just the instructions for the RTM and the TPM initialization functions (reset, etc.). It is typically platform-specific. One example of a TBB is the combination of the CRTM, connection of the CRTM storage to a motherboard, the connection of the TPM to a motherboard, and mechanisms for determining Physical Presence.

Table of Figures

Table of Source Code References

Bibliography

[1] David Challener, Kent Yoder, Ryan Catherman, David Safford and Leendert Van Doorn, “A Practical Guide to Trusted Computing” , IBM Press ISBN 0-13-239842-7 , 2008

[2] David Grawrock, “The Intel Safer Computing Initiative, Building Blocks for Trusted Computing”, Intel Press ISBN 0-9764832-6-2, 2005

[3] Intel ® Trusted Execution Technology (Intel ® TXT) Software Development Guide, Measured Launched Environment Developer’s Guide, http://download.intel.com/technology/security/downloads/315168.pdf , December 2009

[4] Intel ® Trusted Execution Technology White Paper , Hardware Based Technology for Enhancing Server Platform Security, http://www.intel.com/Assets/en_US/PDF/whitepaper/323586.pdf , 2010

[5] McCune, Jonathan M., Bryan Parno, Adrian Perrig, Michael K. Reiter, and Arvind Seshadri. "Minimal TCB Code Execution (Extended Abstract)." In Proceedings of the IEEE Symposium on Security and Privacy, Oakland, California, May 2007

[6] McCune, Jonathan M., Bryan Parno, Adrian Perrig, Michael K. Reiter, and Arvind Seshadri. "How Low Can You Go? Recommendations for Hardware-Supported Minimal TCB Code Execution." In Proceedings of the ACM Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'08), Seattle, Washington, March 1 - 5, 2008

[7] McCune, Jonathan M., Bryan Parno, Adrian Perrig, Michael K. Reiter, and Hiroshi Isozaki. "Flicker: An Execution Infrastructure for TCB Minimization". In Proceedings of the ACM European Conference on Computer Systems (EuroSys'08), Glasgow, Scotland, March 31 - April 4, 2008.

[8] TrouSerS v0.3.6, The open-source TCG Software Stack, http://trousers.sourceforge.net/

[9] TrustedGRUB v1.1.5, Extended GRUB bootloader with TCG support, http://trousers.sourceforge.net/

[10] TPM Manager v0.8, The TPM management grough Graphical User Interface, http://sourceforge.net/projects/tpmmanager/

[11] TNC@FHH v0.7.0, An open source implementation of the TNC architecture, http://trust.inform.fh-hannover.de/joomla/index.php/projects/tncfhh/48-tncatfhh

[12] Trusted Boot (tboot) v20101005, an open source, pre- kernel/VMM module that uses Intel(R) Trusted Execution Technology (Intel(R) TXT) to perform a measured and verified launch of an OS kernel/VMM, http://sourceforge.net/projects/tboot/

[13] Flicker v0.2, Minimal TCB code execution, http://sparrow.ece.cmu.edu/group/flicker.html

[14] PrivacyCA 07 Dec 2009, provides functionality intended to facilitate the use of Trusted Computing technology, http://www.privacyca.com/index.html

[15] strongSwan v4.5.1, The Open source IPSec based VPN solution for Linux, http://www.strongswan.org/index.htm

[16] Trusted Computing Group, TPM Main Specifications, http://www.trustedcomputinggroup.org/resources/tpm_main_specification

[17] Marcel Selhorst, Christian Stüble, Felix Teerkorn Sirrix AG, TSS Study, Introduction and Analysis of the Open Source TCG Software Stack TrouSerS and Tools in its Environment, Study on behalf of the German Federal Office for Information Security (BSI), https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/TSS_Apps/TSS-Apps_en.pdf?__blob=publicationFile

[18] Trusted Computing Group, http://www.trustedcomputinggroup.org/

[19] Sansar Choinyambuu, A Posture Broker protocol, compatible with Trusted Network Connect, Hochschule Für Technik Rapperswil, Jan 2011

[20] P.Sangster, H. Khosravi, M. Mani, K. Narayan, J. Tardo , RFC 5209 „Network Endpoint Assessment (NEA): Overview and Requirements“ , IETF Network Working Group, June 2008

[21] R. Sahita, S. Hanna, R. Hurst, K. Narayan, RFC 5793 “PB-TNC: A Posture Broker (PB) Protocol Compatible with Trusted Network Connect (TNC)” ISSN: 2070-1721 , IETF, March 2010

[22] Richard Stallman, Can you trust your computer? http://www.gnu.org/philosophy/can-you-trust.html, 2002

[23] MSDN, Microsoft, ProtectKeyWithTPMAndPINAndStartupKey Method http://msdn.microsoft.com/en-us/library/bb931362(v=vs.85).aspx , as of 27 Apr, 2011

[24] Das Bundesamt für Sicherheit in der Informationstechnik (BSI, Der Core Root of Trust for Measurement, https://www.bsi.bund.de/ContentBSI/Themen/Sichere_Plattformen/trustedcomputing/TrustedPlatformModul/TrustedPlatformModul/CRTM.html, as of 23, May, 2011

[25] Andrew Martin, The ten-page introduction to Trusted Computing, https://www.cs.ox.ac.uk/files/1873/RR-08-11.PDF, University of Oxford, Nov 2008

[26] David Grawrock, The front door of Trusted Computing (slides), http://cache-www.intel.com/cd/00/00/40/73/407337_407337.pdf, Intel Corporation, 2008

[27] Matthew Gillespie, Intel® Trusted Execution Technology: A Primer, http://software.intel.com/en-us/articles/intel-trusted-execution-technology-a-primer/, Intel Software Network, June 1, 2009

[28] Linux Kernel Documentation, Intel TXT (on kernel version 2.6.38), http://www.mjmwired.net/kernel/Documentation/intel_txt.txt, March 22, 2011

[29] TPM Quote tools, Trousers project's mail tracker, http://sourceforge.net/tracker/?func=detail&atid=704361&aid=3161939&group_id=126012, as of June 13, 2011

Weekly status reports

Calendar week 8

Accomplishments of the week

- Discussion on the Project A report
- Planning of the first 3 weeks of Project B
- Setup of the working environment
- Setup of UML testing environment
- Study of Static Root of Trust

Problems

Installing 32 bit UML testing environment on 64 bit host system introduced some challenge.

Calendar week 9

Accomplishments of the week

- TPM related installations (Trousers, tpm-tools, tpm_manager)
- Study of Attestation IMV/IMC from TNC@FHH
- Study of TSS Study from Bundesamt für Sicherheit
- Finishing of UML testing environment
- Test IMV/IMC skeleton tested
- Installation of networking between host and uml systems
- Testing Host system as Collector (with real TPM) and Virtual UML system as Verifier

Problems

None

Calendar week 10

Accomplishments of the week

- Studying of Attestation IMV/IMC from TNC@FHH
- Studying the TSS API
- Generating AIK key and certificate using identity.c from PrivacyCA.com
- Tested TrustedGrub's checkfile functionality with Attestation IMC on host and IMV on UML guest

Problems

Slightly modified version of identity.c was used in order to generate the AIK key and certificate. TSS_HPOLICY object for TPM owner secret is set as 20 bytes of zero (well known secret).

Calendar week 11

Accomplishments of the week

- Written experimental program that extends PCR of TPM
- Studying of Dynamic Root of Trust
- Research on tboot project
- Research on Intel TXT – Software Development Guide and Preliminary Architecture Specification
- Research on the main board with compatible chipset for SINIT AC Module from tboot project

Problems

The hardware we possessed at this time was Fujitsu Tower Celsius w380 with Core i5 660 processor and Intel 3450 chipset. There is no SINIT AC module released from Intel for this configuration, therefore new, compatible hardware had to be obtained.

Calendar week 12

Accomplishments of the week

- Book study – Intel's Safer Computing Initiative by David Grawrock
- Reading of Software Developers Guide for Intel TXT
- Ordering of the Intel Desktop Board DQ57TM, Intel processor Core i5 660 and case

Problems

None

Calendar week 13

Accomplishments of the week

- Built up the new hardware
- Updated the Linux Kernel from 2.6.35 to 2.6.38
- Configuring tboot through GRUB2
- Booted the Linux Kernel with tboot using Intel TXT with Launch Control and Verified Launch policies

Problems

Kernel update was necessary because the tboot hung at the phase where the control is transferred to linux kernel.

Calendar week 14

Accomplishments of the week

- Explored the Flicker project, studied the papers and slides
- Installed older versions of Ubuntu and 2.6.30 kernel in order the Flicker to work
- Installed 32 bit version of Ubuntu 10.10 in order to compile Flicker on 2.6.38 kernel
- Start the Documentation

Problems

Flicker v0.2 supports Linux kernel 2.6.30 on Intel systems. Thus the Ubuntu version and Linux kernel used had to be downgraded. However the fact that the hardware set that we obtained was shipped later than the Ubuntu versions compatible with the 2.6.30 kernel caused problems. Therefore experimenting to compile Flicker on the latest linux kernel (2.6.38 as of the time of writing) began.

Calendar week 15

Accomplishments of the week

- Explored the Flicker project further
- Installed Ubuntu 10.10 distribution with 32 bit configuration with 32 bit Linux Kernel 2.6.38

Problems

Tboot is set as a prerequisite in order the Flicker to work on a platform. Since we had problems tboot running on the 32 bit configuration, the decision was made to port the Flicker to 64 bit so that it will work on a 64 bit configuration, on which we succeeded to boot with tboot in the earlier phase of a project.

Calendar week 16

Accomplishments of the week

- Explored the Flicker project further
- Ported the Flicker on the 64 bit 2.6.38 Kernel, by adjusting the following
  - ✓ assembly keywords(popl → popq etc.)
  - ✓ 64 bit registers (eax → rax etc.)
  - ✓ virtual memory addressing issue
  - ✓ page table

Problems

As of now, the Flicker could be successfully compiled under 2.6.38 64 bit kernel. However at the run-time the PC is restarted, the following week has to be devoted for debugging this issue.

Calendar week 17

Accomplishments of the week

- Explored the Flicker project further
- Debugging the root cause for the unexpected restart.

Problems

Could get the Kernel log only putting some sleep for current process (ssleep) before the GETSEC[SENTER] is executed, so that Kernel can dump the log before the restart.

Calendar week 18

Accomplishments of the week

- The cause for unexpected restart was found by reading the Intel TXT Error register after the restart.

Problems

The Error Code was : 0xc0002051, which has a following meaning according to the attached sinit_errors.txt in i5_i7_DUAL-SINIT Authenticated Code Module from Intel:

AC module error : acm_type=0x1, progress=0x05, error=0x8

01000 OsSinitDataSize incorrect

Data structure called os_sinit_data_t of size 92 bits (v.4) or 100 bits (v.5) [3] was aligned to 8 bits on 64 bit kernel correspondingly to 96 or 104 bits. Therefore the padding of 4 bits always counted as additional to the OsSinitDataSize, which leads to the TXT error and thus explains the unexpected restart.

Calendar week 19

Accomplishments of the week

- The work on Flicker with Ubuntu 10.10 32 bit distribution on 32 bit Linux Kernel 2.6.38 is restarted again.
- The reason of tboot failure on this configuration is explored

Problems

None

Calendar week 20

Accomplishments of the week

- Found out the reason of tboot failure on 2.6.38 Kernel. Kernel was not disabling the DMA protection put in place by tboot. Intel_iommu=on was supposed to be specified on Kernel command line. However it is to be enabled automatically by kernel if the latter is built with CONFIG_INTEL_TXT=y.
- Built the 2.6.38 32 bit kernel with the security option CONFIG_INTEL_TXT=y
- Succeeded in booting with the tboot on 2.6.38/32 bit kernel.

Problems

Regardless of the success of booting with tboot on 32 bit configuration, the Flicker didn't run successfully. We got freeze at the execution of GETSEC[SENTER] in smx.h. Conclusion was drawn as for the result of project that the experiences and knowledge collected during the project have to be documented.

Calendar week 21

Accomplishments of the week

- Work on the project report.

Problems

None

Calendar week 22

Accomplishments of the week

- Work on the project report. Releasing of first draft.

Problems

None

Calendar week 23

Accomplishments of the week

- Work on the project report.
- Final preparation for the Documentation.

Problems

None

Appendix A: Contents of CD

Attestation IMVC

- Configurations – Configuration files for Attestation IMC/IMV
- Daemon.log – strongSwan log files for each of the testing scenarios
- strongSwan configurations – strongSwan configuration files
- TrustedGRUB – TrustedGRUB bootloader configuration files

Patches

- Flicker 0.2 – patches for each of the modified source files for the compilation on 64 bit kernel
- PrivacyCA – patch for identity.c

Tboot logs

- Log files obtained from tboot tool ***txt-stat*** for both case with and without Launch Control Policy

Flicker-0.2.64bit.tar.gz – tarball of flicker v.0.2 compiled on 64 bit 2.6.38 kernel.

RTM-ReportProjectB_SansarChoinyambuu.docx – Semester project report

RTM-ReportProjectB_SansarChoinyambuu.pdf – Semester project report

Appendix B: diff identity.c

```
diff --git a/identity.c b/identity_mod.c
index 68532fa..b691300 100644
--- a/identity.c
+++ b/identity_mod.c
@@ -8,6 +8,7 @@
 /* Fri Oct  3 11:59:04 PDT 2008 - Add Pragma: no-cache to POST */
 /* Mon Dec  7 14:19:07 PST 2009 - Add MIT copyright notice */
 /* Mon Dec  7 14:19:07 PST 2009 - Output AIK as a blob rather than in database */
+/* Wed Mar 16 10:20:05 PST 2011 - Sansar Choinyambuu: Get owner secret from well-know, defined string */

 /*
  * Copyright (c) 2008 Hal Finney
@@ -49,8 +50,6 @@
 #define CAURL                   "http://www.privacyca.com/"
 #define CERTURL                 CAURL "api/pca/level%d?ResponseFormat=PEM"
 #define REQURL                  CAURL "api/pca/level%d?ResponseFormat=Binary"
-/* Prompt for TPM popup */
-#define POPUPSTRING             "TPM owner secret"


 #ifndef REALEK
@@ -73,11 +72,13 @@ makeEKCert(TSS_HCONTEXT hContext, TSS_HTPM hTPM, UINT32 *pCertLen, BYTE **pCert)
         result = Tspi_TPM_GetPubEndorsementKey (hTPM, TRUE, NULL, &hPubek);
         if (result != TSS_SUCCESS)
                 return result;
+
         result = Tspi_GetAttribData (hPubek, TSS_TSPATTRIB_RSAKEY_INFO,
                 TSS_TSPATTRIB_KEYINFO_RSA_MODULUS, &modulusLen, &modulus);
         Tspi_Context_CloseObject (hContext, hPubek);
         if (result != TSS_SUCCESS)
                 return result;
+
         if (modulusLen != 256) {
                 Tspi_Context_FreeMemory (hContext, modulus);
                 return TSS_E_FAIL;
@@ -260,29 +261,51 @@ main (int argc, char **argv)
                 printf ("Error 0x%x on Tspi_Policy_SetSecret for SRK\n", result);
                 exit(result);
         }
```

```
-	result = Tspi_Context_CreateObject(hContext, TSS_OBJECT_TYPE_POLICY,
-			TSS_POLICY_USAGE, &hTPMPolicy);
-	if (result != TSS_SUCCESS) {
-		printf ("Error 0x%x on Tspi_CreateObject for TPM policy\n", result);
-		exit(result);
-	}
-	result = Tspi_Policy_AssignToObject(hTPMPolicy, hTPM);
-	if (result != TSS_SUCCESS) {
-		printf ("Error 0x%x on Tspi_Policy_AssignToObject for TPM\n", result);
-		exit(result);
-	}
-	popupString = (BYTE *)Trspi_Native_To_UNICODE((BYTE *)POPUPSTRING, &popupLen);
-	result = Tspi_SetAttribData(hTPMPolicy, TSS_TSPATTRIB_POLICY_POPUPSTRING, 0,
-		popupLen, popupString);
+
+	/* Sansar Choinyambuu: Please see Changelog from 16.Mar.2011 */
+	/**
+	 *
+	 *result = Tspi_Context_CreateObject(hContext, TSS_OBJECT_TYPE_POLICY,
+	 *		TSS_POLICY_USAGE, &hTPMPolicy);
+	 *if (result != TSS_SUCCESS) {
+	 *	printf ("Error 0x%x on Tspi_CreateObject for TPM policy\n", result);
+	 *	exit(result);
+	 *}
+	 *result = Tspi_Policy_AssignToObject(hTPMPolicy, hTPM);
+	 *if (result != TSS_SUCCESS) {
+	 *	printf ("Error 0x%x on Tspi_Policy_AssignToObject for TPM\n", result);
+	 *	exit(result);
+	 *}
+	 *popupString = (BYTE *)Trspi_Native_To_UNICODE((BYTE *)POPUPSTRING, &popupLen);
+	 *result = Tspi_SetAttribData(hTPMPolicy, TSS_TSPATTRIB_POLICY_POPUPSTRING, 0,
+	 *	popupLen, popupString);
+	 *if (result != TSS_SUCCESS) {
+	 *	printf ("Error 0x%x on Tspi_SetAttribData for TPM password prompt\n", result);
+	 *	exit(result);
+	 *}
+	 *result = Tspi_Policy_SetSecret(hTPMPolicy, TSS_SECRET_MODE_POPUP, 0, NULL);
+	 *if (result != TSS_SUCCESS) {
+	 *	printf ("Error 0x%x on Tspi_Policy_SetSecret for TPM\n", result);
+	 *	exit(result);
+	 *}
+	 */
+
+
+
```

```
+       /* Begin of additional code, Sansar Choinyambuu: Please see Changelog from 16.Mar.2011 */
+
+       result = Tspi_GetPolicyObject(hTPM, TSS_POLICY_USAGE, &hTPMPolicy);
        if (result != TSS_SUCCESS) {
-               printf ("Error 0x%x on Tspi_SetAttribData for TPM password prompt\n", result);
+               printf ("Error 0x%x on Tspi_GetPolicyObject for TPM\n", result);
                exit(result);
        }
-       result = Tspi_Policy_SetSecret(hTPMPolicy, TSS_SECRET_MODE_POPUP, 0, NULL);
+       result = Tspi_Policy_SetSecret(hTPMPolicy, TSS_SECRET_MODE_SHA1, 20, secret);
        if (result != TSS_SUCCESS) {
                printf ("Error 0x%x on Tspi_Policy_SetSecret for TPM\n", result);
                exit(result);
        }
+       /* End of additional code, Sansar Choinyambuu: Please see Changelog from 16.Mar.2011 */
+

        if (aikpass)
                initFlags |= TSS_KEY_AUTHORIZATION;
```

Project plan