Identity Credential Issuance with Trusted Computing

Norazah Abd Aziz[a], Lucyantie Mazalan[b]

Cyberspace Security Centre, MIMOS Berhad, Technology Park Malaysia, 57000 Kuala Lumpur
[a]azahaa@mimos.my, [b]lucyantie.mazalan@mimos.my

ABSTRACT

In a client-server environment that deals with multiple clients, there is a need to provide a mechanism on the server to manage the issuance of the client credentials for security authorization. Credentials created using a particular own platform identities and functions as an authentication credentials to authenticate the platform itself in a network communication. However, these credentials can easily be shared, copied and stolen. This will led to an anonymous service sharing and worst to come when the stolen credentials is using for phishing attacks to the original user. One solution to the problem is to use tamper-resistant hardware to which a credential is bound such that a credential can only be generated and used in connection with the hardware. For that, manufacturers have started to embed into computers a tamper-resistant piece of hardware, called trusted platform modules (TPM), as specified by the Trusted Computing Group. This mechanism insures that credentials can only be issued with the TPM existence in the platform thus guarantees the platform origins. This paper describes the component involved in the credential issuance method by the server trusted computing domain. To implement our approach, a client server application is used as an interface through the secure communication channel in credential request. The server acts as a Trusted Third Party to verify authorized users in this environment.

Keyword-*Credential, Trusted Computing, Trusted Third Party*

1.0 Introduction

Trusted Computing (TC) is a technology developed and promoted by non-profit industry consortium that aims to enhance the security of the trusted computing hardware and software building blocks. The main goal of the consortium is to come out with the specification for Trusted Platform Module (TPM) [1] and surrounding software architectures like the TCG Software Stack (TSS) [2]. These components have potentials to be used for security and trust related services like remote attestation and key management.

In this paper, we will discuss about credentials to serve the authentication services of a client server network environment. In particular, it describes about the credential issuance and other factors that contribute to the implementation of such activities. Our approach is supported for both Windows and Linux platform. Hence, in the first contribution the question how trust relationships between remote platforms can be established by using TC is addressed. The approach presented in this paper allows establishing trusted communication channels by means of the TCG's specified remote attestation. The approach introduces a so-called attestation proxy that is placed in front of the actual application and performs a mutual platform attestation of the two communication parties.

This paper is organized in the following way. It starts with brief introduction in part one, followed by part two which detail out the client and server environment in security purpose. Part three of the paper mentions about the credential issuance where much of the credential issuer components is discussed. The development component issue are presented in part four and finally is part five which describes the basic architecture of the attempt containing the basic system requirement continued by the current implementation in part six. The paper ends with a conclusion.

2.0 Client Server environment

Client-server environment is an exciting architecture that helps to redefine the end users role in application systems. It also manages computer resources among multiple end users in a computer network environment. Basically the client-server environment is an approach or network design to split an application's processing across multiple processors to gain the maximum benefit at the least cost while minimizing the network traffic between machines. The key phase is to split the application processing. In a client-server mode each processing works independently but in cooperation with other processors. Both rely on each other to perform an independent activity to complete the application process. The distinguishing feature of a client-server system environment is that it contains cooperative processing capabilities through the use of networks.

A server acts as a process that resides at the central location of resource data to provide services to one or more clients. The server is connected to the network and is made available to the client. Whereby, a client is an intelligent workstation processer that is capable on making request to servers to establish certain application processes. Since the server is usually the central location for critical data, adequate trusted securities need to be taken to ensure data safety. Security is implemented to cater the robust access control, data integrity, confidentiality and accountability services in client server environment. The natural question resulted as a

corollary of this, is how do we establish trusted securities in such a system?

One of the approaches available is well known as trusted computing network environment demonstrated by the TCG specifications. The goals of trusted computing are to protect the most sensitive information, such as private and symmetric keys, from theft or use by malicious code. Trusted computing assumes that client software is going to be compromised at some time during its life, and provide protection for its sensitive keys in case this should happened [3]. This concept of trusted computing covers a rather vast set of specifications and standards ranging from the core trusted platform module to both processor and operating system. Within this specification a trusted platform is one that behaves in an expected manner for a particular purpose [4].

Security plays a significant role in a client-server network environment. Users are typically identified with a user account, and system-specific controls can be mounted on these accounts to provide security mechanisms. Security services such as access control and accountability can be implemented in this manner, with the accounts providing a form of stable identity. Other services such as authentication and non-repudiation clearly also rely on the establishment and preservation of stable identities. Thus most of the generic security services are reliant on the provision of stable identities. This environment, by definition provides a function of a trusted third party who can provide assurance as to the identities of entities in the network. With a TPM hardware embedded in the platform and a trusted third party in work, the network provides integrity, creation and use of digital signatures and privacy protection mechanisms.

One of the potential security services is called by the name of attestation [5]. Attestation is a process of assuring that information is accurate and obviously a critical concept for the trusted platform. This is because of the trust in the system is based on taking measurements and checking the measurements. If a system is not able to attest the accuracy of that information, then the trust to the platform does not exist. Attestation is closely related to authentication. In the network environment, anonymous authentication access could facilitate the security mechanism. According to [4], the authentication concept performed by the access requestor requires an access to the facilities without necessarily revealing their identities to external parties. This requirement stems from the possible need for each individual to maintain some degree of plausible deniability as to these presences at a convenor. One of the approaches to perform this requirement is by using Direct Anonymous Attestation (DAA) [6].

3.0 Credential Issuance

Trusted Third Party

In cryptography, a trusted third party (TTP) is an entity which facilitates interactions between two parties who both trust the third party; they use this trust to secure their own interactions. TTPs are common in cryptographic protocols, for example, a certificate authority (CA) [7]. As an example, imagine two people, Alice and Bob wish to communicate securely that use cryptography. Alice may need to obtain a key that use to encrypt messages in order to send it to Bob. In this case, the CA is the trusted third party which sends the key to Alice and Bob. The key then uses by Alice to send secure messages to Bob as she trust the CA.

The Public Key Infrastructure (PKI) depends on the concept of a TTP. Nowadays, the PKI technology is widely used in computing and networking environment. A PKI enables users of a basically unsecure public network such as the Internet to securely and privately exchange data through the use of a public and a private cryptographic key pair. The key pair is obtained and shared through a TTP. Thus, the public key cryptography is also based on the digital signing of public keys. In this case, however, one central authority signs all the public keys, and everybody trusts the central authority. The authority's public key is distributed among the users, who can use it to verify the signatures on public keys of other users [8].

With certificate authorities, every user in the system trusts the CA through a process of digital signing since everything that signed by CA is considered trusted. The CA sends its public keys to the users to let them verify the signature. The users must submit their information (names and public keys) to ensure the trust is mutual to enrol with the CA. Then the CA verifies the authenticity of the submitted information and signs the submitted public key with its private key if everything is correct. Finally, all the signed information is sent back to the end users including with the CA signature.

Privacy CA

In a verification process, each TPM has to generate a unique RSA key pair called Endorsement Key (EK). Authentication of a valid TPM that belongs to the platform is done by certification of EK issued by CA or verifier itself which have knowledge about EKs of all genuine TPM. Basically, if a verifier wants to ensure that the platform runs on a secured operating system, TPM will send its measurements about the platform (PCR value) to the verifier. Then, TPM needs to validate the PCR using the EK. So, verifier knows PCR is valid as well as TPM itself. However, there are problems on privacy issue using this way. All the transaction would become linkable to each other since two different verifiers can tell that they talk to the same platform [9]. Figure 1 illustrates this issue.

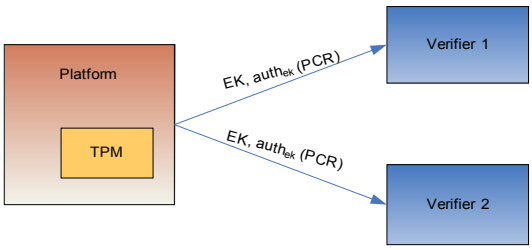

Figure 1: Privacy issue on verification of secure platform.

To solve this problem, trusted computing group (TCG) has proposed two protocols which remotely convince a communication partner that a trusted hardware is indeed a trusted hardware. These protocols will enable two communication partners to establish the other end in a secured computing platform and therefore making sure that it is a safe data exchange. Some degree of privacy is provided by these remote identification protocols to users of the platform. However, the communication partners can only establish the other end using a trusted hardware device but not in a particular device [11]. These specified protocols are Privacy CA and Direct Anonymous Attestation (DAA). In this paper, we are not discussing about DAA protocol since we use Privacy CA in our implementation as prototypes. The Privacy CA protocol was initially used by the group to solve the above mentioned privacy issue for user that needs the verification of the platform containing a TPM.

The Privacy CA involves TTP in each transaction and the party must be fully trusted by all other parties. The Privacy CA is assumed to know the public parts of the Endorsement Keys of all valid TPM. This valid TPM refers to an uncompromised TPM. In contrasts, a rogue TPM is a TPM that has been compromised and had its secrets extracted. Now, whenever a TPM needs to authenticate itself to a verifier, it generates a second RSA key pair, called an Attestation Identity Key (AIK). Then, it sends the AIK public key to the Privacy CA, and authenticates this public key relating to the EK. The Privacy CA will issue a certificate on the TPM's AIK if it finds the EK in its list. Through this protocol, there are two possibilities to detect a rogue TPM [10]:

1. If the distribution of the EK secret key which was extracted from a TPM is detected and announced as a rogue secret key, the Privacy CA can compute the corresponding public key and remove it from its list of valid Endorsement Keys.
2. If the Privacy CA gets many requests that are authorized using the same Endorsement Key, it might needs to reject that requests. In practice, it's probably be determined by some risk-management policy due to the exact threshold on requests that are allowed before a TPM is tagged rogue depending on the actual environment and applications.

4.0 TCG SOFTWARE STACK AND TBS

The TCG also defines an accompanying software infrastructure called the TCG Software Stack (TSS) as well as TPM hardware. The TSS is used by TCG as interface between applications and the TPM (through the TPM driver). TSS 1.2 specification is provided and standardized by the TCG. TSS design goals are to supply one single entry point to the TPM functionality (exclusive TPM access), synchronize concurrent TPM access, TPM resource management (key slots, authorization sessions etc.) and building of TPM commands messages according to TPM specification. TSS is designed as a stack of discreet modules with clearly defined interfaces between them.

TCG Device Driver Library (TDDL), TSS Core Services (TCS) and TSS Service Provider (TSP) are software layers in TSS. Every layer provides different sets of functionalities. Figure 2 shows the TCG Software Layering architecture. The TCG Device Driver Library (TDDL) is an intermediate module between the TCS and the kernel mode TPM Device Driver (TDD). The TDDL supplies the conversion between user mode and kernel mode. TDDL commands are sent at byte level as a stream to the TPM Device Driver. There is typically one TDDL per TPM and is provided by the TPM manufacturer. Access to the TPM is exclusive and synchronized via the TDDL.

TSS Service Providers (TSP) is the top-most modules and provides a rich, object-oriented interface for applications to incorporate the full capabilities of a TCG-enabled platform. The interface used by the applications to access the TSP is the TSS Service Provider Interface (TSPI). While not an architecture requirement, it is intended that the TSP obtain many TCG services such as TPM byte stream generation, key management, etc from the TCS [2]. It provides contexts which are used by applications to manage TPM objects such as policy, key, PCR and others. Furthermore, it is provided as a library and used by all high-level applications, for example a Cryptographic Service Provider (CSP) or user application. Therefore, the application developers do not need to have in depth TPM knowledge.

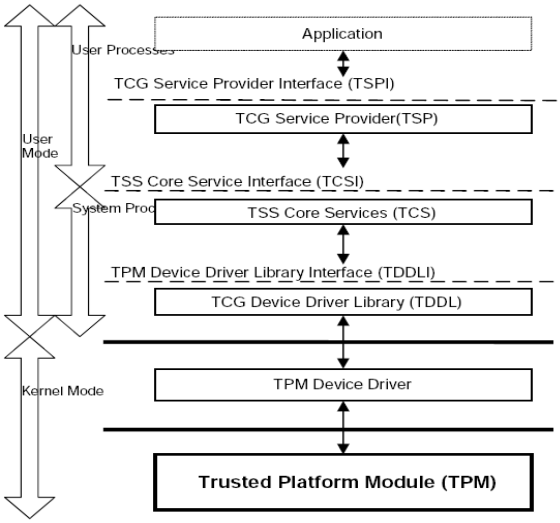

Figure 2: TCG Software Stack Architecture [12]

Microsoft has implemented a good base of functionality in Windows Vista using the TCG specification such as TSS by using Microsoft CryptoAPI as interface and BitLocker which uses TPM to measure boot process attributes and store keys from full volume data encryption. Before Windows Vista, the TPM device driver and TDDL are based on vendor specific as well as some TSS which is supplied by vendor. Then, Vista comes with a basic TPM abstraction layer called TPM Base

Services (TBS) which a Remote Procedure Calls (RPC) based service that only accessible from the local machine.

TBS provides virtualization of TPM resources allowing multiple applications (TSS, OS services etc.) to access the TPM. It allows restricted access to TPM commands on a "per command" basis. In TBS, the resources such as key handles and auth handles are replaced by virtual handles. TBS keeps the mappings of handles and if TPM runs out of resources, TBS takes care of swapping out entities from the TPM. In this situation, the virtual resource handles are not affected. If a swapped out resource is used again (via its virtual resource handle) the TBS tries to reload the entity into the TPM [12].

The TBS component is divided into four functional areas. They are Resource Virtualization, Command Scheduling, Power Management and Command Blocking [13]. Each command submitted to the TBS is associated with a specific entity to ensure that different entities cannot access each other's resources. This is accomplished by creating one or more contexts for an entity, which are then associated with each subsequent command submitted by that entity. Then, the TBS can execute TPM commands under the appropriate context after receive the command which includes a context object. An entity creates the context before it accesses the TBS and maintains the context until it is finished performing TBS accesses. For example, in the case of a TSS, the TCG core services (TCS) component of the TSS would create a TBS context when it starts up, and it would keep that context active until it shuts down [13].

5.0 Architecture

The overall architecture of the proposed implementation is depicted in Figure 3. The implementation has the following characteristics:

- A server machine on Linux platform has the TPM chip hardware.
- Client machines on Linux or Windows platform have TPM chip hardware.
- A trusted third party software embedded in the server and acts as the credential issuer uses the Privacy CA protocol.
- A database connected to the server is to manage the information of the client's credentials.
- The server and the client is connected by the network and coupled by a secured communication.
- TSS and TBS are softwares performing the trusted protocols and are installed in both server and client for TPM application implementation.

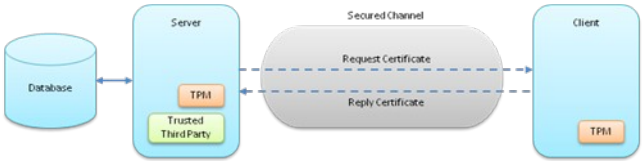

Figure 3: Basic Implementation Architecture

The server serves as the manager to the client in regards to the credential issuance procedure. Request made by the client through the web services is acknowledged by the server for the credential procedure in a secured connection communication. The server inquires the credential from the credential issuer that is the trusted third party. The trusted third party grants the credential to the server while the server replies the credential to the client through secure communication channel.

In order to fulfil the management of credentials for clients, the server needs a database to store and clear the information of the credentials which is requested by the clients. The server has the requirements to request and revoke that offers the possibility to record and eliminate the credential's information from the database.

6.0 Current Implementation

Referring to the above architecture, installation of all components must be established in order to proceed with the development. The current phase of implementation setup is focusing on providing the environment development for server and the trusted third party.

These software packages installation and configuration are required to facilitate the above mentioned components:

- Qt as a cross-platform application framework for desktop and embedded development. This software includes an intuitive API and C++ class library integrated tools for client and server GUI development in this research. More of the Qt can be found here [14]. This research uses Qt4 version.
- Secure communication channel and reliable transport protocol to establish a reliable communication between server and client. This is to ensure the security of the data throughout the application process as well as protecting it during the credential issuance.
- OpenSSL [15] as an open source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols, including the necessary cryptographic operations. The functionality provided by an OpenSSL engine allows the implementation of certificates issuance in this research development.
- TrouSerS TCG Software Stack [2] to provide an API to the operating systems and applications to implement the functionality provided by the TPM.
- The TrouSerS TPM Tools [16] is implemented for testing to check TPM capability on TPM hardware and software. This tool is a set of open source programs that provides commands allowing a platform administrator to manage and diagnose the TPM resident on a platform.
- TPM Device Driver is required in order to use the TPM after Linux bootstrapped. This is necessary for both hardware and software TPM implementation. For TPM hardware module implementation, the device

driver is specified to the TPM manufacturer. While for TPM software implementation, the device driver can be obtained by installing the TPM Emulator. This research development is currently been done on an Intel machine provided with Infineon TPM chip [1] [2]. The experiment outputs are shown in Figure 4 and 5.

```
#/sbin/lsmod | grep tpm
tpm_infineon           11369  1
tpm                    15329  1 tpm_infineon
tpm_bios                9537  1 tpm
#/usr/local/sbin/tpm_version
LOG_DEBUG TSPI rpc/tcstp/rpc.c:358 Sending TSP packet to
host localhost.
LOG_DEBUG TSPI rpc/tcstp/rpc.c:373 Connecting to 127.0.0.1
LOG_DEBUG TSPI rpc/tcstp/rpc_context.c:44
RPC_OpenContext_TP: Received TCS Context: 0xa0ed3786
LOG_DEBUG TSPI rpc/tcstp/rpc_caps_tpm.c:40
RPC_GetTPMCapability_TP: TCS Context: 0xa0ed3786
  TPM 1.2 Version Info:
  Chip Version:        1.2.1.2
  Spec Level:          2
  Errata Revision:     0
  TPM Vendor ID:       IFX?
LOG_DEBUG TSPI rpc/tcstp/rpc_caps_tpm.c:40
RPC_GetTPMCapability_TP: TCS Context: 0xa0ed3786
  TPM Version:         01010000
LOG_DEBUG TSPI rpc/tcstp/rpc_caps_tpm.c:40
RPC_GetTPMCapability_TP: TCS Context: 0xa0ed3786
  Manufacturer Info:   49465800
LOG_DEBUG TSPI rpc/tcstp/rpc_context.c:60
RPC_CloseContext_TP: TCS Context: 0xa0ed3786
gmon.out
#ls -l /usr/local/sbin
tcsd
tpm_changeownerauth
tpm_clear
tpm_createek
tpm_getpubek
tpm_resetdalock
tpm_restrictpubek
tpm_restrictsrk
tpm_revokeek
tpm_selftest
tpm_setactive
tpm_setclearable
tpm_setenable
tpm_setoperatorauth
tpm_setownable
tpm_setpresence
tpm_takeownership
tpm_version
```

Figure 4: Console command to locate existence of the TPM Device Driver.

```
#/usr/local/sbin/tcsd -f
TCSD tcsd_conf.c:90 platform_class_list_append:
platform_class_list_append start:
TCSD tcsd_conf.c:121 platform_class_list_append: Platform
Class Added.
TCSD TCS ps/ps_utils.c:486 init_disk_cache: found 4 valid
key(s) on disk.
TCSD TCS tcsi_caps_tpm.c:43 Entering Get Cap
TCSD TCS tcs_context.c:184 Success: 30000000 is an Internal
Context
To TPM: 00 C1 00 00 00 12 00 00 00 65 00 00 00 1A 00 00
To TPM: 00 00
TCSD TDDL tddl.c:105 Calling write to driver
TCSD TDDL tddl.c:116 ioctl: (25) Inappropriate ioctl for
device
TCSD TDDL tddl.c:117 Falling back to Read/Write device
support.
From TPM: 00 C4 00 00 00 1D 00 00 00 00 00 00 00 0F 00 30
From TPM: 01 02 01 02 00 02 00 49 46 58 00 00 00
TCSD TCS tcsi_caps_tpm.c:43 Entering Get Cap
TCSD TCS tcs_context.c:184 Success: 30000000 is an Internal
Context
To TPM: 00 C1 00 00 00 16 00 00 00 65 00 00 00 01 00 00
To TPM: 00 04 00 00 00 B4
TCSD TDDL tddl.c:105 Calling write to driver
From TPM: 00 C4 00 00 00 0F 00 00 00 00 00 00 00 01 00
TCSD TCS tcsi_caps_tpm.c:43 Entering Get Cap
TCSD TCS tcs_context.c:184 Success: 30000000 is an Internal
Context
To TPM: 00 C1 00 00 00 16 00 00 00 65 00 00 00 01 00 00
To TPM: 00 04 00 00 00 B6
```

```
TCSD TDDL tddl.c:105 Calling write to driver
From TPM: 00 C4 00 00 00 0F 00 00 00 00 00 00 00 01 00
TCSD TCS tcsi_caps_tpm.c:43 Entering Get Cap
TCSD TCS tcs_context.c:184 Success: 30000000 is an Internal
Context
To TPM: 00 C1 00 00 00 16 00 00 00 65 00 00 00 01 00 00
To TPM: 00 04 00 00 00 B6
TCSD TDDL tddl.c:105 Calling write to driver
From TPM: 00 C4 00 00 00 0F 00 00 00 00 00 00 00 01 00
TCSD TCS tcsi_caps_tpm.c:43 Entering Get Cap
TCSD TCS tcs_context.c:184 Success: 30000000 is an Internal
Context
To TPM: 00 C1 00 00 00 16 00 00 00 65 00 00 00 05 00 00
To TPM: 00 04 00 00 01 01
TCSD TDDL tddl.c:105 Calling write to driver
From TPM: 00 C4 00 00 00 12 00 00 00 00 00 00 00 04 00 00
From TPM: 00 18
TCSD TCS tcsi_caps_tpm.c:43 Entering Get Cap
TCSD TCS tcs_context.c:184 Success: 30000000 is an Internal
Context
To TPM: 00 C1 00 00 00 16 00 00 00 65 00 00 00 05 00 00
To TPM: 00 04 00 00 01 02
TCSD TDDL tddl.c:105 Calling write to driver
From TPM: 00 C4 00 00 00 12 00 00 00 00 00 00 00 04 00 00
From TPM: 00 01
TCSD TCS tcsi_caps_tpm.c:43 Entering Get Cap
TCSD TCS tcs_context.c:184 Success: 30000000 is an Internal
Context
To TPM: 00 C1 00 00 00 16 00 00 00 65 00 00 00 05 00 00
To TPM: 00 04 00 00 01 04
TCSD TDDL tddl.c:105 Calling write to driver
From TPM: 00 C4 00 00 00 12 00 00 00 00 00 00 00 04 00 00
From TPM: 00 0A
TCSD TCS tcsi_caps_tpm.c:43 Entering Get Cap
TCSD TCS tcs_context.c:184 Success: 30000000 is an Internal
Context
To TPM: 00 C1 00 00 00 16 00 00 00 65 00 00 00 05 00 00
To TPM: 00 04 00 00 01 03
TCSD TDDL tddl.c:105 Calling write to driver
From TPM: 00 C4 00 00 00 12 00 00 00 00 00 00 00 04 49 46
From TPM: 58 00
TCSD TCS tcsi_caps_tpm.c:43 Entering Get Cap
TCSD TCS tcs_context.c:184 Success: 30000000 is an Internal
Context
To TPM: 00 C1 00 00 00 16 00 00 00 65 00 00 00 05 00 00
To TPM: 00 04 00 00 01 0D
TCSD TDDL tddl.c:105 Calling write to driver
From TPM: 00 C4 00 00 00 12 00 00 00 00 00 00 00 04 00 00
From TPM: 00 14
TCSD TCS tcs_caps.c:138 get_max_auths reports 20 auth
contexts found
TCSD TCS tcsi_caps_tpm.c:43 Entering Get Cap
TCSD TCS tcs_context.c:184 Success: 30000000 is an Internal
Context
To TPM: 00 C1 00 00 00 12 00 00 00 65 00 00 00 07 00 00
To TPM: 00 00
TCSD TDDL tddl.c:105 Calling write to driver
From TPM: 00 C4 00 00 00 10 00 00 00 00 00 00 00 02 00 00
TCSD svrside.c:280 trousers 0.3.1: TCSD up and running.
TCSD svrside.c:60 Caught SIGINT. Cleaning up and exiting.
```

Figure 5: Console display the detected PCR value.

- TPM Manager as a realization of open source TPM management software that provides an easy-to-use graphical user interface. TPM Manager is installed and run to for experiment. The experiment outputs are shown in Figure 6, 7, and 8.

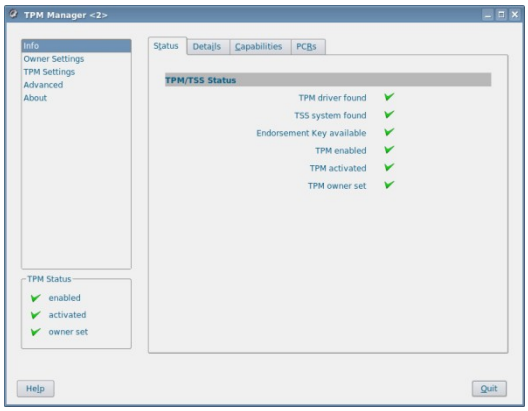

Figure 6: TPM Manager interface display the TPM TSS Status.

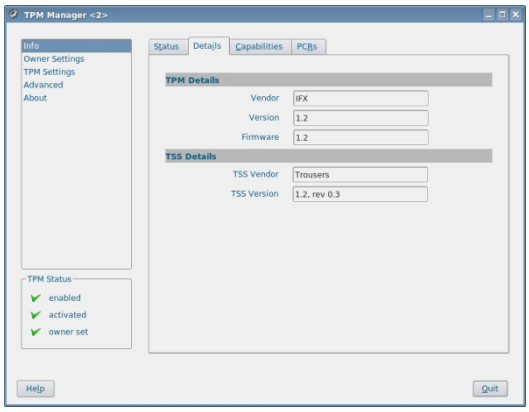

Figure 7: TPM Manager interface display the TPM and TSS Details.

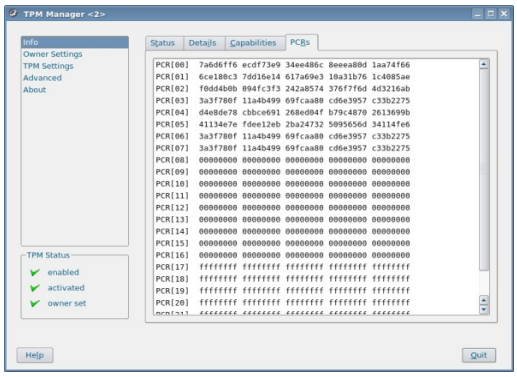

Figure 8: TPM Manager interface display the detected PCR value.

7.0 Conclusion and Future works

This paper describes in general the implementation setup requirement of current research progress in identity credential issuance system. The software components that have been installed and configured are the Qt software, the OpenSSL engine, the TrouSerS TCG Software Stack, the TrouSerS TPM Tools, the TPM Device Driver, the TPM Emulator and the TPM Manager. The future works will focus on the challenge in implementing the trust and secure of the trusted third party in order to generate credentials as well as dealing with different types of attestation protocol. In addition, this research will also reach on the development of the client application in Windows platform.

8.0 References

TPM Main: Part 1 Design Principles. 1.2 revision 85 edition, 2005.

Trusted Computing Group. Retrieved from http://trustedcomputinggroup.org.

Challener, D., Yoder, K., Catherman, R., Safford, D. and Van Doorn, L. (2007). *A Practical Guide to Trusted Computing*: Pearson Education, Inc.

Balfe, S., Lakhani, A.D., Paterson, K.G. (2005). Trusted Computing: Providing security for Peer-to-Peer Networks. *In Proceedings of the 5th IEEE Conference on Peer-to-Peer Computing,* 117-124.

Sadeghi, A.-R. and Stüble, C. (2004). Property-based attestation for computing platforms*:* Caring about properties*,* not mechanisms. *In Proceedings of the New Security Paradigms Workshop*, 67-77.

Balfe, S., Shiqun Li and Jianying Zhou. (2006). Pervasive Trusted Computing. *In Proceedings of the 2nd IEEE Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing*, -94.

Trusted Third Party. Retrieved from http://en.wikipedia.org/wiki/Trusted_third_party

Guynes, C. S. and Ron G. Thorn. (1995). Network Security in a Client/Server Environment. *In Proceedings of the ACM SIGSAC Review*, 6-12.

Camenisch, J. (2004). *Direct Anonymous Attestation: Achieving Privacy in Remote Authentication* [Presentation]. ZISC Information Security Colloquium. Zurich Research Laboratory.

IBM Research - Direct Anonymous Attestation. Retrieved from http://www.zurich.ibm.com/security/daa/

Camenisch, J. (2004). Better Privacy for Trusted Computing Platforms. *In Proceedings of the 9th European Symposium On Research in Computer Security*, 73-88.

TCG Software Stack (TSS) Specification Version 1.2 Level 1 Errata A, Part1: Commands and Structures, March 7, 2007

TPM Based Services. Retrieved from http://msdn.microsoft.com/en-us/library/aa446796(VS.85).aspx

Qt for Embedded Linux. Retrieved from http://trolltech.com/products/qt/features

OpenSSL. Retrieved from http://www.openssl.org/

TrouSerS - An open-source TCG Software Stack implementation, created and released by IBM. Retrieved from http://sourceforge.net/projects/trousers/