Trusted Computing & Pseudonymity : An Introduction and Survey

Gaurav Veda
Email: gveda@cse.iitk.ac.in
Supervisors: Dr. Deepak Gupta, Dr. Dheeraj Sanghi
Supervisors' Email: {deepak, dheeraj}@cse.iitk.ac.in

Department of Computer Science
and Engineering
Indian Institute of Technology
Kanpur, UP, INDIA - 208016

***Abstract*— This report is in two parts. In the first part I talk about Trusted Computing, while in the second part the focus is on pseudonymity.**

In today's world, security is of primary concern. Data of ever increasing value is being created and stored on PC's. At the same time, more and more vulnerabilities are being found in existing software . Till now, the main focus of security has been servers and networks, while clients have remained relatively unprotected. Also, most of the mechanisms in place for client security are software based. It is increasingly being felt that client security should be given much more importance and that a purely software based mechanism is inadequate in providing the required level of security. It was with this in mind that people and organizations started exploring the idea of security through hardware enhancements and a whole new paradigm of Trusted Computing was born. The idea was to come up with appropriate hardware modifications that would help in providing security against software attacks on clients. Here, I will present the suggested enhancements and look at the proposed security mechanism. This technology is no longer on paper and companies such as Intel, IBM etc. have started selling PC's with some of the required enhancements.

An important component of Trusted Computing is Attestation. This is the ability of a system to prove its security properties to a remote system. However, this immediately raises questions of privacy. Although we want to prove our credentials to a remote host, we do not want it to know any other identifying information about us. This leads us to the more general problem of protecting user privacy. Users want the ability to control the information that others know about them and also to be able to monitor and control its use. Currently, the way computers are used to carry out transactions, user privacy is being compromised since organisations and other users get to know much more information than what is necessary. We need anonymity in transactions. At the same time, we want to safeguard against malicious users who try to exploit the system. In this report, I will present a simple anonymous credential system proposed by Chaum [1]. A credential system is a system in which users can obtain credentials from some organizations and demonstrate their possession. It is said to be anonymous when no one (apart from maybe a few trusted third parties) can say whether two transactions are being carried out by the same user. I will conclude by presenting the basic mathematics underlying the anonymous credential system proposed by Camenisch and Lysyanskaya [2] that is actually used in the above security framework (Trusted Computing) for achieving anonymous attestation.

Trusted Computing

I. Introduction and Motivation

A. Client security is important

In today's world, security is of primary concern. Data of ever increasing value such as passwords, credit card numbers etc. is being created and stored on PC's. At the same time, attacks are outpacing today's protection models. Highly sophisticated tools are now readily available. Moreover due to the internet and ubiquitous connectivity (wired as well as wireless), attackers have remote access to the clients. Many times, they also have financial incentives for attaking a client. More and more vulnerabilities are being found in existing software and attackers are exploiting them continually to their advantage. Till now, the main focus of security has been servers and networks, while clients have remained relatively unprotected. Due to the above reasons and many more, it has become important to protect clients as well.

B. Software alone is not good enough

Today, most of the mechanisms in place for client security are purely software based eg. passwords, antivirus etc. It is increasingly being felt that a purely software based approach is not good enough. Many of these approaches rely on the OS. So, if the OS is compromised, then they are rendered ineffective. Since the OS is a large piece of code (easily over a million lines of code), there are many vulnerabilities in the OS code itself that are being continually found and exploited. Also, many malicious software (such as viruses, keyloggers etc.) hide themselves behind normal programs and often the user doesn't even know that they have been installed. We come to know of a virus only after it has been there for some time on the system and has (potentially) done some damage. Instead, we must have some pre-emptive protection mechanism that would not even allow such programs to get installed. Due to all these reasons, it is increasing being felt that we need to look at other approaches to security.

C. Hardware based security & the TCG

Due to the above reasons, people started looking towards hardware based approaches to security. Soon, smart cards were introduced. However, they only provided authetication. They were largely useless against software based attacks and attacks over the network. After this, the big companies like IBM, Intel, Microsoft, HP etc. started thinking about bringing additional changes to hardware. It soon became apparent that unless there was some standardization, there would be no compatibility between the new PC's and that this would be highly inacceptable to most people. Thus, the TCPA (Trusted Computing Platform Alliance) was formed in October 1999 by Intel, IBM, Compaq, HP and Microsoft. In April 2003, the name of TCPA was changed to TCG (Trusted Computing Group). The stated goal of this group is to "develop and promote open industry standard specifications for trusted computing hardware building blocks and software interfaces across multiple platforms, including PC's, servers, PDA's, and digital phones". This is an open industry association and now has over 40 members.

The approach taken by this organization is to bring about a host of hardware and software changes. It mandates changes in the CPU, chipset and IO devices and the introduction of a completely new hardware component - the TPM (Trusted Platform Module). Along with this, a small secure OS kernel (the nexus in MS terminology) needs to be added. The applications also need to be changed if they want to take advantage of this new secure platform. The normal OS needs just the introduction of a new device driver component. In such a system, there would be a parallel operating environment in which both the normal OS and this secure OS kernel will run *simultaneously*.

D. Threat model

The threat model that the TCG has in mind is as follows. Malicious software can

- Read Memory / HDD
  - Expose secrets
- Change Memory
  - Change values of data or programs
- Manipulate input and output
- Can change request for information

It is important to note that this technology is aimed at providing protection against software based attacks *only*. It does not intend to provide protection against hardware based attacks.

E. Major Security Enhancements

The major security enhancements provided by the TCG are

- Strong Process Isolation (Protected Execution)
  - The protected operating environment isolates a secure area of memory (RAM) that is used to process data with higher security requirements
  - Even the normal OS *can not* access this portion of RAM directly
- Sealed Storage
  - An application can ensure that no other application can read what it stores
  - The storage is also platform specific
- Secure IO
  - The input and output to the applications running in secure mode is guaranteed to be secure. That is, no other application can get to know what it is and no other application can change it.
- Attestation
  - This is the ability of a system to prove its security properties to a remote system.

II. Hardware Changes Required

The TCG specifications mandate a host of changes in the PC hardware. These are as follows.

A. Changes in the CPU

The following changes are needed in a TCG enabled computer CPU.

- A new mode flag that enables the CPU to run in both a standard mode and a nexus mode.
- Support for nexus-mode initialization
  The TCG specifications mandate a new CPU instruction (the SENTER instruction in Intel terminology). The job of this instruction is to bring up the nexus and pass control onto it after verifying that it (the nexus) has not been compromised with (ie. the code identity (a hash value) matches with some stored value). This instruction is executed by a program in the normal OS that wants to do some computation or start some application in secure mode. When this instruction comes to the CPU, it must pause execution of all other processes and process this instruction completely (without any context switches).
- CPU must be able to context switch between standard and nexus modes.
- Pages of physical memory are marked as trusted and the CPU should ensure that they are accessible *only* when the CPU is running in nexus mode

B. Changes in the chipset

- It must support page-granular DMA protection in collaboration with the CPU
  This means that the pages of memory that are marked protected, must not be allowed access by the DMA controller.
- Memory, I/O controller, and bus bridges must follow the parameters set by the DMA exclusion vector, which is an in-memory, 1-bit-per-page table entity that may be cached and that ensures that DMA devices do not read or write to the secure area of memory
- DMA exclusion vector programming must be under nexus control
- It must provide memory reset (ie. writing a fixed quantity (say 0) at the specified locations in the memory) on an

improper shutdown or if the protected environment fails. This ensures that the protected data can not be snooped from memory.
- It must be connected to the TPM

C. Trusted Platform Module (TPM)

This is a completely new hardware component introduced by the TCG TPM specifications. It is the most important component of the Trusted Computing architecture.

Its main components are

- Hardware Random Number Generator
- Cryptographic co-processor and RSA engine
  It can generate 2048-bit RSA public and private key pairs. This is needed for attestation and some other functions.
- SHA-1 and HMAC engine
  In the TC architecture, before an application is started in secure mode, it is verfied that the application has not been compromised with. This is done by hashing its code and checking with some previously stored value. Hashing is a very important aspect of the TC architecture and it is used repeatedly for a number of things (more examples are there in the sections that follow). Hence, the TPM also provides a SHA-1 (160 bit) and HMAC engine.
- Volatile memory
  The TPM has some small amount of volatile meory that is used to store the Platform Configuration Registers (see below)
- Non-volatile memory
  The TPM needs to store certain things like the Storage Root Key (for sealed storage) and Endorsement Key (unique for every TPM). These have to be maintained across boots. Moreover, an application can also ask the TPM to store some amount of information in the non-volatile memory.
- Opt-In
  The TPM plays an important role in nexus mode initialization and has a component for opting into the nexus mode.

The TPM serves the following functions

- Performs certain cryptographic operations as requested by the nexus and the CPU
- It stores the cryptographic keys used to provide the sealed storage and attestation functions
- It is involved in nexus-mode initialization

1) Platform Configuration Registers (PCR's): A Platform Configuration Register (PCR) is a 160-bit storage location inside the volatile memory area of the TPM. It is used for integrity measurements. There are a minimum of 16 PCR's inside the TPM. A PCR is designed to hold an unlimited number of measurements. It does this by using a cryptographic hash and hashing all updates to a PCR.

$$PCR_i^{New} = \mathcal{H}(PCR_i^{Old} \mid\mid value\ to\ add)$$

A log is maintained about the values stored in every PCR. Since the order is important for verification, the order in which these values came is also stored. During boot, some values (related to system integrity) are continually stored in some PCR's (for details, refer [3]) and checked against the values for an authentic system. If there is any mismatch, then the nexus will not be loaded and although the computer will boot into the normal mode, no application will be able to bring up the nexus mode.

Once in nexus mode, an application can also request that some value be stored in a PCR. This is used in sealed storage (discussed in detail below).

D. Secure Input devices

We need a secure session between the input devices (mouse, keyboard etc.) and the nexus. Due to this, the input devices must support the following changes.

- They must support the use of Triple Data Encryption Standard (3DES) or some other encryption algorithm to encrypt keystrokes and mouse movements so they are never openly exposed to the OS or to potentially malicious programs.
- They can include a certificate from the device manufacturer which can be used to set up session keys between the device and the nexus.

E. Secure Output devices

For secure output, we want to ensure that all unsecured hardware and software components, including the standard-mode operating system, video driver, and applications, must be unable to access the secure pixel data. For this, we need one of the following modifications depending on the nature of the graphics adaptor.

1) A secure graphics adaptor can be integrated in the chipset with a special closed path between it and the nexus. For example, as part of this solution, the graphics adaptor could offer a set of registers at a fixed address, accessible only when the system is running in nexus mode.
2) Discrete graphics adaptors : In this case, we need an encrypted path between the nexus and the graphics adaptor. One method to establish this encrypted path would be to use a digital certificate on a graphics adaptor that has public/private keys and is signed by a manufacturer. These public/private keys can be used to set up session keys between the graphics adaptor and the nexus.

Apart from a secure session, we also need an additional property for secure output. The video hardware must support a mechanism to always force secure window data "on top" so it cannot be overlaid or obscured by unsecured video windows. This would protect against spoofing attacks.

III. Software Changes Required - the Nexus

The nexus is a small security kernel. It manages the security hardware and the protected operating environment. The nexus is authenticated during system boot. After the authentication is done, the nexus creates a protected operating environment

within the normal OS. Once this has been done, an application can request that some other application be started in nexus mode. An application can also ask a part of itself (that needs security) be executed in the nexus mode. It is the job of the nexus to authenticate a process before it starts it.

The nexus executes in kernel mode within the curtained memory. It uses standard ring and virtual memory protections to isolate itself from the trusted applications and to isolate those applications from each other. It provides the system services that are required by an application executing in the secure mode. The nexus also provides a mechanism for Inter Process Communication between the secure applications and between the applications running in normal mode and secure mode. Apart from that, it also provides the trusted applications with a limited set of API's and services such as sealed storage and attestation functions. It also provides a primitive user interface that the applications can use for displaying graphics.

It replicates the following OS functionality

- Process and Thread Loader
- Memory Manager
- I/O Manager
- Interrupt handling

However, it is not a complete OS and does not have the following OS features

- File System
- Networking
- Kernel Mode/Privileged Device Drivers
- Scheduling

Therefore, the nexus has to use some of the features of the normal OS like storage (retrieval) of data on (from) the hard disk.

IV. Security Enhancements

A. Protected Execution

This feature is also known as Curtained Memory or Strong Process Isolation. This is basically an implementation of domain separation (virtual memory) in hardware.

Currently, the RAM is divided into two parts

1) Ring 0 (Operating System Memory)
   This portion contains important system functions like Memory management, scheduling etc.
2) Ring 3 (Space for user programs)
   Presently, only virtual memory protection is achievable to prevent two programs from accessing each others memory.

Presently, it is relatively easy for an attacker to add malicious programs to both the operating system and user space memory (eg. the user can be coaxed into opening an attachment that may actually contain a virus). TC addresses this problem by isolating a specific portion of RAM within the user address space to create curtained memory

A nexus addressing-mode bit is set to address this portion of memory. This memory can now be accessed only when the CPU is running in nexus mode. TCG specifications block direct memory access (DMA) devices from reading and writing to curtained memory by using a special structure called the DMA exclusion vector. This is an in-memory, 1-bit-per-page table entity that may be cached (see subsection II B above). This ensures that trusted applications running "behind the curtain" are not modified or observed by any other program or even the operating system itself. Consequently, information from the computer's hard disk must be read into operating system or user space memory first and then moved by the nexus into curtained memory, if necessary.

B. Sealed Storage

This is a mechanism to ensure secure storage of information. Currently, users or programs are prevented from accessing others' data by using file access control mechanisms like permission bits. However, since this check is done by the OS, this is only as secure as the OS. In case of a root or OS compromise, there is no way to prevent loss of information security. Moreover, even if no such breach takes place, one can easily remove the hard disk and take it to some other computer where we have root access and gain access to the information.

Sealed storage provides protection in all such cases. In addition to this, an application can also mandate that the data it sores is only accesssible to itself and no other application on this platform or to any application (including itself) on any other platform. This is implemented as follows.

Every TPM has a secret assymetric key pair known as the Storage Root Key (SRK). When an application asks for its data to be encrypted, the data is encrypted using a symmetric key and stored on the HDD. This key is in turn encryppted using some other key and stored along with the data. In this way their is a heirarchy of keys and each key is encrypted using the parent key (in this heirarchy). At the top of this heirarchy is the SRK. Since the SRK is TPM specific, the storage is automatically platform specific.

Moreover, an application can mandate that the data be stored in such a way that no other application should be able to read it. To do this, the application stores some values in the PCR's and along with the data, these PCR values are also stored. The nexus knows whether the PCR values are stored or not. If they are indeed stored, then when an application asks for decryption, the nexus will do the decryption and first check whether the current PCR values match with the stored PCR values. If yes, it gives the data to the application. If not, it refuses to give the data to the application. Now the application that asked for data storage knows what exactly did it store in the PCR's when the data was stored and can hence store the same values and thus get access to the data.

C. Secure IO

To prevent applications like keyloggers and window grabbers, we need secure IO. This is easily ensured in TC enabled platforms since we use new secure IO devices (see subsections II D and II E above). There is either an encrypted channel

using which data is transferred or there is a direct link between the TPM and the IO device.

D. Attestation

Attestation is the ability of a system to prove its security properties to a remote system. However, this immediately raises questions of privacy. Although we want to prove our credentials to a remote host, we do not want it to know any other identifying information about us.

To prove the security properties of the system, we want to send the values of the PCR's securely to the remote system. One way to do this is to encrypt using the Endorsement Key (this is a 2048-bit RSA key pair and is given by the platform manufacturer. It is unique for every TPM) and send the EK (Endorsement Key) certificate (issued by the manufacturer and containing the public component of the EK). But this gives away our anonymity and privacy (since the EK is unique). Another solution is to use a Trusted Third Party (TTP). However, this is also not the best solution. TCG mandates the use of Direct Anonymous Attestation (DAA) which does not involve a TTP and is described in the second part of this report (refer section IX).

V. Current status

- CPU
  Intel has already launched its prescott series of processors which probably have a TC enables CPU. However, this feature is currently turned off (if it exists at all). Intel has refused to either confirm or deny whether the chips have this feature or not.
- TPM
  Amtel, Infineon and National are already selling TPM chips (also called Fritz chips)
- Platform
  IBM has started shipping some of its laptops with TPM chips since 2002.
- Nexus
  Microsoft is all set to include this technology in its upcoming OS - codenamed Longhorn (earlier called Palladium). Microsoft calls this complete technology the NGSCB (Next Generation Secure Computing Base).

VI. Future Work and Conclusions

In its current form, the Trusted Computing (TC) architecture has some very important limitations. Some of these are

- It does not support DMA access to memory
- Sharing of memory (mmap) is not allowed
- No support for paging
- Dynamic linking not allowed

Removing atleast some of these limitations is extremely imortant if this technology is intended to be used widely. Otherwise, it would be a step in the backward direction.

Also, right now there do not exist any applications that make use of this technology. Unless some applications are written and it is proved that they benefit due to the use of this technology, it would be difficult to find buyers for Trusted Computing. The onus for this lies in the hands of the promoters and adopters of this technology ([4]).

A key road block in the adoption of this technology is the concern raised by people belonging to the GNU and open source community. They believe that the primary motive of this technology is to curb users' freedom and impose Digital Rights Management (DRM). They are against the feature of Attestation. They demand that there should be an owner override ([5]) whereby, the owner of a TC platform should be able to override the normal authentication mechanism. eg. Although a person uses the Mozilla web browser, he must be able to fool the remote party into believing that he is actually using Internet Explorer.

Some of the concerns raised by these people are indeed genuine and must be addressed.

Nevertheless, TC is a very bold step and promises to provide a host of security features. Whether it would find widespread acceptance is a question that can not be answered right now. As this is a nascent technology, it can be easily changed right now to address some of the issues pointed above and ensure its widespread adoption. It also has many unexplored aspects and has tremendous scope for research.

Pseudonymity

VII. Introduction and Motivation

An important component of Trusted Computing is Attestation. This is the ability of a system to prove its security properties to a remote system. This might be necessary, for example, before the remote system starts sending/receiving data to/from the host system and enters into a transaction with it. However, this immediately raises questions of privacy. Although we want to prove our credentials to a remote host, we do not want it to know any other identifying information about us. This leads us to the more general problem of protecting user privacy.

Users want the ability to control the information that others know about them and also want to be able to monitor and control its use. For example, the user might not want an organization A and an organization B to know what transaction it did with the other organization and when. Currently, because of the way in which computers are being used to carry out transactions, user privacy is being compromised. Consider, for example, digital signature schemes based on digital certificates asserting the genuineness of the host system and its public key. This certificate typically contains information about who owns the certificate (name, contact address, etc) as well as information about the issuing authority (VeriSign, Thawte, etc). So, if a host uses this scheme, he has to unnecessarily give away information such as its name, contact address and unique public key. This is not acceptable for all transactions and we need anonymity in transactions. At the same time, we also need to protect the system against malicious users who may try to take advantage of this anonymity.

A. Anonymous credential system

This points towards the need of an anonymous credential system. A credential system is a system in which users can obtain credentials from some organizations and demonstrate their possession to others (including organizations other than the one which issued this credential). A credential granting organization (O) can mandate that the user possess certain credetials from some other organizations before it seeks a credential from itself (O). Such a system is said to be anonymous when no one (apart from maybe a few trusted third parties) can say whether two transactions are being / were carried out by the same user (ie. the unlinkability of transactions). In such a system, an organization knows a user only via a pseudonym and it grants a credential to the pseudonym. A pseudonym is a fictitious name chosen by the user with or without the involvement of the organization. Unlinkability implies that the user have different pseudonyms with different organizations.

A basic credential system was introduced by Chaum [1] that satisifies these properties and is still the basis of most of the new protocols. We will soon have a look at it.

Apart from user anonymity, there are some more properties that are needed in an anonymous credential system. These can be summarised as follows.

1) Basic Desirable Properties:

- It should be impossible for other users and organizations to forge the credential of a user even if they connive.
- It should also not be possible for users to pool in their credentials and show some of them to obtain a credential from an organization for one of them which they would have been unable to obtain individually.
- Since the organizations are independent entities, they should be able to choose their public/private key pairs themselves and independently of the other organizations.
- Another important property is that such a system must be able to function without the involvement of a Trusted Third Party. This not only enhances the trust in the system, but also brings down its cost.

2) Additional Desirable Properties: In addition to the properties mentioned above, there are some more properties that are desirable in such a system to prevent its misuse.

- Users must be discouraged to share their pseudonyms and credentials with others.
- To safeguard the system against malicious users, it should be possible to revoke the anonymity of users who are found to take part in illegal transactions and those who misuse their credential.

The architects of Trusted Computing recognized the importance of user privacy and hence they went in for a system that allows Direct Anonymous Attestation (DAA). They use the anonymous credential system proposed by Camenisch and Lysyanskaya [2]. Their approach is also based on Chaum's [1] work. Here, I will state the assumptions made by them and will also provide the mathematical preliminaries needed to follow their work.

VIII. Chaum's Solution

Chaum's solution is based on an envelope analogy [6]. Suppose you have a number of carbon-paper-lined envelopes. An envelope is such that if you put a piece of paper inside it and make a signature mark on the outside of the envelope, the mark is transferred onto the paper. Also, the envelopes have slits at appropriate places so that only certain parts of the paper (put inside) are visible. At the beginning, you come up with a number of randomly chosen pseudonyms and write them on a piece of paper.

To obtain a credential from an organization, you put the paper in such an envelope in which, only that portion of the paper is visible where, the pseudonym you want to use with this organization is written. You then send this to the organization. To grant a credential, the organization will make a repeating signature pattern on the outside of the envelope and send it back to you. The kind of signature pattern determines the type of credential that the organization wishes to grant. The organization then returns the envelope to you. To show this credential to some other organization (O'), you put the paper in such an envelope, in which only the pseudonym

you wish to use with O' and part of the signature pattern is visible. You then send the envelope to O'. O' verifies that the signature pattern is indeed that of the right organization and then proceeds just as above.

It is trivial to see that such a system guarantees unconditional anonymity and unlinkability of credential showings. We can extend this analogy to prevent users from using multiple pseudonyms with the same organization. This can be done by firstly restricting that a single person can get only one such slip of paper. This can be done by issuing such slips based on fingerprints or some such unique identifying mechanism. Since the slips are initially empty (apart from having the signature of the slip issuing organization), nothing in terms of privacy is lost by this restriction. In addition to this, the slip can be divided into zones and each zome be assigned to a single organization. An organization can now mandate that the user show the comlpete zone corresponding to it. Due to this, the user would be limited to using only a single pseudonym with an organization.

A. Putting in numbers

The above analogy can be converted into numbers as follows. We first of all introduce a few restrictions. We assume that only a particular organization Z can issue credentials on behalf of all the organizations. Z issues a credential for an organization O to a user U having a presudonym P when, it is instructed by the organization to do so. But still, Z does not get to know any more information about a user U than does an organization. We also assume that every user U has a unique number u associated with it.

The system is based on RSA signatures. Z uses only a single RSA modulus m to issue the credentials. In the following text, all computation is done $modulo\ m$ when not explicitly stated. If computation is done to some other modulus, it is explicitly stated. Every organization O_i has a public key e_i and a private key d_i associated with it for the RSA modulus m. However, only Z has knowledge of the private components of the organizations' RSA key pair. Also, since Z is the only organization issuing credentials on the behalf of all the other organizations, it is the only entity that knows the factorization of m. Suppose we have k organizations. Then, we will have k such RSA key pairs. A signature on a message t using the ith RSA key pair is t^{d_i}. To verify the signature, the verifier only needs to raise the signature to the e_ith power. This is because:

$$e_i * d_i = 1(mod\ \phi(m))$$

where $\phi(m)$ denotes the order of the group Z_m^* and any element of a group raised to the order of the group is equal to identity. Therefore,

$$(t^{d_i})^{e_i} = t$$

(remember all computation is done modulo m). Let us denote the product of all e_i's as C ie.

$$C = e_1 * ... * e_k$$

Since the public exponents are publicly known, the quantity C is known to everyone.

1) Forming a pseudonym: To form a pseudonym to be used with the organization O_i, the user U computes a random number r_i and then computes

$$P_i = u * (r_i^C)$$

The quantity P_i thus computed (obviously, modulo m), is the pseudonym of the user U with the organization O_i.

2) Granting a credential: If an organization O_i wants to grant a credential to the user with the pseudonym P_i, it will instruct Z to do so. Z will then compute $P_i^{d_i}$ and send it to the user U.

3) Storing a credential: The user receives the credential as $P_i^{d_i}$. This can be expressed as

$$P_i^{d_i} = (u * r_i^C)^{d_i} = u^{d_i} * r_i^{C*d_i}$$

Also,

$$r_i^{C*d_i} = (r_i^{C/e_i})^{e_i*d_i} = r_i^{C/e_i}$$

Therefore,

$$P_i^{d_i} = u^{d_i} * r_i^{C/e_i}$$

Now, the user U knows the value of r_i, C, e_i. Hence, it calculates the value r_i^{C/e_i}. After this, it divides the quantity received as the credential $P_i^{d_i}$ by this and is thus able to compute the value u^{d_i}. It stores this as the credential from organization O_i.

4) Demonstrating possession of a credential: Suppose the user U wants to demonstrate the possession of a credential from organization O_i to an organization O_j. It has got the pseudonym

$$P_j = u * (r_j^C)$$

with the organization O_j. If the user has obtained a credential from the organization O_i, it must have stored the value u^{d_i}. Now the user is in a position to compute the quantity

$$u^{d_i} * r_j^{C/e_i} = (u * r_j^C)^{d_i} = P_j^{d_i}$$

It sends this to the organization O_j. To verify possession of the credential given by O_i, by the user, O_j simply raises the received quantity to the e_ith power. If U indeed possessed the credential, then the quantity that would be obtained by O_j after this computation would simply be P_j - the pseudonym by which it knows U. If the quantities do not match, it means that the user with pseudonym P_j does not possess a valid credential from the organization O_i.

Since the user U chooses the blinding factors (r_i's) on its own and no one apart from the user has knowledge of them, even if all other organisations connive, they cannot link the transactions made by this user. Also, since the user alone

knows its unique secret u, no other user or organisation can forge its credential. Therefore, we have an anonymous credential system satisfying some of the basic desirable properties.

IX. Camenisch and Lysyanskaya's approach

The above protocol given by David Chaum does not satisfy many of the basic and the additional desirable properties mentioned in the Introduction. However, it does form the basis of many protocols that do satisfy some or most of these properties.

One such protocol is that given by Camenisch and Lysyanskaya [2]. Their basic protocol does not require the presence of a Trusted Third Party (Z). However, a modification that provides safety to the system via revokable anonymity does require such an organization. However, the work of such an organisation is greatly reduced as compared to that in Chaum's protocol. Also, this protocol is quite efficient than the other available protocols (which are also based on Chaum's work). This protocol has also been included as part of the TPM 1.2 (Trusted Platform Module) specifications by the TCG (Trusted Computing Group). There, it is used to provide Direct Anonymous Attestation (DAA). Here, I will present the mathematical preliminaries needed to understand the protocol and will also delve into the notation used in [2].

A. The Discrete Logarithm Problem

The protocol is mainly based on the discrete logarithm (DL) problem. This problem is defined as follows.

Given a finite group G and two elements $g, y \epsilon G$, find an integer x (if it exists) such that

$$g^x = y$$

The DL problem for a general group is believed to be hard. Also, note that it is quite easy to compute exponentiation is a finite group. Due to this, the DL problem yields a good one way function. Due to these properties, the DL problem forms the basis of many public key cryptosystems. However, the DL problem is efficiently solvable in certain groups. Hence, the choice of the finite group in which we compute the DL, is of paramount importance.

1) Double discrete logarithm and eth root of the discrete logarithm: Let G be a cyclic group of order n and let a be an element of Z_n^*. Let g be the generator ofthe group. The double discrete logarithm of $y \epsilon G$ to the bases g and a is the smallest positive integer x (if it exists), satisfying

$$g^{(a^x)} = y$$

An eth root of the discrete logarithm of $y \epsilon G$ to the base g is an integer x (if it exists), satisfying

$$g^{(x^e)} = y$$

If the factorization of n is unknown (eg. if it is an RSA modulus), then computing the eth roots in Z_n^* is assumed to be infeasible.

B. Group of Quadratic Residues modulo a composite n

In their protocol, Camenisch and Lysyanskaya apply all proofs of knowledge over the group of quadratic residues modulo a composite number n. It is the clever choice of this particular class of groups that is being considered as a novelty of their work by some [7].

A number q is said to be a quadratic residue $(modulo\ n)$ if there exists an integer x such that

$$x^2 = q\ (modulo\ n)$$

C. Assumptions

The protocol is based on two main assumptions : the strong RSA assumption and the decision diffie hellman assumption. The assumptions are as follows.

1) Strong RSA Assumption: The Strong RSA assumption is the assumption that the flexible RSA problem is diffucult to solve for large RSA modulus m. The flexible RSA problem is defined as follows.

Given an RSA modulus m, and a number $z \epsilon Z_m^*$, find an integer $e \geq 2$ and an element $u \epsilon Z_m^*$, such that

$$u^e = z\ (modulo\ m)$$

2) Decision Diffie Hellman Assumption: The protocol also assumes that the Decision Diffie Hellman Problem is hard. The problem is stated as follows.

Given a cyclic group G with a generator g, and given three elements g^u, g^v and g^w, decide whether

$$w = u * v\ (modulo \mid G \mid)$$

Refer [8] for a concise, yet insightful discussion of the problem and a proof that it is infact not as hard as the Diffie Hellman problem (given g^u and g^v, come up with an element g^w such that $w = u * v\ (modulo \mid G \mid)$).

D. Proofs of Knowledge of Discrete Logarithms

Here, I will use notation similar to that used in [9]. The symbol $||$ denotes the concatenation of two binary strings. $\mathcal{H}$ is a one way hash function which takes binary strings of any length to a fixed binary string of length k bits $(k \approx 160)$.

1) Proof of knowledge of the discrete logarithm of y to the $base\ g$: A pair $(c, s) \epsilon \{0,1\}^k \times Z_n^*$ satisfying

$$c = \mathcal{H}(y \mid\mid g \mid\mid g^s y^c)$$

is a proof of knowledge of the element $y \epsilon G$ to the $base\ g$.

Note that y is raised to the power of the hash value c in the last concatenated string (of the quantity that is hashed). It is this feature that is giving us a proof of knowledge, since such a hash value c can only be computed by a person who knows x - the discrete logarithm of y to the $base\ g$.

Verification of this signature is trivial since the hash function $(\mathcal{H})$, y, g, s and c are either publicly known, or explicitly communicated to the verifier.

Such a signature can be created as follows.

- Choose a random number r and compute c as
$$c = \mathcal{H}(y \mid\mid g \mid\mid g^r)$$
- The prover knows the secret key x given by
$$x = log_g(y)$$
and hence can compute the parameter s as
$$s = r - cx \ (modulo\ n)$$
- Now,
$$g^r = g^{s+cx} = g^s g^{cx}$$
and
$$g^{cx} = (g^x)^c = y^c$$
$$\Rightarrow g^r = g^s y^c$$

2) Proof of knowledge of representation of y to the bases g and h: We now want to prove knowledge of (α, β) such that

$$y = g^\alpha h^\beta$$

where $h, g \ \epsilon \ G$. This is denoted as

$$PKREP[(\alpha, \beta) : y = g^\alpha h^\beta]$$

Now, let me further explain the notation that is used here. Consider

$$PKREP[(\alpha, \beta) : y = g^\alpha \wedge z = g^\beta h^\alpha]$$

This means that we want to prove the knowledge of the discrete logarithm of y to the base g and of a representation of z to the bases g and h. In addition, the h-part of this representation equals the discrete logarithm of y to the base g. This is equivalent to proving knowledge of (α, β) satisfying the equations on the rigt side of the colon.

We now look at the general case of proving the knowledge of representation. We want to prove the knowledge of the representations of $y_1, ..., y_w$ with respect to the bases $g_1, ..., g_v$.

This is denoted as

$$PKREP\Bigg[(\alpha_1, ..., \alpha_u) : \left(y_1 = \prod_{j=1}^{l_1} g_{b_{1j}}^{\alpha_{e_{1j}}}\right) \wedge ... \\ ... \wedge \left(y_w = \textstyle\prod_{j=1}^{l_w} g_{b_{wj}}^{\alpha_{e_{wj}}}\right)\Bigg]$$

where the indices $e_{ij} \ \epsilon \ \{1, ..., u\}$ refer to the elements $\alpha_1, ..., \alpha_u$ and the indices $b_{ij} \ \epsilon \ \{1, ..., v\}$ refer to the base elements $g_1, ..., g_v$. The signature consists of a $(u+1)$ tuple $(c, s_1, ..., s_u \ \epsilon \ \{0,1\}^k \times Z_n^u$ satisfying the equation

$$c = \mathcal{H}\Bigg(y_1 \mid\mid ... \mid\mid y_w \mid\mid g_1 \mid\mid ... \mid\mid g_v \mid\mid \{\{e_{ij}, b_{ij}\}_{j=1}^{l_i}\}_{i=1}^{w} \mid\mid \\ y_1^c \prod_{j=1}^{l_1} g_{b_{1j}}^{s_{e_{1j}}} \mid\mid ... \mid\mid y_w^c \prod_{j=1}^{l_w} g_{b_{wj}}^{s_{e_{wj}}}\Bigg)$$

As in 1) above, the verification of this signature is trivial since all the quantities involoved are either publicly known or explicitly communicated to the verifier.

Such a signature can be created on similar lines as in 1). This can be done as follows.

- For $i \ \epsilon \ \{1, ..., u\}$, choose random numbers r_i and compute c as
$$c = \mathcal{H}\Bigg(y_1 \mid\mid ... \mid\mid \{\{e_{ij}, b_{ij}\}_{j=1}^{l_i}\}_{i=1}^{w} \mid\mid \prod_{j=1}^{l_1} g_{b_{1j}}^{r_{e_{1j}}} \mid\mid ... \\ ... \mid\mid \prod_{j=1}^{l_w} g_{b_{wj}}^{r_{e_{wj}}}\Bigg)$$
- The prover knows the secret u-tuple $(\alpha_1, ..., \alpha_u)$ and can hence compute the parameters s_i as
$$s_i = r_i - c\alpha_i \ (modulo\ n)$$
- Now,
$$\prod_{j=1}^{l_i} g_{b_{ij}}^{r_{e_{ij}}} = \prod_{j=1}^{l_i} g_{b_{ij}}^{s_{e_{ij}} + c\alpha_{e_{ij}}} = \left(\prod_{j=1}^{l_i} g_{b_{1j}}^{s_{e_{1j}}}\right)\left(\prod_{j=1}^{l_i} g_{b_{1j}}^{c\alpha_{e_{ij}}}\right)$$
and
$$\prod_{j=1}^{l_i} g_{b_{1j}}^{c\alpha_{e_{ij}}} = \left(\prod_{j=1}^{l_i} g_{b_{1j}}^{\alpha_{e_{ij}}}\right)^c = y^c$$
$$\Rightarrow \prod_{j=1}^{l_i} g_{b_{ij}}^{r_{e_{ij}}} = y_i^c \prod_{j=1}^{l_i} g_{b_{ij}}^{s_{e_{ij}}}$$

This scheme can be further extended to provide more complex proofs of knowledge such as that of the double discrete logarithm and the eth root of the discrete logarithm. For details, refer [9].

X. Future Work

There is an excellent reference ([10]) on the use of discrete logarithms for payment systems and signature schemes. It is a 174 page thesis that starts out by explaining the preliminaries. It should be read (atleast in part) to get a thorough understanding of the protocol proposed by Camenisch and Lysyanskaya and to develop further on it.

XI. Conclusion

Anonymity is a very important requirement in today's transactions. Although very simple and elegant protocols (such as [1]) can be devised, they can not be used due to various additional requirements (see Introduction) that are needed to make the system robust and fool-proof. However, systems based on such simple primitives, that also use other proofs of knowledge can be effectively deployed for providing anonymity. One interesting approach is via the use of proofs of knowledge of discrete logarithms ([2]).

Acknowledgment

I would like to thank Dr. Deepak Gupta and Dr. Dheeraj Sanghi (both professors at IIT Kanpur) for allowing me to undertake this venture as a part of my CS397 (Special topics in Computer Science) course. They introduced me to Trusted Computing. As guides, they were very encouraging and provided complete freedom for me to pursue whatever I wanted to, in the course of this research.

REFERENCES

[1] D. Chaum, "Showing credentials without identification : Transferring signatures between unconditionally unlinkable pseudonyms," in *Advances in Cryptology - AUSCRYPT '90*, ser. Lecture Notes in Computer Science, vol. 453. Berlin, Germany: Springer Verlag, 1990, pp. 246–264.

[2] J. Camenisch and A. Lysyanskaya, "An efficient system for non-transferable anonymous credentials with optional anonymity revocation," in *Advances in Cryptology - EUROCRYPT 2001*, ser. Lecture Notes in Computer Science, vol. 2045. Berlin, Germany: Springer Verlag, 2001, pp. 93–118.

[3] *TCG PC Specific Implementation Specification v1.1*, Trusted Computing Group, Aug. 2003.

[4] Trusted computing group: Current members. [Online]. Available: http://www.trustedcomputinggroup.org/about/members/

[5] Electronic frontier foundation - give tcpa an owner overdrive. [Online]. Available: http://www.eff.org/Infra/trusted_computing/20031001_tc.php

[6] D. Chaum, "Security without identification: Transaction systems to make big brother obsolete," *Communications of the ACM*, vol. 28, no. 10, oct 1985.

[7] A. Stiglic. (2002, Sept.) Re: Cryptographic privacy protection in tcpa. The Cryptography Mailing List. [Online]. Available: http://www.mail-archive.com/cryptography@wasabisystems.com/msg02708.html

[8] U. Maurer and S. Wolf, "On the hardness of the diffie-hellman decision problem," 1998. [Online]. Available: http://www.iro.umontreal.ca/ wolf/MauWol98b.ps

[9] J. Camenisch and M. Stadler, "Efficient group signature schemes for large groups," in *Advances in Cryptology - CRYPTO '97*, ser. Lecture Notes in Computer Science, vol. 1294. Berlin, Germany: Springer Verlag, 1997, pp. 410–424.

[10] J. Camenisch, *Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem*, ser. ETH-Series in Information Security and Cryptography. Konstanz, Germany: Hartung-Gorre Verlag, 1998, vol. 2.

[11] *TPM 1.2 Main Part 1 Design Principles*, Trusted Computing Group, Oct. 2003.

[12] *TCG Software Stack (TSS) 1.1 Specification*, Trusted Computing Group, Aug. 2003.

[13] Trusted computing group: Home. [Online]. Available: http://www.trustedcomputinggroup.org/

[14] "Intel developer forum," Safer Computing Track, 2003. [Online]. Available: http://wwwp.intel.com/idf/us/fall2003/presentations/index.htm

[15] Microsoft ngscb homepage. [Online]. Available: http://www.microsoft.com/resources/ngscb/default.mspx

[16] Ibm tj watson center - global security analysis lab. [Online]. Available: http://www.research.ibm.com/gsal/tcpa/

[17] Ross anderson's tcpa faq. [Online]. Available: http://www.cl.cam.ac.uk/ rja14/tcpa-faq.html

[18] Richard stallman's essay on trusted computing (or treacherous computing). [Online]. Available: http://www.gnu.org/philosophy/can-you-trust.html