# Trusted Computing

## part I — Motivation

June 29, 2004

# Trust and Security

Miriam Webster's dictionary defines:

- **Trust -** [n] assured reliance on the character, ability, strength or truth of someone or something. (origin: true)
- **Secure -** [adj] assured in opinion or expectation. Having no doubt. Free from danger. Free from risk of loss.

# Trust and Security (cont.)

Trust is a very elusive term. In our electronic life, we need to trust various entities and programs:

- A trusted Certifying Authority (e.g., Verisign),
- A trusted server (your bank's server),
- A trusted utility (e.g., WINZIP).

Security is also a very elusive term as "secure" is in the eye of the attacker. In our daily Internet use, we assume that various sites, programs and mechanisms are secure:

- A secure block cipher (e.g., Rijndael),
- A secure public key encryption algorithm (e.g., RSA),
- A secure server that can be used by your bank,
- A secure Transport Layer Protocol (e.g., SSL).

# Trusting Your Machine

When we use our personal computer we have no idea what is truly done.

- Viruses can affect the way our operating system behaves.
- Trojan horses might send all our secret files to some IP.
- Spyware might check what files we download.
- Key-loggers can log whatever password we use (and where they were used).
- Remote control utilities (like Back Orifice) cause the computer to behave much differently than expected.
- and much, much more.

We would like to be able to trust our own machine – to know exactly what it does and that what it does is the things we expect it to do.

# Trusting Your Friend's Machine

When we use our personal computer it interacts with other computers. The other side of the interaction can be a web server, remote server, mail server, or even a node in a Peer-to-Peer network (e.g., Kazaa).
We would like to know that those computers:

- Are who they claim to be.
- Give the services they claim to give in the right way (e.g., a mail server can run anti-virus before putting the mail in the user's mail box).
- Respect and protect our privacy.

# Making Sure Your Computer is Truly Yours

There are various ways to know if your computer behaves like it should:

- Use applications from a trusted vendor. For example, Winamp is developed by a group, trusted by most of the users.
- Use only code you know and trust. For example, when you use open source, you know what the program does. Note that this approach requires a trusted compiler.
- You can debug your machine the entire time. That way any change will be noticed by your debug process.

The main problem is that those three approaches assume that the state of the computer is trusted before any of them is executed. Also those methods require the ability to distinguish between trusted and untrusted.

# Running a Server

We restrict our attention to servers, as any computer today is a server.
Each server is composed of many layers:

- The bottom layer is the hardware (CPU, Ethernet card, etc.).
- Next are the regular kernel functions – scheduling, interrupt handling, etc.
- A little above that are the drivers (part of the kernel that handles hardware).
- Next are the system's services (including daemons in UNIX, and DLLs in Windows).
- Finally come the user applications.

# Running a Server (cont.)

Consider for example a movie server that manages playing movies to clients of the system.
We shall examine the various levels that run in the server once a request for a movie is processed:

| | |
|---|---|
| `Server context (user mode)` | Parsing the request<br>Clarifying user's status<br>Requesting the file<br>Generating a reply |
| `Operating system context` | Taking care of TCP/IP connections<br>Handling the file system |
| `Hardware` | Network card, CPU, memory |

# Running a Server (cont.)

Breaking a layer, we can alter the behavior of that layer, and of all layers above it. For example, replacing the Ethernet card of a computer to one sending all the received data to the attacker, affects all the layers. Breaking into a user's account, does not break the hardware.

Note however, that sometimes breaking an higher level helps the attacker to attack lower levels. For example, breaking into some account, the attacker can use a wider range attacks to gain control of the system.

# Running a Server Revisited – Storing Sensitive Information

Many servers has sensitive information, e.g., passwords to access the server, media accessible only to users, credit card numbers. We need to decide where to store this information. Our options are as follows:

- The memory of the server process.
- A file accessible by the server process (only).
- In the kernel environment (kernel mode).
- Removable hardware.

# The Memory of the Server

Keeping sensitive information in the regular context of a process is very problematic:

- Breaking into the process means the information can be accessed.
- In some operating systems the user space memory is accessible by all processes (those OSes are actually multi-threaded).
- The information might be swapped out to the disk. As accessing the swap area might be easier than accessing the memory of the process.

Note that sometimes, we can consider the scenario when the attacker has the machine (e.g., a Chinese surfing to Fallon-Gong website, and wishes to remove traces for that action, while the Chinese police is taking the computer away).

# Files, Permissions, etc.

We can also store the secret information under a file accessed only by the server process. We can either run the server under the account of root (administrator) or under a regular user's account. However, once we break into a server, we can access all the files the server can access.

Many servers require that they be used as root. Most of them do not require root's privileges and permissions most of the time. For example, an FTP server needs to be able to access all the files of the users of the system. This can be accomplished only if the ftp server has some of root permissions. But most of the time it does not access those files. More precisely, most of the time the ability to read the contents of some account is not needed for the FTP server.

# Kernel Mode

User mode is the name for the running environment of a user program (or process). This mode is very constrained. It cannot access physical devices and access files directly. All those operations are done by asking the kernel to do these operations (using system calls).

Kernel mode is the name for the running environment of the kernel. In this environment all the process table and information about the system is found. The kernel can access whatever device it wishes, and in whatever method it desires, including memory.

Breaking a server that runs at user mode implies that only the memory of that server is compromised. All secrets stored in that server's memory or with the permissions that allow the server to access them (files) are no longer secure. Note that the remaining data on the machine is not compromised.

# Using Kernel Mode for Storing Sensitive Information

- Modern operating systems support virtual memory. Even the kernel memory can be swapped out.
- Virtual memory is the concept of having more memory than your true physical memory. The operating system simulate the additional memory by using hard drives.
- Data written on the hard drive is more susceptible to compromise. In many cases, reading the virtual memory from the hard drive is easier than reading the RAM.
- For example, if a secret is swapped out (written to the hard drive) and an electrical shortage occurs, the secret is still stored on the hard drive, and someone might access it.

# Using Kernel Mode for Storing Sensitive Information (cont.)

There are various ways to implement this separation between kernel mode and user mode. Most of them are based on catching the change of the mode and forcing those changes of modes to "behave":

- Hardware can check what mode the code is currently in (CPL field in x86 processors).
- Hardware can check that a switch from user mode to kernel mode occurs only after a software interrupt.
- Entering kernel mode can be done only in specific hook-points, controlled by the operating system.
- Accessing those addresses for write purposes can be done only by the operating system, thus the code that resides there is given, and can not be altered by the user program.

Note that these checks can help, but they do not guarantee a successful separation between kernel mode and user mode (and are not implemented at all).

# Using Kernel Mode for Storing Sensitive Information (cont.)

Thus, if we want to store secrets even in the case of breaking to the system, we would like to store them in kernel context.

This has the advantage of keeping the information secret even if the server is compromised.

However, the disadvantage is that we need to store things in the kernel context. They are still stored in memory accessible by someone (operating system) without any checks. If someone breaks the server, by using the right manipulation she can still access those secrets.

# On the Fragility of Operating Systems

As those slides are written, bugtraq's website (http://www.securityfocus.com/bid/) reports 122 vulnerabilities in Microsoft Windows XP home edition. In Linux kernel version 2.4 there are 46 vulnerabilities.

Not all of these vulnerabilities mean that the system's security can be compromised, but most of them do. Some are only denial of service conditions on the system (10 in Linux and 16 in Windows). Some are boundary conditions – things that shouldn't be there, but currently pose no immediate threat to the system (2 in Linux).

Thus, even storing secret information solely in the operating system's context is still very problematic.

A possible solution is to use removable hardware (disk-on-key, smart cards, etc.). However, those are exploitable to other attacks, are forgotten or stolen easily. Besides – once they are connected they are also susceptible to attacks, and when not connected they are not usable.

# So Why to Enhance Trust?

If we could trust the server to behave as it should, we can be sure it was not broken. If the server can verify that the operating system does what it should, then it can decide to continue functioning, knowing the system was not compromised.

If the boot loader could have trusted the BIOS, then it could check that there are no viruses present in memory before it loads the operating system. If the operating system could have trusted the boot loader, then the operating system could start running knowing that no virus is present in the memory.

If . . .

# More Ideal World Scenarios

Assume that we could bind a key to a specific platform (CPU). For example, we encrypt a file under a key $K$, and using some mechanisms, bind the key $K$ to our computer. To read the file on our system we need to type in a password (to access $K$ just like in PGP).But on other systems the file could not be accessed (even if the key $K$ is known).

The advantage of this scenario is that data can then be encrypted using a key that is bound to the platform and requires the user's authentication. Since the key is bound to the platform, even if an attacker obtains the encrypted data and the password used by the user, she cannot use it without using that specific platform. Thus, to obtain the access to the data an attacker must possess both the authentication information and the specific platform.

# More Ideal World Scenarios (cont.)

Assume that we could prove to some other computer that our computer is secure.

In that case, servers can decide to service us only if our computer is secure.

For example, the mail server might decide it is willing to receive messages from a user, only if the user proves that the mail contains no viruses.

Moreover, in that ideal world, where we can prove that our computer is secure, it might be even possible to define the level of security. Then, the server can require higher levels of security for delicate or classified e-mails.

# More Ideal World Scenarios (cont.)

Assume that we can prove that our system has not been altered. Assume that we can even prove this statement to someone else.

In that case, all the IP-spoofing, MAC-spoofing, XXXX-spoofing can be solved without the need of changing the underlying protocols. When you have a TCP session with some other host, you could be rest assured that this host's IP is the IP it claims to possess.

In many local networks (e.g., wireless networks) there is a MAC-based authentication. A user is allowed to connect only if her MAC address is in the "allowed" list. If a user can send a MAC and a proof it has not been altered, such authentication schemes would become secure (changing the MAC address of a machine is quite easy).

# More Ideal World Scenarios (cont.)

Assume that we could prove that our system behaves in some manner. For example, if we play a movie, we could prove to **the movie** that we are not capturing it and recording it.

That way, movie vendors could release their movies without us being able to copy them and re-distribute them. Music will not be copied into illegal mp3 files (though the mechanisms might allow making music into a legal mp3, for personal use only).

Digital Rights Management (DRM) mechanism would be simplified greatly. The hard labor involved in preventing the theft of intellectual property would decrease to just making sure the computer handles the content as it should.

# DRM Techniques

During the years, many techniques to enforce digital rights have been suggested. There is a continuous battle between DRM designers and DRM breakers:

- Access Code (Registration Number) — Easy to bypass, key generators, copied values, disabling the code that does the check, etc.
- Encrypting Data — Access the decrypted data, find the key coded into the binary loader, etc.
- HASP (dongles) — Disabling the checking code, removing the HASP calling from the executable, duplicating HASPs.
- Code Obfuscation — Invest more time, getting the source files. There exist a theoretical proof that this method cannot work under complexity assumptions.

Hence, history shows us that as long the user control his system, and can tweak the executable, alter the computer's behavior — no DRM would succeed.

# Trusted Computing

## Part II — Concepts and Mechanisms

# Basic Concepts – A Secure Subsystem

As we have seen, software-only based solution to enhance trust will fail. Thus, we must introduce a special, irremovable, un-bypassed hardware device to handle the concept of trust.

We need a secure subsystem that can be trusted. This subsystem will help us by managing all the trust and security requirements we have.

We require this subsystem to be:

- Minimal – The bigger the system is, the larger the probability of someone breaking it.
- Immune to Physical attacks – User tampering with the hardware must not achieve anything useful.
- Immune to Software attacks – A program cannot tamper, disable or destroy the module.
- Fast – All operations should take small amount of time.

We shall call this subsystem **Trust Platform Module** (TPM).

# Basic Concepts – A Secure Subsystem (cont.)

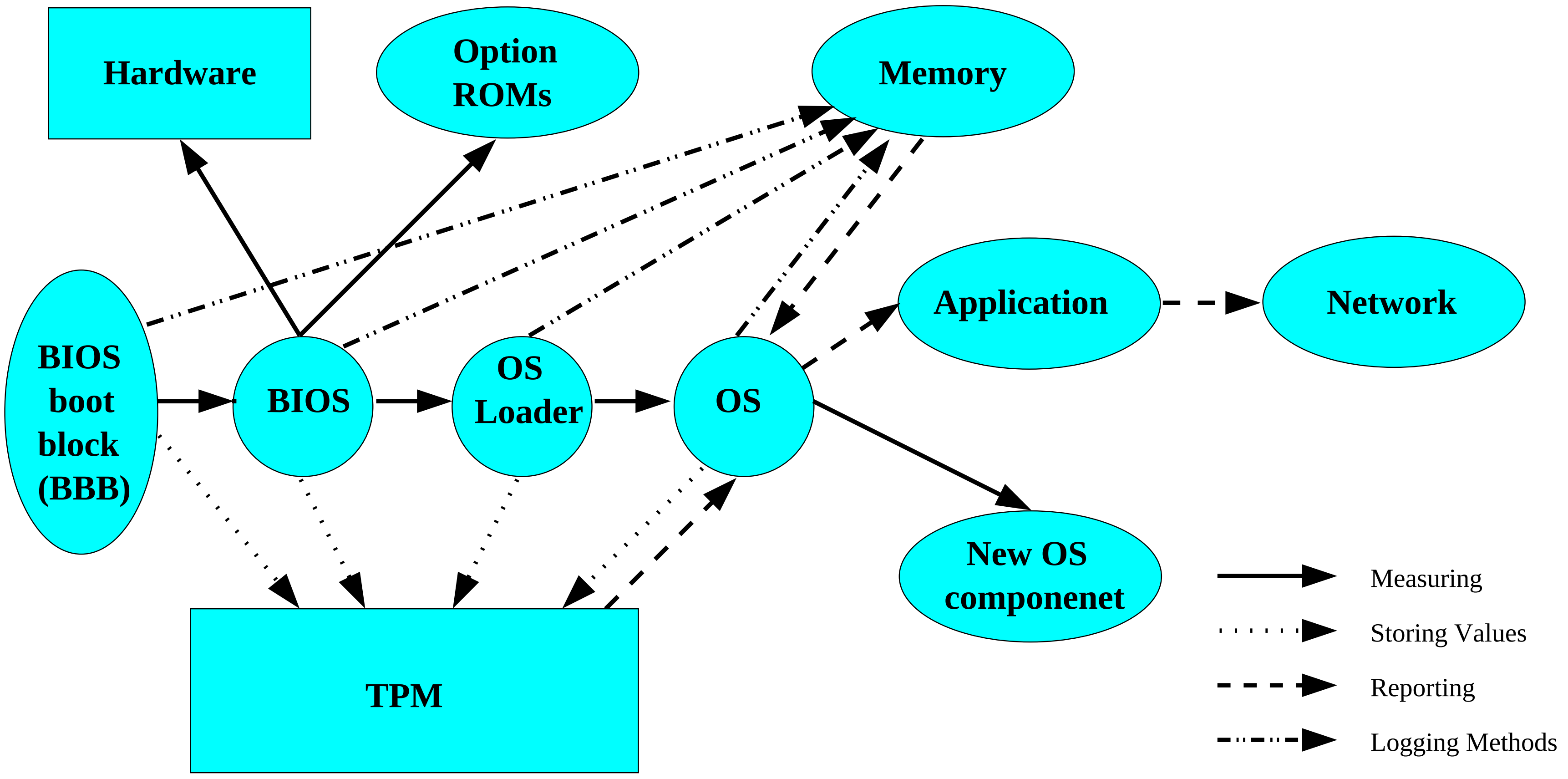
The TPM is used to check that the state of the computer is as it should be. For example, the operating system can check that the only things that have run on the computer before it loads were the BIOS and the boot loader. Not only that, the OS could check whether the legitimate BIOS and boot loader were used (thus detecting viruses that reside on the BIOS or in the boot sector).

The TPM will keep track of what is going on in the computer, will interact with various hardware devices (BIOS, motherboard, etc.) and various software handlers (boot loader, kernel, etc.). It is desired that the TPM will be able to prove the "trust" state of the computer to remote hosts.

We shall divide the discussion to the two tasks at hand:

- Making sure that the local computer is trustable.
- Being able to prove this to someone.

# Making Sure Your Computer is Truly Yours



# Making Sure Your Computer is Truly Yours (cont.)

We shall use the TPM to trust our computer in the following way:

1. The TPM will be activated before the BIOS.
2. Before the BIOS is loaded, the TPM will scan it, and verify it is a valid and trustable BIOS.
3. Once the BIOS finishes loading, it will ask the TPM permission to load the operating system loader.
4. The TPM will scan the operating system loader, making sure it is not infected by viruses and any other harmful animals.
5. Once the operating system loader is loaded, and the user chooses an operating system, the TPM will check that the operating system is allowed to run on that computer, by that user in the mode he choose.
6. Then the operating system will be loaded, and will know (after consulting the TPM) that everything is fine, it is allowed to load, and that the memory contains only what it should (no viruses).

# Making Sure Your Computer is Truly Yours (cont.)

After the operating system loads successfully (and securely) we can continue to keep the trust in our machine:

- Each hardware driver will prove to be original and unaltered.
- Each physical device will prove to be what is claims to be (in similar methods used by the BIOS).
- Everyone will communicate with the TPM, and prove to it their claims.
- The TPM will verify the claims, and announce the status to the operating system.
- The operating system will mark those drivers and devices as trustable.

In case some driver (or hardware) refuses to prove its trustiness, the operating system will decide whether to change its status to untrustable or to refrain from using this driver.

# Making Sure Your Computer is Truly Yours (cont.)

We can treat each driver and hardware device as entities proving that they are what they claim to be. For example, the network card can be viewed as an entity trying to prove that it corresponds to the networking standards.
If this network card was manufactured by a trusted company, then we can trust the card if it proves its origin. The simplest solution in this case is to have the hardware manufacturer sign the identity of the network card.
Problems (not necessarily hard ones) that need to be addressed in that case:

- What to do with unknown manufacturers?
- How to minimize dependency on manufacturer trust?
- How to make sure that the network card was not tampered?
- How to make sure that the card is standard-complaint (security standards)?

Given the above problems, the model of our solution needs to have many entities. Each vouching for a different thing.

# Who is Who in the Model?

There are many entities in the model: users, system owners, applications, servers, etc. Each has a role, a subsystem and its environment (described in general terms).
Each entity may have a role, or several roles. In some circumstances, a single entity may perform all roles. We have two kinds of entities – local and global. The local entities are those related specifically with the machine, while the global entities are not bound to a specific machine.
However, some global entities may be found in the process of proving trust in the local system, and vice versa. For example, to prove our identity to some server, we ask for a certificate. When we do so, we (local entity) send proof to a certifying authority (global entity).
The local entities are:

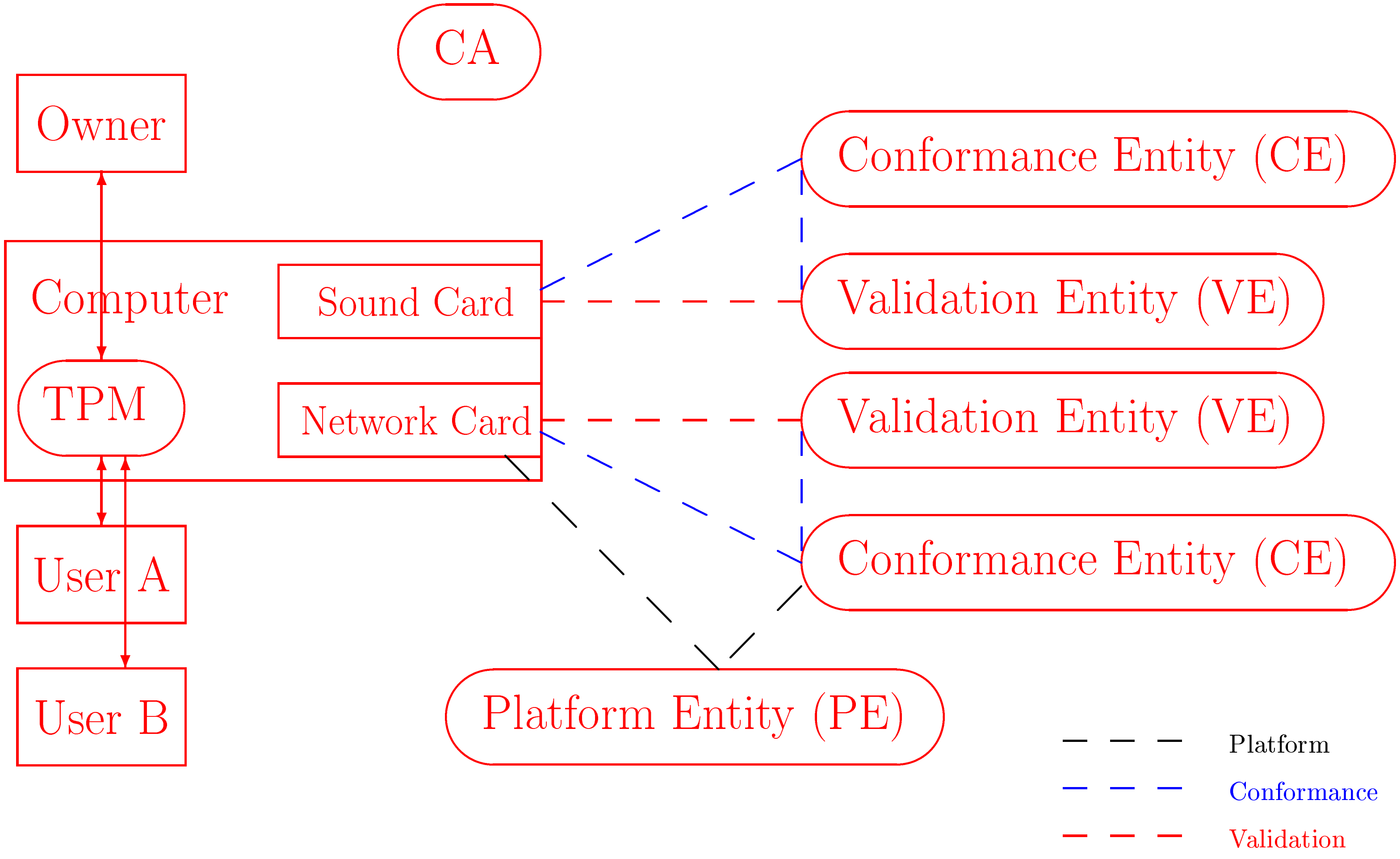
- Owner – the entity that controls the TPM (administrator).
- TPM Entity (TPME) – the entity related to the TPM.
- User – entity that uses the platform.

# Who is Who in the Model? (cont.)

The global entities are:

- Certification Authority – the entity that vouches for the identity of subsystems.
- Validation Entity (VE) – the VE states that the part can be trusted, and produces values that should be verified before the platform is trusted. There can be multiple VEs associated with a platform. In addition, the VE states **when** the part can be trusted.
- Conformance Entity (CE) – this entity vouches that a subsystem **was designed** according to the trusted computing design principals.
- Platform Entity (PE) – this entity vouches that a **specific instance** (a given device or module) was built according to the accredited design.

# Who is Who in the Model? (cont.)



# Who is Who in the Model? (cont.)

In the above example we have a computer whose owner allows two users to use it. The computer has also a sound card and a network card.

Assume that the sound card was produced by Sound Inc. Sound Inc. can assure that its sound card will not be used for illegal purposes (for example, copying MP3) under various conditions, and signs the statement as a validation entity for this sound card. Thus, Sound Inc. is the VE for their cards.

The design of the sound card was transferred to Sound-Assure Inc. They can verify that the design of the sound card is such that it assures trustiness of sound card produced by such a design. They sign that a sound card with that design can be trusted. Sound-Assure Inc. is the CE for Sound Inc. cards.

When the TPM wants to check the trustiness of the sound card, it access those two signatures and verifies the signatures. The signatures themselves can be found in many places: on the sound card itself, with the sound card drivers, in Sound Inc. website, etc.

## Who is Who in the Model? (cont.)

Similarly, the network card was designed by Network Inc., and was approved by Network-Assure Inc. (Network Inc. is the VE and Network-Assure Inc. is the CE).
Moreover, Network Inc. is very proud in the security of its products. Before selling this specific network card, they gave the network card (along with the design criteria and Network-Assure Inc. signature) to Network-Very-Assure Inc.

Network-Very-Assure Inc. examined the specific network card according to the design principals declared by Network Inc. and approved by Network-Assure Inc. and provides a signature stating that this specific card satisfies all the design criteria. Hence, Network-Very-Assure Inc. is the PE for each specific network card.
Note that Network-Very-Assure Inc. checks thousands of network cards a day made by Network Inc.
Thus, not only the TPM can trust the company Network Inc. for having a secure and trusty design, but also that the specific card in the computer was truly manufactured according to those design criteria. This gives us high level of trustiness in the network card functions.

# The Chain of Trust

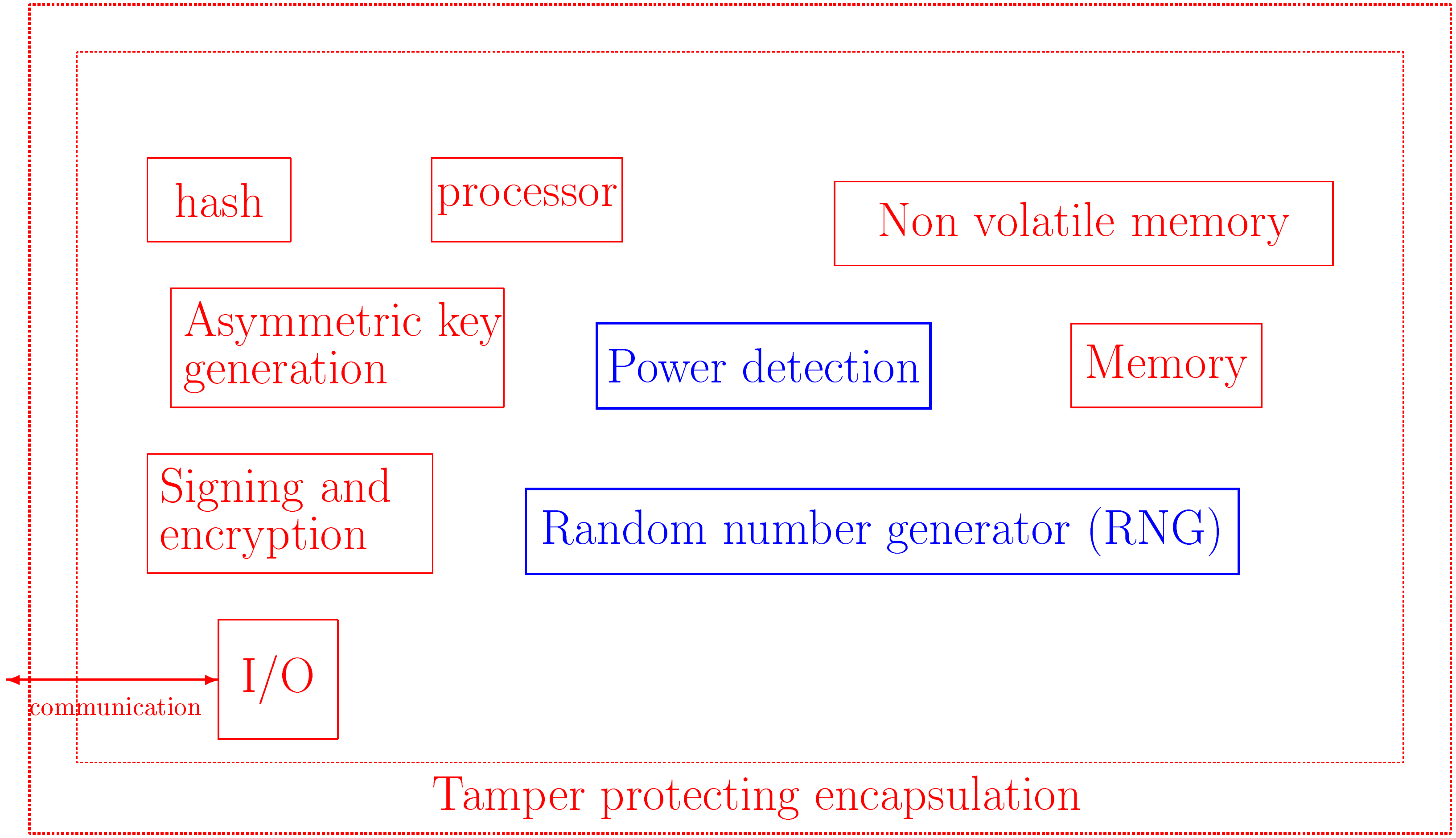
Any trust mechanism must start with some unproved trust. We need to trust some entity to some extent to start the chain of trust. for example, in the world of X.509, we trust the CAs that they issue certificates according to real life identities.

1. The certifying authority is the root of the trust chain.
2. The conformance entity (CE) vouches for the right design principals.
3. The validation entities are the roots of trust in the platform. Each VE vouches for a particular part of the platform. This is done by issuing a "validation credential". Those can be either stored on the part of the system (like in case of BIOS) or in a designated place in the system.
4. The platform entity vouches for an individual platform. It attests the entire trust mechanism (i.e., that the trust subsystem of that platform was built according to the design principals). The statement includes a reference to the conformance credentials.

# The Chain of Trust (cont.)

5. The TPM vouches for many things. It vouches to the trust-state of the machine. It also defines a public key of the platform – a public key to send private messages to the TPM. Those message are restricted to be messages related to the trust mechanisms, e.g., proving trust, validating trustworthiness of other entities.

# A Typical Trusted Platform Module



# A Typical Trusted Platform Module (cont.)

A usual TPM should have the following abilities and features:

- Symmetric cryptography support – for secure communication.
- Asymmetric cryptography support – for proving identities, secure communication, proving the trust state, checking trust states of other entities, etc.
- Hash – for various cryptographic schemes.
- Power detection unit – for detecting when the system power is on or when the unit is being powered on by an adversary.
- Random number generator – for generating truly random strings.
- Non volatile memory – for storing information in memory for long term (independent with electricity).
- Processor, memory and I/O functions – for integrating everything and communication with the outer world.

# A Typical Trusted Platform Module (cont.)

All those modules are wrapped in a special hardware that cannot be attacked physically.

We would like to have immunity to physical attacks – a user that opens the subsystem to discover its internal keys, or to manipulate its behavior using electrical devices. If such attacks exist, then remote entities could not trust the system where the TPM is installed.

Of course, the TPM should not be suspectible to software attacks as well. If the subsystem is not immune to software attacks, the user could alter the behavior of the subsystem using software. That way she can trick the subsystem, rip MP3, and do other illegal stuff.

Another reason for this requirement is that if a software attack against this subsystem exist, it may be used by a hacker to change the behavior of the operating system.

# What if a Measurement Fail?

There are two possible states in case of a failed measurement. Each is determined by the mode of the boot process:

- Authenticated Boot Process – The process without considering failures as a problem.
- Secure Boot Process – This process is used whenever the system needs to boot to the same state. In case of a mismatch between the integrity of the data and the integrity measured, the system deals with the mismatch according to its configuration.

# Palladium – NGSCB

Palladium is the name of the TCG (trusted computing group) supporting operating system from Microsoft. As Palladium became a target for lots of criticism, Microsoft changed the name of that initiative to be Next-Generation Secure Computing Base for Windows, or shortly NGSCB.
Microsoft's announced that there will be some similarities between TCG and NGSCB, but they do not guarantee that.
The main idea is to have a nexus, a security agent, running in parallel to the OS, and providing the security services required:

1. Memory Curtaining – Each process can access only its memory.
2. Sealed Storage – Storing information in a way only the application can access.
3. Secure communication with the user – Verifying no key loggers are present, that the keyboard is truly the one used by the user, etc.
4. Attestation – The ability to attest the trust/security state to other entities.

# Palladium – NGSCB (cont.)

In the world of TCG, the TPM hardware key is used to store keys. If a failure occurs in the TPM, we could not retrieve the data. Microsoft declares that NGSCB will address this problem. Note, that the problem of recovering encrypted data in the case of key loss is a hard problem if the encryption mechanisms are strong.

Another feature NGSCB offers, is to have secure communication with the input/output devices. For example, the interaction between the keyboard and the OS will be encrypted under a session key. Thus, even if we listen to the communication lines between the keyboard and the OS, we can not learn anything besides the fact a key was pressed.

# On Privacy and Freedom of Choice

As far as security is concerned, the TCG assures that the computer would be secure. So where is the problem?

Many accuse the TCG initiative of causing problems while solving security issues. These problems have much to do with privacy, freedom of choice, etc.

One main problem is the initial trust. You need to trust the CEs and PEs used in the TCG model, to start the trust chain. What if no CEs and PEs are truly trusted?

As the task of validation of the security requirements is not computationally feasible, even the most dedicated CE could not sign its statement with 100% assurance.

Another main problem is the fact that the TCG could be used to reduce the freedom of choice an owner has on her system.

# On Privacy and Freedom of Choice (cont.)

**Problem: Third-party uncertainty about your software environment is normally a feature, not a bug**

The remote attestation is one of the features that jeopardize an honest user. This can be used against the user (or owner) to impose restrictions on what she can (or cannot) do.

When we design a security solution, we must specify the threat model. The TCG assumes that the owner is also a threat. It is usually not the case.

Another concern is the fact that preventing one attack may lead to new attacks. For example, adding heavy cryptographic computations for verifying trust, can be used for denial-of-service attacks.

# On Privacy and Freedom of Choice (cont.)

EFF's suggests in their website that:

Attestation is appropriate for the purpose of preventing the software on a computer from being changed without the knowledge of the computer's owner (for instance, by a virus). Unfortunately, the attestation model in TCG's current design can equally effectively prevent the software on a computer from being changed deliberately by the computer owner with his or her full knowledge and consent. While the owner is always free to alter software, attestation adds a new risk: doing so may now eliminate the computer's ability to interoperate with other computers.

Because third parties currently have no reliable way to tell what software you are using, they have no reliable way to compel you to use the software of their choice. This aspect of the status quo is almost always a benefit for computer owners – in terms of improved competition, software choice, software interoperability, and owners' ability to control their computers – and therefore no attestation scheme that changes this situation is likely to be beneficial to consumers.

## Examples of Hurting Privacy and Freedom of Choice

1. Forcing users to use a specific program – many sites today support Internet Explorer only. Not only this is against any Internet standard, it prevents people from using other browsers. However, many browsers know to present themselves as Internet Explorer, thus, bypassing the browser check imposed by the web site. In a TCG-world this will not work.

2. Preventing interoperability – by demanding attestation that the OS is of one kind and not another, we can prevent running programs in the "wrong" OS. We can also lock data to be opened only by a specific platform in a specific environment. Thus, preventing migration from one OS to another.

3. Tethering – locking information to a specific platform can be good, but may have down sides. Assume you find out that your corporation is exploiting transistors in hard labor, and you wish to tell the press about it (in order to stop the exploit). If all the documents related to the transistor-labor are tethered to the corporation's computers, you cannot do it.

# Examples of Hurting Privacy and Freedom of Choice (cont.)

4. Forced change of version – currently you can use whatever version of software you would like. In the TCG-world, someone can force you to upgrade to a newer software, by binding data to the software.

5. Preventing privacy – today there is anonymity to some extent in the Internet. A Chinese person who uses the Internet to surf into "democratic" sites can do so without the risk of the government finding that. In the TCG-world, as communications will be authenticated, then the government can find this out.

6. Automatic document destruction – we can add a feature saying that any email 6 month old is deleted forever from the records. Some companies would like that ability (Enron case, Microsoft anti-trust case, etc.). TC can also implement fancier controls: for example, if you send an email that causes embarrassment to your boss, she can broadcast a cancellation message that will cause it to be deleted wherever it's got to.

(Those scenarios are taken from Ross Anderson's website).

# More Reading

Whether you think trusted computing is the solution, or you think it is the problem, there are many voices out there:

- Trusted Computed Platform Alliance — http://www.trustedcomputing.org
- Trusted Computed Group — http://www.trustedcomputinggroup.org
- Microsoft's Next Generation Secure Computing Base – Technical FAQ — http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/news/NGSCB.asp
- Ross Anderson's TCPA FAQ — http://www.cl.cam.ac.uk/~rja14/tcpa-faq.html
- EFF's Trusted Computing: Promise and Risk — http://www.eff.org/Infra/trusted_computing/20031001_tc.php

# Acknowledgments

I would like to thank Prof. Eli Biham for his great help in writing the lecture. Many talks and conversations among us have helped me to grasp the trusted computing initiative.