Modeling TCG-based Secure Systems with Colored Petri Nets[⋆]

Liang Gu[†], Yao Guo[†⋆⋆], Yanjiang Yang [§], Feng Bao[§], Hong Mei[†]

[†]Key Laboratory of High Confidence Software Technologies (Ministry of Education), Institute of Software, School of EECS, Peking University, China.
[§]Institute for Infocomm Research, Singapore.
[†]{guliang05,yaoguo,meih}@sei.pku.edu.cn, [§]{yangyj, fengbao}@i2r.a-star.edu.sg

Abstract. With the rapid progresses in trusted computing related research and application, many trusted computing based security mechanisms have been proposed to defend against threats in open, dynamic and distributed environments. These mechanisms are supposed to serve as the security foundations in the underlying systems. However, the correctness of these security mechanisms still require further examination and validation. We propose a Colored Petri Nets (CPN or CP-nets) based approach to model the trusted computing based secure system. In particular, with CPN, we model process management, data protection and late launch mechanisms in the systems. Further, as case studies we use these models to investigate the memory protection mechanism in TrustVisor and remote attestation based on dynamic root of trust, respectively; and the results demonstrate that our models are indeed capable of depicting real secure system based on trusted computing. With the advantages of CPN based modeling and analysis (e.g., graphical representation, well defined semantics and a large number of formal analysis methods), our models can well serve as the foundation for formal analysis on the security properties of trusted computing enhanced systems.

1 Introduction

Internet has become a critical infrastructure for information dissemination and sharing in our society. In this open, dynamic and distributed environment, it is, however, difficult to defend against software attacks in systems that span mutually distrust domains. As a countermeasure, Trusted Computing Group (TCG) has proposed a series of specifications on trusted computing [44], in which the computer hardware is extended with a Trusted Platform Module (TPM) [43]. The main idea of trusted computing is as follows: using the TPM as the root of trust, a trust chain can be built from the TPM to the

⋆ This work is supported by the National Basic Research Program of China (973) under Grant No. 2009CB320703, the Science Fund for Creative Research Groups of China under Grant No. 60821003, National Key S & T Special Projects under Grant No. 2009ZX01039-001-001 and the National High-Tech Research and Development Plan of China under Grant No.2009AA01Z139, and National Natural Science Foundation of China under Grant No. 60873238.

⋆⋆ corresponding author

final users through the operating system and upper layer applications. Since the introduction of TCG specifications, many TCG-based secure systems[1] have been proposed from both the industrial and academical domains, such as the IBM Integrity Measurement Architecture [40], the BitLocker [35] from Microsoft, Terra [13], Flicker [34], TrustVisor [33]. Needless to say, the effectiveness of these secure systems is largely dependent upon the *correct* usage of the trusted computing related technologies within. Thus, it is crucially important to correctly build Trusted Computing Base with TCG technologies, and guarantee that the trust chain is built up to the applications.

However, the theoretical study on trusted computing lags far behind its development. Indeed, for most of the existing trusted computing based secure systems, there still lacks formal verification on the (system) correctness from a theoretical point of view. From the perspective of formal study, we refer to the *correctness* of a secure system as that it functions according to the expected specifications. In this paper, our particular concerns are security related specifications. Recent formal studies on trusted computing mainly fall into two categories: on TCG specifications and on TCG-based secure systems. For the former (correctness study on TCG specifications), the majority of the existing formal modeling and analysis studies focus on key security mechanisms in the specifications [47, 18], besides testing based analysis techniques [38, 46]. Usually formal verification [47] and the universally composable (UC) framework [18] are employed to analyze the security properties. For the latter, analysis and verification of the TCG-based security mechanisms mainly rely on testing based methods, and only a few formal studies were proposed [1]. It is thus fair to say that formal verification on TCG-based secure systems still has a long way to go. On the positive side, the recent study of Logic of Secure System [6] based on the Protocol Composition Logic (PCL) [5] demonstrates the possibility of employing formal methods to analyze TCG-based secure systems.

In this paper, we propose a formal study on TCG-based secure systems with Colored Petri Nets (CPN)[22–24]. In particular, CPN is employed to model and analyze the key security mechanisms in the systems, including process management, data protection and late launch. With the proposed models, we further conduct two case studies: the memory protection mechanism in TrustVisor [33]; and remote attestation based on dynamic root of trust [19, 44, 3].

The advantages of CPNs make it appropriate for modeling and analyzing TCG-based secure systems. Our CPN based approach has the following benefits: (1) the capability of graphical representation of CPN makes it possible to simulate the key mechanisms in a complex TCG-based system in a graphical way; with the colored token in CPN, it is possible to model complex system states; (2) the well defined semantics of CPN can exactly define the behavior of a target system that is augmented with TCG, and the widely studied CPN provides solutions to check properties of the target system including liveness and safety; (3) with a large number of formal analysis methods and tools that admit CPN, it is possible/convient to model and verify different security properties of a TCG-based system.

[1] Without special notice, we will use the terms: secure system, secure system based on trusted computing and TCG-based secure system interchangeably in this paper.

2 Preliminaries

2.1 Trusted Computing

In the TCG specifications, a trusted platform is supposed to provide features such as secure storage and platform remote attestation. Trusted Platform Module (TPM) is the core and root of trust for a trusted platform. TPM is a tamper resistant, self-contained coprocessor, capable of providing a series of security functions, such as random number generator, key management, digital signature and hash function, etc. TPM contains a group of internal Platform Configuration Registers (PCRs). Equipped with a SHA-1 engine, TPM employs PCRs to provide PCR extend operation on specified machine state data m: $PCR\_Extend(m) : PCR_i \leftarrow SHA - 1(PCR_i||m)$. The content of the PCRs represents a integrity measurement of the state of the underlying machine. *Remote attestation* is a mechanism enabling the platform to attest to a remote entity of its integrity measurement contained in the PCRs.

There are two types of trust chain for TCG based secure systems: one is built based on the static root of trust for measurement since the system's booting up [44]; the other is based on the dynamic root of trust for measurement which is supported by the late launch [19, 3]. The late launch can dynamically create a secure execution environment by setting up a secure virtual machine or secure kernel. The late launch mechanism works as follows: after receiving the request of late launch from the system, the CPU executes a privileged instruction ($GETSEC[SENTER]$ for Intel, $SKINIT$ for AMD); the privileged instruction will enable several hardware protection mechanisms, including disabling Direct Memory Access by setting the relevant bits in the system's Device Exclusion Vector (DEV), disabling the interrupts and debugging access; then the process will execute a specified loader module(Secure Loader Block (SLB) for AMD, Authenticated Module for Intel) at a given address; then the specified loader module will securely measure and load a secure execution environment (Secure Virtual Machine for AMD, Measured Launched Environment for Intel). Late launch allows to set up a secure execution environment without rebooting the system, thus enabling an application to run in a protected environment.

2.2 Colored Petri Nets

Colored Petri Nets (CPNs) were introduced as a full-fledged language for the design, specification, simulation, validation and implementation of large software systems. CPNs combine the strength of Petri nets with the strength of programming languages and are widely employed in both academical and industrial areas for software system design, implementation, simulation and validation. CPNs have a series of good features that make it suitable for modeling and analyzing complex systems [22, 24].

Below, we will give a brief introduction to CPN. For the definition of *multi-set*, please refer to [22, 24]. For a set A, we use A_{MS} to denote the set of all multi-sets over A, and 0 to denote the empty multi-set. Let A be a set. The power set of A, denoted as $PowerSet(A)$ or 2^A, is the set of all subsets of A including the empty set.

A **Colored Petri Net (CPN)** can be represented as a tuple $(\Sigma, P, T, A, N, C, G, E, M_0)$, where

- Σ is a finite set of *color sets*, P, T and A are finite sets of *places*, *transitions* and *arcs*, respectively. $P\bigcap T$=$T\bigcap A$ = $A\bigcap P$ =$\emptyset$, and $A\subseteq P\times T\bigcup T\times P$. There are two types of arcs for a transition: incoming arc and outgoing arc. When a transition takes place, incoming arcs indicate that the input places shall remove specified number of tokens, while outgoing arcs mean that the output places should add the specified number of tokens. The exact number is determined by the *arc expression*, defined by the expression function E.
- N is a *node* function $N : A \rightarrow P\times T\bigcup T\times P$ and it specifies the source and destination of an arc.
- C is a *color* function, and $C : P \rightarrow PowerSet(\Sigma)$. $C(p)$ specifies the set of allowed colors for any token of place p. A token element is a pair (p, c), where $p\in P$ and $c\in C(p)$. The set of all toke elements is denoted as $TE = \{(p, c)|p \in P\bigwedge c\in C(p)\}$.
- G is a *guard* function, mapping each transition t to a boolean expression $G(t)$. Let $\mathbb{B}$ stand for boolean type, which contains the elements $\{true, false\}$. Let $Type(v)$ denote the type of the variable v, and $Type(expr)$ stands for the type of the expression $expr$. $Var(expr)$ denotes the set of variables in expression $expr$. So $\forall t\in T : Type(G(t)) = \mathbb{B}\bigwedge Type(Var(G(t)))\subseteq \Sigma$.
- E is an *arc expression* function, mapping each arc into an expression with type $C(p)_{MS}$ (multi-set over $C(p)$), where p is the place of the given arc. That is $\forall a\in A : Type(E(a)) = C(p)_{MS}\bigwedge Type(Var(E(a)))\subseteq\Sigma$.
- M_0 is the *initial marking* of CPN, and $M_0\in(TE)_{MS}$.

In order to describe the behavior of CPN, we use the following notations for all $t\in T$ and for all node pairs $(x_1, x_2)\in P\times T\bigcup T\times P$: $A(t) = \{a\in A|N(a)\in P\times\{t\}\bigcup\{t\}\times P\}$; $Var(t) = \{v|v\in Var(G(t))\bigvee\exists a\in A(t) : v\in Var(E(a))$; $A(x_1, x_2) = \{a\in A|N(a) = (x_1, x_2\}$; $E(x_1, x_2) = \sum_{a\in A(x_1,x_2)} E(a)$.

A *binding* of a *transition* t is a function b defined on $Var(t)$, which satisfies the following conditions: $\forall v\in Var(t) : b(v)\in Type(v)\bigwedge G(t) < b >$. For variable set $Var(t)$, b associates each variable $v\in Var(t)$ with an element $b(v)\in Type(t)$, and a binding b is usually expressed in the form $< v_1 = c_1, v_2 = c_2, ..., v_n = c_n >$, where $Var(t) = \{v_1, v_2, ..., v_n\}$ and $c_1, c_2, ..., c_n$ are the data values such that $c_i\in Type(v_i)$ for $1\leq i\leq n$. Expression $expr < b >$ means the value obtained by evaluating the expression $expr$ by substituting the variables in $Var(expr)$ as their values in b. $G(t) < b >$ means that the binding b satisfies the corresponding guard and its value should be $true$. The set of all bindings for transition t is denoted as $B(t)$. A binding element is a pair (t, b), where $t\in T$ and $b\in B(t)$. The set of all binding elements is denoted by BE.

A *marking* is a multi-set over TE and a *step* is a non-empty and finite multi-set over BE. The sets of all markings and steps are denoted as $\mathbb{M}$ and $\mathbb{Y}$. Let $M(p)$ denote the marking of a place p in the marking M. A *binding element* is *enabled* in a marking M iff the following property is satisfied: $\forall p\in P : \sum_{(t,b)} E(p, t) < b >\leq M(p)$. When a binding element (t, b) is enabled in the marking M_1, which is denoted by $M_1[(t, b)\rangle$, the transition t may be activated and the marking may be changed from M_1 to M_2. We write this change as $M_1[(t, b)\rangle M_2$. When a step Y is enabled in the marking

M:$\forall p \in P : \sum_{(t,b)\in Y} E(p,t) < b >\leq M(p)$. Similarly, when a step Y is enabled to change the marking from M_1 to M_2, then the change is denoted as $M_1[Y\rangle M_2$.

3 Modeling TCG-based Secure Systems with CPN

A TCG-based secure system builds the trust chain from the TPM up to the application layer. As shown in Figure 1, the upper layer applications send requests to the secure system via the interfaces provided by the secure system, and the secure system handles these requests and provides the corresponding security services to the applications, such as TPM based sealed storage, setting up secure execution environment based on late launch. The secure system serving as the security foundation for the upper layer applications, can present itself in various forms, e.g., virtual machine monitor, secure kernel, and operating systems. A formal study on the correctness of the secure system can provide definite confidence on the prescribed security functionalities promised by this foundation. In this section, we will model the mechanisms in TCG-based secure system with CPN.

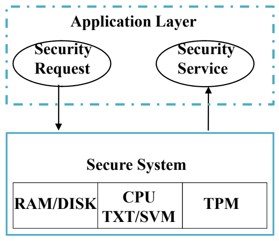

Fig. 1. TCG-based Secure Systems

The security properties of a computer system are mostly reflected by its behaviors, that is how the system behaviors handle specific information and whether they are in compliance with some given specifications. Computer systems usually deal with two types of information from the logical point of view: the data and program code, all of which physically exist in the form of data on specific storage devices; from the physical existence point of view, all data are located in RAM, DISK, CPU registers, TPM registers and other storage units. The operations that may change the states of data on computers are as follows: write, read, system reset, CPU computing.

Isolation is the basic method for information protection in systems. For example, the virtual machine monitor isolates different domains and protects the processes running in a domain from attacks from another domain. For data protection, some privileged tags are usually employed to represent its protected states, thus restricting access to the protected data. In the system level, these privileged tags are directly bound with the locations of storage units. The CPN is capable of depicting the behaviors of secure systems and reasoning about their properties.

Focusing on the internal mechanisms of TCG-based secure systems, we employ the Constrained by the System Interface Adversary model (CSI_ ADVERSARY model) [14] for modeling and analyzing secure systems. All the attacks on the secure system should only be conducted by invoking the interfaces provided by the secure system, and there should be no other approaches for attacking or manipulating the secure system. For example, an attacker who wants to modify some specified information in the secure system, can only operate by invoking the system interfaces. It is also possible to conduct injection, interception and replay attacks on the secure system, and all these attacks are assumed to take place based on the operations on its open APIs. With this adversary model, the security property analysis on the secure system can concentrate on the system behaviors.

With CPN, we try to make our model expressive enough to depict the main characteristics of the system, and reason with enough semantic support. Meanwhile, we try to keep the system model at a reasonable level of abstraction without dealing with too much lower level details. With the hierarchy representation capability of CPN, we will first give a high level model and then refine the elements in the model to more fine-grained models. We assume that some basic operations in a system are correct or verified in advance, such as read and write operations in the system, TPM operations like TPM extend, DMA operations etc. However, these lower level operations can still be further studied with CPN or other formal methods.

3.1 Basic Model

The formal analysis on the secure system concerns whether the data and process related behaviors in the system are in compliance with the given specifications. The TCG-based secure system can be modeled as a Colored Petri Net: $(\Sigma, P, T, A, N, C, G, E, M_0)$. We use colored *tokens* and *places* in CPN to denote the states of data and processes in the secure system, and the operations on data and processes can be denoted as *transitions* in CPN. The *place* sets denote the major state categories of data and processes, and *color* sets denote the types of data and processes in these categories. The links from actions to states and states to actions are denoted by *arcs*. The required states of processes and data for enabling an action, and the states of processes and states after the action finishes can be modeled as an *arc expressions*. The constraints on system behaviors can be modeled as the *guard* function on transitions.

Data is the most important concept in secure systems, and the target objects of all operations in the system can be considered as data. From the perspective of security objectives, data expression $e \in Data$ can be denoted as following types:

$$Data = \{Num, K, K^{-1}, Sig\{e\}, Enc_K\{e\}, Symenc_K\{e\}, H(e), Prog, \\ Bool, Rec, Loc, NULL\}$$

Num stands for numbers. K and K^{-1} stand for a public/private key pair, and K^{-1} is the inverse of key K. $Sig\{e\}$ is the value of the data e signed by private key K. $Enc_K\{e\}$ is the value of date e encrypted with public key K. $Symenc_K\{e\}$ is the value of date e encrypted with symmetric key K (in this case, $K == K^{-1}$). $H(e)$ is the hash value of date. $Prog$ stands for program in the form of data. $Bool$ stands for data of boolean type, and $Bool = \{true, false\}$. REC stands for data of record type, which is used to record the states of objects. For example, the record of a data expression e can be denoted as $\{H(e), Sig_{AIK^{-1}}\{H(e)\}\}$. We use r to denote a record instance. $NULL$ is a special notation standing for no data.

Loc is the storage location of data, which can be denoted by two factors: the location type and the location address. We consider the following types of physical storage location on computer: RAM, Disk, TPM static registers and dynamic registers, CPU registers. We use the following location types $LocType$= $\{RAM, Disk, TPM.spcr, TPM.dpcr, CPU.reg\}$ to denote the types of the above storage locations. The location address of storage can be denoted by two numbers: $LocAddr \to Num \times Num$, where the former is the start address and the latter is the end address: (st, ed). Then we have $Loc \to LocType \times LocAddr$.

If $\exists(st_1, ed_1), (st_2, ed_2) : st_1 \leq st_2 \bigwedge ed_1 \geq ed_2$ then we say (st_1, ed_1) is located in the address of (st_2, ed_2), which is denoted as $(st_1, ed_1) \subseteq (st_2, ed_2)$. We use l to denote a location instance. When a location l_1 physically exists in the area of another address l_2, we say $l_1 \subseteq l_2$. That is : $\exists l_1, l_2 : l_2.st \leq l_1.st \bigwedge l_2.ed \geq l_1.ed \Rightarrow l_1 \subseteq l_2$.

Process is an important runtime concept in secure systems, and it is denoted as $Proc \to Prog \times Next \times Para_{in} \times Para_{out}$, where $Next : \{prog_i | prog_i \in Prog\}$, $Para_{in} : Data$, $Para_{out} : Data$.

In the following discussions, we will use pro to denote a variable or instance of process type, e for a variable or instance of data type, con for a variable or instance of boolean type.

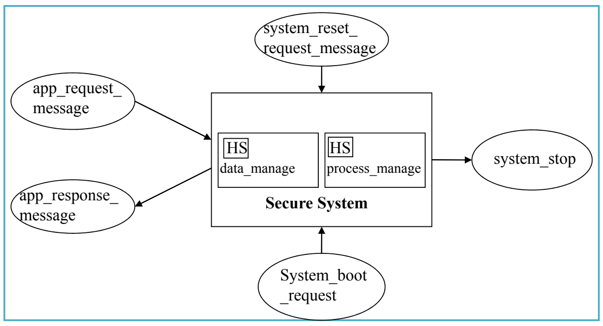

Fig. 2. Overview of the CPN model for a TCG-based Secure System

A higher level of the CPN model of a TCG-based secure system is shown as Figure 2. The secure system commonly deals with the following events: accept and handle the data and process management requests from the upper layer applications; return the service or processing results to requesters; handle the system boot/reboot/shutdown requests. We consider the system boot and reboot as parts of the secure system because they can change the states of data and process in a system. The service requests from the upper layer applications have two types: one is the data protection request which is about security operations on specified data, such as data encryption or decryption, signing data with TPM, lock or unlock specified data area; the other type is the process handling request, such as the request for creating a secure execution environment for the specified program, or for terminating a specific process. These two types of requests are corresponding to the two places in Figure 2: $app\_request\_message$ and $app\_response\_message$, respectively.

The secure system provides the above services by combining two mechanisms: the data management and the process management. The $data\_manage$ and $process\_manage$ transitions in Figure 2 are corresponding to these two mechanisms.

The secure system based on trusted computing supports two types of system boot: secure boot and authenticated boot. Secure boot guarantees the system will boot into a secure state, while authenticated boot will record all the boot information to support attestation on the boot process. The system reboot and boot requests are corresponding to the places $system\_boot$ and $system\_reset$. The place $system\_stop$ stands for the system shutdown.

I/O operation is an important factor in real systems. However, in most of the existing studies [34, 33], the I/O operations are considered as being untrusted in secure systems. For example, the DMA operations are considered to be unsafe for secure systems. Usually the asynchronous I/O should be disabled in privileged mode of the secure system by disabling all interrupts, and it is in fact the case for existing secure system implementations. When synchronous I/O are involved, the driver of the device should also be considered as part of the secure system, like the drivers for TPM security chip.

For simplicity, Figure 2 does not specify the arc expression, as the later sections will further refine these transitions.

3.2 Process Management

One of the key mechanisms in secure systems is process management, by which the secure system provides security services to the upper layer applications. From a structural perspective, the secure system usually consists of different programs or modules. From the runtime point of view, most of the existing secure systems are designed to execute in a single thread mode, e.g., Flicker[34] and TrustVisor[33]. In this paper, we also first consider the secure system executing in single thread mode. However, as CPN can naturally reason on the concurrency in systems, we may also discuss some concurrency features of the secure system in the future, and thus reflecting multi-cores for secure systems.

We consider the execution of the secure system in two categories: one is the situation when the system is operating on some business logics, like the computational tasks, and the operation does not change the security state of the secure system; The other is when security operations change the security state of the secure system, such as data encryption and decryption, late launch and seal storage, etc. For the first kind of execution states, we record the system state as $proc\_ex$. Figure 3 is the CPN model for the process management mechanism in a TCG-based secure system. This model is generated by refining the higher level model in Figure 2.

As mentioned above, the place $proc\_ex$ stands for the executing state of the secure system when no security operations are involved. The tokens in places app_rq_msg and app_rs_msg denote the requests and responses between the secure system and upper

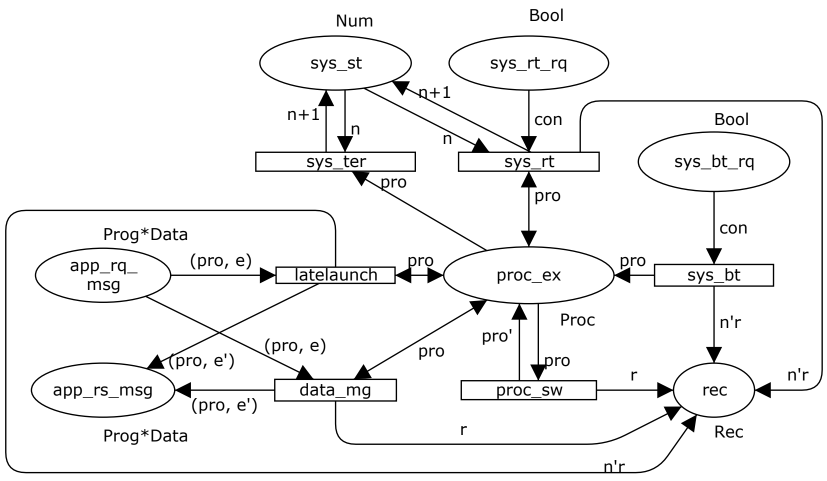

Fig. 3. Process management

layer applications. The tokens in place sys_rb_rq and sys_bt_rq stand for the requests on system reboot and boot, respectively. The token in place sys_st stands for the number of system shutdown. The token in sys_st also indirectly reflects the number of system boot, because during the execution of the secure system, the number of system boots is always one more than the number of system shutdowns. The tokens in place rec stand for the runtime records of objects in the secure system based on the TPM. Thus we get the place set for the process management mechanism: $P_1 = \{proc\_ex$, app_rq_msg, app_rs_msg, sys_rb_rq, sys_bt_rq,sys_st, rec $\}$.

In order to depict the tokens in these places, we introduce the following color sets:

$$\begin{gathered} C_1 = \{\ C_1(sys\_st) = Num, C_1(sys\_rt\_rq) = Bool, C_1(rec) = Rec, \\ C_1(sys\_bt\_rq) = Bool, C_1(app\_rs\_msg) = Prog \times Data, \\ C_1(app\_rq\_msg) = Prog \times Data, C_1(proc\_ex) = Proc\} \end{gathered}$$

In the single-threaded secure system, the following operations can change the security state of the system: process/module switch, system boot, system termination, system reset, late launch, data management. Thus we get the *transition* set for the CPN model of process management mechanism:

$$T_1 = \{\ sys\_bt, proc\_sw, sys\_ter, sys\_rb, latelaunch, data\_mg\ \}$$

The *arcs* set and *arc expression* function are shown in the Figure 3. Specially, there is a constraint on the arc expressions between the place $proc\_ex$ and the transition $process\_switch$:

$$E_1(proc\_ex, proc\_sw).Next == E_1(proc\_sw, proc\_ex).Prog$$

which means the order constraint of process switch.

Tokens in place sys_rt_rq and place sys_bt_rq denote the requests corresponding to system reboot and boot respectively, and they are of the type $Bool$. Only when the value of the token is $true$, the corresponding transition can be enabled. We use the *guard* function to express this constraint:

$$G_1(sys\_rt) = (con == true), G_1(sys\_bt) = (con == true)$$

When the system reboot request is true or the system boot is true, the corresponding transition should be enabled under any condition. That is :

$$\forall(sys\_bt, b) \in BE, \forall M_1 \in \mathbb{M} : (con == false) \Rightarrow \nexists M_2 \in \mathbb{M} : M_1[(sys\_bt, b)\rangle M_2$$
$$\forall(sys\_rt, b) \in BE, \forall M_1 \in \mathbb{M} : (con == false) \Rightarrow \nexists M_2 \in \mathbb{M} : M_1[(sys\_rt, b)\rangle M_2$$

which means: in any condition, if a condition con is true and the transitions sys_bt and sys_rt are enabled, then they must occur, i.e., the boot or reboot action must be allowed. sys_bt and sys_rt are enabled means that beside the condition is true, the system also requests these actions.

The secure system consists of different programs and modules, and it is common that program/module switch happens during the system execution. The CPN model for process management should specially depicts the process switch. The process switch happens in two situations: when a module terminates its execution and starts another module/program; the executing module jumps to another module by invoking instructions like “JMP”. The process switch can only happen when the executing process has at least one next program to execute, and this is denoted as a guard function on transition

$$proc\_sw\text{: } G_1(proc\_sw) = (pro.Next \neq NULL)$$

Considering the secure system as a single-threaded system, the CPN model in Figure 3 only depicts the process switch while without taking into account the context switch. If the context switch is considered, it is natural to refine the transition $proc\_sw$ into two parts: with and without context switch.

System reboot may change the security state of the secure system. For example, in a remote attestation scenario, after the platform configurations are transferred to the challenger, the system can immediately reset the system to boot into a compromised system. The solution for this reboot is to introduce a boot counter which records the number of system boots. Our CPN model depicts this mechanism with an alternative solution: the sys_st place holds the token whose color stands for the number of system shutdowns.

The late launch mechanism is responsible for setting up a secure execution environment for given program codes. The data management mechanism is based on the hardware memory protection techniques to manage and protect the access to data. We will model these two transitions in the following subsections.

3.3 Synchronous I/O

There are two types of I/O: synchronous and asynchronous. In the designs of existing secure systems, interrupts are usually disabled in privileged mode, thus the asynchronous

I/O is not applicable for modeling secure systems. The I/O devices are usually accessed in the synchronous way, such as the TPM security chip.

We will take the TPM as an example to depict the synchronous I/O operations in the secure system. Let's consider the TPM_PCR_read operation in TPM. When place $proc\_ex$ has a token $pro_1 = (program_1, \{TPM\_PCR\_read\}, para_1, para_2)$; then after the transition $proc\_sw$ occurs, the token in $proc\_ex$ is replaced by $pro_2 = (\ TPM\_PCR\_read, \{program_1\}, para_2, para_3)$; when TPM_PCR_read finishes execution, the transition $proc\_sw$ occurs again and the token in $proc\_ex$ is replaced by $pro_3 = (program_1, \{program_2\}, para_3, para_4)$, where $program_2$ is the next process for $program_1$.

3.4 Data Management

Restriction on data access is an important mechanism to guarantee the integrity of program code and data. By modeling the data access control, we may analyze the data protection mechanism in secure systems. Secure systems usually use tags to identify the access privileges on specific data areas, and restrict applications from accessing to these areas. In this subsection, we model the data protection mechanism by refining the transition $data\_mg$ in Figure 3. Consequently, Figure 4 is the CPN model for data management mechanism in the secure system.

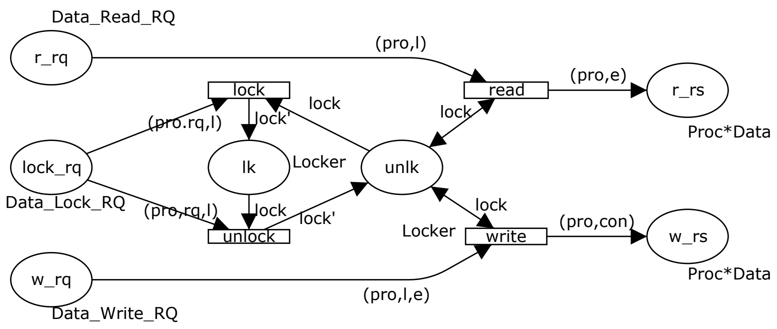

Fig. 4. Data management

As mentioned in section 3.1, we consider the following storage locations for data $LocType$= $\{RAM, Disk, TPM.spcr, TPM.dpcr, CPU.reg\}$. There are two kinds of operations: read and write. We consider two types of access control restrictions on data: read lock and write lock, which restrict the corresponding actions on the target data. Then we have two operations on these restrictions respectively: lock and unlock. So, we have the following color sets to model the data management mechanism:

$$Lock\_Type = \{r\_lk, w\_lk\};\ Lock\_OP = \{lk, unlk\};$$
$$Locker = Loc \times Lock\_Type \times Prog;\ Data\_Read\_RQ = Proc \times Loc;$$
$$Data\_Write\_RQ = Proc \times Loc \times Data;$$
$$Data\_Lock\_RQ = Loc \times Lock\_Type \times Lock\_OP;$$

In the following discussions, we use the following variables : $rq : Data\_Lock\_RQ$; $l : Loc$; $lock : Locker$, where $Loc \rightarrow LocType \times LocAddr$.

We have the place set P_2 $=\{r\_rq,\ w\_rq,\ lk\_rq,\ r\_rq,\ w\_rs,\ ld,\ unlk\ \}$. Tokens in r_rq,w_rq and lk_rq stand for the corresponding action requests. Tokens in r_rs and w_rs stand for the results of read and write actions. Tokens in lk, $unlk$ stand for the locked and unlocked states of storage locations.

We have the color set:

$$C_2 = \{\ C_2(r\_rw) = Data\_Read\_RQ,\ C_2(w\_rq) = Data\_Write\_RQ,\\ C_2(lk\_rq,) = Data\_Lock\_RQ,\ C_2(r\_rs) = Proc \times Data,\ C_2(w\_rs) =\\ Proc \times Data,\ C_2(lk) = Locker, C_2(unlk) = Locker\}$$

and the transition set:

$$T_2 = \{lock, unlock, read, write\}$$

The arc set and the arcs expression function are shown in Figure 4.

We assume that all the storage locations are unlocked at the initial time of the system, except some privileged locations, such as the boot address of the system, and the TPM dynamic registers. When the place $unlk$ holds all the tokens, it means that all storage locations are unlocked.

The data can only be accessed when it is unlocked. We use the guard function on read and write transition to depict the model restrictions as follows:

$$G_2(read) =\\ (\exists(pro_1 \times l_1) \in Type(M(r\_rq)) \bigwedge \exists(l_2 \times r\_lk \times prog_2) \in Type(M(unlk) : l_1 \subseteq l_2)$$

$$G_2(write) =\\ (\exists(pro_1 \times l_1) \in Type(M(w\_rq)) \bigwedge \exists(l_2 \times r\_lk \times prog_2) \in Type(M(unlk) : l_1 \subseteq l_2)$$

Only when the target location is unlocked, the system may carry out read or write operation. This property of CPN model for data protection mechanism in secure systems can be expressed as follows:

$$\forall(read, b) \in BE, \forall M \in \mathbb{M} : b(l) \subseteq b(locker).Loc\\ \Rightarrow \nexists M' \in \mathbb{M} : M[(read, b)\rangle M'$$

$$\forall(write, b) \in BE, \forall M \in \mathbb{M} : b(l) \subseteq b(locker).Loc\\ \Rightarrow \nexists M' \in \mathbb{M} : M[(write, b)\rangle M'$$

It is straightforward to employ the above model to depict security related data operations, e.g., the PCR operations in TPM. These operations can be expressed by the read and write model. For example, the PCR read can directly expressed by the read operation, while the PCR reset operation by privileged instructions can be expressed as write operation on specific registers.

3.5 Late Launch

The late launch mechanism can set up a secure execution environment for the given application. We will refine the late launch transition in Figure 3 to model the late launch mechanism. The CPN model of late launch mechanism is shown in Figure 5.

When an application requests the secure system to set up a secure execution environment for the given program code, the secure system stores the context and then invokes the privileged CPU instruction to launch the secure environment and records related information; then the given program code executes in the secure environment, and after finishing the execution, it returns the control to the secure system; Then the secure system records related information and then restores the context to resume execution.

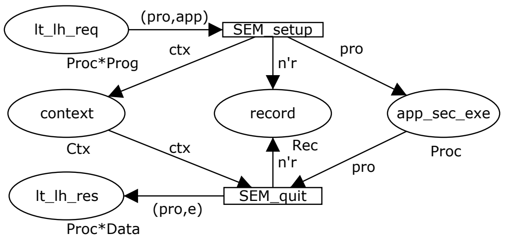

Fig. 5. Late Launch

We have the places set:

$$P_3 = \{lt\_lh\_rq, context, record, lt\_lh\_res, app\_sec\_exe\}$$

Tokens in place lt_lh_rq stand for the late launch request from the upper layer applications. Tokens in place lt_lh_res stand for the state after late launch finishes its execution. Tokens in place app_sec_exe stand for the execution state of the given program code in secure execution environment. As only one thread is allowed, we may restrict the number of tokens in app_sec_exe to be no more than 1.

We add the following color set to depict the tokens in the late launch model (Ctx stands for Context):

$$Ctx = Prog \times Next; ctx : Ctx; C_3 = \{C_3(lt\_lh\_req) = Proc \times Prog,$$
$$C_3(context) = Ctx, C_3(lt\_lh\_res) = Proc \times Data, C_3(app\_sec\_exe) = Proc,$$
$$C_3(record) = Rec\}$$

The transition set is $T_3 = \{SEM\_setup, SEM\_quit\}$, where SEM_setup and SEM_quit stand for the set up and quit of secure execution environment. The arcs set and arc expression function are shown as in Figure 5.

4 Case Studies

In this section, we will employ the above models to study the memory protection mechanism in TrustVisor [33] and remote attestation based on dynamic root of trust [19, 3], respectively.

4.1 Memory Protection in TrustVisor

TrustVisor is a special purpose secure hypervisor. It provides fine-grained protection mechanism to guarantee the integrity of program code and data for applications, under the assumption that the operating system may be malicious or tampered. TrustVisor employs the following mechanisms to provide stronger protection with acceptable overhead: hardware based virtualization [20], late launch [19, 3], DMA protection [4] , 2D page walking [8] and virtualized TPM [7]. With these mechanisms, TrustVisor stands to setting up a secure execution environment to support the execution of the given program codes PAL specified by the application APP.

In TrustVisor, a system state has three modes: host mode, legacy guest mode and secure guest mode. In host mode, TrustVisor handles the control and it can access any location on the computer; in legacy guest mode, the operating system or the applications are executing, while the TrustVisor and the PAL of the APP are protected; in secure guest mode, the PAL is executing in the secure execution environment, and all other parts of the system are protected. The states of objects in TrustVisor for these three modes are shown in Table 1.

Table 1. The protection status of objects in TrustVisor based system

Mode \ Object	TrustVisor	Untrusted Legacy OS	APP	PAL
host mode	executing	accessible	accessible	accessible
legacy guest mode	protected	executing	executing	protected
secure guest mode	protected	protected	protected	executing

We use the data management model presented in Section 3.4 to study the memory protection mechanism in TrustVisor. We can map the accessible and protected states to the locked and unlocked states in the model. The involved operations in the process of the system mode change can be considered as actions of lock, unlock, read and write, respectively. With these mappings, we are entitled to model the memory protection mechanism in TrustVisor as a CPN and analyze its properties accordingly. We consider an application app, which sends a request to TrustVisor for setting up a secure execution environment for the given program codes pal. To see whether the memory protection mechanism is designed correctly, the CPN model of the memory protection mechanism can be verified according to the following properties (TV stands for TrustVisor):

1. Host mode:

$\forall(read, b) \in BE, \forall M \in \mathbb{M} : b(pro).Prog == TV$
$\Rightarrow \nexists M' \in \mathbb{M} : M[(read, b)\rangle M'$
$\forall(write, b) \in BE, \forall M \in \mathbb{M} : b(pro).Prog == TV$
$\Rightarrow \nexists M' \in \mathbb{M} : M[(write, b)\rangle M'$

2. Legacy guest mode:
$\forall(read, b) \in BE, \forall M \in \mathbb{M} : b(pro).Prog == OS \bigvee b(pro).Prog == app$
$\bigwedge(b(l) \subseteq TV \bigvee b(l) \subseteq pal) \Rightarrow \nexists M' \in \mathbb{M} : M[(read, b)\rangle M'$
$\forall(write, b) \in BE, \forall M \in \mathbb{M} : b(pro).Prog == OS \bigvee b(pro).Prog == app$
$\bigwedge(b(l) \subseteq TV \bigvee b(l) \subseteq pal) \Rightarrow \nexists M' \in \mathbb{M} : M[(write, b)\rangle M'$

3. Secure guest mode:
$\forall(read, b) \in BE, \forall M \in \mathbb{M} : b(pro).Prog == pal \bigwedge(b(l) \subseteq TV \bigvee b(l) \subseteq OS \bigvee b(l) \subseteq app) \Rightarrow \nexists M' \in \mathbb{M} : M[(read, b)\rangle M'$
$\forall(write, b) \in BE, \forall M \in \mathbb{M} : b(pro).Prog == pal \bigwedge(b(l) \subseteq TV \bigvee b(l) \subseteq OS \bigvee b(l) \subseteq app) \Rightarrow \nexists M' \in \mathbb{M} : M[(write, b)\rangle M'$

4.2 Remote Attestation based on Dynamic Root of Trust

Remote attestation is a fundamental issue in trusted computing, and has attracted considerable attention [10, 15–17, 21, 36, 39–41]. In an open and dynamic environment, remote attestation provides a reliable way to establish trust among different platforms in mutually distrusted domains. The dynamic root of trust based remote attestation has been studied in recent years [3, 15, 19, 33, 34]. It is important to validate the correctness of these security mechanisms. Next, we will extend our above models to study remote attestation based on dynamic root of trust.

A typical dynamic root of trust based remote attestation scenario is as follows: the application app runs on an open platform P, and a challenger CH requests to attest the execution of specific program codes pal. The remote attestation protocol runs as follows: first the challenger CH sends a request with a random nonce non, which is used to eliminate replay attack; when app is going to execute the pal, app will requests the secure system to set up a secure execution environment for the pal with late launch; when the secure system sets up the environment for pal, the states of related information will be recorded, including the environment, the pal and its inputs; then pal executes in the secure environment; after pal finishes its execution, the secure system will record the outputs of pal; the system resumes its execution and sends all records to the CH.

The CPN model for remote attestation based on dynamic root of trust is shown in Figure 6. This model is based on the model of late launch in Section 3.5. The newly added elements are $P_4 = \{RA\_rq, nonce, RA\_res, known - good\_record, RA\_result, exe\_finished\}$, which are remote attestation request, random number, response of remote attestation from challenger, known-good measurements, result of remote attestation, respectively. The newly added transitions $T_4 = \{RA\_challenge, RA\_response, verify\}$ denote remote attestation request from CH, remote attestation response from the target platform, and the verification on the returned records, respectively.

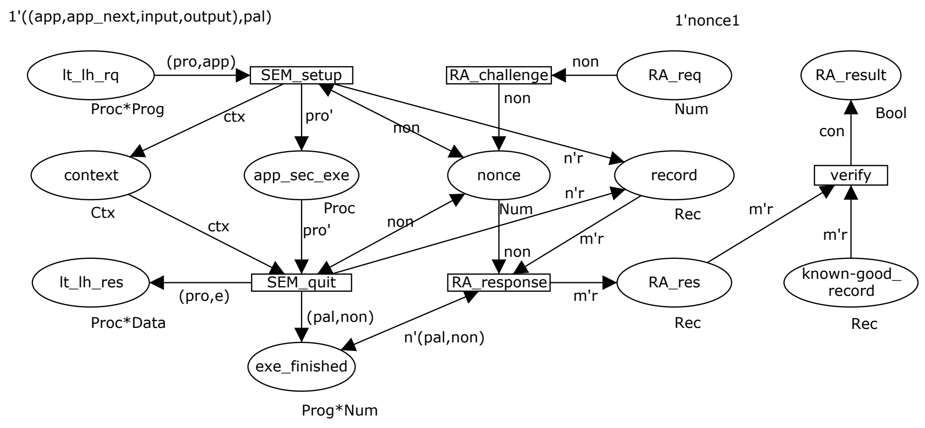

Fig. 6. Remote attestation based on dynamic root of trust

5 Related Work

In this section, we will briefly review the related work in three areas: the TCG-based secure systems, CPN based formal studies (especially in security related applications), formal methods for TCG-based security mechanisms/systems.

5.1 Secure Systems based on Trusted Computing

Since the introduction of TCG specifications in 1999, many secure systems have been built around trusted computing. Terra [13] introduced a TPM based trusted virtual machine architecture geared towards both open and closed domains, thus supporting both legacy systems and security sensitive computations on the same hardware platform. The Integrity Measurement Architecture (IMA) [40] was essentially a TPM based integrity measurement and report mechanism in Linux. OSLO [29] establishes a secure boot mechanism with the assistance of the late launch technique, and Flicker [34] leverages on late launch to set up a secure execution environment for given program codes. With the debug facilities on CPU, [15] presents a fine-grained attestation scheme for legacy binary program at the function level. TrustVisor [33] is implemented as a special purpose security hypervisor with acceptable performance overhead.

With the increasing emergence of trusted computing based systems and security mechanisms, formal studies on the design and implementation are essential in guaranteeing the correctness of these security foundations.

5.2 CPN based Formal Studies

Petri Nets have been widely used to model, analyze and validate various security mechanisms and systems, such as the security requirements [2], mandatory access control (MAC) [48, 45, 49, 25], discretional access control (DAC) [31], and role based access control (RBAC) [37].

With the features of Petri Nets and the support of advanced programming languages, Colored Petri Nets is a good tool to model, analyze and validate complex systems [12]. For example, in BETA[32], an object-oriented language supporting distributed program execution, CPN is used to model and analyze the protocol of remote object invocation[26]; Nokia research center has conducted a series of formal studies on mobile computing and network related systems. In security related applications, CPN were also extensively used to study the security properties of systems. For instance, CPN and its tools have been used to study access control models, such as RBAC [37], Chinese Wall policy [48], and UCON [28]. CPN are also utilized to analyze and validate security models, such as the Bell-LaPadula model [31], the Biba model[49]. Shin [42] introduced an extended role based access control mechanism for a trusted operating system: E-RBAC, and conducted a formal study on E-RBAC with CPN. [27] makes use of CPN to model and analyze the Stateful Reference Monitor. It is now a proven fact that CPN is a wieldable tool for modeling and analyzing complex systems (including security related systems).

5.3 Formal Verification on Trusted Computing

There exist some formal studies on TCG specifications and TCG based systems. Among many others, below are some examples. Bruschi *et al.* [9] conducted a formal study on the authorization protocol in TPM, and analyze the vulnerabilities in the protocol by model checking. [38] presented a test based method to verify the compliance between TPM and the TCG specifications. Zhang *et al.* [47] analyzed the trusted network connect (TNC) protocols within the universally composable (UC) framework. Gürgens *et al.* [18] conducted a systematical security analysis of a large part of the TCG specifications with a formal automata model based on asynchronous product automata APA and a finite state verification tool SHVT. Predict logic was used to model and analyze the bootstrapping trust [11]. Based on the Protocol Composition Logic (PCL), Logic of Secure System is employed to analyze the remote attestation schemes based on static root of trust and dynamic root of trust [6]. While some progresses have been made, many of the existing trusted computing based systems and security mechanisms still require further formal study on their correctness, such as the IMA, Flicker, and TrustVisor. As such, the proposal in this paper is dedicated to contribute along this line of work.

6 Conclusion

The trusted computing technology by TCG is proposed as a countermeasure to software attacks in an open environment. In practice, a (secure) system built upon the TPM must guarantee *correctness*. However, existing work still falls short of formal methods that model, analyze, and validate the correctness of TCG based systems. Aimed at contributing along this line of work, in this paper we proposed CPNs to be an appropriate tool, because of its inherent advantages in the formal study of complex systems. In particular, we established CPN models for various mechanisms in a TCG based secure system: process management, data management and late launch. With these models, we further conducted case studies on the memory protection mechanism in TrustVisor and remote attestation based on dynamic root of trust, and the results demonstrate that these models are indeed capable of modeling real secure systems. We stress that our models can serve as the foundation to support further formal analysis on the security properties of TCG-based secure systems.

One of the future work is to simulate and analyze the properties of the derived CPN models, with the help of the relevant CPN tools. In addition, we shall further model TrustVisor to study its security properties based on its design. According to the recent progress in system verification [30], we also plan to employ CPNs to conduct formal study on the implementation of secure systems, e.g., Flicker.

References

1. M. Abadi and T. Wobber. A Logical Account of NGSCB. In *Formal Techniques for Networked and Distributed Systems (Forte '04)*, 2004.
2. T. Ahmed and A. R. Tripathi. Static verification of security requirements in role based cscw systems. In *the eighth ACM symposium on Access control models and technologies (SACMAT)*, pages 196–203, 2003.
3. AMD. Amd64 virtualization codenamed "pacifica" technology-secure virtual machine architecture reference manual. Technical report, AMD, May 2005.
4. AMD. Amd64 architecture programmer's manual, volume 2: System programming, 2007.
5. D. Anupam, D. Ante, C. M. John, and R. Arnab. Protocol Composition Logic (PCL). *Electron. Notes Theor. Comput. Sci.*, 172:311–358, 2007.
6. D. Anupam, F. Jason, G. Deepak, and K. Dilsun. A logic of secure systems and its application to trusted computing, 2009. 1608135 221-236.
7. S. Berger, R. Caceres, K. A. Goldman, R. Perez, R. Sailer, and L. van Doorn. vTPM: Virtualizing the Trusted Platform Module. In *Proceedings of the 15th USENIX Security Symposium*, pages 305–320. USENIX, Aug. 2006.
8. R. Bhargava, B. Serebrin, F. Spadini, and S. Manne. Accelerating two-dimensional page walks for virtualized systems. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'08)*, pages 26–35, Seattle, WA, USA, Mar. 2008. ACM.
9. D. Bruschi, L. Cavallaro, A. Lanzi, and M. Monga. Replay attack in TCG specification and solution. In *ACSAC*, pages 127–137. IEEE Computer Society, 2005.
10. L. Chen, R. Landfermann, H. Löhr, M. Rohe, A.-R. Sadeghi, and C. Stüble. A protocol for property-based attestation. In *STC '06*, pages 7–16, New York, NY, USA, 2006. ACM Press.
11. S. Chen, Y. Wen, and H. Zhao. Formal analysis of secure bootstrap in trusted computing. In *Autonomic and Trusted Computing, LNCS*, volume Volume 4610/2007, pages 352–360. Springer Berlin / Heidelberg, 2007.
12. CPN group at the Department of Computer Science in University of Aarhus. Examples of Industrial Use of CP-nets, 2010. `http://www.daimi.au.dk/CPnets/intro/example_indu.html`.
13. T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A Virtual Machine-Based Platform for Trusted Computing. In *SOSP03*, Bolton Landing, New York, USA, October 19C22, 2003.
14. D. Garg, J. Franklin, D. Kaynar, and A. Datta. Compositional system security with interface-confined adversaries. In *Mathematical Foundations of Programming Semantics (MFPS) Conference*, April 6th, 2010.

15. L. Gu, Y. Cheng, X. Ding, R. H. Deng, Y. Guo, and W. Shao. Remote attestation on function execution. In *First International Conference on Trusted Systems (INTRUST)*. Lecture Notes in Computer Science 6163, pages 60-72, Beijing, China, December 17-19th, 2009.
16. L. Gu, X. Ding, R. H. Deng, B. Xie, and H. Mei. Remote attestation on program execution. In *ACM workshop on Scalable Trusted Computing (STC), held in conjunction with the 15th ACM Conference on Computer and Communications Security (ACM CCS'08)*, 2008.
17. L. Gu, X. Ding, R. H. Deng, B. Xie, and H. Mei. Remote platform attestation - the testimony for trust management. In *Trust Modeling and Management in Digital Environments: from Social Concept to System Development*. 2010. (Editor: Zheng Yan), IGI Global, DOI: 10.4018/978-1-61520-682-7, ISBN-13: 978-1615206827.
18. S. Gürgens, C. Rudolph, D. Scheuermann, M. Atts, and R. Plaga. Security Evaluation of Scenarios Based on the TCG's TPM Specification. In J. B. Lopez and Javier, editors, *12th European Symposium On Research In Computer Security,*, volume 4734, pages 438–453. Springer, 2007.
19. Intel Corporation. Intel Trusted Execution Technology — Preliminary Architecture Specification. Technical Report Document Number: 31516803, Intel Corporation, 2006.
20. Intel Corporation. Intel Virtualization Technology for Directed I/O Architecture Specification. Technical Report Order Number: D51397-001, Intel Corporation, Feb. 2006.
21. T. Jaeger, R. Sailer, and U. Shankar. PRIMA: policy-reduced integrity measurement architecture. In *SACMAT '06*, pages 19–28, New York, NY, USA, 2006. ACM Press.
22. K. Jensen. Colored Petri Nets. *Lecture Notes Comp. Sci.: Advances in petri nets*, 254:248–299, 1986.
23. K. Jensen. *Coloured Petri Nets : Basic Concepts, Analysis Methods and Practical Use, Vol. 1*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1992.
24. K. Jensen. An Introduction to the Theoretical Aspects of Coloured Petri Nets. *Lecture Notes in Computer Science*, 803, 1994.
25. Y. Jiang, C. Lin, H. Yin, and Z. Tan. Security analysis of mandatory access control model. In *IEEE International Conference on Systems, Man and Cybernetics*, volume vol. 6, page 5013C5018, 2004.
26. J. B. Jørgensen and K. H. Mortensen. Modelling and Analysis of Distributed Program Execution in BETA Using Coloured Petri Nets. In *Proceedings of the 17th International Conference on Application and Theory of Petri Nets*, pages 249–268, London, UK, 1996. Springer-Verlag.
27. B. Katt, M. Hafner, and X. Zhang. Building a stateful reference monitor with Coloured Petri Nets. In *CollaborateCom*, pages 1–10, 2009.
28. B. Katt, X. Zhang, and M. Hafner. Towards a Usage Control Policy Specification with Petri Nets. In *On the Move to Meaningful Internet Systems: OTM*, volume Volume 5871/2009, pages 905–912, 2009.
29. B. Kauer. OSLO: improving the security of trusted computing. In *Proceedings of 16th USENIX Security Symposium*, pages 1–9, Berkeley, CA, USA, 2007. USENIX Association.
30. G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4: Formal verification of an OS kernel. In *Proceedings of the 22nd Symposium on Operating Systems Principles (22nd SOSP'09), Operating Systems Review (OSR)*, pages 207–220, Big Sky, MT, Oct. 2009. ACM SIGOPS.
31. K. Knorr. Dynamic access control through Petri Net workflows. In *16th Annual Computer Security Applications Conference (ACSAC)*, page 159, 2000.
32. O. L. Madsen, B. Moller-Pedersen, and K. Nygaard. *Object Oriented Programming in the Beta Programming Language*. Addison-Wesley Publishing Company, 1993.
33. J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig. TrustVisor: Efficient TCB Reduction and Attestation. In *IEEE Symposium on Security and Privacy*, 2010.

34. J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: an execution infrastructure for TCB minimization. In *3rd ACM SIGOPS/EuroSys European Conference on Computer Systems*. ACM, 2008. 1352625,315-328.
35. Microsoft Corporation. BitLocker Drive Encryption: Technical Overview, 2006. New (optionally) TPM-based secure boot and filesystem encryption from MS Vista.
36. J. Poritz, M. Schunter, E. Van Herreweghen, and M. Waidner. Property attestation—scalable and privacy-friendly security assessment of peer computers. Technical Report RZ 3548, IBM Research, May 2004.
37. H. Rakkay and H. Boucheneb. Security analysis of role based access control models using colored petri nets and CPNtools. *Transactions on Computational Science*, 5430:149–176, 2009.
38. A.-R. Sadeghi, M. Selhorst, C. Stüble, C. Wachsmann, and M. Winandy. TCG inside?: a note on TPM specification compliance, 2006. 1179487,47-56.
39. A.-R. Sadeghi and C. Stüble. Property-based attestation for computing platforms: caring about properties, not mechanisms. *New security paradigms*, 2004.
40. R. Sailer, X. Zhang, T. Jaeger, and L. v. Doorn. Design and Implementation of a TCG-based Integrity Measurement Architecture. In *Proceedings of the 13th USENIX Security Symposium*, San Diego, CA, USA, August 9C13, 2004.
41. E. Shi, A. Perrig, and L. V. Doorn. BIND: A Fine-Grained Attestation Service for Secure Distributed Systems . In *2005 IEEE Symposium on Security and Privacy (S&P'05)*, 2005.
42. W. Shin. *An extension of role based access control for trusted operating systems and its coloured petri net model*. PhD thesis, 2005. Gwangju Institute of Science and Technology, Korea, 2005, `http://itslab.csce.kyushu-u.ac.jp/ssr/SSR-1/thesis-main.pdf`.
43. Trusted Computing Group. TPM main specification. Main Specification Version 1.2 rev. 85, Trusted Computing Group, Feb. 2005.
44. Trusted Computing Group. TCG Specification Architecture Overview (Revision 1.4) , 2007.
45. V. Varadharajan. Hook-up property for information flow secure nets. In *Computer Security Foundations Workshop IV*, page 154C175, 1991.
46. J. Zhan, H. Zhang, B. Zou, and X. Li. Research on automated testing of the trusted platform model. In *ICYCS*, pages 2335–2339. IEEE Computer Society, 2008.
47. J. Zhang, J. Ma, and S. Moon. Universally composable secure tnc model and eap-tnc protocol in if-t. *SCIENCE CHINA Information Sciences*, 53(3):465–482, 2010.
48. Z. Zhang, F. Hong, and J. Liao. Modeling Chinese Wall Policy Using Colored Petri Nets. In *CIT '06: Proceedings of the Sixth IEEE International Conference on Computer and Information Technology*, page 162, Washington, DC, USA, 2006. IEEE Computer Society.
49. Z. Zhang, F. Hong, and J.-G. Liao. Verification of strict integrity policy via petri nets. In *the International Conference on Systems and Networks Communication (ICSNC)*, page p. 23., 2006.