Fine-Grained Refinement on TPM-Based Protocol Applications

Wenchao Huang, Yan Xiong, Xingfu Wang, Fuyou Miao, Chengyi Wu, Xudong Gong, and Qiwei Lu

***Abstract*—Trusted Platform Module (TPM) is a coprocessor for detecting platform integrity and attesting the integrity to the remote entity. There are two obstacles in the application of TPM: minimizing trusted computing base (TCB) for reducing risk of flaws in TCB, for which a number of convincing solutions have been developed; formal guarantees on each level of TCB, where the formal methods on analyzing the application level have not been well addressed. To the best of our knowledge, there is no general formal framework for developing the TPM-based protocol applications, which not only guarantees the security but also makes it easier for design. In this paper, we make fine-grained refinement on TPM-based security protocols to illustrate our formal solution on the application level by using the Event-B language. First, we modify the classical Dolev-Yao attacker model, which assumes normal entity's compliance with the protocol even without TPM's protection. Thus, the classical security protocols are vulnerable in this modified attacker model. Second, we make stepwise refinement of the security protocol by refining the protocol events and adding security constraints. From the fifth refinement, we make a case study to illustrate the entire refinement and further formally prove the key agreement protocol from DAAODV, the TPM-based routing protocol, under the extended Dolev-Yao attacker model. The refinement provides another way of formal modeling the TPM-based security protocols and a more fine-grained model to satisfy with the rigorous security requirement of applying TPM. Finally, we prove all the proof obligations generated by Rodin, an Eclipse-based IDE for Event-B, to ensure the soundness of our proposal.**

***Index Terms*—Event-B, formal method, key agreement protocols, refinement, security protocols, TPM.**

I. Introduction

TRUSTED platform module (TPM) [1], [2] is a secure coprocessor which is embedded on motherboard of many new laptop computers since 2006. The primary feature of TPM is to measure and to ensure the integrity of its platform. The TPM contains several PCRs (Platform Configuration Registers) that allow a secure storage and reporting of security relevant metrics. Starting from power-on boot process, the TPM stores the measurement of the boot loader to PCRs. After that, the boot loader takes control of the platform, extends the measurement of OS block to PCRs, and transfers the control to the OS. Similarly, the measurement is extended to application level. Thus, the measurement stored in PCRs represents the integrity of a platform: if the platform is contaminated, the value of PCRs changes. The next feature of TPM is to attest its PCRs to the remote entity. The remote entity checks the PCRs and the signature to verify the integrity of the platform.

Though it seems more secure to use TPM in platform and protocol design, a design flaw of the platform would cause the protection failure, which is an obstacle in the application of TPM. Therefore, many approaches on TPM focus on the correctness of platform design [3]. For instance, some approaches minimize the Trusted Computing Base (TCB) [4] in order to reduce the risk of flaws in the TCB, which is the set of all hardware, firmware, and/or software components that are critical to the platform's security. Recently, convincing solutions have been made on minimizing TCB by both hardware [5] and software design [6], [7]. Though the TCB could be minimized, the problem of correctness of TCB still exists. Other approaches formally verify or prove the correctness on each level of TCB, including the TPM APIs [8]–[12], OS [13], and communication protocols [14]–[16].

However, to the best of our knowledge, the formal methods on the application level of TCB have not been well addressed, where two major problems need to be solved. The first one is redundancy. While the bottom level of TCB is designed as common components which are shared by upper applications, there are also common ways of developing on the application level. Since the formal development is much more sophisticated than normal ones, a novel model, which has done the common work shared by most applications, will greatly reduce the redundancy and make the formal development much easier. The second problem is the formal correctness of fine-grained models. Most of the formal approaches, such as [14]–[16], verify or prove the correctness of TPM-based protocols in an abstract way, rather than a fine-grained one. However, it is not guaranteed that fine-grained software is still sound, even if the soundness of the abstract protocol is proved. Because an inexperienced developer is prone to mistakes in implementation. Unfortunately, in TPM-based environment, the fine-grained correctness needs to be reassured, which determines the validity of TPM's protection.

In this paper, we make fine-grained refinement on TPM-based security protocols to illustrate our formal solution on the application level of the TCB. This paper follows the work [17] while

Manuscript received October 07, 2012; revised February 05, 2013; accepted April 13, 2013. Date of publication April 18, 2013; date of current version May 16, 2013. This work was supported in part by the National Natural Science Foundation of China (61170233; 61232018; 61272472; 61202404; 61272317) and in part by China Postdoctoral Science Foundation (2011M501060). The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Ramesh Karri. *(Corresponding author: Xingfu Wang.)*

The authors are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China (e-mail: huangwc@ustc.edu.cn; yxiong@ustc.edu.cn; wangxfu@ustc.edu.cn; mfy@ustc.edu.cn; wuchengyi@mail.ustc.edu.cn; lzgxd@mail.ustc.edu.cn; luqiwei@mail.ustc.edu.cn).

Digital Object Identifier 10.1109/TIFS.2013.2258915

they made formal analysis of classical security protocols rather than TPM-based protocols. Another difference is that we make *fine-grained* refinement where the code in final refinement step is closer to C language. By refinement, the above problem of redundancy and fine-grained correctness can be solved. We also learned lessons from [18], which reviewed the recent development on formal modeling security protocols. [12] also verified the implementation of security protocols by refinement, which is similar to our work. But they mainly verified the security protocol of TPM APIs in the case study, i.e., the key management inside the TPM, while we verify the protocol applications that only use TPM APIs as interfaces.

The basic idea of our approach is as follows. First, we modify the classical Dolev-Yao attacker model, which assumes that the normal entity is not compromised. In modified attacker model, the entity is not compromised only if its platform is protected by the TPM. So, the classical security protocols are vulnerable in our model, for the normal entity, which are not protected by the TPM, can be controlled by adversaries. Second, we refine the TPM-based security protocols using Event-B [19], [20]. In early refinement steps, we model the basic function of TPM, communication channels, communication operations and some of the security constraints. The purpose is to build common steps of developing such that these steps don't need to be modified and are the base for building new protocol applications. In other words, we try to solve the redundancy problem in early refinement steps. In the following steps, we make a case study on a TPM-based key agreement protocol aiming to illustrate the entire formal solution and to ensure the security on the fine-grained level, which deals with the second major problem.

In the case study, we solve the key agreement problem which is stated as follows [21]: two entities assuming i and j wish to agree on keying information in secret over a distributed network. Each party desires an assurance that no party other than i and j can possibly compute the keying information agreed. This is the authenticated key agreement (AK) problem [22], [23]. Several techniques related to the Diffie-Hellman (DH) key-exchange protocol [24] have been proposed to solve the AK problem, and formal techniques on analyzing these protocols have also been proposed [21], [25], [26]. However, few TPM-based security protocols and related formal analysis have been proposed. In our former work, we proposed a TPM-based secure routing protocol, named DAAODV [27]. The protocol contains two steps: the key agreement step and the routing step. On the key agreement step, the authentication of each node is based on TPM and the key exchanging is based on DH algorithm. Therefore, from the 5th refinement (R5), we make a case study on the key agreement process of DAAODV and solve the AK problem in the final refinement.

Finally, we show the soundness of our proposal by proving all the proof obligations (POs) in the refinements. The POs are generated by Rodin, an Eclipse-based IDE for Event-B. If all the POs are successfully proved, the protocol complies with the constraints that are specified in the model.

The paper is organized as follows: In Section II, we give a brief overview of TPM and Event-B. Section III presents attacker model and security assumptions. Section IV details the refinement on general TPM-based security protocols. Section V proposes the fine-grained refinement on a TPM-based key agreement protocol. Section VI shows the result in refinement and proving. Finally, some concluding remarks are made in Section VIII.

II. Preliminaries

A. Trusted Computing

Trusted Computing [2] is a technology developed and promoted by the Trusted Computing Group. It is based on a coprocessor Trusted Platform Module (TPM) which is embedded on the motherboard. TPM is shielded with several components, including the Platform Configuration Registers (PCRs), Attestation Identity Key (AIK), Endorsement Key, nonvolatile storage, cryptographic engines, etc.

One feature of the TPM is integrity measurement. The TPM obtains metrics of platform characteristics that affect the integrity of a platform and puts digests of those metrics in its PCRs. There are at least 16 PCRs in a TPM and a PCR is a 160-bits shielded location to hold an unlimited number of measurements by the operation *extend* as the following:

$$PCR_{new} = SHA-1(PCR_{old} \| measurement)$$

Based on the measurement feature, one way of enforcing the platform integrity is to *extend* the trusted boundary from the root level to OS and application level. It is implemented by measuring the target code before execution control is transferred. This is called the Static Root of Trust Measurement (SRTM). The other way is the Dynamic Root of Trust Measurement (DRTM): the application directly takes control of TPM, CPU, and physical memory, and works in isolation with OS and other applications. It is implemented by calling the CPU instruction SKINIT/SENTOR. The CPU extends measurement of the application block to PCR 17, isolates the block from OS and other applications and then transfers the control to the application. DRTM is adopted by TPM specification 1.2, known as Intel Trusted Execution Technology (TXT) [5] and AMD Secure Virtual Machine (SVM).

Another feature of the TPM is remote attestation. TPM is uniquely identified by the 2048 bits key pairs, named Endorsement Key (EK). EK is generated by manufacturers and permanently bound to the TPM. EK can be recognized by manufacturers but cannot be identified by others. Instead, the entity i generates Attestation Identity Key (AIK) inside TPM for identification. Then, i gets the certificate of AIK by communicating with the privacy CA and uses the certificate for attesting that the AIK comes from the valid TPM. Meanwhile, the PCRs are signed by AIK. Hence, on receiving i's public key of AIK, the certificate and the signature, remote entity j is capable of verifying i's platform integrity.

However, TPM has to communicate with privacy CA when a new AIK pair is generated, such that the original scheme faces the potential bottleneck problem which is caused by privacy CA. To solve this, a new remote attestation scheme named Direct Anonymous Attestation, (DAA) [28] was proposed. Instead of privacy CA, TPM t first gets the membership certificate by the operation *Join* with an issuer i. By using the certificate, t can sign the AIK with a nonce named *challenge* every time

a new AIK pair is generated. It is called $\mathrm{DAASign}(c; AIK_p)$ in this paper. As a result, the running privacy CA is no longer needed in remote attestation. Additionally, for Zero-Knowledge-Proofs are used in the scheme, the signature is anonymous that it doesn't expose the identity of TPM.

The integrity measurement in remote attestation has also been improved. As the TPM directly signs the PCRs with AIK, the values of PCRs are exposed. There is a chance for an attacker to guess the identity of the entity's platform through the analyzation of the values of PCRs. So, [29] proposed a scheme named Property-based Attestation (PBA). By using the $\mathrm{PBASign}(c)$, where c is the *challenge* sent from TPM, the TPM signs the PCRs without exposing the values of PCRs, but can attest that the values belong to a specific set of configurations. We model the interface of DAA and PBA scheme in the refinement.

Additionally, the term *trust*, which is defined by TCG (Trusted Computing Group) [2], is adopted and modeled in the paper. It is the expectation that a device will behave in a particular manner for a specific purpose [30]. In other words, the platform is trusted if it preserves its integrity according to its PCRs. It is worth noting that the platform may also be vulnerable even if it is trusted. For instance, if there is a design flaw on the application level of the platform, the remote entity may also attack the platform without modifying the platform. In this case, the platform is still trusted but insecure.

B. Event-B

Event-B [19], [20], [31] is considered an evolution of the B method [32]. It is defined in terms of a few simple concepts that describe a discrete event system and proof obligations (POs) that permit verification of properties of the event system. Key features of Event-B are the use of refinement to represent systems at different abstraction levels and the use of mathematical proof to verify consistency between refinement levels.

An Event-B *model* consists of several components of two kinds: *machines* and *contexts*.

Contexts contain the static parts of a model. These are *constants* and *axioms* that describe the properties of these constants.

Machines contain the dynamic parts of a model. A machine is made of a *state*, which is defined by means of *variables* denoted by v. They are constrained by *invariants* $I(v)$. Invariants are supposed to hold whenever variables values change. But this must be proved first.

Besides its state, a machine contains a number of *events* which specify the way the state may evolve. Each event is composed of a *guard* and an *action*. The guard is the necessary condition under which the event may occur. The action, as its name indicates, determines the way in which the state variables are going to evolve when the event occurs.

An event may be executed only when its guard holds. Events are *atomic* and when the guards of several events hold simultaneously, then *at most one of them* may be executed at any moment. The choice of event to be executed is nondeterministic. An event, named E, has the following form:

$$E = \text{when } G(v) \text{ then } S(v) \text{ end}$$

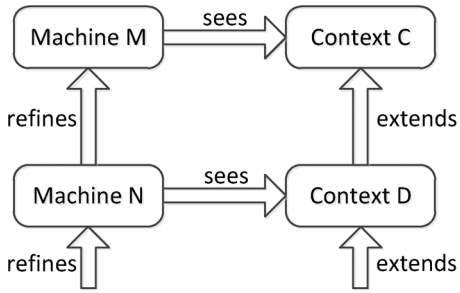


Fig. 1. Model and context refinements in Event-B.

If the guard $G(v)$ is true, then E may be executed, v evolves to v' by the state transition $S(v)$. Informally in this case, by mathematical induction, two types of POs are generated: (1) on initialization, $I(v)$ is true, (2) if $I(v)$ is true, then $I(v')$ is preserved after E is executed.

From a given machine M, a new model N can be built and asserted to be a *refinement* of M. Likewise, context C seen by a model M, can be extended to a context D, which may be seen by N. A typical example of the machine and context relationship is shown as Fig. 1.

The extension of a context consists of adding new sets and new constants. These are defined by means of new axioms. Consequently, no specific proof obligations are associated with context extension.

Unlike the situation of extending context, the concrete machine N has a collection of state variables w, which must be *completely distinct* from the collection v of variables in the abstraction M. Instead, Event-B only allows the transparent reuse of variables in refinement. Machine N also has an invariant dealing with these variables w. But contrary to the case of abstract machine M, where the invariant exclusively depended on the local variables of this machine, this time it is possible to have the invariant of N also depending on the variables v of its abstraction M. So this invariant J of N "glues" the state of the concrete machine N to that of its abstraction M. Informally, on refining an existing event E, two POs should be proved: (1) the guard G in the abstract event E can be implied from the guard H in the refined event F, (2) given the before-after state variable w, w' in refined event F and the corresponding state variable v in E, there exists an instance v' which is evolved from v in E and v', w' still satisfy the glue invariant J. Other techniques on refinements (e.g. the refinement on new events, dead lock prevention), which are not used in our approach, are not introduced in the section.

III. Attacker Model and Security Assumptions

In classical Dolev-Yao attacker model [33], the adversary can overhear, intercept, and synthesis any message and is only limited by the constraints of the cryptographic methods used. But the model excludes the possibility of breaking the platform's integrity. On the other side, the main feature of TPM is to measure and attest the integrity of the platform. Hence, the existence of TPM doesn't affect the result of security analysis under the Dolev-Yao model. Besides, the TPM-based security protocol is

intuitively more secure than the one without a TPM. The contradiction is caused by the unbalance relation between the weaker attacker model and the stronger protection.

Some approaches, based on Dolev-Yao model, made additional analysis, such as adding the *Oops* events [18] in security protocols. At *Oops* events, the communication entity has a chance to violate the protocols, e.g., the session key is lost by accidents. In other words, by introducing the *Oops* events, it is assumed that the attackers can make specific attacks. In addition, extra analysis is made for evaluating the potential loss caused by *Oops* events. However, it's the last resort unless the TPM is utilized: the definition of *Oops* events relies on the specific violation and the incomplete definition makes formal verification unreliable. So, the other purpose of modifying the Dolev-Yao model is to get rid of *Oops* events to reduce the sophisticated additional analysis.

We define the attacker model as follows:

1) *The adversary controls the network and can perform any message operation except cryptanalysis*. We inherit it from Dolev-Yao attacker model. The adversary gets every message that is sent on the network, and can prevent delivery of any message or redirect it to agents other than the intended recipient. In addition, the adversary can break up messages into components by splitting up concatenated ones, and opening up cipher-texts sealed with the key that the adversary knows. Finally, the adversary can form new messages at free will by concatenation and encryption. It is also assumed that the adversary cannot use any cryptanalytic technique to derive significant information from cipher-texts, which is also a part of the Dolev-Yao model. We make this assumption for two reasons. One reason is that the length of the RSA keys inside TPM, such as AIK, EK, SRK, is 2048 bits, and the TCG group specifies some techniques for preventing such attacks. For example, TCG group chooses OAEP for RSA-padding, which meets indistinguishability under chosen-plaintext attack (IND-CPA) [34]. The other reason is that though there are still possibilities of such attacks, our current work only focuses on the security of the application level, which merely uses the crypto commands as interfaces.
2) *The adversary controls the entity without TPM's protection and the entity with the fake or tampered TPM, but cannot get the AIK credential of the fake or tampered TPM*. We assume that there is no other protection mechanism, except that the TPM can measure the integrity and check if the platform is compromised. Thus, if the entity is not protected by the TPM or is protected by a tampered TPM, we assume that the entity's platform is already compromised to maximize the capability of the adversary. When the TPM is tampered, the adversary can access the TPM, where the private key of AIK can be read and used to sign any PCR values. The adversary can also forge a fake TPM, (e.g., the software-based TPM emulator [35]). There are also limitations. We assume only the real and unbroken TPM can get the credential of its AIK from the issuer in the DAA scheme or privacy CA. Otherwise, the adversary can perform any remote attestation with success.
3) *The adversary controls the entity which is protected by real and unbroken TPM if the value of the TPM's PCRs is not in the trust list*. In this case, the platform of the entity may be tampered. We assume that there are various of trusted platforms and their measurement values stored in PCRs are recorded in a trust list. On the contrary, we assume the entities whose platform are not in the trust list are already tampered.
4) *The adversary cannot modify the platform of the entity which is protected by real and unbroken TPM if the value of the TPM's PCRs is in the trust list*. That is, the adversary can only control it through the network if it has design flaws, rather than controlling the platform directly.

There are also security assumptions related to the TCB. We assume the bottom level of the TCB has no flaws in the design.

1) *The adversary cannot illegally control the TPM*. Though there were formal analysis on the TPM APIs [8]–[12] and flaws have been found, the security of the TPM is not concerned in the paper. Besides, we assume that there is no flaw in remote attestation of TPM (e.g., DAA scheme, the scheme based on Privacy CA) and the platform (e.g., PBA scheme, the scheme in TPM specification).
2) *The adversary also cannot illegally control the level between TPM and protocol application*. The middle level can be implemented in two ways, the SRTM and DRTM, as illustrated in Section II-A. Though, it is hard to guarantee the correctness in designing the OS in SRTM, it is realizable in DRTM as the OS is excluded from TCB and the code size in this level becomes small.
3) *The adversary can illegally control the protocol application if there is a flaw in the application*. In this case, the adversary can illegally control the application through the flaws in all refinement steps. In many approaches, the protocol events were first abstracted and then analyzed. On the contrary, we model and refine the protocol event to ensure that the implementation is still correct in this work.

We further assume that the platform runs only the TPM-based security protocol that the TPM only measures the protocol software and the bottom computing base. Though the TPM can measure numerous pieces of software, we simplify the TPM's function. In other words, we assume that the PCRs are constants and unique to the specific platform. Thus, in the refinement, the bijection of the TPM and platform can be built such that PCRs in TPM corresponds to the configuration of the platform and the protocol software. The purpose of these assumptions is to prune the conditions which are not related to the AK problem in TPM-based protocols.

IV. Early Refinement on General TPM-Based Security Protocols

Our refinement is mainly composed of two parts: refining the machine, where we mainly refine the protocol events and finally build the pseudocode of each event; extending the context, where we define the environment and axioms that never change.

Roughly speaking, starting from the initial context, two sets representing the *entities* and *messages* are defined. In the initial machine, we model the communication *channels* through which

the entities transfer the messages to others. Naturally, the events *Send* and *Receive* are also modeled. In first extension of context (E1), the entities are divided into *TPMs* and *platforms* that are protected by TPMs. In first refinement of the machine (R1), the communication channels and the events are divided as well. In E2, the abstract messages are refined into several subsets which represent different types. The cryptographic functions are successively modeled according to different crypto types, e.g., encryptions and signatures, etc. In addition, the operations on these types of messages are modeled, such as breaking up and concatenating messages, which are the abilities of the adversaries as defined in the attacker model. E3 further models the TPM environment in which relevant crypto sets are derived from the sets in E2. Back to R2, confidentiality is constrained on several types of abstract variables which are maintained by each entity. These abstract variables are then refined into concrete ones in R3. Finally, R4 does the additional work for sufficient preparation on concrete protocols.

A. Initial Machine and Context

The initial machine and context provide the basic abstract elements of the communication infrastructure.

In the context, two carrier sets are defined: *AGENT* and *MSG* respectively. *AGENT* represents set of communication entities. *MSG* represents the set of messages which are sent, received and processed by the entities in *AGENT*.

The machine is composed by the event *Send* and *Receive* with the variable *channel* where $channel \subseteq AGENT \times AGENT \times MSG$. When the message m is sent from a_1 to a_2, a new element $a_1 \mapsto a_2 \mapsto m$ is added to *channel*. When a_2 receives the message m from a_1, the element $a_1 \mapsto a_2 \mapsto m$ is removed from *channel* accordingly. For instance, the event *Send* is described in Algorithm 1.

Algorithm 1 Event *Send* in initial abstract machine

Event *Send* $\widehat{=}$
any
a_1
a_2
m
where
grd1 : $a_1 \in AGENT$
grd2 : $a_2 \in AGENT$
grd4 : $a_1 \neq a_2$
grd3 : $m \in MSG$
then
act1 : $channel := channel \cup \{a_1 \mapsto a_2 \mapsto m\}$
end

It is worth mentioning that the type of variables in Event-B. Variables of one type only represent global states of the machine which may change when an event occurs. They do not exist in the protocol application of each entity. For instance, in Algorithm 1, a_1 and a_2 in the tuple $a_1 \mapsto a_2 \mapsto m$ represent the *real* sender and receiver of the message m, which means m is definitely sent from a_1 and is definitely received by a_2 if $a_1 \mapsto a_2 \mapsto m$ is in *channel*. That is, if a_1 intends to send m to a_2 but the adversary a captures m, then the tuple $a_1 \mapsto a \mapsto m$, other than $a_1 \mapsto a_2 \mapsto m$, is put in the channel. Thus, *channel* only represents a state in the abstract machine. Variables of the other type not only represent the state but also represent the data maintained by each entity, which will be explained in the next subsections.

TABLE I
DEFINITION OF SUBSETS DIVIDED FROM MSG

subset	the element in the subset
STRUCT	structure composed by multiple elements
$AKEY_p$	public asymmetric key
$AKEY_s$	private asymmetric key
AENCRYPT	asymmetric encryption of a structure
SKEY	symmetric key
SENCRYPT	symmetric encryption of a structure
SIGN	signature of a structure
DH_s	private part of DH algorithm
DH_p	public part of DH algorithm
HASH	the hash value of a structure
NORMAL	element not in the upper subset

B. First Refinement and First Extension

The TPM is independent of its platform since it has its own secure storage and cryptographic engines and can communicate with the platform. Thus, in 1st extension of context (E1), the carrier set *AGENT* is divided into *PLATFORM* and *TPM* respectively. It is implemented by the statement:

$$\text{partition}(AGENT, TPM, PLATFORM).$$

That is, $AGENT = TPM \cup PLATFORM$ and $TPM \cap PLATFORM = \varnothing$. As assumed in Section III, we define the bijection $t2p$ from th$^{t2p \in TPM \rightarrowtail PLATFORM}$s follows:

For simplicity, if $s \in PLATFORM$ has no TPM, there is an equivalent value $t2p^{-1}(s)$ which represents tampered TPM.

Hence, in 1st refinement (R1), $chan_p$ and $chan_s$ are refined from *channel*, with $chan_p$, $chan_s \subseteq channel$, and $chan_p \cap chan_s = \varnothing$. $chan_p$ represents the public channel between platforms, whereas $chan_s$ represents the private channel between TPMs and their platforms.

Furthermore, the *Send* event is refined to *SendPub*, *SendfromTPM*, *SendtoTPM*. In *SendPub*, new tuple $a_1 \mapsto a_2 \mapsto m$ is added in $chan_p$, while in *SendtoTPM* and *SendfromTPM*, $a_1 \mapsto a_2 \mapsto m$ is added to $chan_s$. The *Receive* event is refined to *ReceivePub*, *ReceivefromTPM* and *ReceivebyTPM* in a similar way.

C. Second Extension

In the second extension of context (E2), in order to express the message types, the carrier set *MSG* is divided into subsets following with related functions defined. So the set *MSG* are divided into subsets by the clause *partition*: *STRUCT*, $AKEY_p$, $AKEY_s$, *AENCRYPT*, *SKEY*, *SENCRYPT*, *SIGN*, DH_s, DH_p, *HASH*, *NORMAL*. The definitions of the subsets are shown in Table I.

Then the constant $TYPE \subseteq \mathbb{P}(MSG)$ and $TYPE = \{STRUCT, AKEY_p, \ldots\}$ and the function *type* is defined as follows:

$$type \in MSG \rightarrow TYPE$$

For all the $m \in MSG, type(m) = T$ iff $m \in T$.

Next, to differentiate structures further in *STRUCT*, we define the *struct* as follows:

$$struct \in STRUCT \rightarrow \left(\bigcup z \cdot z \in \mathbb{N}_1 | 1 \ldots z \rightarrow MSG\right)$$

If $m \in STRUCT$, m is a structure which is combined with several messages. Since the number of messages in the structure is undetermined, we use z to represent the size of the structure, i.e., if the size of m is z, $struct(m)$ is total function that $struct(m) \in 1 \ldots z \rightarrow MSG$, and $struct(m)(i)$ means the ith message in m for $i \in 1 \ldots z$.

STRUCTTYPE and *structType* are also defined for differentiating the types of the structures:

$$STRUCTTYPE = \left(\bigcup z \cdot z \in \mathbb{N}_1 | 1 \ldots z \rightarrow TYPE\right)$$
$$structType \in STRUCT \rightarrow STRUCTTYPE$$

For $m \in STRUCT$, $structType(m) \in STRUCTTYPE$. If the size of m is z, $structType(m) \in 1 \ldots z \rightarrow TYPE$, which means for $i \in 1 \ldots z$, $type(struct(m)(i)) = structType(m)(i)$. *TYPE* will be extended to $TYPE_1, TYPE_2, \ldots$ in further extension when the set *MSG* is further partitioned. Likewise, *STRUCTTYPE*, *structType* will also be extended.

Note that $AKEY_p$, $AKEY_s$ denote the type of public and private keys of both ciphertext of type *AENCRYPT* and signature of type *SIGN* respectively. All the plain texts of encryptions and signatures are structures. For $m \in AENCRYPT$, $aencrypt_d(m)$ and $aencrypt_k(m)$ denote the structure of plaintext that is encrypted and the corresponding private key. For $m \in SIGN$, $sig_d(m)$, $sig_s(m)$, $sig_p(m)$ and $sig_c(m)$ denote the corresponding data structure, private key, public key, and the challenge of the signature m. For $m \in SENCRYPT$, $sencrypt_d(m)$ and $sencrypt_k(m)$ denote the structure of plain text that is encrypted and the symmetric key encrypting $sencrypt_d(m)$.

In DH algorithm, Alice generates a private part $s_1 \in DH_s$ and the public part $p_1 \in DH_p$, while Bob generates a private part $s_2 \in DH_s$ and the public part $p_2 \in DH_p$. They exchange the public parts, and calculate the symmetric key k_1 and k_2, where $skey_1(k_1) = s_1$, $skey_2(k_1) = p_2$, $skey_1(k_2) = s_2$, $skey_2(k_2) = p_1$. According to the property of DH algorithm, the value of k_1 equals the value of k_2, we define the function $dh_{equal} \in SKEY \times SKEY \rightarrow BOOL$ as follows:

$$\begin{array}{l}\forall x, y \cdot x \in SKEY \wedge y \in SKEY \Rightarrow \\ \quad dh_{equal}(x \mapsto y) = \\ \qquad bool\,((skey_1(x) = skey_1(y) \wedge \\ \qquad\quad skey_2(x) = skey_2(y)) \vee \\ \qquad (dh_p\,(skey_1(x)) = (skey_2(y)) \wedge \\ \qquad\quad dh_p\,(skey_1(y)) = (skey_2(x))))\end{array}$$

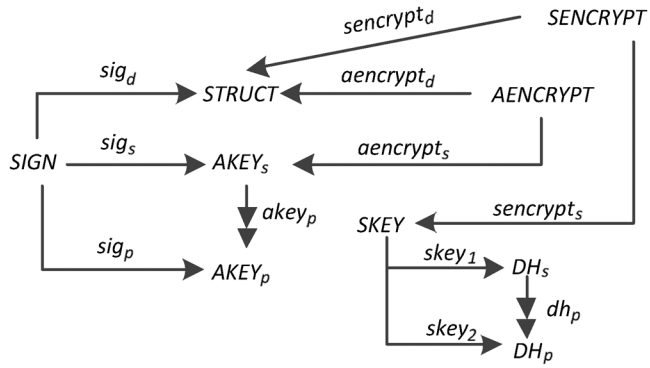


Fig. 2. Refined messages and related functions. The normal arrows represent total functions and the two headed arrows represent bijections.

where the $dh_p \in DH_s \twoheadrightarrow DH_p$, i.e., $dh_p(s_1) = p_1$ and $dh_p(s_2) = p_2$.

For $m \in MSG$, we define $hash \in MSG \twoheadrightarrow HASH$, where $hash(m)$ is the hash value of m and $akey_p(s)$ is the public key corresponding to the private key s. Thus, we have the relation of these bijections and functions in Fig. 2. We model parts of the first assumption in attacker model: the adversary can perform any message operation except crypto analysis. That is, the adversary can break up messages into components by the function $analz \in \mathbb{P}(MSG) \rightarrow \mathbb{P}(MSG)$, and can form new messages by $compose \in \mathbb{P}(MSG) \rightarrow \mathbb{P}(MSG)$. This idea is drawn from [18] which is based on higher-order logic. Instead, we define *analz* and *compose* recursively in Event-B, which is based on first-order logic. However, the definitions of *analz* and *compose* is not enough to prove confidentiality in further refinements. For instance, for $x \in \mathbb{P}(MSG)$, it is hard to determine whether $y \in MSG$ is in $analz(x)$, such that the result of $analz(x)$ still cannot be solved by the recursive definition. Thus, we define $parts \in MSG \leftrightarrow MSG$ and the related additional axioms to assist proving. For $x \in \mathbb{P}(MSG)$, $parts[x]$ represents the set of messages possible to be analyzed. For instance, for $y \in SENCRYPT$, if $x \in MSG$ is the element in $sencrypt_d(y)$, then $x \in parts[\{y\} \cup S]$ no matter whether $sencrypt_s(y)$ can be analyzed from S. So, if $x \notin parts[\{y\} \cup S]$, then $x \notin analz(\{y\} \cup S)$. Else if $x \in parts[\{y\} \cup S]$, it can also be judged by *parts*, though it is more complicated than the former condition.

The verification process is also defined as a function: $verify \in SIGN \times STRUCT \rightarrow BOOL$. For $s \in SIGN$ and $m \in STRUCT$, if s is the valid signature of m, $verify(s \mapsto m) = TRUE$, otherwise $verify(s \mapsto m) = FALSE$. In the next extension, *verify* is extended to several types, including *DAAVerify* and *PBAVerify*, etc.

It is worth noting that the message operations and verification processes are defined as functions in context whereas the sending and receiving processes are modeled as events in a machine. Because the former processes do not affect the security status of the entities while the goal in the machine is to check if the invariants hold when the state changes. Instead, the result of the former processes is used in the events. For instance, in

TABLE II
PARTITIONING THE SETS IN THIRD EXTENSION

Sets	Partition in 3rd extension
$AKEY_p$	AIK_p, DAA_p, $HSIGN_p$, $AKEY_{p_1}$
$AKEY_s$	AIK_s, DAA_s, $HSIGN_s$, $AKEY_{s_1}$
$SIGN$	$SIGN_{DAA}$,$SIGN_{PBA}$,$SIGN_{HASH}$,$SIGN_{NORMAL}$
$HASH$	PN, $HASH_1$
TPM	TPM_t, TPM_f
PCR	PCR_t, PCR_f

ReceivePub events, the platform alters the trust state of others according to the result of $verify(s \mapsto m)$.

D. Third Extension

The third extension (E3) focuses on modeling the TPM-based environment.

Like the inheritance mechanism of Java or C++ program, the type of *MSG* is refined in depth. The key pairs embedded in TPM, such as AIK, DAA key pairs, are derived from $AKEY_p$ and $AKEY_s$. For instance, $AKEY_p$ is refined as follows:

$$\text{partition}(AKEY_p, AIK_p, DAA_p, HSIGN_p, AKEY_{p_1})$$

Similar to *NORMAL*, $AKEY_{p_1}$ is abstract set of public keys which can be extended in further extensions. Likewise, we partition several sets in Table II.

For convenience, some sets used in the case study are defined, e.g., the key pairs $HSIGN_p$ and $HSIGN_s$, which are the key pairs of $HASH_{SIGN}$ and PN, which collects the pseudo names of entities.

In addition, the key word *trust* is modeled. In the definition of TPM main specification [2], trust is the expectation that a device will behave in a particular manner for a specific purpose. More specifically, the device is armed with unbroken TPM, and the PCR value is in the trust list which records the ones of the trusted platforms. Thus, for $trust \in PLATFORM \rightarrow BOOL$:

$$\forall a \cdot a \in PLATFORM \Rightarrow (trust(a) = TRUE \Leftrightarrow t2p^{-1}(a) \in TPM_t \wedge pcr\left(t2p^{-1}(a)\right) \in PCR_t)$$

where TPM_t represents the set of TPMs which are not tampered, PCR_t represents the set of PCRs which are in the trust list and $pcr \in TPM \rightarrow PCR$ for $t \in TPM$, $pcr(t)$ represents the PCR value of t. In our model, it is assumed that the adversary's platform is also not trusted. Because if the platform is trusted, the platform will act according to the protocol application, and is not capable to attack other entities.

On verifying if $t2p^{-1}(a)$ is in TPM_t, we define the DAA verification $dverify \in SIGN_{DAA} \times STRUCT \rightarrow BOOL$, i.e., if a TPM succeeds in DAA verification, it is not tampered:

$$\forall s, m \cdot s \in SIGN_{DAA} \wedge m \in STRUCT \wedge dverify(s \mapsto m) = TRUE \Rightarrow d2t\,(sig_s(s)) \in TPM_t$$

On verifying if $pcr(t2p^{-1}(a))$ is in PCR_1, we define the PBA verification $pverify \in SIGN_{PBA} \times STRUCT \rightarrow BOOL$, that if a TPM succeeds in PBA verification, its platform is not tampered:

$$\forall s, m \cdot s \in SIGN_{PBA} \wedge m \in STRUCT \wedge pverify(s \mapsto m) = TRUE \wedge a2t\,(sig_s(s)) \in TPM_t \Rightarrow pcr\,(a2t\,(sig_s(s))) \in PCR_t$$

where $d2t$ and $a2t$ are bijections that for $t \in TPM$, $d2t^{-1}(t)$ and $a2t^{-1}(t)$ represent the private keys of DAA and AIK which are shielded in t. Noted that it is not the complete implication if t succeeds in DAA or PBA verification. There are other properties of DAA and PBA, such as anonymity. In our model, we only consider the authenticity and confidentiality related to AK problem, and the other properties of DAA and PBA are out of scope and not discussed in this paper.

E. Second Refinement

In the second refinement of machine (R2), the variables which are maintained by TPMs and platforms are defined. Following that, the ability of adversaries on capturing the variables and constraints on the variables for preserving the confidentiality are modeled as follows.

First, there are public and private messages stored in *TPM* and *PLATFORM*: t_p, t_s, s_p, s_s. For t_p, $t_p \in TPM \rightarrow \mathbb{P}(MSG)$ and $t_p(t)$ denotes the public messages maintained by t and can be sent to $t2p(t)$ via $chan_s$. For t_s, $t_s \in TPM \rightarrow \mathbb{P}(MSG)$ and $t_s(t)$ denotes the private messages maintained by t. Messages in $t_s(t)$ cannot be analyzed from its platform $t2p(t)$ or by adversaries if $t \in TPM_t$, according to the first assumption in Section III. Likewise, $s_s(s)$ and $s_p(s)$ represent the private and public messages maintained by s respectively. Besides the states in the machine, t_p, t_s, s_p, s_s indicate the variables which can be refined to concrete ones and corresponds to the variables in application implementation of C language.

Next, we model the *knowledge* maintained by the adversary. In the first assumption in attacker model, the adversary can eavesdrop messages. In our model, these messages are stored in variable m_p representing the knowledge. Thus, in the event *SendPub* (see Algorithm 1) message m which is put in $chan_p$ is also stored in m_p, i.e., in THEN clause, the substitution $m_p := m_p \cup \{m\}$ is added. To guarantee this attacker's ability, we add the following invariant:

$$\forall a_1, a_2, m \cdot a_1 \in PLATFORM \wedge a_2 \in PLATFORM \wedge m \in MSG \wedge a_1 \mapsto a_2 \mapsto m \in chan_p \Rightarrow m \in m_p$$

Hence, to prevent the attacks by adversaries, a constraint on confidentiality is defined as an invariant:

Inv_confidentiality_1: Messages maintained by the platform x cannot be analyzed or composed from the adversaries if x is trusted:

$$\forall x, m \cdot x \in PLATFORM \wedge trust(x) = TRUE \wedge m \in MSG \wedge m \in s_s(x) \Rightarrow m \notin compose\,(analz\,(m_p \cup (\bigcup a \cdot a \in PLATFORM \wedge trust(a) = FALSE \mid s_s(a) \cup s_p(a) \cup t_s\left(t2p^{-1}(a)\right) \cup t_p\left(t2p^{-1}(a)\right))))$$

In this invariant, the adversary analyzes and composes messages not only from m_p, but also from the untrusted platforms.

We assume that if the platform a is not trusted, then messages in $s_s(a), s_p(a), t_s(t2p^{-1}(a)), t_p(t2p^{-1}(a))$ are obtained by the adversary as well.

To satisfy the constraint, we add guards in the events to prevent the specific messages from sending. For example, in *SendPub* event, the guards are added in *where* clause:

$$trust(a_1)=TRUE \Rightarrow (\forall x \cdot x \in s_s(a_1) \Rightarrow x \notin parts\,[\{m\}])$$
$$trust(a_1)=TRUE \Rightarrow (\forall x \cdot x \in SKEY \wedge skey_1(x) \in s_s(a_1) \Rightarrow x \notin parts\,[\{m\}])$$

For the trusted platform a_1, the first guard prevents the messages in $s_s(a_1)$ from sending, and the second guard prevents the private parts of DH algorithm from sending. We also constrain the capability of sending messages according to their own *knowledge*. For instance, the following guards are added in *where* clause:

$$trust(a_1)=TRUE \Rightarrow m \in compose\,(s_p(a_1) \cup s_s(a_1))$$
$$trust(a_1)=FALSE \Rightarrow m \in compose\,(analz\,(m_p \cup (\bigcup a \cdot a \in PLATFORM \wedge trust(a)=FALSE | s_p(a) \cup s_s(a) \cup t_s\,(t2p^{-1}(a)) \cup t_p\,(t2p^{-1}(a)))))$$

For the platform a_1, a_1 only possesses and composes messages from $s_s(a_1)$ and $s_p(a_1)$, if a_1 is trusted, otherwise a_1 can analyze and compose messages from m_p and the variables in untrusted platforms.

To both satisfy Inv_confidentiality_1 and make it achievable in further refinements, we make additional constant S_m in E3 and related invariants in R2. As shown in the first assumption in attacker model, the attacker cannot perform crypto analysis, i.e., it is not allowed that the adversary generates the same asymmetric keys owned by the trusted entities by accident. In this model, S_m is defined as the set of messages which cannot be generated from adversaries:

$$\forall a \cdot a \in PLATFORM \wedge trust(a)=FALSE \Rightarrow (s_s(a) \cup s_p(a) \cup t_s\,(t2p^{-1}(a)) \cup t_p\,(t2p^{-1}(a))) \cap S_m = \varnothing$$

Next, we make additional invariants on S_m: for every platform $a, (s_p(a) \cup t_s(t2p^{-1}(a)) \cup t_p(t2p^{-1}(a))) \cap S_m = \varnothing$; messages $s_s(a) \subseteq S_m$ if the platform a is trusted. As a result, we make and prove the invariant $parts[m_p] \cap S_m = \varnothing$ following which it is helpful to prove many proof obligations on R6.

To make it easier for proving in the next refinements, we also make constraints on s_s, s_p, t_s, t_p such that only the specific types are allowed to be stored in the specific variables. Besides, it is not allowed that the secret messages generated by different nodes intersect.

F. Third Refinement

In the third refinement of the machine (R3), the abstract variables t_s, t_p, s_s, s_p are refined and replaced with concrete variables.

TABLE III
REFINED VARIABLES IN THIRD REFINEMENT

R2	R3	Definition
t_s	t_{aiks}	private key of AIK
	t_{daas}	private key of DAA
t_p	t_{aikp}	public key of AIK
	t_{daap}	public key of DAA
	t_{pn}	*PN* value from its platform
	t_{pbac}	PBA challenge from its platform
s_s	a_{hsigns}	private key of Hash-Sign
	as_{skey1s}	private part of DH algorithm
	as_{skey}	symmetric keys
s_p	a_{hsignp}	public key of Hash-Sign
	a_{pn}	the Pseudo Name of its platform
	a_{aikp}	public key of AIK from its TPM
	a_{dsign}	DAA signature from its TPM
	as_{psign}	PBA signatures from its TPM
	as_{skey1p}	public part of DH generated from itself
	as_{skey2}	public part of DH from others
	as_{pbaci}	PBA challenge from others
	as_{pbaco}	PBA challenge sent to others
	as_{aikp}	public key of AIK from others

These variables are named with three prefixes:

- t_* represents the variables owned by specific TPM. For instance, $t_{aiks} \in TPM \rightarrow AIK_s$ means that $t_{aiks}(t)$ represents the private key of AIK in t.
- a_* represents the variables owned by specific platform. For instance, $a_{dsign} \in PLATFORM \nrightarrow SIGN_{DAA}$ that $a_{dsign}(a)$ represents DAA signature owned by a and obtained from $t2p^{-1}(a)$.
- as_* represents the variables which are maintained by the platforms. For instance, $as_{skey} \in PLATFORM \times PN \nrightarrow SKEY$ tells that $as_{skey}(a \mapsto pn)$ is maintained by a and represents symmetric key between a and the platform whose pseudo name is pn. Noted that all the maintained variables are partial functions. They are empty after initialization, and then added with new values when the specific events occur.

Then, t_s, t_p, s_s, s_p are refined by glue invariants. For example, for t_s:

$$\forall t \cdot t \in TPM \Rightarrow t_s(t) = t_{aiks}\,[\{t\}] \cup t_{daas}\,[\{t\}]$$

The refined variables are listed in Table III:

Corresponding to Inv_confidentiality_1 in R2, this property on concrete variables in R3 is constrained by the above glue invariants. If the glue invariants hold when the events in R3 are executed, Inv_confidential-ity_1 also holds in R3.

G. Fourth Refinement

The fourth refinement of the machine (R4) does the additional work for the sufficient preparation on concrete protocols.

The event *SendPub* is refined to *SendPubt* and *SendPubf*. *SendPubt* denotes the messages sent by a_1, where $trust(a_1) = TRUE$, and the *SendPubf* denotes the messages sent by a_1 where $trust(a_1) = FALSE$. Thus, the guards of *SendPub* in R3 are separated depending on whether a_1 is trusted. The *SendPubt* is further refined to several events according to the

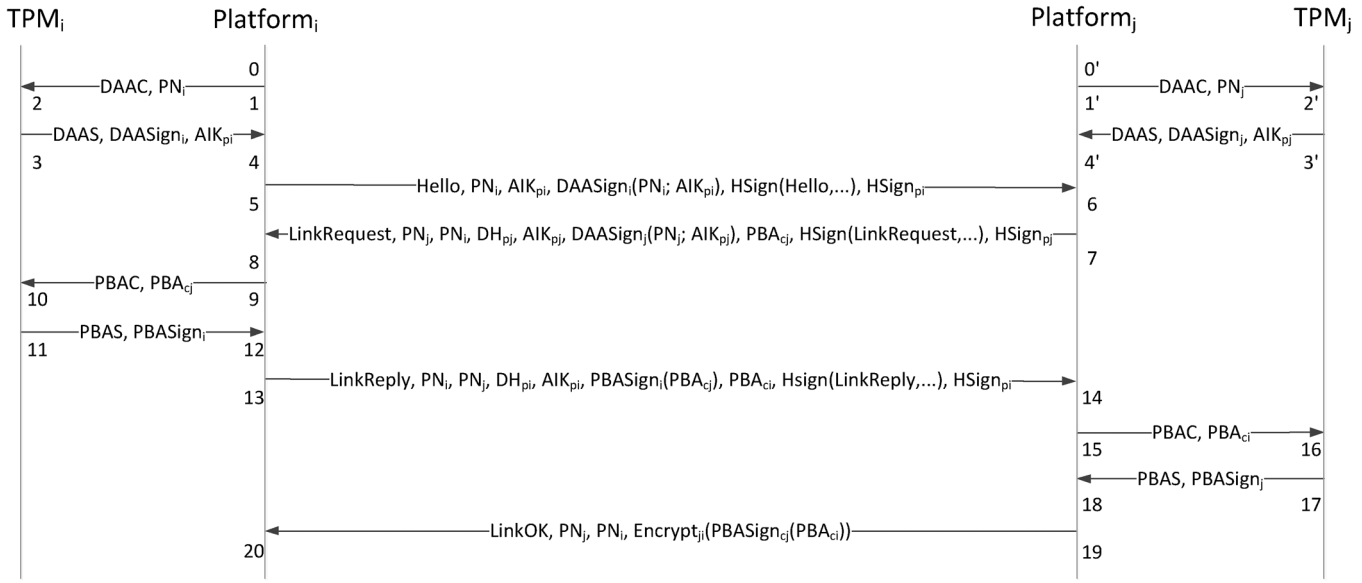


Fig. 3. Simplified key agreement process in DAAODV.

concreted protocol. The event *SendPubf* is finished modeling at this level and will not be further refined. Thus, *SendPubf* can analyze and compose messages from the adversary's *knowledge* and send messages to any platforms.

We define the constant *PMTYPE* in E4, and constrain the types of messages in *SendtoTPM*, *SendfromTPM*, *SendPubt* and *SendPubf* by *PMTYPE*. $PMTYPE \in \mathbb{P}(STRUCTTYPE_2)$ and denotes protocol message types which is defined in concrete protocols. It is also reasonable to constrain *SendPubf*, as the receiver first checks the types before processing. The purpose is to make it easier for proving by the additional constraint.

V. Case Study

We make fine-grained refinement on the key agreement process of DAAODV [27] protocol. DAAODV is a TPM-based routing protocol. It contains two processes. One is the key agreement process based on TPM and the other is the secret routing process where all the messages transferred between neighbors are encrypted with symmetric keys generated in the former process. To solve the AK problem, the key agreement process are further modeled in the case study.

A. The Key Agreement Process of DAAODV

In DAAODV, the platforms are identified by the pseudo name (PN), instead of the real ID. They periodically alter the PN and the AIKs to achieve the anonymity. In the key agreement process, a platform authenticates the neighbor using the DAA and PBA scheme. At the same time, they exchanges the public part of the DH algorithm. If they both succeed in authentication, they use the symmetric key generated by the DH algorithm for exchanging the messages in the routing process.

Additionally, as the computation overhead of DAA and PBA scheme is large, if a platform a_1 directly verifies the DAA and PBA signature of pn_2 and does not save the results of verification, a_1 is easy to suffer the DoS attacks. Because a_1 has to verify pn_2 continually if a_1 finds the false signature of pn_2, for there may be attackers forging the messages tagged with pn_2, though pn_2 may be trusted. Thus, to prevent the potential DoS attacks, the light-weighted signature and verification scheme (Hash-Sign, HSign) are used before verifying the DAA and PBA signatures. In the protocol, PN and plaintexts are signed together by HSign scheme and PN is bound to the public key of HSign signature at the same time, i.e., $PN = Hash^{CHT}(CHV)$ where CHV is the public key and CHT equals to the specific times on hashing CHV. The purpose is to prevent the adversary from forging the messages with the PN of the trusted platforms. As a result, if a platform a_1 finds pn_2's DAA or PBA signature a false one, a_1 puts pn_2 into the blacklist, and will not verify the signature of pn_2 again.

For simplicity, the relationship between PN and public key of HSign signature is equivalently defined in the refinement:

$$\forall a \cdot a \in PLATFORM \Rightarrow akey_p(a_{hsigns}(a)) = a_{hsignp}(a) \\ \wedge\, a_{pn}(a) = hash\,(a_{hsignp}(a))$$

Here CHT is eliminated and the entity's public key of HSign $a_{hsign}(a)$ is considered constant though CHT and CHV changes at short intervals in real protocol. The reason for us to do this is that $Hash^{CHT}$ is still a one way function and each private key of HSign cannot be captured by adversaries for effectively sign the messages.

We illustrate the simplified key agreement process of DAAODV as follows and in Fig. 3:

Case 0: Each platform broadcasts Hello messages periodically. The message is identified by PN, and contains the public key of AIK and the DAA signature. The DAA signature signs the public key of AIK with the value of PN as the DAA challenge. The message also contains the public key of HSign and the HSign signature, which signs the content of the Hello message.

Case 1: $trustlevel(j \mapsto PN_i) = -1$ and the platform of j receives the messages tagged with PN_i. j drops the message. In the refinement, $trustlevel \in PLATFORM \times PN \rightarrow \mathbb{N}$, and $trustlevel(a \mapsto pn)$ represents the record

maintained by a and is assigned with a new value after a verifies the pn's DAA or PBA signature. Specifically, $trustlevel(a \mapsto pn) = 0$ at initialization, 1 when pn succeeds in DAA verification, 2 when pn succeeds in PBA verification, and -1 when pn fails in DAA or PBA verification.

Case 2: $trustlevel(j \mapsto PN_i) = 0$ and the platform of j receives Hello tagged with PN_i. If the HSign and DAA signature are true, $trustlevel(j \mapsto PN_i) := 1$ and j sends i the message LinkRequest. Different from the Hello message, the public part of DH (DH_{pj}) and PBA challenge (PBA_{cj}) are added. That is, j asks i to reply the DH_{pi} to build the symmetric key between j and i, and the PBA signature to verify the platform of i. If HSign is false, j drops the Hello message. If HSign is true and DAA signature is false, $trustlevel(j \mapsto PN_i) := -1$.

Case 3: $trustlevel(i \mapsto PN_j) = 0$ and the platform of i receives the LinkRequest tagged with PN_j. If the HSign and DAA signature are true, $trustlevel(i \mapsto PN_j) := 1$, i generates the symmetric key k_{ij} by DH_{si} and DH_{pj}, and sends j the message LinkReply. If HSign is false, i drops the LinkRequest message. If HSign is true and DAA signature is false, $trustlevel(i \mapsto PN_j) := -1$.

Case 4: $trustlevel(j \mapsto PN_i) = 1$ and the platform of j receives LinkReply tagged with PN_i. If the HSign and PBA signature is true, $trustlevel(j \mapsto PN_i) := 2$, j generates the symmetric key k_{ji} by DH_{sj} and DH_{pi} and sends i the message LinkOK. If HSign is false, j drops the LinkReply message. If HSign is true and PBA signature is false, $trustlevel(j \mapsto PN_i) := -1$.

Case 5: $trustlevel(i \mapsto PN_j) = 1$ and the platform of i receives the LinkOK tagged with PN_j. If the encrypted content can be decrypted by k_{ij}, and the PBA signature is true, $trustlevel(i \mapsto PN_j) := 2$. If the encrypted content cannot be decrypted by k_{ij}, i drops the LinkOK message. If the encrypted content can be decrypted by k_{ij} and the PBA signature is false, $trustlevel(i \mapsto PN_j) := -1$.

Case 6: $trustlevel(i \mapsto PN_j) = 2$ and $trustlevel(j \mapsto PN_i) = 2$. i and j communicate by encrypting the messages by k_{ij} and k_{ji} and start the routing process with each other.

B. Fifth Refinement

The fifth refinement of the machine (R5) refines *SendPubt* and *ReceivePub* to several events which will be shared by events in R6.

In DAAODV, there are several types of messages sent in the public channel. In these types of messages, several common types of submessages are generated, composed and verified, such as signatures in $SIGN_{HASH}, SIGN_{DAA}, SIGN_{PBA}$. Developers have to repeatedly implement these processes in the final implementation, which may trigger more faults and make the developing processing exhausting. In this refinement, we introduce our method to solve the problem.

In Fig. 4, The *SendPubt* events is refined to *SendPubt_HSIGN*, *SendPubt_DSIGN* and *SendPubt_PSIGN* and each event adds the process of generating the specific signature. These events can be shared by events in R6. In a similar way, the *ReceivePub* event is refined to *ReceivePub_HVERIFY*, *ReceivePub_DVERIFY* and *ReceivePub_PVERIFY*.

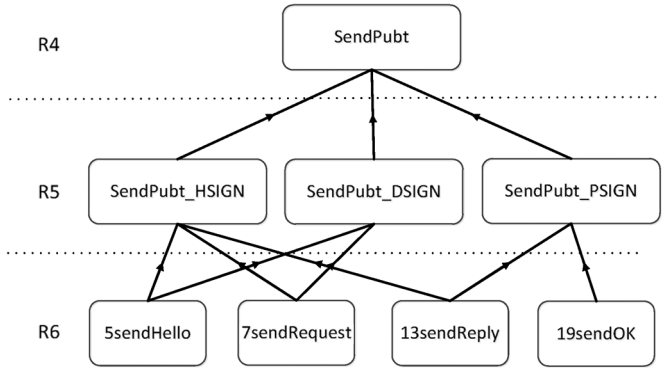


Fig. 4. Generating sharable events from *SendPubt*.

C. Final Refinement

In the final refinement of the machine (R6), the fine-grained protocol events are modeled and the extra security constraints are presented and proved.

We tagged the protocol events with number in Fig. 3, including *1sendDAAc*, *2receiveDAAc*, *3sendDAAs*, *4receiveDAAs*, *5sendHello*, *6receiveHello*, *7sendRequest*, *8receiveRequest*, *9_15sendPBAc*, *10_16receivePBAc*, *11_17sendPBAs*, *12_18receivePBAs*, *13sendReply*, *14receiveReply*, *19sendOK*, *20receiveOK*, and *INITIALIZATION* and *SendPubf*.

Specifically, the events which are refined from *SendPubt* compose and send the messages, and the events which are refined from *ReceivePub* verify the signatures and generate secret data in s_s, such as the parts of DH algorithm and symmetric keys.

We take event *8receiveRequest* in Algorithm 2 as an example. The event *8receiveRequest* occurs only when the conditions in *where* clause are satisfied:

- recv: The actual receiver a_2's PN equals to the 3rd item in message m, i.e., $struct(m)(3)$.
- tl: a_2 has not verified DAA or PBA signature of $struct(m)(2)$.
- mt: The message is LinkRequest, and the message format is valid.
- hv: a_2 verifies Hash-Sign of $struct(m)(2)$, and the signature is true.
- dkey: The AIK_p in DAA signature is $struct(m)(5)$.
- a2k: a_2 hasn't generated public or private part of DH algorithm for $struct(m)(2)$.

Hence, new values, i.e., $newlevel, dhs, dhp, pbac, key$, are generated and assigned to the variables maintained by a_2. The constraints on these values are also in *where* clause:

- nl: a_2 verifies if the DAA signature of $struct(m)(2)$. If it is true, $newlevel = 1$, otherwise, $newlevel = -1$.
- dh: a_2 generates new dhs and dhp, which are the private and public part of DH algorithm.

Algorithm 2 Fine-grained Refinement on Event *8receiveRequest*

Event *8receiveRequest* $\widehat{=}$
  any
    $\mathtt{a_1}, \mathtt{a_2}, \mathtt{m}, \mathtt{newlevel}, \mathtt{dhs}, \mathtt{dhp}, \mathtt{pbac}, \mathtt{key}$
  where
    grd : $\mathtt{a_1} \mapsto \mathtt{a_2} \mapsto \mathtt{m} \in \mathtt{chan_p} \wedge \mathtt{a_1} \in \mathtt{PLATFORM} \wedge \mathtt{a_2} \in \mathtt{PLATFORM}$
    recv : $\mathtt{a_{pn}}(\mathtt{a_2}) = \mathtt{struct}(\mathrm{m})(3)$
    tl : $\mathtt{trustlevel}(\mathtt{a_2} \mapsto \mathtt{struct}(\mathrm{m})(2)) = 0$
    mt : $\mathtt{m} \in \mathtt{STRUCT} \wedge \mathtt{struct}(\mathtt{m})(1) = \mathtt{tLINKREQUEST} \wedge \mathtt{structType_2}(\mathtt{m}) = \{1 \mapsto \mathtt{TAG}, 2 \mapsto \mathtt{PN}, 3 \mapsto \mathtt{PN}, 4 \mapsto \mathtt{DH_p}, 5 \mapsto \mathtt{AIK_p}, 6 \mapsto \mathtt{SIGN_{DAA}}, 7 \mapsto \mathtt{PBA_c}, 8 \mapsto \mathtt{SIGN_{HASH}}, 9 \mapsto \mathtt{HSIGN_p}\}$
    hv : $\mathtt{hverify}(\mathtt{struct}(\mathtt{m})(8) \mapsto \mathtt{m}) = \mathtt{TRUE} \wedge \mathtt{struct}(\mathtt{m})(9) = \mathtt{sig_p}(\mathtt{struct}(\mathtt{m})(8))$
    dkey : $\mathtt{struct}(\mathtt{m})(5) = \mathtt{struct}(\mathtt{sig_d}(\mathtt{struct}(\mathtt{m})(6)))(1)$
    a2k : $\mathtt{a_2} \mapsto \mathtt{struct}(\mathtt{m})(2) \notin \mathtt{dom}(\mathtt{as_{skey1s}}) \wedge \mathtt{a_2} \mapsto \mathtt{struct}(\mathtt{m})(2) \notin \mathtt{dom}(\mathtt{as_{skey1p}})$
    nl : $\mathtt{newlevel} \in \mathbb{N} \wedge (\mathtt{newlevel} = 1 \vee \mathtt{newlevel} = -1) \wedge (\mathtt{dverify}(\mathtt{struct}(\mathtt{m})(6) \mapsto \mathtt{m}) = \mathtt{TRUE} \wedge \mathtt{struct}(\mathtt{m})(2) = \mathtt{sig_c}(\mathtt{struct}(\mathtt{m})(6)) \Leftrightarrow \mathtt{newlevel} = 1) \wedge (\mathtt{dverify}(\mathtt{struct}(\mathtt{m})(6) \mapsto \mathtt{m}) = \mathtt{FALSE} \wedge \mathtt{struct}(\mathtt{m})(2) = \mathtt{sig_c}(\mathtt{struct}(\mathtt{m})(6)) \Leftrightarrow \mathtt{newlevel} = -1)$
    dh : $\mathtt{dhs} \in \mathtt{DH_s} \wedge \mathtt{dhs} \notin \mathtt{ran}(\mathtt{as_{skey1s}}) \wedge \mathtt{dhp} \in \mathtt{DH_p} \wedge \mathtt{dhp} \notin \mathtt{ran}(\mathtt{as_{skey2}}) \cup \mathtt{ran}(\mathtt{as_{skey1p}}) \wedge \mathtt{dhp} = \mathtt{dh_p}(\mathrm{dhs})$
    Sm : $\mathtt{trust}(\mathtt{a_2}) = \mathtt{TRUE} \Leftrightarrow \mathtt{dhs} \in \mathtt{S_m}$
    pbac : $\mathtt{pbac} \in \mathtt{PBA_c} \wedge \mathtt{pbac} \notin \mathtt{ran}(\mathtt{as_{pbaci}}) \cup \mathtt{ran}(\mathtt{as_{pbaco}})$
    key : $\mathtt{key} \in \mathtt{SKEY} \wedge \mathtt{skey_1}(\mathtt{key}) = \mathtt{dhs} \wedge \mathtt{skey_2}(\mathtt{key}) = \mathtt{struct}(\mathtt{m})(4)$
  then
    act1 : $\mathtt{chan_p} := \mathtt{chan_p} \setminus \{\mathtt{a_1} \mapsto \mathtt{a_2} \mapsto \mathtt{m}\}$
    pbaci : $\mathtt{as_{pbaci}}(\mathtt{a_2} \mapsto \mathtt{struct}(\mathtt{m})(2)) := \mathtt{struct}(\mathtt{m})(7)$
    aikp : $\mathtt{as_{aikp}}(\mathtt{a_2} \mapsto \mathtt{struct}(\mathtt{m})(2)) := \mathtt{struct}(\mathtt{m})(5)$
    tl′ : $\mathtt{trustlevel}(\mathtt{a_2} \mapsto \mathtt{struct}(\mathtt{m})(2)) := \mathtt{newlevel}$
    skey2 : $\mathtt{as_{skey2}}(\mathtt{a_2} \mapsto \mathtt{struct}(\mathtt{m})(2)) := \mathtt{struct}(\mathtt{m})(4)$
    dhs : $\mathtt{as_{skey1s}}(\mathtt{a_2} \mapsto \mathtt{struct}(\mathtt{m})(2)) := \mathtt{dhs}$
    dhp : $\mathtt{as_{skey1p}}(\mathtt{a_2} \mapsto \mathtt{struct}(\mathtt{m})(2)) := \mathtt{dhp}$
    pbaco : $\mathtt{as_{pbaco}}(\mathtt{a_2} \mapsto \mathtt{struct}(\mathtt{m})(2)) := \mathtt{pbac}$
    skey : $\mathtt{as_{skey}}(\mathtt{a_2} \mapsto \mathtt{struct}(\mathtt{m})(2)) := \mathtt{key}$
  end

Sm: If a_2 is a trusted platform, it generates dhs in S_m, otherwise dhs is not in S_m. The statement will not be implemented in real application as it is an assumption inferred from attacker model. Instead, it is only used for proving.

pbac: a_2 generates new $pbac$ in PBA_c.

key: a_2 generates the symmetric key key by composing dhs and $struct(m)(4)$.

Thus, instead of abstracting on events in classical formal analysis, the refinement in this level can be directly translated to real application codes.

To solve the AK problem, we prove the following invariants:

Inv_confidentiality_2: The trusted platform a uses a symmetric key shared with pn for encryption and sends the encryption to public, only if $trustlevel(a \mapsto pn) = 2$.

$$\begin{array}{l}\forall a, pn, m \cdot a \in PLATFORM \wedge pn \in PN \wedge \\ \quad m \in SENCRYPT \wedge a \mapsto pn \in \mathrm{dom}(as_{skey}) \wedge \\ \quad sencrypt_s(m) = as_{skey}(a \mapsto pn) \wedge \\ \quad trust(a) = TRUE \wedge m \in analz(m_p) \\ \quad \Rightarrow trustlevel(a \mapsto pn) = 2\end{array}$$

Since there are three variables $as_{skey}, m_p, trustlevel$ in events in the invariant, the POs is generated if one of these variables alters in the events. The POs demand that the invariants hold on transition of these variables. These are the general ideas of proving the invariants: when the new key k is added to as_{skey} in some events, we prove that k has not been used in encryption yet; when the new message m is added to m_p, we prove that for every encryption $m \in parts[m_p \cup m]$, $trustlevel(a \mapsto pn) = 2$, and for $m \in analz(m_p \cup m) \Rightarrow m \in parts[m_p \cup m]$, the invariant holds; when $trustlevel'(a \mapsto pn) = -1$ or 1, we prove that there is a contradiction in conditions; when $trustlevel'(a \mapsto pn) = 2$, the invariant is directly proved. The basic idea on leveraging POs of the following invariants is similar to this one.

In Inv_confidentiality_2, it is not clear that PN is trusted when $trustlevel(a \mapsto pn) = 2$. Here, we prove the invariant Inv_authenticity:

Inv_authenticity: For the trusted platform a, if $trustlevel(a \mapsto pn) = 2$, then $a^{-1}(pn)$ is trusted.

$$\begin{array}{l}\forall a, pn \cdot a \in PLATFORM \wedge pn \in PN \wedge \\ \quad trust(a) = TRUE \wedge trustlevel(a \mapsto pn) = 2 \Rightarrow \\ \quad pn \in ran(a_{pn}) \wedge trust\left(a_{pn}^{-1}(pn)\right) = TRUE\end{array}$$

It is delicate to prove this invariant that we add and prove dozens of supported invariants. If readers are interested in proving, please refer to our source code.

TABLE IV
STATISTICS OF PROOF OBLIGATIONS PROVED ON REFINEMENT AND EXTENSION STEPS

Machine	Total	Manual	Context	Total	Manual
Total	873	636	Total	224	215
R0	3	0	E0	0	0
R1	52	24	E1	1	1
R2	76	30	E2	81	79
R3	134	85	E3	22	15
R4	41	29	E4	120	120
R5	51	48			
R6	516	420			

Hence, by Inv_confidentiality_2 and Inv_authenticity, the messages encrypted by a's key are sent only when the $a^{-1}(pn)$ is trusted.

Inv_confidentiality_3: If the platform a is trusted and sends an encrypted message, the encrypted message cannot be decrypted by the adversary, since the key used for encryption cannot be analyzed by the adversary.

$$\begin{aligned}&\forall a, pn, m \cdot a \in PLATFORM \wedge pn \in PN \wedge \\ &\quad a \mapsto pn \in \mathrm{dom}(as_{skey}) \wedge m \in SENCRYPT \wedge \\ &\quad sencrypt_s(m) = as_{skey}(a \mapsto pn) \wedge \\ &\quad trust(a) = TRUE \wedge m \in analz(m_p) \Rightarrow \\ &\quad (\forall k \cdot k \in SKEY \wedge dh_{equal}(k \mapsto sencrypt_s(m)) \\ &\quad = TRUE \Rightarrow k \notin analz(m_p))\end{aligned}$$

That is, the adversary cannot get the key of the trusted platform. Besides, the key cannot be got by other trusted entities. Because the *trusted* platforms only act according to the protocol application and we have added and proved the invariants in R3 that for the different trusted platforms a_1 and a_2, $as_{skey1s}(a_1 \mapsto pn_1) \neq as_{skey1s}(a_2 \mapsto pn_2)$ and $as_{skey}(a_1 \mapsto pn_1) \neq as_{skey}(a_2 \mapsto pn_2)$. So the AK problem is solved.

VI. RESULTS

We have proved all the proof obligations (POs) generated by Rodin 2.4, and the statistics are listed in Table IV. Most of the POs are proved manually, and only a few are proved automatically by Rodin's tool. As R6 and E4 start the implementation of the concrete key agreement process, the rest of the levels are reusable for other protocols. Though it is still more sophisticated in proving R6 than the other levels, the percentage of POs in the reusable levels is 42.0%.

VII. FUTURE WORK

As the TPM-based security protocol, such as DAAODV, does not require the entirety of the available TPM functions, we only formalize parts of the interfaces of TPM in our proposal. Thus, one direction of our future work is to expand our refinement framework to more general applications by formalizing all the interfaces of TPM, such as the following:

- *Extend*: When a new measurement is extended to the PCRs, the state of the platform changes. To prevent the design flaws, we need to make formal invariants to ensure that the specific state only allows the specific operations.
- *Seal*: The command encrypts data and specifies a state in which the TPM must be in order for the data to be decrypted. We need to ensure the adversary cannot get the data in this state before sealing is called.

In addition, in our case study, we only prove the authenticity and confidentiality in order to show how to use the lower level of the refinement framework such that our refinement could be extended. One possible extension is the general refinement framework in prevention of deadlock. The goal is that when users design new protocols, the constraint on deadlock is already proven on the lower level of the refinement that users need not to know how to make the invariant and prove it again. Though the problem is not related to the security issue or TPM, it is also a challenge of providing a general framework to guarantee this property. The other one is to prove the anonymity of DAAODV protocol. DAAODV is based on DAA and PBA, which are both anonymous schemes. The purpose of applying these schemes is to hide all private information, such as the configuration of the platform. Thus, the anonymity of DAAODV needs to be proved in the future work. Note that for illustrating the framework more conveniently, we simplify the original DAAODV protocol, therefore, the other future work is to prove the security of more sophisticated original DAAODV.

It is worth mentioning that if the assumptions 1) or 2) in Section III become invalid, the soundness of TPM and the upper TCB need to be further verified. Since the implementation code of TPM is large [12], it is hard to completely fulfill the assumption 1). However, if the code is given publicly, there is a chance to verify the specific commands of TPM before we use them. Besides, some commands such as DAA scheme have been formally verified [8], [9], and some other TPM APIs have been verified as well [12]. It becomes much easier to only use the verified commands instead of verifying all the commands of TPM first. So, if the assumption 1) is invalid, we should additionally verify the soundness of the TPM commands which are used in our protocol. As [12] provides a refined implementation on the TPM commands, there are collaboration opportunities that we could build a more complete refinement architecture which includes not only the components for application design but also the verified TPM commands, such that the developers are more confident about security of their application design.

To the assumption 2), another direction of the future work is to formally verify the TCB. Since there have been a few approaches on designing and minimizing the TCB [6], [7], [36], we could verify the TCB before applying it to our protocol application.

VIII. CONCLUSIONS

In this paper, we make C-code-like formal modeling to ensure the security on the application level. The properties of TPM-based security infrastructure, the extended Dolev-Yao attacker model and the abstract communication environment are presented at the early refinement. In the case study, we refine a TPM-based key agreement protocol and prove the POs to solve AK problem. As a result, this approach ensures the security of the protocol not only at the abstract protocol level, but also at the concrete software level.

APPENDIX A
NOTATIONS

The notations in Event-B are shown in Table V.

TABLE V
NOTATIONS IN EVENT-B

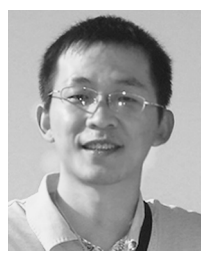
Notation	Definition
$\mathbb{P}(S)$	power set, $\mathbb{P}(S) = \{s \mid s \subseteq S\}$
$S \times T$	$S \times T = \{x \mapsto y \mid x \in S \wedge y \in T\}$
$r[S]$	$r[S] = \{y \mid \exists x \cdot x \in S \wedge x \mapsto y \in r\}$
$S \leftrightarrow T$	relations, $S \leftrightarrow T = \mathbb{P}(S \times T)$
r^{-1}	$r^{-1} = \{y \mapsto x \mid x \mapsto y \in r\}$
$p; q$	$\forall p, q \cdot p \in S \leftrightarrow T \wedge q \in T \leftrightarrow U \Rightarrow$ $p; q = \{x \mapsto y \mid (\exists z \cdot x \mapsto z \in p \wedge z \mapsto y \in q)\}$
$p \circ q$	$p \circ q = q; p$
$S \lhd r$	$S \lhd r = \{x \mapsto y \in r \wedge x \in S\}$
$\mathrm{dom}(r)$	domain, $\forall r \cdot r \in S \leftrightarrow T \Rightarrow$ $\mathrm{dom}(r) = \{x \cdot (\exists y \cdot x \mapsto y \in r)\}$
$\mathrm{ran}(r)$	range, $\forall r \cdot r \in S \leftrightarrow T \Rightarrow$ $\mathrm{ran}(r) = \{y \cdot (\exists x \cdot x \mapsto y \in r)\}$
$S \nrightarrow T$	partial functions, $S \nrightarrow T = \{r \cdot r \in S \leftrightarrow T \wedge r^{-1}; r \subseteq T \lhd \mathrm{id}\}$
$S \rightarrow T$	total functions, $S \rightarrow T = \{f \cdot f \in S \nrightarrow T \wedge \mathrm{dom}(f) = S\}$
$S \rightarrowtail T$	bijections, $S \rightarrowtail T = \{f \cdot f \in S \nrightarrow T \wedge$ $f^{-1} \in T \nrightarrow S \wedge \mathrm{dom}(f) = S \wedge \mathrm{ran}(f) = T\}$
$\bigcup z \cdot P \mid S$	P must constrain the variables in z $\forall z \cdot P \Rightarrow S \subseteq T \Rightarrow$ $\bigcup z \cdot P \mid S = \{x \mid x \in T \wedge \exists z \cdot P \wedge x \in S\}$ where $\neg\mathrm{free}(x, z, T)$, $\neg\mathrm{free}(x, P)$, $\neg\mathrm{free}(x, S)$, $\neg\mathrm{free}(x, z)$

ACKNOWLEDGMENT

The authors sincerely thank the anonymous reviewers for their invaluable feedback which helped improve this paper.

REFERENCES

[1] Wiki of trusted platform module [Online]. Available: http://en.wikipedia.org/wiki/Trusted_Platform_Module
[2] T.C.G. Tpm main specification. Trusted Computing Group [Online]. Available: http://www.trustedcomputinggroup.org/developers
[3] B. Parno, J. M. McCune, and A. Perrig, "Bootstrapping trust in commodity computers," in *Proc. IEEE Symp. Security and Privacy*, 2010, pp. 414–429.
[4] Department of Defense Trusted Computer System Evaluation Criteria, Department of Defense, Dec. 1985.
[5] D. Grawrock, *Dynamics of A Trusted Platform: A Building Block Approach*, ser. Books by engineers, for engineers. USA: Intel Press, 2009.
[6] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and A. Seshadri, "Minimal tcb code execution," in *Proc. IEEE Symp. Security and Privacy*, 2007, pp. 267–272.
[7] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. D. Gligor, and A. Perrig, "Trustvisor: Efficient tcb reduction and attestation," in *Proc. IEEE Symp. Security and Privacy*, 2010, pp. 143–158.
[8] M. Backes, C. Hritcu, and M. Maffei, "Type-checking zero-knowledge," in *Proc. ACM Conf. Computer and Communications Security*, 2008, pp. 357–370.
[9] M. Backes, M. Maffei, and D. Unruh, "Zero-knowledge in the applied picalculus and automated verification of the direct anonymous attestation protocol," in *Proc. IEEE Symp. Security and Privacy*, 2008, pp. 202–215.
[10] S. Gürgens, C. Rudolph, D. Scheuermann, M. Atts, and R. Plaga, "Security evaluation of scenarios based on the TCG's TPM specification," in *Proc. ESORICS*, 2007, pp. 438–453.
[11] A. H. Lin, R. L. Rivest, and A. H. Lin, "Automated Analysis of Security Apis," Master's thesis, MIT, Cambridge, MA, USA, 2005.
[12] N. Polikarpova and M. Moskal, "Verifying implementations of security protocols by refinement," in *Proc. VSTTE*, 2012, pp. 50–65.
[13] 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 *Proc. SOSP*, 2009, pp. 207–220.
[14] S. Delaune, S. Kremer, M. D. Ryan, and G. Steel, "Formal analysis of protocols based on TPM state registers," in *Proc. CSF*, 2011, pp. 66–80.
[15] Y. Yang, Y. Li, R. H. Deng, and F. Bao, "Shifting inference control to user side: Architecture and protocol," *IEEE Trans. Depend. Secure Comput.*, vol. 7, no. 2, pp. 189–202, Apr./Jun. 2010.
[16] S. Delaune, S. Kremer, M. D. Ryan, and G. Steel, "A formal analysis of authentication in the tpm," in *Proc. 7th Int. Conf. Formal Aspects of Security and Trust (FAST'10)*, 2011, pp. 111–125.
[17] C. Sprenger and D. A. Basin, "Developing security protocols by refinement," in *Proc. ACM Conf. Computer and Communications Security*, 2010, pp. 361–374.
[18] G. Bella, *Formal Correctness of Security Protocols*, ser. Information Security and Cryptography. Dordrecht: Springer, 2007.
[19] J.-R. Abrial and S. Hallerstede, "Refinement, decomposition, and instantiation of discrete models: Application to event-b," *Fundam. Inform.*, vol. 77, no. 1–2, pp. 1–28, 2007.
[20] J.-R. Abrial, *Modeling in Event-B—System and Software Engineering*. Cambridge, U.K.: Cambridge University Press, 2010.
[21] S. Blake-Wilson, D. Johnson, and A. Menezes, "Key agreement protocols and their security analysis," in *Proc. IMA Int. Conf.*, 1997, pp. 30–45.
[22] W. Diffie, P. C. V. Oorschot, and M. J. Wiener, Authentication and Authenticated Key Exchanges 1992.
[23] T. Matsumoto, Y. Takashima, and H. Imai, "On seeking smart public-key distribution systems," *Trans. IECE Japan*, vol. E69, no. 2, pp. 99–106, 1986.
[24] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theory*, vol. 22, no. 6, pp. 644–654, Nov. 1976.
[25] J. C. Herzog, "The diffie-hellman key-agreement scheme in the strandspace model," in *Proc. CSFW*, 2003, pp. 234–247.
[26] L. Ngo, C. Boyd, and J. G. Nieto, "Automated proofs for diffie-hellman-based key exchanges," in *Proc. CSF*, 2011, pp. 51–65.
[27] W. Huang, Y. Xiong, and D. Chen, "Daaodv: A secure ad hoc routing protocol based on direct anonymous attestation," in *Proc. CSE (2)*, 2009, pp. 809–816.
[28] E. F. Brickell, J. Camenisch, and L. Chen, "Direct anonymous attestation," in *Proc. ACM Conf. Computer and Communications Security*, 2004, pp. 132–145.
[29] L. Chen, H. Löhr, M. Manulis, and A.-R. Sadeghi, "Property-based attestation without a trusted third party," in *Proc. ISC*, 2008, pp. 31–46.
[30] T.C.G., TCG Specification Architecture Overview 2007.
[31] J. Abrial, M. Butler, S. Hallerstede, T. Hoang, F. Mehta, and L. Voisin, "Rodin: An open toolset for modelling and reasoning in event-b," *Int. J. Software Tools for Technology Transfer (STTT)*, vol. 12, no. 6, pp. 447–466, 2010.
[32] J.-R. Abrial, *The B-Book: Assigning Programs to Meanings*. Cambridge, U.K.: Cambridge Univ. Press, 1996.
[33] D. Dolev and A. C.-C. Yao, "On the security of public key protocols," *IEEE Trans. Inf. Theory*, vol. 29, no. 2, pp. 198–208, Mar. 1983.
[34] E. Kiltz, A. O'Neill, and A. Smith, "Instantiability of RSA-OAEP under chosen-plaintext attack," in *Proc. CRYPTO*, 2010.
[35] M. Strasser and H. Stamer, "A software-based trusted platform module emulator," in *Proc. TRUST*, 2008, pp. 33–47.
[36] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and H. Isozaki, "Flicker: An execution infrastructure for TCB minimization," in *Proc. EuroSys*, 2008, pp. 315–328.



Wenchao Huang received the B.S. and Ph.D. degrees in computer science from the University of Science and Technology of China in 2006 and 2011, respectively. He is currently a Postdoc in the School of Computer Science and Technology, University of Science and Technology of China.

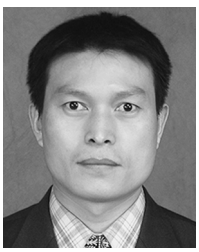
His current research interests include information security, trusted computing, formal methods, and mobile computing.

Yan Xiong received the B.S., M.S., and Ph.D. degrees from the University of Science and Technology of China in 1983, 1986, and 1990, respectively.

He is a professor with the School of Computer Science and Technology, University of Science and Technology of China. His main research interests include distributed processing, mobile computing, computer network, and information security.

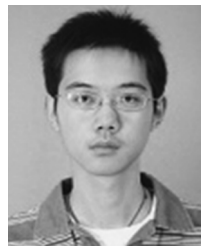
Xingfu Wang received the B.S. degree in electronic and information engineering from Beijing Normal University of China in 1988, and the M.S. degree in computer science from the University of Science and Technology of China in 1997.

He is an associate professor in the School of Computer Science and Technology, University of Science and Technology of China. His current research interests include information security, data management, and WSN.



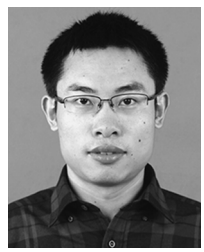
Fuyou Miao received the Ph.D. degree in computer science from the University of Science and Technology of China in 2003.

He is an associate professor with the School of Computer Science and Technology, University of Science and Technology of China. His research interests include applied cryptography, trusted computing, and mobile computing.



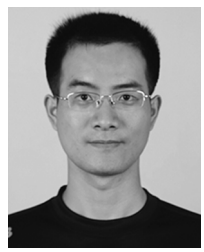
Chengyi Wu received the B.S. and M.S. degrees in computer science from the University of Science and Technology of China, in 2009 and 2012, respectively.

He is currently working for Baidu Online Network Technology (Beijing) Co. Ltd. His current work interests include big data and data management.



XuDong Gong received the B.S. degree in computer science from the University of Science and Technology of China in 2010. He is currently working toward the Ph.D. degree at the Department of Computer Science and Technology, University of Science and Technology of China.

His current research interests include information security and data management.



Qiwei Lu received the B.S. degree in information security from Nanjing University of Posts and Telecommunications, China, in 2010. He is currently working toward the Ph.D. degree at the Department of Computer Science and Technology, University of Science and Technology of China.

His current research interests include information security, privacy-preserving data publishing, mining and computation, geo-social networks, and LBS.