Verifying TPM DevID Provisioning*

Sarah Johnson and Perry Alexander

Institute for Information Sciences
The University of Kansas
Lawrence, KS 66045
{sarahjohnson,palexand}@ku.edu

Abstract. Provisioning of TPM-based secure device identifiers must be verified to ensure such identifiers are strongly bound to their devices. Lack of such assurance jeopardizes high-integrity decisions and reporting that rely upon associating data with devices. We develop models of two TCG provisioning protocols and verified they provide the assurances necessary to produce a strong cryptographic binding of TPM key to device. As a result we can be assured that initial and local device identifiers are strongly bound to their associated devices.

Keywords: Secure Device Identifiers · TPM · Verification.

1 Introduction

Development and deployment of trusted systems requires strong device identification (Martin et al., 2008). An entity should have confidence that a remote device is the device it claims to be. An ideal method for fulfilling this need uses asymmetric keys in the Trusted Platform Module (TPM) as secure device identifiers (Trusted Computing Group, 2021). A secure device identifier (DevID) is defined as an identifier that is cryptographically bound to a device by a trusted party (Institute of Electrical and Electronics Engineers, 2018). As a secure Root of Trust for Storage and Reporting, a TPM provides the necessary protections for generating and storing DevIDs with suitable integrity guarantees.

The Trusted Computing Group (TCG) describes protocols for assuring a key to be resident in a specific device's TPM (Trusted Computing Group, 2021). These protocols are performed by a trusted Certificate Authority (CA) in communication with a certificate-requesting device and maintain a cryptographic evidentiary chain linking a DevID to a specific TPM-containing device (Trusted Computing Group, 2021). For each provisioning procedure, the TCG outlines a protocol for the CA and the certificate-requesting entity. Furthermore, the TCG

* This work is funded in part by the NSA Science of Security initiative contract #H98230-18-D-0009 and Defense Advanced Research Project Agency contract #HR0011-18-9-0001 and Honeywell FMT Purchase Order #N000422909. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

claims that each protocol provides certain assurances. These assurances are the basis for the resulting cryptographic evidentiary chain. Each assurance manifests as an assertion regarding either TPM-residency, key attributes, or previously-issued certificates.

Our work examines correctness of two key certification protocols: (i) OEM Creation of an IAK Certificate based on an EK Certificate; and (ii) Owner Creation of an LAK Certificate based on an IAK Certificate. These DevID establishment protocols are essential for our work in remote attestation (Coker et al., 2011; Petz and Alexander, 2022) where device appraisal requires strong identification of an attestation source. Specifically, that the signing key for attestation is bound to the attestation manager responsible for attestation. Formal verification is required for integration into our formally verified/synthesized attestation environment(Petz and Alexander, 2022). Our contributions of this research include: (i) design and implementation of an abstract formal model of TPM command execution; (ii) an in-depth analysis of the TCG-provided IAK and LAK certification protocols, and (iii) a recommendation for clarifying the IAK provisioning procedure.

2 Related Work

The Privacy CA protocol for the TPM 1.2 has been extensively studied using various formal methods. The work of Chen and Warinschi (2010) analyzes the Privacy CA protocol in the presence of an adversary. They suggest a small modification to the protocol that enhances its security without changing the existing functionality of the TPM 1.2. Our work similarly suggests a small modification to the IAK provisioning procedure which requires no change to the existing functionality of the TPM 2.0. In contrast, earlier work of Halling and Alexander (2013); Halling (2013) analyzes the Privacy CA protocol but does not consider any specific adversary. Instead, they consider the functional correctness of the protocol and specifically examine the protocol implementation to ensure that it produces correct results. Our current work has many parallels to earlier work from Halling and Alexander. Here we attempt to determine whether the implementations of certain key certification protocols result in the assurances that they should. Furthermore, Halling and Alexander abstractly model a large subset of the TPM 1.2 commands in the PVS specification language. They model TPM command execution as a transition system over an abstract system state using a monadic approach. Here we implement a state transition system directly over a smaller subset of TPM 2.0 commands and non-TPM commands targeted to device identifiers.

The TPM 1.2 Privacy CA protocol was replaced by the Direct Anonymous Attestation (DAA) scheme in the TPM 2.0. The work of Whitefield et al. (2019) and Wesemeyer et al. (2020) thoroughly study the TPM 2.0 DAA scheme. Collectively, they develop a symbolic model, a C++ implementation, and a formally-verified fix of the scheme. Both the Privacy CA and DAA protocols aim to allow remote authentication of a device while maintaining its anonymity. As such these

protocols differ from the key certification protocols analyzed here that provide individual device identification.

Other work uses formal techniques to examine various aspects of both the TPM 1.2 (Delaune et al., 2011a,b) and 2.0 (Shao et al., 2015, 2018). These include analyzing authentication (Delaune et al., 2011a), platform configuration registers (PCRs) (Delaune et al., 2011b), enhanced authorization (Shao et al., 2015), and hash-based message authentication code (HMAC) mechanisms (Shao et al., 2018). They utilize tools and techniques such as SAPIC (Kremer and Künnemann, 2016), Tamarin (Meier et al., 2013), and stateful applied pi calculus (Milner, 1999), amongst others.

3 Background

The Trusted Platform Module (TPM) is a microcontroller that complies with the ISO/IEC 11889:2015 international standard. The TPM and its specification were designed by the Trusted Computing Group (TCG) to act as a hardware root-of-trust for PC system security. To this end, TPMs have capabilities for secure generation of keys, algorithm agility, secure storage of keys, enhanced authorization, measurement storage and reporting, device identification, and NVRAM storage (Arthur et al., 2015).

TPM keys have attributes that are set at creation-time and restrict how they may be used. Attributes include `FixedTPM` to control duplication, `Sign` and `Decrypt` to indicate use as a signing or decryption key, and `Restricted` to limit operations to TPM-generated data.

All TPMs are shipped with a root storage key, the endorsement key (EK). The EK is a fixed-TPM, restricted, decryption key installed by the TPM Manufacturer in an immutable, shielded TPM location. The EK has an associated certificate that binds the EK to a specific TPM. Although not a DevID itself, the EK plays a significant role as a root-of-trust in provisioning a TPM-based device identity.

A secure device identifier (DevID) is an asymmetric TPM key that is cryptographically bound to a device by a DevID certificate. When using a TPM key for secure device identity, restrictions on attributes enforce best security practices. The `Sign` attribute must be set and the `Decrypt` attribute not set to indicate the key is a signing key. Critically, a DevID must have the `FixedTPM` attribute set to ensure it will never be duplicated, transferred, or copied to another TPM. The `Restricted` attribute is optional. When the `Restricted` attribute is set the key becomes an attestation key (AK). The AK is so named for its unique ability to prove that a data object is contained in the same TPM as itself.

An AK may only sign objects produced by the TPM. This includes TPM-internal structures such as platform configuration registers, digests, and keys. TPM-based attestation involves high-integrity reporting of system measurements stored in the TPM's platform configuration registers (PCRs). PCRs store a measurement chain composite in a way that guarantees integrity. Rather than being set like a traditional register, a PCR is extended by concatenating a new hash

Key	`FixedTPM`	`Sign`	`Decrypt`	`Restricted`	Creator
EK	X		X	X	TPM Manufacturer
IAK	X	X		X	OEM
IDevID	X	X			OEM
LAK	X	X		X	Owner
LDevID	X	X			Owner

Table 1. EK and DevID Attribute Requirements

to the current PCR value, hashing and storing the result. Thus, a PCR captures both measurement value and order over an arbitrarily long measurement sequence. A TPM attestation is called a quote and minimally consists of a PCR composite signed to guarantee integrity and authenticity. To tie attestation data to a specific device, the signing key must be an AK. An AK's `Restricted` attribute and associated certificate results in cryptographic binding of signatures to a specific device.

The acronyms DevID and AK are prefixed by the letter I or L denoting initial or locally significant respectively. Initial device identifiers are installed by OEMs on TPM-containing devices at manufacturing time. Initial device identifiers are permanent identifiers of TPM-containing devices, intended to be long-lived and usable for the lifetime of the device. As such, initial identifiers should be used sparingly outside the TPM to limit risk of compromise. Locally significant device identifiers are installed by device owners and serve as aliases for the device. Unlike initial device identifiers they need not be long-lived, may be created and installed at any time, and shared externally.

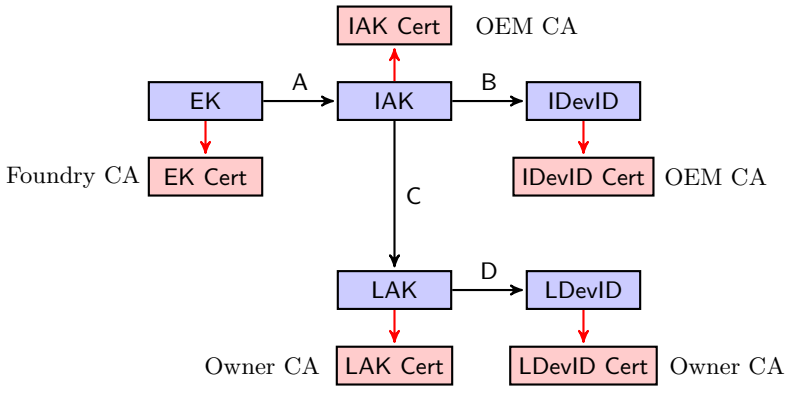

Fig. 1. Keys and certificates created by CA provisioning. Labeled black arrows are trust relationships. Red arrows denote key inclusion in certificates. Certificates are labeled with their associated CA (Trusted Computing Group, 2021).

Trust for new DevIDs is inherited from an existing certificate forming a chain-of-trust from the new DevID back to a trust anchor (Trusted Computing Group, 2021). The IAK certificate is the first and most important certificate to identify a device. Proving that an IAK belongs to a specific device requires first binding the IAK to a specific TPM using the EK and then binding the TPM to a specific device. According to the TCG, the IAK certificate is intended to provide definitive evidence that a key belongs to a specific device. Therefore, an IAK certificate serves as the trust anchor and root node in a chain of DevID certificates. In issuing an IAK certificate, the OEM's CA makes an assertion that is a primary security dependency for the provisioning of all future DevIDs. All subsequent certificates in a chain identify the device represented by the IAK certificate.

Since a key with the `Restricted` attribute set has the ability to prove that some unknown key is loaded on the same TPM as itself, AK certificates are used as the existing certificate in the provisioning of new DevIDs. This means that only IAK and LAK certificates are parent nodes in a chain of DevID certificates and that IDevID and LDevID certificates are leaf nodes. A basic AK and DevID certificate configuration is shown in Figure 1.

4 Protocol Execution Model

We use a labeled transition system to model abstract TPM and TSS command execution. State is represented as a collection of objects installed in the TPM while transitions perform operations over state as defined by commands. Labels represent individual commands and label sequences represent protocols. DevID-provisioning protocols are modeled as sequences of state transformations corresponding with commands executed by the CA and the OEM or Owner.

4.1 Keys and Certificates

Our model inductively defines a `pubKey` and `privKey` type for public keys and private keys respectively. A key of either type requires a unique identifier and a sequence of boolean values describing whether a particular attribute is set or not set. A key pair consists of a `pubKey` and a `privKey` with the same identifier and attributes.

Certificates are defined by the inductive type `signedCert` consisting of a public key, an identifier, and a private key. The identifier represents the certificate's Subject field and may include information describing either the TPM or the device. The private key parameter denotes the CA's key that performed the signature over the certificate.

4.2 Commands

Protocols used to create DevID certificates require performing TPM, TSS and non-TPM commands. A command may rely on a variety of parameters such as

keys, nonces, certificates, as well as messages that define structures an entity may use or produce. The `message` type is an abstract representation of these structures shown in Figure 2.

```
Inductive message : Type :=
| publicKey : pubKey → message
| privateKey : privKey → message
| hash : message → message
| signature : message → privKey → message
| TPM2B_Attest : pubKey → message
| encryptedCredential : message → randType → pubKey → message
| randomNum : randType → message
| TCG_CSR_IDevID : identifier → signedCert → pubKey → message
| TCG_CSR_LDevID : message → signedCert → message
| signedCertificate : signedCert → message
| pair : message → message → message.
```

Fig. 2. TPM Message Model

An entity may derive additional messages from a collection of known messages. For example, given a message of the form `signature m k`, `m` may be deduced while given a message in the form `encryptedCredential n g k`, no messages may be deduced. Message inference is modeled in two equivalent ways: (i) a recursive function `inferFrom`; and (ii) an inductive proposition `inferrable`. Additional messages may be derived from signed messages, `TPM2B_Attest` structures, certificate signing requests (CSRs), certificates, and message pairs. Other messages either contain no additional information or information is concealed.

`tpm_state` and `state` are lists of messages. The `tpm_state` type contains messages that a restricted signing key may operate on. Such messages must be in one of two forms: (i) objects constructed from TPM-internal data such as private keys and PCRs; or (ii) digests produced by signature hash operations. The `state` type contains messages visible to the system.

A labeled transition system models command execution in the canonical fashion as a triple $(S, L, \rightarrow)$ where S is a set of states, L is a set of labels, and $\rightarrow \subseteq S \times L \times S$ is a labeled transition relation. In this case, S is defined as `tpm_state * state` pairs; L is the set of `command` values; and $\rightarrow$ is the `execute` relation defining single command execution. Specifically, `execute`is defined using an inductive proposition relating a current state pair, a command, and a next state pair.

While most TPM command functions are easily inferred from their name, several are worth explaining in some detail. The `TPM2_Certify` command provides proof that an object is loaded in the TPM by producing a signed `TPM2B_Attest` structure. The command requires two inputs: (i) a public key to be certified; and (ii) a private key to sign the attestation structure. The private key must have

```
Inductive command : Type :=
| TPM2_Hash : message → command
| CheckHash : message → message → command
| TPM2_Sign : message → privKey → command
| TPM2_Certify : pubKey → privKey → command
| CheckSig : message → pubKey → command
| TPM2_MakeCredential : message → randType → pubKey → command
| TPM2_ActivateCredential : message → privKey → privKey → command
| MakeCSR_IDevID : identifier → signedCert → pubKey → command
| MakeCSR_LDevID : message → signedCert → command
| CheckCert : signedCert → pubKey → command
| CheckAttributes : pubKey → Restricted → Sign → Decrypt → FixedTPM →
command
| MakePair : message → message → command.
```

Fig. 3. TPM Command Data Structure

the `Sign` attribute set and must be loaded. To execute, the TPM verifies the public key parameter's inverse is also loaded.

The `TPM2_MakeCredential` command is used when a remote entity, desires to affirm that some private key is loaded on the same TPM as a particular EK. This command produces an `encryptedCredential` structure and requires three inputs: the cryptographic name of a key to be credentialed; a secret; and a public EK. The `TPM2_ActivateCredential` command is used by the recipient of an encrypted credential blob to release its secret. The secret contained in an encrypted credential blob is only released if the credentialed key is loaded on the same TPM as the EK. When executing the `TPM2_ActivateCredential` command, the TPM first decrypts the blob with the EK then verifies that the private key corresponding with the name field is also loaded. The secret is only released if both steps succeed. The secret value may then be returned to the remote CA to validate the result.

The `MakeCSR_IDevID` command produces a `TCG_CSR_IDevID` structure. A `TCG_CSR_IDevID` is a certificate signing request (CSR) that contains the data required to couple an initial DevID to a TPM-containing device. This structure is used any time a provisioning procedure uses the EK certificate. The `MakeCSR_LDevID` command is similar except that it produces a `TCG_CSR_LDevID` structure that includes the certification information for a locally significant DevID.

4.3 Protocols

Commands are sequenced linearly by the `sequence` type to form protocols. Protocol execution is defined as an inductive proposition relating a current state pair, a command sequence, and a next state pair. Sequential execution first executes the command at the head of a sequence using the single command execution relation `execute` followed by executing the remaining sequence using the

sequential execution relation `seq_execute`. The next state pair produced by the single command becomes the current state pair for the following sequence.

```
Inductive sequence : Type :=
| Sequence : command → sequence → sequence
| Done : sequence.

Inductive seq_execute : tpm_state * state → sequence → tpm_state * state →
Prop :=
| SE_Seq : ∀ ini mid fin c s,
    execute ini c mid →
    seq_execute mid s fin →
    seq_execute ini (Sequence c s) fin
| SE_Done : ∀ ini, seq_execute ini Done ini.
```

Fig. 4. Sequential command execution model

We prove two intermediate properties of sequential execution. First, sequential execution is deterministic. Given an initial state pair and a command sequence, there is at most one final state pair that satisfies the `seq_execute` relation implying that command sequence execution is a partial function. Second, sequential execution is monotonic over state. Given a current state pair, command sequence, and initial state pair, the initial state is always a subset of the final state implying that command sequence execution is monotonic.

5 Proof Techniques

The TCG describes several provisioning protocols (Trusted Computing Group, 2021) that aim to maintain a cryptographic evidentiary chain linking a DevID to a specific TPM and device. For each protocol, the specification outlines steps for the CA and the certificate-requesting entity. The specification claims that each protocol provides certain assurances. Each assurance manifests as an assertion regarding either TPM-residency, key attributes, or previously-issued certificates and provides the basis for the resulting cryptographic evidentiary chain formed by a chain of certificates. Therefore, it is important to verify that each protocol guarantees its associated assurances.

When verifying these assurances, we consider two scenarios: (i) the certificate-requesting entity and the CA are both trusted to execute their steps correctly; and (ii) only the CA is trusted to execute its steps correctly. We use a novel proof technique for each scenario to verify TPM-residency — minimal initial state and supersequence respectively.

5.1 Minimal Initial State

A minimal initial state aims to quantitatively describe the prerequisites for executing a command sequence. Given a command sequence, a minimal initial state is defined as the smallest initial state that allows successful execution of the sequence. The proof obligation describing this property in Figure 5 has two parts: (i) the minimal initial state is a lower bound on the set of possible initial states and (ii) the minimal initial state is sufficient for successful execution.

```
Definition lower_bound seq minTPM min : Prop :=
  ∀ iniTPM ini fin,
  seq_execute (iniTPM, ini) seq fin →
  (minTPM ⊆ iniTPM) ∧ (min ⊆ ini).

Definition sufficient seq minTPM min : Prop :=
  ∃ fin, seq_execute (minTPM, min) seq fin.
```

Fig. 5. Minimal Initial State

We will use the minimal initial state to demonstrate that an unknown key is loaded on the same TPM as some previously-certified key if the certificate-requesting entity successfully follows the TCG's recommended procedure.

5.2 Supersequence

A CA will only issue a certificate if it receives a valid certificate request. Arbitrary certificate request production is described in Figure 6. The certificate-requesting entity executes some arbitrary command sequence; the initial state pair contains only messages that cannot be generated; the command sequence produces some message in the final state; and a CA determines that the message is a valid certificate request.

```
Definition arbitrary_CSR_production (steps_CA : message → Prop) : Prop :=
  ∀ seq iniTPM ini finTPM fin,
  seq_execute (iniTPM, ini) seq (finTPM, fin) →
  (∀ m', needsGenerated m' → ¬In m' iniTPM) →
  (∀ m', needsGenerated m' → ¬In m' ini) →
  In csr fin →
  steps_CA csr.
```

Fig. 6. Arbitrary CSR production definition.

We constrain the initial state of the certificate-requesting entity to include only those messages needed for the request that cannot be generated by a com-

mand or command sequence. To ensure the contents of the certificate request are fresh, the initial state may contain only keys and certificates.

To produce a valid certificate request, specific commands must be executed in a specific order. Due to the shape of a valid request and the checks performed by a CA, we can prove that any command sequence that the certificate-requesting entity performed to generate the request is a supersequence of the TCG-recommended procedure. A list is a supersequence of another list if and only if all the elements of the second list occur in order in the first — the elements need not occur consecutively. With this knowledge of the certificate-requesting entity's behavior, we can apply an abbreviated version of the minimal initial state technique to complete verification.

6 Identity Provisioning

We consider two DevID provisioning protocols in detail: (i) Owner creation of an LAK certificate based on an IAK certificate; and (ii) OEM creation of an IAK certificate based on an EK certificate. For each protocol, the TCG specification outlines steps for the CA and the certificate-requesting entity. The specification claims that each protocol provides certain assurances. Each assurance manifests as an assertion regarding either TPM-residency, key attributes, or previously-issued certificates and provides the basis for the resulting cryptographic evidentiary chain formed by a chain of certificates. We verify that each protocol guarantees its associated assurances, thereby determining their correctness.

When verifying these goals, we consider two scenarios: (i) the certificate-requesting entity and the CA are both trusted to execute their steps correctly; and (ii) only the CA is trusted to execute its steps correctly. Because trust for a new certificate is rooted in an existing certificate, these scenarios assume the precondition that previously-issued certificates imply the assurances guaranteed by their provisioning protocols.

6.1 Owner Creation of LAK Certificate based on IAK Certificate

The TCG's specification claims the protocol for creating an LAK certificate in Figure 7 provides these assurances: (A) the LAK has good attributes; and (B) the LAK is loaded on the same TPM as the IAK. These assurances correspond with the Chain of Trust line C in Figure 1.

Modeling this protocol begins by initializing states for the Owner and CA. The Owner must have its LAK, IAK, and IAK certificate, and the CA must have its own key and the public key of the OEM's CA. To enforce the randomness of cryptographic keys, all key values are specified to be pairwise distinct.

The protocol is modeled in two parts: the Owner's steps (Steps 0-5) followed by the CA's steps (Steps 6-7). Each part of the procedure is modeled using the initialized state and the sequential command construction defined previously. We construct a `sequence` value for the Owner and a function for the CA. Constructing the Owner's steps is straightforward and shown in Figure 8. Step 0

0. The Owner creates and loads the LAK
1. The Owner certifies the LAK with the IAK
2. The Owner builds the CSR
3. The Owner takes a signature hash of the CSR
4. The Owner signs the resulting hash digest with the LAK
5. The Owner sends the CSR paired with the signed hash to the CA
6. The CA verifies the recieved data
7. If all checks succeed, the CA issues the LAK certificate to the Owner

Fig. 7. LAK Certificate Creation Protocol

is assumed to have been performed previously since its results are already encapsulated in the Owner's initialized state. Each remaining step of the Owner corresponds with exactly one command in the model, namely `TPM2_Certify` for Step 1, `MakeCSR_LDevID` for Step 2, `TPM2_Hash` for Step 3, `TPM2_Sign` for Step 4, and `MakePair` for Step 5.

```
Definition steps1to5_Owner : sequence :=
TPM2_Certify pubLAK privIAK
;; MakeCSR_LDevID (signature (TPM2B_Attest pubLAK) privIAK) certIAK
;; TPM2_Hash
   (TCG_CSR_LDevID (signature (TPM2B_Attest pubLAK) privIAK) certIAK)
;; TPM2_Sign
   (hash (TCG_CSR_LDevID (signature (TPM2B_Attest pubLAK) privIAK) certIAK))
   privLAK
;; MakePair
   (TCG_CSR_LDevID (signature (TPM2B_Attest pubLAK) privIAK) certIAK)
   (signature
     (hash
       (TCG_CSR_LDevID
         (signature (TPM2B_Attest pubLAK) privIAK) certIAK)) privLAK)
;; Done.
```

Fig. 8. Owner's LAK Provisioning Protocol

The CA's steps in Figure 9 are more complex as they rely on the certificate request produced by the Owner. First, the CA waits to receive a certificate request from the Owner. The request must be in a specific format to be considered valid. The CA then executes Step 6 of the procedure defined by the sequence within `seq_execute`. If execution succeeds, the CA issues the LAK certificate to the Owner. The model includes several additional parameters and criteria to serve as a method for referencing elements of the certificate request within proof statements.

```
Definition steps_CA (msg : message) (iak lak : pubKey) (cert : signedCert)
 : Prop :=
  match msg with
  | (pair (TCG_CSR_LDevID (signature (TPM2B_Attest k) k0') (Cert k0 id k_ca'))
          (signature m k')) ⇒
       iak = k0 ∧ lak = k ∧ cert = (Cert k0 id k_ca') ∧
       seq_execute (iniTPM_CA, inferFrom msg ++ ini_CA)
         CheckHash m
           (TCG_CSR_LDevID (signature (TPM2B_Attest k) k0') (Cert k0 id k_ca'))
           ;; CheckSig (signature m k') k
           ;; CheckSig (signature (TPM2B_Attest k) k0') k0
           ;; CheckCert (Cert k0 id k_ca') pubOEM
           ;; CheckAttributes k Restricting Signing NonDecrypting Fixing
           ;; Done)
           (iniTPM_CA, inferFrom msg ++ ini_CA)
  | _ ⇒ False
  end.
```

Fig. 9. CA's LAK Provisioning Protocol

Assurance A is proved under both scenarios simultaneously by using only the CA's function. The `CheckAttributes` command in the CA's function corresponds with Step 6 of the provisioning procedure. It follows that successful execution implies the LAK has all attributes required to be an attestation key.

We now verify Assurance B under the conditions of the first scenario where the Owner and the CA are both trusted to execute their steps correctly. Recall that this proof trusts that the previously-performed IAK provisioning guarantees its associated assurances, specifically that the IAK has good attributes.

We construct a minimal initial state pair for `steps1to5_Owner` using the following intuition: (i) the private LAK and private IAK are loaded on the same TPM because the LAK is certified by the IAK; and (ii) the IAK certificate is known to the Owner because it is included in the CSR. The proof of the lower bound property uses this intuition. The proof of the sufficiency property uses the constructors of the `execute` relation to demonstrate that a next state pair exists that satisfies the `seq_execute` relation for the minimal initial state pair and `steps1to5_Owner`. This proof relies on two conditions, the IAK has good attributes and that the CA issues the LAK certificate.

This analysis of the Owner's steps' prerequisites in the form of a minimal initial state leads to the conclusion that the LAK and IAK must be loaded on the same TPM for the Owner to execute its steps. Therefore, we have now confirmed that Assurance B is in fact guaranteed by the protocol when we assume that both the Owner and the CA are trusted to execute their steps correctly.

We now attempt verification of the same goal under the second scenario where only the CA is trusted to execute its steps correctly. We define a cascading collection of recursive functions to determine whether a given command

sequence is a supersequence of `steps1to5_Owner`. Using the properties provided by the `arbitrary_CSR_production` definition, we prove the Owner's arbitrary command sequence satisfies this function.

From here we need only demonstrate that the private LAK and private IAK must be loaded on the same TPM to execute a command sequence with this property. We do this by reusing the intuition described in the first scenario that the private LAK and private IAK are loaded on the same TPM if the LAK is certified by the IAK.

Due to extra parameters and criteria of the `steps_CA` function, the formal theorem states that the inverse of the key contained in the IAK certificate and the inverse of the key contained in the new certificate are loaded on the same TPM. This is the assertion made by Assurance B. Therefore, we have proven that Assurance B is guaranteed by the protocol assuming only the CA is trusted to execute its steps correctly.

The procedure uses the previously provisioned IAK to prove the new LAK is loaded on the same TPM and thus contained in the device identified by the IAK certificate. Thus, when issuing the LAK certificate, the CA should use the same device identifying information as the IAK certificate's Subject field. To summarize, Assurance A is guaranteed by the attribute check performed by the CA and Assurance B is guaranteed by the IAK's `Restricted` attribute and the signed `TPM2B_Attest` structure.

6.2 OEM Creation of IAK Certificate based on EK Certificate

The TCG's specification claims the procedure in Figure 10 guarantees three assurances: (A) the IAK has good attributes; (B) the IAK is loaded on the same TPM as the EK; and (C) the EK certificate is valid. These assurances correspond with the Proof of Residency line A in Figure 1.

0. The OEM creates and loads the IAK
1. The OEM builds the CSR
2. The OEM takes a signature hash of the CSR
3. The OEM signs the resulting hash digest with the IAK
4. The OEM sends the CSR paired with the signed hash to the CA
5. The CA verifies the received data
6. If all of the checks succeed, the CA issues a challenge blob to the OEM
7. The OEM releases the secret nonce by decrypting the challenge blob and verifying the IAK name
8. The CA checks the returned nonce against the one generated in Step 6b
9. If the check succeeds, the CA issues the IAK certificate to the OEM

Fig. 10. IAK Certificate Creation Protocol

The procedure from Figure 10 is similar to the one described in Figure 7. We focus on the details that differ from the previous verification process.

The provisioning procedure may be viewed as the composition of four parts: the OEM's initial steps (Steps 0-4) followed by the CA's initial steps (Steps 5-6) followed by the OEM's final step (Step 7) followed by the CA's final steps (Steps 8-9). The initial steps of the OEM and the OEM's CA are constructed similarly to the steps of the Owner and the Owner's CA respectively. The OEM's final step is naturally constructed as a simple function. First the OEM waits to receive a challenge blob from the CA, then the OEM executes Step 7. The CA's final steps are implicit in the proof statements and do not require an explicit definition.

Proving Assurance A is trivial and proceeds identically to the corresponding proof in the previous section. Proving Assurance C is also quite similar since the CA checks the signature on the EK certificate with the public key of the TPM Manufacturer's CA.

We now verify Assurance B when the OEM and the CA are both trusted to execute their steps correctly. We implement the same strategy as before showing that the private IAK and private EK are contained in the OEM's minimal initial `tpm_state`. We build a minimal initial state pair for the composition of `steps1to4_OEM` and `step7_OEM` using the following intuition: (i) the EK certificate and public IAK are known to the Owner because they are included in the CSR; and (ii) the private IAK and private EK are loaded on the same TPM because the IAK is credentialed by the EK. The two required proofs to determine minimality proceed similarly to the corresponding proof in the previous section.

We now attempt to verify the same goal assuming only the CA is trusted to execute its steps correctly. We modify the `arbitrary_CSR_production` definition so that we can describe the additional interaction that takes place between the OEM and the CA. We now say that the OEM executes some arbitrary sequence of commands `seq1`; `seq1` produces some message `csr` in the OEM's intermediate state; the OEM's initial state contains only messages that need not be generated; the CA executes its steps on `csr` and sends the challenge blob `encryptedCredential (hash (publicKey iak)) g ek` to the OEM; the OEM executes some other arbitrary command sequence `seq2`; and `seq2` releases the secret nonce value `g` into the OEM's final state. The CA's final steps are implicit in that the nonce in the challenge blob is the same nonce in the OEM's state.

From here we prove that `seq2` is a supersequence of the correct final steps of the OEM (i.e., Step 7). This proof relies on two important properties of sequential execution. The first being that one cannot produce a random number in its final state without an encrypted credential containing that number in its initial state. The second being approximately the converse of the first, one cannot produce an encrypted credential containing a particular random number in its final state without that number or an encrypted credential containing that number in its initial state.

We reuse the intuition from the first scenario to show that the private IAK and private EK are loaded on the same TPM if the IAK is credentialed by the EK. The composition of these intermediate proofs leads to our goal shown in Figure 11. Due to the extra parameters and criteria of the `steps_CA` function, the

```
Theorem iak_and_ek_in_TPM :
  ∀ seq2 seq1 iniTPM ini midTPM mid finTPM fin csr ident ek iak cert g,
  seq_execute (iniTPM, ini) seq1 (midTPM, mid) →
  In csr mid →
  (∀ m', needsGeneratedTPM m' → ¬In m' iniTPM) →
  (∀ m', needsGenerated m' → ¬In m' ini) →
  steps_CA csr ident ek iak cert →
  seq_execute
   (midTPM, inferFrom (encryptedCredential (hash (publicKey iak)) g ek)++mid)
    seq2
   (finTPM, fin) →
  In (randomNum g) fin →
  In (privateKey (pubToPrivKey iak)) iniTPM ∧
  In (privateKey (pubToPrivKey ek)) iniTPM.
```

Fig. 11. Final Verification Goal for IAK Provisioning Procedure

proof statement shows that the inverse of the key contained in the EK certificate and the inverse of the key contained in the new certificate are loaded on the same TPM. This is precisely the assertion described by Assurance B.

In conclusion, we use the previously-certified EK to prove that the new IAK is loaded on the same TPM as itself. When issuing the IAK certificate, the CA uses the device identifying information from the recieved CSR. Assurance A is guaranteed by the attribute check performed by the CA, Assurance B is guaranteed by the EK's `Restricted` attribute and the secret contained in the challenge blob, and Assurance C is guaranteed by the signature check on the EK certificate performed by the CA.

Our analysis revealed a potential inconsistency in how the IAK provisioning procedure is described. The procedure provides no assurance regarding IAK device-residency as implied by its definition. The IAK certificate is intended to provide evidence that a key belongs to a specific device, yet the procedure makes no guarantees that the IAK is actually on the device. Proving an IAK belongs to a specific device requires first binding the IAK to a specific TPM using the EK and then binding the TPM to a specific device (Trusted Computing Group, 2021). However, the CA makes no attempt at performing the second part of this process. Instead the CA fully trusts the OEM to have provided the correct device identifying information.

Even the attestation variation of the IAK provisioning procedure does not attempt verification. This variation uses a quote over selected PCRs to show only that the device is running the appropriate, trusted firmware. All devices with the same model number have an identical golden hash value that represents its expected PCR digest. Although this check guarantees that the TPM containing the IAK is on a specific *type* of device it does not guarantee that the TPM is on a specific *device*.

A small change to the attestation variation of IAK provisioning procedure will overcome this shortcoming. The TPM should store a measurement that contains a device's unique identifier. In this way, each individual device has a unique golden hash value and its identity can be uniquely determined. Due to integrity guarantees of the TPM's PCR mechanism, this modification provides definitive evidence that a key belongs to a specific device. A weakness of this modification is it requires a large database of golden hash values. Maintaining such a database in dynamic environments with many devices would be challenging. however, the approach will be useful in critical systems where it is imperative to know the precise identity of a specific device.

7 Conclusions and Future Work

Contributions of this research include: (i) design and implementation of an abstract formal model of TPM command execution; (ii) an in-depth analysis of the TCG-provided IAK and LAK certification protocols, and (iii) a recommendation for clarifying the IAK provisioning procedure. To perform this verification we used two primary proof techniques. The first uses a minimal initial state to ensure key residency while the second uses a command supersequence to characterize arbitrary construction of a certificate request.

This work is motivated by the need for strong identifiers in performing layered remote attestation Coker et al. (2011); Ramsdell et al. (2019); Helble et al. (2021). It is essential that an attestation result be strongly bound to the attestation manager that produced it. LAKs are ideal for identifying attestation managers and providing a secure method for signing attestation results. Verification results reported here allow safe integration of LAKs into our verified attestation tool suite Petz and Alexander (2022).

Further improvements are necessary should we choose to extend our approach to a broader range of DevID protocols. An initial step would expand the command library to include additional TPM and TSS commands necessary for modeling the attestation variation of both provisioning protocols. The attestation variations use PCRs to inform a remote CA of the internal state of a certificate-requesting entity. This variation is strongly encouraged during IAK certification to assure the CA it is issuing a certificate to a device running trusted software. It also contributes to our overall attestation goal. To include necessary PCR-related commands, the structure of abstract state should be modified from a pair to a record similar to that of the work of Halling and Alexander (2013). These advancements to the model also provide the mechanisms necessary to verify our recommended changes to the IAK provisioning procedure.

All verification models and LaTeX source associated with this work may be found at git@github.com:ku-sldg/vstte23.git.

Bibliography

Arthur, W., Challener, D., Goldman, K.: A Practical Guide to TPM 2.0: Using the Trusted Platform Module in the New Age of Security. Apress (2015), https://link.springer.com/content/pdf/10.1007/978-1-4302-6584-9.pdf

Chen, L., Warinschi, B.: Security of the TCG Privacy-CA Solution. In: 2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing. pp. 609–616 (2010)

Coker, G., Guttman, J., Loscocco, P., Herzog, A., Millen, J., O'Hanlon, B., Ramsdell, J., Segall, A., Sheehy, J., Sniffen, B.: Principles of remote attestation. International Journal of Information Security 10(2), 63–81 (June 2011)

Delaune, S., Kremer, S., Ryan, M.D., Steel, G.: A Formal Analysis of Authentication in the TPM. In: Degano, P., Etalle, S., Guttman, J. (eds.) Formal Aspects of Security and Trust. pp. 111–125. Springer Berlin Heidelberg, Berlin, Heidelberg (2011a)

Delaune, S., Kremer, S., Ryan, M.D., Steel, G.: Formal Analysis of Protocols Based on TPM State Registers. In: 2011 IEEE 24th Computer Security Foundations Symposium. pp. 66–80 (2011b)

Halling, B.: Towards a Formal Verification of the Trusted Platform Module. Master's thesis, University of Kansas (2013)

Halling, B., Alexander, P.: Verifying a Privacy CA Remote Attestation Protocol. In: Brat, G., Rungta, N., Venet, A. (eds.) NASA Formal Methods. pp. 398–412. Springer Berlin Heidelberg, Berlin, Heidelberg (2013), http://cae.ittc.ku.edu/papers/nfm13.pdf

Helble, S.C., Kretz, I.D., Loscocco, P.A., Ramsdell, J.D., Rowe, P.D., Alexander, P.: Flexible mechanisms for remote attestation. ACM Transactions on Privacy and Security (TOPS) 24(4), 1–23 (2021)

Institute of Electrical and Electronics Engineers: IEEE Standard for Local and Metropolitan Area Networks - Secure Device Identity. IEEE Std 802.1AR-2018 (2018), https://ieeexplore.ieee.org/document/8423794

Kremer, S., Künnemann, R.: Automated analysis of security protocols with global state. Journal of Computer Security 24(5), 583–616 (2016)

Martin, A., et al.: The ten page introduction to trusted computing. Tech. Rep. CS-RR-08-11, Oxford University Computing Labratory, Oxford, UK (2008)

Meier, S., Schmidt, B., Cremers, C., Basin, D.: The tamarin prover for the symbolic analysis of security protocols. In: Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings 25. pp. 696–701. Springer (2013)

Milner, R.: Communicating and Mobile Systems: The π-calculus. Cambridge University Press (1999)

Petz, A., Alexander, P.: An infrastructure for faithful execution of remote attestation protocols. Innovations in Systems and Software Engineering (2022)

Ramsdell, J., Rowe, P.D., Alexander, P., Helble, S., Loscocco, P., Pendergrass, J.A., Petz, A.: Orchestrating layered attestations. In: Principles of Security and Trust (POST'19). Prague, Czech Republic (April 8-11 2019)

Shao, J., Qin, Y., Feng, D.: Formal Analysis of HMAC Authorisation in the TPM 2.0 Specification. IET Information Security 12(2), 133–140 (2018), https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-ifs.2016.0005

Shao, J., Qin, Y., Feng, D., Wang, W.: Formal Analysis of Enhanced Authorization in the TPM 2.0. In: Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security. pp. 273–284. ASIA CCS '15, Association for Computing Machinery, New York, NY, USA (2015), https://doi.org/10.1145/2714576.2714610

Trusted Computing Group: TPM 2.0 Keys for Device Identity and Attestation (2021), https://trustedcomputinggroup.org/wp-content/uploads/TPM-2p0-Keys-for-Device-Identity-and-Attestation_v1_r12_pub10082021.pdf

Wesemeyer, S., Newton, C.J., Treharne, H., Chen, L., Sasse, R., Whitefield, J.: Formal Analysis and Implementation of a TPM 2.0-Based Direct Anonymous Attestation Scheme. In: Proceedings of the 15th ACM Asia Conference on Computer and Communications Security. pp. 784–798. ASIA CCS '20, Association for Computing Machinery, New York, NY, USA (2020), https://doi.org/10.1145/3320269.3372197

Whitefield, J., Chen, L., Sasse, R., Schneider, S., Treharne, H., Wesemeyer, S.: A Symbolic Analysis of ECC-Based Direct Anonymous Attestation. In: 2019 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 127–141 (2019)