Formal analysis of protocols based on TPM state registers

Stéphanie Delaune[1], Steve Kremer[1], Mark D. Ryan[2], and Graham Steel[1]

[1] LSV, ENS Cachan & CNRS & INRIA Saclay Île-de-France, France
[2] School of Computer Science, University of Birmingham, UK

Thursday, June 9th, 2011

TPM - What is it?

Trusted Platform Module

Hardware chip designed to enable commodity computers to achieve greater levels of security than is possible in software alone.

TPM - What is it?

Trusted Platform Module

Hardware chip designed to enable commodity computers to achieve greater levels of security than is possible in software alone.

- more than 200 millions currently in existence (mostly in laptops)
  $\longrightarrow$ already used by some applications (*e.g.* Disk encryption)
- specified by the Trusted Computing Group
  $\longrightarrow$ more than 700 pages of specification

`http://www.trustedcomputinggroup.org`

TPM functionality

Secure storage:

- TPM stores keys and other sensitive data in its shielded memory
- A user can store content that is encrypted by keys only available to the TPM.

TPM functionality

Secure storage:

- TPM stores keys and other sensitive data in its shielded memory
- A user can store content that is encrypted by keys only available to the TPM.

Platform authentication:

- Each TPM chip has a unique and secret key
- A platform can obtain keys by which it can authenticate itself reliably.

TPM functionality

Secure storage:

- TPM stores keys and other sensitive data in its shielded memory
- A user can store content that is encrypted by keys only available to the TPM.

Platform authentication:

- Each TPM chip has a unique and secret key
- A platform can obtain keys by which it can authenticate itself reliably.

Platform measurement and reporting:

- TPM contains some internal memory slots called PCRs, and some keys can be locked to a particular PCR value
- PCR values can be modified using some specific command (*e.g.* command Extend).

TPM - How is it used?

Application programming interface:

- create new keys (*e.g.* CreateWrapKey), and load them into the device (*e.g.* LoadKey2);
- manipulate these keys, and the PCRs
  $\longrightarrow$ *e.g.* UnBind allows one to decrypt a ciphertext using a key that is stored into the TPM and locked to the current PCR value
  $\longrightarrow$ *e.g.* Quote allows one to obtain a certificate attesting that a key is locked to a particular PCR value
  $\longrightarrow$ *e.g.* Extend allows one to extend the current value of a PCR with some data x, *i.e.* $p := SHA1(p||x)$.

TPM - How is it used?

Application programming interface:

- create new keys (*e.g.* CreateWrapKey), and load them into the device (*e.g.* LoadKey2);
- manipulate these keys, and the PCRs
  $\longrightarrow$ *e.g.* UnBind allows one to decrypt a ciphertext using a key that is stored into the TPM and locked to the current PCR value
  $\longrightarrow$ *e.g.* Quote allows one to obtain a certificate attesting that a key is locked to a particular PCR value
  $\longrightarrow$ *e.g.* Extend allows one to extend the current value of a PCR with some data x, *i.e.* $p := SHA1(p||x)$.

The TPM provides a root of trust for a variety of protocols: *e.g.* Microsoft's hard drive encryption system "BitLocker", Direct Anonymous Attestation protocol, . . .

Related Work

Several attempts to formally analyse the TPM itself

- using a theorem prover [Lin, 2005];
- using ProVerif [Delaune *et al.*, 2010]; or
- in some specific models (with no tool support), *e.g.* [Gürgens *et al.*, 2007, Coker *et al.*, 2010]

Related Work

Several attempts to formally analyse the TPM itself

- using a theorem prover [Lin, 2005];
- using ProVerif [Delaune *et al.*, 2010]; or
- in some specific models (with no tool support), *e.g.* [Gürgens *et al.*, 2007, Coker *et al.*, 2010]

$\longrightarrow$ These results do *not* consider TPM state registers.

Related Work

Several attempts to formally analyse the TPM itself

- using a theorem prover [Lin, 2005];
- using ProVerif [Delaune *et al.*, 2010]; or
- in some specific models (with no tool support), *e.g.* [Gürgens *et al.*, 2007, Coker *et al.*, 2010]

$\longrightarrow$ These results do *not* consider TPM state registers.

Modelling state is challenging [Herzog, 2006]

- extension of the strand space model to analyse optimistic fair exchange protocol [Guttman, 2011]
- extension of ProVerif to take global state into account [Modersheim, 2010, Arapinis *et al.*, 2011]

Related Work

Several attempts to formally analyse the TPM itself

- using a theorem prover [Lin, 2005];
- using ProVerif [Delaune *et al.*, 2010]; or
- in some specific models (with no tool support), *e.g.* [Gürgens *et al.*, 2007, Coker *et al.*, 2010]

$\longrightarrow$ These results do *not* consider TPM state registers.

Modelling state is challenging [Herzog, 2006]

- extension of the strand space model to analyse optimistic fair exchange protocol [Guttman, 2011]
- extension of ProVerif to take global state into account [Modersheim, 2010, Arapinis *et al.*, 2011]

$\longrightarrow$ These results are *not* suitable to analyse protocols based on TPM state registers.

Our contributions

Formal analysis of protocols based on TPM registers using an automatic tool

Our contributions

Formal analysis of protocols based on TPM registers using an automatic tool

Our approach:

- we use Horn clauses and rely on the ProVerif tool;
- we solve non-termination issues by providing a transformation that is sound and complete for the class of k-stable clauses; and
- we provide a syntactic criterion to conclude to k-stability.

Our contributions

Formal analysis of protocols based on TPM registers using an automatic tool

Our approach:

- we use Horn clauses and rely on the ProVerif tool;
- we solve non-termination issues by providing a transformation that is sound and complete for the class of k-stable clauses; and
- we provide a syntactic criterion to conclude to k-stability.

Some case studies:

- a simplified version of the Micosoft BitLocker protocol
- a secure envelope protocol [Ables & Ryan, 2010]

$\longrightarrow$ both protocols crucially rely on the use of PCR

Outline

Outline

TPM key hierarchy

Cryptographic key

Keys are arranged in a tree structure and stored in the TPM memory
$\longrightarrow$ Storage Root Key created by a special command

Authdata, PCR

In particular, to each TPM key is associated an authdata value and also some PCR values

- authdata is a password shared between the user process and the TPM
- PCR values constrain the state of the TPM. The TPM will use the key only if certain PCRs currently have certain values.

CertifyKey command

Goal: allow a user to obtain a certificate on a key that is stored in the device.

CertifyKey command

Goal: allow a user to obtain a certificate on a key that is stored in the device.

Description:

USER

TPM

current pcr value: pcr

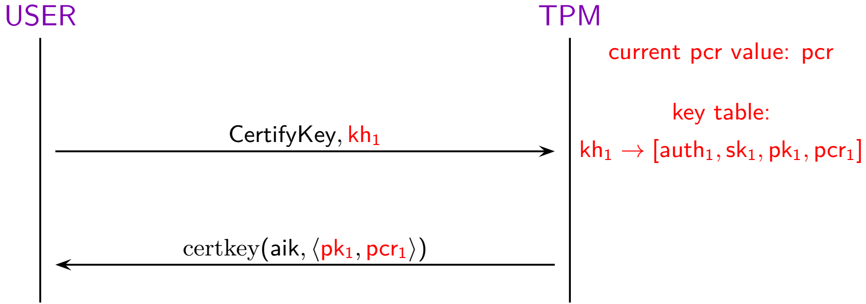
key table:

$\mathsf{kh}_1 \rightarrow [\mathsf{auth}_1, \mathsf{sk}_1, \mathsf{pk}_1, \mathsf{pcr}_1]$

CertifyKey command

Goal: allow a user to obtain a certificate on a key that is stored in the device.

Description:



UnBind command

Goal: allow a user to retrieve the content of an encryption provided that the decryption key is stored in the key table of the TPM.

UnBind command

Goal: allow a user to retrieve the content of an encryption provided that the decryption key is stored in the key table of the TPM.

Description:

USER

TPM

current pcr value: $\mathbf{pcr_1}$

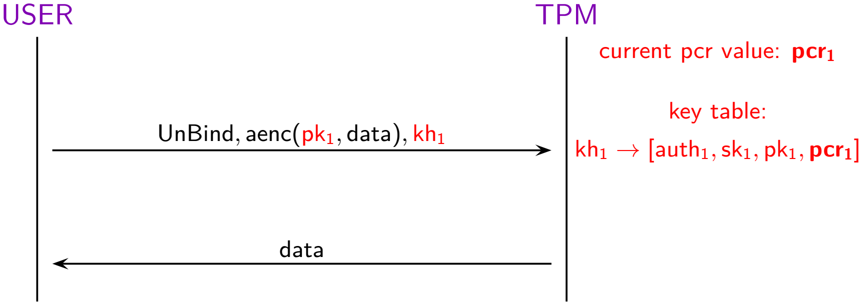
key table:

$\mathsf{kh}_1 \rightarrow [\mathsf{auth}_1, \mathsf{sk}_1, \mathsf{pk}_1, \mathbf{pcr_1}]$

UnBind command

Goal: allow a user to retrieve the content of an encryption provided that the decryption key is stored in the key table of the TPM.

Description:



Extend command

Goal: allow a user to update the value stored in one of the platform configuration register (PCR).

Extend command

Goal: allow a user to update the value stored in one of the platform configuration register (PCR).

Description:

USER

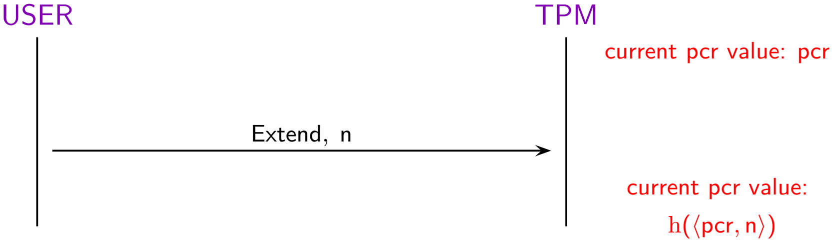
TPM

current pcr value: pcr

Extend command

Goal: allow a user to update the value stored in one of the platform configuration register (PCR).

Description:



Outline

An introductory example

Goal: Alice has two secrets s_1 and s_2. First, she interacts with Bob, and then Bob can learn one of the secrets (he chooses) but not both.

An introductory example

Goal: Alice has two secrets s_1 and s_2. First, she interacts with Bob, and then Bob can learn one of the secrets (he chooses) but not both.

Description:

1. create and load a key pair $(k_1, \mathsf{pk}(k_1))$ locked to $\mathrm{h}(u_0, a_1)$ in Bob's TPM;
2. create and load a key pair $(k_2, \mathsf{pk}(k_2))$ locked to $\mathrm{h}(u_0, a_2)$ in Bob's TPM;

$\longrightarrow$ For sake of simplicity, we assume that the keys are already in Bob's TPM.

An introductory example

Goal: Alice has two secrets s_1 and s_2. First, she interacts with Bob, and then Bob can learn one of the secrets (he chooses) but not both.

Description:

1. create and load a key pair $(k_1, \mathrm{pk}(k_1))$ locked to $\mathrm{h}(u_0, a_1)$ in Bob's TPM;
2. create and load a key pair $(k_2, \mathrm{pk}(k_2))$ locked to $\mathrm{h}(u_0, a_2)$ in Bob's TPM;

   $\longrightarrow$ For sake of simplicity, we assume that the keys are already in Bob's TPM.
3. Bob provides some certificates to Alice (using CertifyKey);

An introductory example

Goal: Alice has two secrets s_1 and s_2. First, she interacts with Bob, and then Bob can learn one of the secrets (he chooses) but not both.

Description:

1. create and load a key pair $(k_1, \mathsf{pk}(k_1))$ locked to $\mathrm{h}(u_0, a_1)$ in Bob's TPM;
2. create and load a key pair $(k_2, \mathsf{pk}(k_2))$ locked to $\mathrm{h}(u_0, a_2)$ in Bob's TPM;

   $\longrightarrow$ For sake of simplicity, we assume that the keys are already in Bob's TPM.

3. Bob provides some certificates to Alice (using CertifyKey);
4. Alice sends $\mathsf{aenc}(\mathsf{pk}(k_1), s_1)$ and $\mathsf{aenc}(\mathsf{pk}(k_2), s_2)$ to Bob;

An introductory example

Goal: Alice has two secrets s_1 and s_2. First, she interacts with Bob, and then Bob can learn one of the secrets (he chooses) but not both.

Description:

1. create and load a key pair $(k_1, \mathsf{pk}(k_1))$ locked to $\mathrm{h}(u_0, a_1)$ in Bob's TPM;
2. create and load a key pair $(k_2, \mathsf{pk}(k_2))$ locked to $\mathrm{h}(u_0, a_2)$ in Bob's TPM;

   $\longrightarrow$ For sake of simplicity, we assume that the keys are already in Bob's TPM.

3. Bob provides some certificates to Alice (using CertifyKey);
4. Alice sends $\mathsf{aenc}(\mathsf{pk}(k_1), s_1)$ and $\mathsf{aenc}(\mathsf{pk}(k_2), s_2)$ to Bob;
5. Using Extend and UnBind, Bob can obtain either s_1 or s_2, but not both.

Modelling the attacker

Predicate att

$\mathsf{att}(u, v)$ means that there is a reachable state in which the PCR has value u and the attacker knows v.

Some rules:

$$\begin{array}{rcl}
\mathsf{att}(x_p, x) & \rightarrow & \mathsf{att}(x_p, \mathsf{pk}(x)) \\
\mathsf{att}(x_p, x) \wedge \mathsf{att}(x_p, y) & \rightarrow & \mathsf{att}(x_p, \mathsf{aenc}(x, y)) \\
\mathsf{att}(x_p, \mathsf{aenc}(\mathsf{pk}(x), y)) \wedge \mathsf{att}(x_p, x) & \rightarrow & \mathsf{att}(x_p, y)
\end{array}$$

Initial knowledge:

$$\mathsf{att}(\mathsf{u}_0, \mathsf{a}_1)$$

$$\mathsf{att}(\mathsf{u}_0, \mathsf{a}_2)$$

Modelling the key table

Predicate key

key(u, sk, pk, v) means that there is a reachable state in which the PCR has value u, and the key table has an entry for the key pair (sk, pk) locked to the PCR value v.

Modelling the key table

Predicate key

$\mathrm{key}(u, sk, pk, v)$ means that there is a reachable state in which the PCR has value u, and the key table has an entry for the key pair (sk, pk) locked to the PCR value v.

Some initial facts:

$$\mathrm{key}(\mathrm{u}_0, \mathrm{k}_1, \mathrm{pk}(\mathrm{k}_1), \mathrm{h}(\mathrm{u}_0, \mathrm{a}_1))$$

$$\mathrm{key}(\mathrm{u}_0, \mathrm{k}_2, \mathrm{pk}(\mathrm{k}_2), \mathrm{h}(\mathrm{u}_0, \mathrm{a}_2))$$

Modelling the key table

Predicate key

$\mathsf{key}(u, sk, pk, v)$ means that there is a reachable state in which the PCR has value u, and the key table has an entry for the key pair (sk, pk) locked to the PCR value v.

Some initial facts:

$$\mathsf{key}(\mathsf{u}_0, \mathsf{k}_1, \mathsf{pk}(\mathsf{k}_1), \mathrm{h}(\mathsf{u}_0, \mathsf{a}_1))$$

$$\mathsf{key}(\mathsf{u}_0, \mathsf{k}_2, \mathsf{pk}(\mathsf{k}_2), \mathrm{h}(\mathsf{u}_0, \mathsf{a}_2))$$

Remarks:

- we do not allow keys to be deleted from the memory of the TPM;
  $\longrightarrow$ we allow an unbounded number of keys to be loaded
- the attacker is not allowed to modify the key table (only through the API).

Modelling the TPM commands (1/2)

CertifyKey

$$\mathsf{key}(x_p, x_{sk}, x_{pk}, x_{pcr}) \rightarrow \mathsf{att}(x_p, \mathrm{certkey}(\mathsf{aik}, \langle x_{pk}, x_{pcr}\rangle))$$

UnBind

$$\mathsf{att}(x_p, \mathsf{aenc}(x_{pk}, x_{data})) \wedge \mathsf{key}(x_p, x_{sk}, x_{pk}, x_p) \rightarrow \mathsf{att}(x_p, x_{data})$$

Modelling the TPM commands (2/2)

The TPM rule for extending and rebooting the PCR is treated in a particular way. We have a dedicated set of inheritance rules.

Modelling the TPM commands (2/2)

The TPM rule for extending and rebooting the PCR is treated in a particular way. We have a dedicated set of inheritance rules.

Key table:

$$\mathsf{key}(x_p, x_{sk}, x_{pk}, x_{pcr}) \wedge \mathsf{att}(x_p, x_v) \rightarrow \mathsf{key}(\mathrm{h}(x_p, x_v), x_{sk}, x_{pk}, x_{pcr})$$

$$\mathsf{key}(x_p, x_{sk}, x_{pk}, x_{pcr}) \rightarrow \mathsf{key}(\mathrm{u}_0, x_{sk}, x_{pk}, x_{pcr}) \quad \text{(optional)}$$

Modelling the TPM commands (2/2)

The TPM rule for extending and rebooting the PCR is treated in a particular way. We have a dedicated set of inheritance rules.

Key table:

$$\mathsf{key}(x_p, x_{sk}, x_{pk}, x_{pcr}) \wedge \mathsf{att}(x_p, x_v) \rightarrow \mathsf{key}(\mathrm{h}(x_p, x_v), x_{sk}, x_{pk}, x_{pcr})$$

$$\mathsf{key}(x_p, x_{sk}, x_{pk}, x_{pcr}) \rightarrow \mathsf{key}(\mathsf{u}_0, x_{sk}, x_{pk}, x_{pcr}) \quad \text{(optional)}$$

Knowledge of the attacker:

$$\mathsf{att}(x_p, x_v) \wedge \mathsf{att}(x_p, x) \rightarrow \mathsf{att}(\mathrm{h}(x_p, x_v), x)$$

$$\mathsf{att}(x_p, x) \rightarrow \mathsf{att}(\mathsf{u}_0, x)$$

Modelling the protocol

Protocol rules:

Considering our introductory example, the role of Alice can be described by the following two rules:

$$\mathsf{att}(x_p, \mathrm{certkey}(\mathsf{aik}, \langle x_{pk}, \mathrm{h}(\mathsf{u}_0, \mathsf{a}_1)\rangle)) \rightarrow \mathsf{att}(x_p, \mathsf{aenc}(x_{pk}, \mathsf{s}_1))$$

$$\mathsf{att}(x_p, \mathrm{certkey}(\mathsf{aik}, \langle x_{pk}, \mathrm{h}(\mathsf{u}_0, \mathsf{a}_2)\rangle)) \rightarrow \mathsf{att}(x_p, \mathsf{aenc}(x_{pk}, \mathsf{s}_2))$$

Modelling the protocol

Protocol rules:

Considering our introductory example, the role of Alice can be described by the following two rules:

$$\mathsf{att}(x_p, \mathrm{certkey}(\mathsf{aik}, \langle x_{pk}, \mathrm{h}(\mathsf{u}_0, \mathsf{a}_1)\rangle)) \rightarrow \mathsf{att}(x_p, \mathsf{aenc}(x_{pk}, \mathsf{s}_1))$$

$$\mathsf{att}(x_p, \mathrm{certkey}(\mathsf{aik}, \langle x_{pk}, \mathrm{h}(\mathsf{u}_0, \mathsf{a}_2)\rangle)) \rightarrow \mathsf{att}(x_p, \mathsf{aenc}(x_{pk}, \mathsf{s}_2))$$

Query

Is Bob able to learn both secrets?

$$Q = \{\mathsf{att}(x, \mathsf{s}_1),\ \mathsf{att}(x, \mathsf{s}_2)\}$$

Outline

The ProVerif tool (B. Blanchet)

Available on line:

`http://www.proverif.ens.fr/`

Input: protocols written in Horn clauses

Characteristics

- unbounded number of sessions
- primitives given by an equational theory
- security properties: (strong) secrecy, correspondence properties, equivalence properties
- sound but not complete, termination is not guaranteed
  $\longrightarrow$ the tool works well in practice

Termination problem

The termination problem seems due to the way PCR is modeled:

$$\mathsf{att}(x_p, x_v) \wedge \mathsf{att}(x_p, x) \rightarrow \mathsf{att}(\mathrm{h}(x_p, x_v), x)$$

$$\mathsf{key}(x_p, x_{sk}, x_{pk}, x_{pcr}) \wedge \mathsf{att}(x_p, x_v) \rightarrow \mathsf{key}(\mathrm{h}(x_p, x_v), x_{sk}, x_{pk}, x_{pcr})$$

Termination problem

The termination problem seems due to the way PCR is modeled:

$$\mathsf{att}(x_p, x_v) \wedge \mathsf{att}(x_p, x) \rightarrow \mathsf{att}(\mathrm{h}(x_p, x_v), x)$$

$$\mathsf{key}(x_p, x_{sk}, x_{pk}, x_{pcr}) \wedge \mathsf{att}(x_p, x_v) \rightarrow \mathsf{key}(\mathrm{h}(x_p, x_v), x_{sk}, x_{pk}, x_{pcr})$$

Main idea

1. Could we bound the length of the PCR, *i.e. the number of times a PCR may be extended between two resets*?
2. If the answer is 'yes', can we compute such a bound?

Notion of k-stability

Definition k-stable

A rule R is *k-stable* if for any substitution θ grounding for R, for any PCR value $u = \mathrm{h}(u_1, u_2)$ such that $\mathsf{length}_{\mathsf{pcr}}(u) > k$ we have that:

- either $(\mathsf{R}\theta)[\mathrm{h}(u_1, u_2) \rightarrow u_1] = \mathsf{R}(\theta[\mathrm{h}(u_1, u_2) \rightarrow u_1])$,
- or $(\mathsf{R}\theta)[\mathrm{h}(u_1, u_2) \rightarrow u_1]$ is a tautology.

Notion of k-stability

Definition k-stable

A rule R is *k-stable* if for any substitution θ grounding for R, for any PCR value $u = \mathrm{h}(u_1, u_2)$ such that $\mathsf{length}_{\mathsf{pcr}}(u) > k$ we have that:

- either $(\mathsf{R}\theta)[\mathrm{h}(u_1, u_2) \to u_1] = \mathsf{R}(\theta[\mathrm{h}(u_1, u_2) \to u_1])$,
- or $(\mathsf{R}\theta)[\mathrm{h}(u_1, u_2) \to u_1]$ is a tautology.

Examples

- $\mathsf{att}(x_p, \mathrm{certkey}(\mathsf{aik}, \langle x_{pk}, \mathrm{h}(\mathsf{u}_0, \mathsf{a}_1)\rangle)) \to \mathsf{att}(x_p, \mathsf{aenc}(x_{pk}, \mathsf{s}_1))$
- $\mathsf{att}(x_p, x_v) \wedge \mathsf{att}(x_p, x) \to \mathsf{att}(\mathrm{h}(x_p, x_v), x)$

Notion of k-stability

Definition k-stable

A rule R is *k-stable* if for any substitution θ grounding for R, for any PCR value $u = \mathrm{h}(u_1, u_2)$ such that $\text{length}_{\text{pcr}}(u) > k$ we have that:

- either $(\mathrm{R}\theta)[\mathrm{h}(u_1, u_2) \to u_1] = \mathrm{R}(\theta[\mathrm{h}(u_1, u_2) \to u_1])$,
- or $(\mathrm{R}\theta)[\mathrm{h}(u_1, u_2) \to u_1]$ is a tautology.

Examples

- $\mathrm{att}(x_p, \mathrm{certkey}(\mathrm{aik}, \langle x_{pk}, \mathrm{h}(\mathrm{u}_0, \mathrm{a}_1)\rangle)) \to \mathrm{att}(x_p, \mathrm{aenc}(x_{pk}, \mathrm{s}_1))$
- $\mathrm{att}(x_p, x_v) \wedge \mathrm{att}(x_p, x) \to \mathrm{att}(\mathrm{h}(x_p, x_v), x)$

Proposition

Let $\mathcal{R}$ be a finite set of rules and Q be a query such that $\mathcal{R}$ and Q are k-stable. If Q is satisfiable then there exists a k-bounded derivation witnessing this fact.

Syntactic criterion to check k-stability

Lemma

Let $k \geq 0$ be an integer and $\mathrm{R} = H \rightarrow C$ be a rule such that:

1. for all $\mathrm{h}(v_1, v_2) \in st(\mathrm{R})$, $\mathrm{length}_{\mathrm{pcr}}(v_1, v_2) \leq k$;
2. for all $\mathrm{h}(v_1, v_2) \in st(H)$, we have that $v_1 \notin \mathcal{X}$;
3. for all $\mathrm{h}(v_1, v_2) \in st(C)$ such that $v_1 \in \mathcal{X}$, we have that $C[\mathrm{h}(v_1, v_2) \rightarrow v_1] \in H$.

Then, we have that the rule R is k-stable.

Syntactic criterion to check k-stability

Lemma

Let $k \geq 0$ be an integer and $\mathsf{R} = H \rightarrow C$ be a rule such that:

1. for all $\mathrm{h}(v_1, v_2) \in st(\mathsf{R})$, $\mathsf{length}_{\mathsf{pcr}}(v_1, v_2) \leq k$;
2. for all $\mathrm{h}(v_1, v_2) \in st(H)$, we have that $v_1 \notin \mathcal{X}$;
3. for all $\mathrm{h}(v_1, v_2) \in st(C)$ such that $v_1 \in \mathcal{X}$, we have that $C[\mathrm{h}(v_1, v_2) \rightarrow v_1] \in H$.

Then, we have that the rule R is k-stable.

Examples

- $\mathsf{att}(x_p, \mathrm{certkey}(\mathsf{aik}, \langle x_{pk}, \mathrm{h}(\mathsf{u}_0, \mathsf{a}_1)\rangle)) \rightarrow \mathsf{att}(x_p, \mathsf{aenc}(x_{pk}, \mathsf{s}_1))$
- $\mathsf{att}(x_p, x_v) \wedge \mathsf{att}(x_p, x) \rightarrow \mathsf{att}(\mathrm{h}(x_p, x_v), x)$

Syntactic criterion to check k-stability

Lemma

Let $k \geq 0$ be an integer and $\mathsf{R} = H \to C$ be a rule such that:

1. for all $\mathrm{h}(v_1, v_2) \in st(\mathsf{R})$, $\mathsf{length}_{\mathsf{pcr}}(v_1, v_2) \leq k$;
2. for all $\mathrm{h}(v_1, v_2) \in st(H)$, we have that $v_1 \notin \mathcal{X}$;
3. for all $\mathrm{h}(v_1, v_2) \in st(C)$ such that $v_1 \in \mathcal{X}$, we have that $C[\mathrm{h}(v_1, v_2) \to v_1] \in H$.

Then, we have that the rule R is k-stable.

Examples

- $\mathsf{att}(x_p, \mathrm{certkey}(\mathsf{aik}, \langle x_{pk}, \mathrm{h}(\mathsf{u}_0, \mathsf{a}_1)\rangle)) \to \mathsf{att}(x_p, \mathsf{aenc}(x_{pk}, \mathsf{s}_1))$
- $\mathsf{att}(x_p, x_v) \wedge \mathsf{att}(x_p, x) \to \mathsf{att}(\mathrm{h}(x_p, x_v), x)$

$\longrightarrow$ Going back to our running example, it is sufficient to consider 1-bounded derivation when checking satisfiability of a query.

Our transformation

Goal: A set of k-stable rules can be transformed into another "equivalent" set of rules that is more suitable for analysis with ProVerif.

Our transformation

Goal: A set of *k*-stable rules can be transformed into another "equivalent" set of rules that is more suitable for analysis with ProVerif.

Transformation: we replace each rule R by the set of rules:

$$\{\mathsf{R}[x \mapsto u] \mid x \in \mathcal{X}, p(x, t_1, \ldots, t_\ell) \in \mathsf{R},\ u \in U_k\}$$

$$\text{where } U_k = \{ \begin{array}{l} \mathsf{u}_0, \\ \mathrm{h}(\mathsf{u}_0, x_1), \\ \ldots, \\ \mathrm{h}(\ldots\mathrm{h}(\mathsf{u}_0, x_1), \ldots, x_k)\}. \end{array}$$

Our transformation

Goal: A set of *k*-stable rules can be transformed into another "equivalent" set of rules that is more suitable for analysis with ProVerif.

Transformation: we replace each rule R by the set of rules:

$$\{R[x \mapsto u] \mid x \in \mathcal{X}, p(x, t_1, \ldots, t_\ell) \in R,\ u \in U_k\}$$

$$\text{where } U_k = \{ \begin{array}{l} \mathrm{u}_0, \\ \mathrm{h}(\mathrm{u}_0, x_1), \\ \ldots, \\ \mathrm{h}(\ldots\mathrm{h}(\mathrm{u}_0, x_1), \ldots, x_k)\}. \end{array}$$

This transformation effectively bounds the PCR length of possible PCR values that may appear as the first argument of a predicate.

Theorem

If the initial set of rules is *k*-stable, then the initial and transformed set of rules are equivalent w.r.t. satisfiability of queries.

Outline

1. Overview of the TPM
2. Modelling using Horn clauses
3. Analysing with ProVerif
4. Case studies

TPM commands

TPM's commands – We consider the following commands.

- Read
- Quote
- CreateWrapKey
- LoadKey2
- CertifyKey
- UnBind
- Seal
- UnSeal

TPM commands

TPM's commands – We consider the following commands.

- Read
- Quote
- CreateWrapKey
- LoadKey2
- CertifyKey
- UnBind
- Seal
- UnSeal

Simplifications and/or abstractions

TPM commands

TPM's commands – We consider the following commands.

- Read
- Quote
- CreateWrapKey
- LoadKey2
- CertifyKey
- UnBind
- Seal
- UnSeal

Simplifications and/or abstractions

1. we do not consider authdata;
   $\longrightarrow$ this is equivalent to giving all the authdata to the attacker

TPM commands

TPM's commands – We consider the following commands.

- Read
- Quote
- CreateWrapKey
- LoadKey2
- CertifyKey
- UnBind
- Seal
- UnSeal

Simplifications and/or abstractions

1. we do not consider authdata;
   $\longrightarrow$ this is equivalent to giving all the authdata to the attacker
2. the key AIK (attestation identity key) is initially and permanently loaded in the TPM;
   $\longrightarrow$ In reality, we have to create it (MakeIdentity) and to load it (ActivateIdentity)

TPM commands

TPM's commands – We consider the following commands.

- Read
- Quote
- CreateWrapKey
- LoadKey2
- CertifyKey
- UnBind
- Seal
- UnSeal

Simplifications and/or abstractions

1. we do not consider authdata;
   $\longrightarrow$ this is equivalent to giving all the authdata to the attacker
2. the key AIK (attestation identity key) is initially and permanently loaded in the TPM;
   $\longrightarrow$ In reality, we have to create it (MakeIdentity) and to load it (ActivateIdentity)
3. we only consider one PCR, instead of 24.

A simplified version of the Bitlocker protocol (1/2)

Goal: protect the data that are stored on your disk.
$\longrightarrow$ your data are encrypted using VEK, which is in turn encrypted with VMK.

A simplified version of the Bitlocker protocol (1/2)

Goal: protect the data that are stored on your disk.
$\longrightarrow$ your data are encrypted using VEK, which is in turn encrypted with VMK.

Description of the set-up phase:

- A new key pair (sk,pk) is generated and loaded in Alice's TPM
  $\longrightarrow$ using CreateWrapKey and LoadKey2;
- VMK is encrypted under the key pk locked to $\mathrm{h}(\mathrm{h}(\mathsf{u_0}, \mathsf{bios}), \mathsf{loader})$
  $\longrightarrow$ using Seal

$$\mathsf{seal}(\mathsf{pk}, \mathsf{vmk}, \mathsf{tpmproof}, \mathrm{h}(\mathrm{h}(\mathsf{u_0}, \mathsf{bios}), \mathsf{loader}))$$

A simplified version of the Bitlocker protocol (1/2)

Goal: protect the data that are stored on your disk.
$\longrightarrow$ your data are encrypted using VEK, which is in turn encrypted with VMK.

Description of the set-up phase:

- A new key pair (sk,pk) is generated and loaded in Alice's TPM
  $\longrightarrow$ using CreateWrapKey and LoadKey2;
- VMK is encrypted under the key pk locked to $\mathrm{h}(\mathrm{h}(\mathsf{u_0}, \mathsf{bios}), \mathsf{loader})$
  $\longrightarrow$ using Seal

$$\mathsf{seal}(\mathsf{pk}, \mathsf{vmk}, \mathsf{tpmproof}, \mathrm{h}(\mathrm{h}(\mathsf{u_0}, \mathsf{bios}), \mathsf{loader}))$$

Description of the retrieval phase:

- a trust chain is built: Pre-BIOS $\rightarrow$ BIOS $\rightarrow$ loader
- retrieve VMK using Unseal
- prevent unauthorised retrievals, by extending "deny" into the PCR

Modelling - Bitlocker protocol (2/2)

Alice's role setting up the drive encryption in a trusted state:

$$\mathrm{key}(x_p, x_{sk}, \mathrm{pk}(x_{sk}), \mathrm{nil}) \rightarrow \mathrm{att}(x_p, \mathrm{seal}(\mathrm{pk}(x_{sk}), \mathrm{vmk}[x_p], \mathrm{tpmproof}, \mathrm{h}(\mathrm{h}(\mathrm{u}_0, \mathrm{bios}), \mathrm{loader})))$$

PCR reboot rules:

$$\begin{array}{l} \mathrm{att}(x_p, x) \rightarrow \mathrm{att}(\mathrm{h}(\mathrm{h}(\mathrm{h}(\mathrm{u}_0, \mathrm{bios}), \mathrm{loader}), \mathrm{deny}), x) \\ \mathrm{att}(x_p, x) \rightarrow \mathrm{att}(\mathrm{h}(\mathrm{h}(\mathrm{u}_0, \mathrm{bios}), \mathrm{loader\_rogue}), x) \\ \mathrm{att}(x_p, x) \rightarrow \mathrm{att}(\mathrm{h}(\mathrm{u}_0, \mathrm{bios\_rogue}), x) \end{array}$$

Results of our analysis: $\mathrm{att}(x_p, \mathrm{vmk}[x])$

- the rules are 3-stable
- ProVerif quickly concludes that the protocol is safe (using the set of rules obtained by applying our transformation).

Envelope protocol (1/3) [Ables & Ryan, 10]

Goal: provide some data (secret) to Bob in such a way that Bob can either access the data or revoke his right to access the data.
$\longrightarrow$ Now, we consider the fact that the TPM can be rebooted.

Description

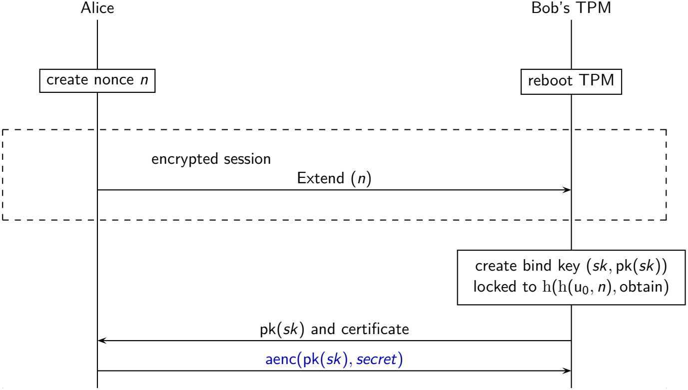
1. Sealing the envelope:

Goal: provide some data (secret) to Bob in such a way that Bob can either access the data or revoke his right to access the data.

$\longrightarrow$ Now, we consider the fact that the TPM can be rebooted.

Description

1. Sealing the envelope:



Envelope protocol (1/3) [Ables & Ryan, 10]

Goal: provide some data (secret) to Bob in such a way that Bob can either access the data or revoke his right to access the data.

$\longrightarrow$ Now, we consider the fact that the TPM can be rebooted.

Description

1. Sealing the envelope:
2. Opening the envelope:
   $\longrightarrow$ use Extend to extend obtain into the PCR,
   $\longrightarrow$ use UnBind to decrypt the ciphertext $\mathsf{aenc}(\mathsf{pk}(sk), secret)$;

Envelope protocol (1/3) [Ables & Ryan, 10]

Goal: provide some data (secret) to Bob in such a way that Bob can either access the data or revoke his right to access the data.
$\longrightarrow$ Now, we consider the fact that the TPM can be rebooted.

Description

1. Sealing the envelope:
2. Opening the envelope:
   $\longrightarrow$ use Extend to extend obtain into the PCR,
   $\longrightarrow$ use UnBind to decrypt the ciphertext $\mathsf{aenc}(\mathsf{pk}(sk), secret)$;
3. Returning the envelope:
   $\longrightarrow$ use Extend to extend deny into the PCR,
   $\longrightarrow$ use Quote to obtain a signature attesting that the current value of the PCR is $\mathrm{h}(\mathrm{h}(\mathsf{u}_0, \mathsf{n}), \mathsf{deny})$. This certificate can be used as a proof that Bob will never have access to secret.

Envelope protocol (2/3)

Alice's role

$$\mathsf{att}(x_p, x) \rightarrow \mathsf{att}(\mathrm{h}(x_p, \mathsf{n}[x_p]), x)$$

$$\mathsf{key}(x_p, x_{sk}, x_{pk}, x_{pcr}) \rightarrow \mathsf{key}(\mathrm{h}(x_p, n[x_p]), x_{sk}, x_{pk}, x_{pcr})$$

$$\mathsf{att}(x_p, \mathrm{certkey}(\mathsf{aik}, \mathsf{pk}(\mathsf{sk}), \mathrm{h}(\mathrm{h}(\mathsf{u}_0, \mathsf{n}[y]), \mathsf{obtain}))) \\ \rightarrow \mathsf{att}(x_p, \mathsf{aenc}(\mathsf{pk}(\mathsf{sk}), \mathsf{secret}[y]))$$

Envelope protocol (2/3)

Alice's role

$$\mathsf{att}(x_p, x) \rightarrow \mathsf{att}(\mathrm{h}(x_p, \mathsf{n}[x_p]), x)$$

$$\mathsf{key}(x_p, x_{sk}, x_{pk}, x_{pcr}) \rightarrow \mathsf{key}(\mathrm{h}(x_p, n[x_p]), x_{sk}, x_{pk}, x_{pcr})$$

$$\mathsf{att}(x_p, \mathrm{certkey}(\mathsf{aik}, \mathsf{pk}(\mathsf{sk}), \mathrm{h}(\mathrm{h}(\mathsf{u}_0, \mathsf{n}[y]), \mathsf{obtain}))) \rightarrow \mathsf{att}(x_p, \mathsf{aenc}(\mathsf{pk}(\mathsf{sk}), \mathsf{secret}[y]))$$

Query

- $\mathsf{att}(x_p, \mathsf{secret}[y])$, and
- $\mathsf{att}(x_p, \mathrm{certpcr}(\mathsf{aik}, \mathrm{h}(\mathrm{h}(\mathsf{u}_0, \mathsf{n}[y]), \mathsf{deny}), x))$.

Envelope protocol (2/3)

Alice's role

$$\mathsf{att}(x_p, x) \rightarrow \mathsf{att}(\mathrm{h}(x_p, \mathsf{n}[x_p]), x)$$

$$\mathsf{key}(x_p, x_{sk}, x_{pk}, x_{pcr}) \rightarrow \mathsf{key}(\mathrm{h}(x_p, n[x_p]), x_{sk}, x_{pk}, x_{pcr})$$

$$\mathsf{att}(x_p, \mathrm{certkey}(\mathsf{aik}, \mathsf{pk}(\mathsf{sk}), \mathrm{h}(\mathrm{h}(\mathsf{u}_0, \mathsf{n}[y]), \mathsf{obtain}))) \\ \rightarrow \mathsf{att}(x_p, \mathsf{aenc}(\mathsf{pk}(\mathsf{sk}), \mathsf{secret}[y]))$$

Query

- $\mathsf{att}(x_p, \mathsf{secret}[y])$, and
- $\mathsf{att}(x_p, \mathrm{certpcr}(\mathsf{aik}, \mathrm{h}(\mathrm{h}(\mathsf{u}_0, \mathsf{n}[y]), \mathsf{deny}), x))$.

All the rules are 2-stable and ProVerif terminates on the set of rules obtained after applying our transformation.

Envelope protocol (2/3)

Alice's role

$\mathsf{att}(x_p, x) \rightarrow \mathsf{att}(\mathrm{h}(x_p, \mathsf{n}[x_p]), x)$

$\mathsf{key}(x_p, x_{sk}, x_{pk}, x_{pcr}) \rightarrow \mathsf{key}(\mathrm{h}(x_p, n[x_p]), x_{sk}, x_{pk}, x_{pcr})$

$$\mathsf{att}(x_p, \mathrm{certkey}(\mathsf{aik}, \mathsf{pk}(\mathsf{sk}), \mathrm{h}(\mathrm{h}(\mathsf{u}_0, \mathsf{n}[y]), \mathsf{obtain}))) \\ \rightarrow \mathsf{att}(x_p, \mathsf{aenc}(\mathsf{pk}(\mathsf{sk}), \mathsf{secret}[y]))$$

Query

- $\mathsf{att}(x_p, \mathsf{secret}[y])$, and
- $\mathsf{att}(x_p, \mathrm{certpcr}(\mathsf{aik}, \mathrm{h}(\mathrm{h}(\mathsf{u}_0, \mathsf{n}[y]), \mathsf{deny}), x))$.

All the rules are 2-stable and ProVerif terminates on the set of rules obtained after applying our transformation.
$\longrightarrow$ false attack due to the nonce abstraction.

Envelope protocol (3/3)

$\longrightarrow$ Add freshness by adding an additional boot parameter to the att and key predicates.

$$\mathsf{att}(x_b, x_p, x) \rightarrow \mathsf{att}(x_b, \mathrm{h}(x_p, \mathsf{n}[x_b]), x)$$

$$\ldots$$

Envelope protocol (3/3)

$\longrightarrow$ Add freshness by adding an additional boot parameter to the att and key predicates.

$$\mathsf{att}(x_b, x_p, x) \rightarrow \mathsf{att}(x_b, \mathrm{h}(x_p, \mathsf{n}[x_b]), x)$$

...

PCR reboot rules:

$$\begin{array}{rcl}
\mathsf{att}(x_b, x_p, x) & \rightarrow & \mathsf{att}(\mathrm{b}(x_b, x_p), \mathsf{u}_0, x) \\
\mathsf{key}(x_b, x_p, \mathsf{srk}, \mathsf{pk}(\mathsf{srk}), \mathsf{nil}) & \rightarrow & \mathsf{key}(\mathrm{b}(x_b, x_p), \mathsf{u}_0, \mathsf{srk}, \mathsf{pk}(\mathsf{srk}), \mathsf{nil}) \\
\mathsf{key}(x_b, x_p, \mathsf{aik}, \mathsf{pk}(\mathsf{aik}), \mathsf{nil}) & \rightarrow & \mathsf{key}(\mathrm{b}(x_b, x_p), \mathsf{u}_0, \mathsf{aik}, \mathsf{pk}(\mathsf{aik}), \mathsf{nil})
\end{array}$$

Envelope protocol (3/3)

$\longrightarrow$ Add freshness by adding an additional boot parameter to the att and key predicates.

$$\mathsf{att}(x_b, x_p, x) \rightarrow \mathsf{att}(x_b, \mathrm{h}(x_p, \mathsf{n}[x_b]), x)$$
$$\ldots$$

PCR reboot rules:

$$\begin{array}{rcl} \mathsf{att}(x_b, x_p, x) & \rightarrow & \mathsf{att}(\mathrm{b}(x_b, x_p), \mathsf{u}_0, x) \\ \mathsf{key}(x_b, x_p, \mathsf{srk}, \mathsf{pk}(\mathsf{srk}), \mathsf{nil}) & \rightarrow & \mathsf{key}(\mathrm{b}(x_b, x_p), \mathsf{u}_0, \mathsf{srk}, \mathsf{pk}(\mathsf{srk}), \mathsf{nil}) \\ \mathsf{key}(x_b, x_p, \mathsf{aik}, \mathsf{pk}(\mathsf{aik}), \mathsf{nil}) & \rightarrow & \mathsf{key}(\mathrm{b}(x_b, x_p), \mathsf{u}_0, \mathsf{aik}, \mathsf{pk}(\mathsf{aik}), \mathsf{nil}) \end{array}$$

Result of our analysis:

$\longrightarrow$ Due to the boot parameter, ProVerif encounters termination problems.

$\longrightarrow$ ProVerif confirms that the protocol is secure (around 30 min) for 1 reboot.

Conclusion

Formal Horn clauses-based framework for modelling protocols of the TPM that use PCRs.

Our method:

1. model everything using Horn clauses;
2. show that the set of clauses needed are k-stable, and apply our attack-preserving transformation;
3. launch ProVerif (or another tool) on the resulting set of clauses.

Conclusion

Formal Horn clauses-based framework for modelling protocols of the TPM that use PCRs.

Our method:

1. model everything using Horn clauses;
2. show that the set of clauses needed are k-stable, and apply our attack-preserving transformation;
3. launch ProVerif (or another tool) on the resulting set of clauses.

Case studies:

1. A simplified version of the Microsoft Bitlocker protocol
   $\longrightarrow$ rules are 3-stable, ProVerif quickly conludes on the resulting set of rules, the VMK remains secret for unbounded reboots and PCR extends
2. The envelope protocol [Ables & Ryan, 10]
   $\longrightarrow$ add freshness through the boot parameter to avoid a false attack, rules are 2-stable, ProVerif concludes for one reboot.