Formal Analysis of Key Management APIs

Graham Steel

with Matteo Bortolozzo, Matteo Centenaro, Riccardo Focardi

INRIA & LSV, ENS de Cachan

and Università Ca' Foscari, Venezia

Cryptographic key management

The ‘elephant in the room’ of cryptographic security

Cryptographic key management

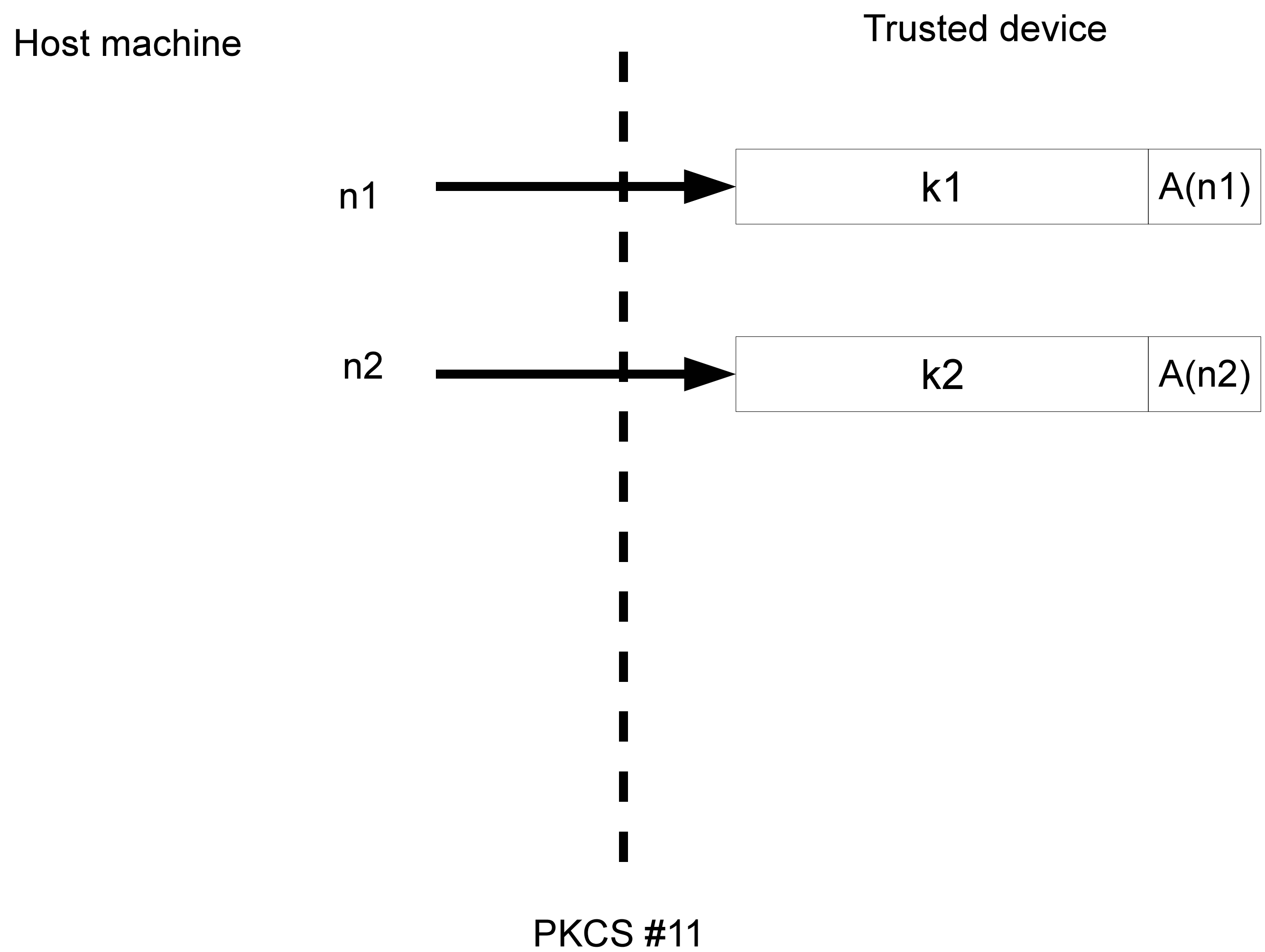
The ‘elephant in the room’ of cryptographic security

- Key creation and destruction
- Key establishment and distribution
- Key storage and backup
- Key use according to policy
- For many hundreds of keys (every employee laptop, smartcard, credential, ticket, token, device, ...)
- .. and all in a secure, robust way in a distributed system in a hostile environment

Host machine

Trusted device

Security API


Host machine
Trusted device
n1
k1
A(n1)
n2
k2
A(n2)
PKCS #11

Before we go on. . .

PKCS#11 is specified in a 400 page document

Funtions defined by headers and long textual descriptions

We go straight to abstract model

Before we go on...

PKCS#11 is specified in a 400 page document

Funtions defined by headers and long textual descriptions

We go straight to abstract model

$\mathsf{h}(\mathsf{n1}, \mathsf{k1})$ - a handle $\mathsf{n1}$ for key $\mathsf{k1}$ (h is a *private symbol*)

$\mathsf{a1}(\mathsf{n1})$ - setting of attribute $\mathsf{a1}$ for handle $\mathsf{n1}$

Before we go on...

PKCS#11 is specified in a 400 page document

Funtions defined by headers and long textual descriptions

We go straight to abstract model

$\mathsf{h}(\mathsf{n1}, \mathsf{k1})$ - a handle $\mathsf{n1}$ for key $\mathsf{k1}$ (h is a *private symbol*)

$\mathsf{a1}(\mathsf{n1})$ - setting of attribute $\mathsf{a1}$ for handle $\mathsf{n1}$

$$\text{Command}: \quad \mathsf{input}; \mathsf{state} \xrightarrow{\mathsf{new}} \mathsf{output}; \mathsf{state}'$$

Key Management - 1

$$\mathsf{KeyGenerate}:$$

$$\xrightarrow{\text{new n,k}} \quad h(n,k); L$$

Where $L = \neg\mathsf{extractable}(n), \neg\mathsf{wrap}(n), \neg\mathsf{unwrap}(n),$
$\neg\mathsf{encrypt}(n), \neg\mathsf{decrypt}(n), \neg\mathsf{sensitive}(n)$

Key Management - 2

$$
\begin{array}{lrcl}
\mathsf{Set\_Wrap}: & h(x_1, y_1);\ \neg wrap(x_1) & \rightarrow & ;wrap(x_1) \\
\mathsf{Set\_Encrypt}: & h(x_1, y_1);\ \neg encrypt(x_1) & \rightarrow & ;encrypt(x_1) \\
\vdots & & \vdots & \\
\mathsf{UnSet\_Wrap}: & h(x_1, y_1);\ wrap(x_1) & \rightarrow & ;\neg wrap(x_1) \\
\mathsf{UnSet\_Encrypt}: & h(x_1, y_1);\ encrypt(x_1) & \rightarrow & ;\neg encrypt(x_1) \\
\vdots & & \vdots &
\end{array}
$$

Some restrictions, e.g. can’t unset sensitive

Key Management - 3

Wrap :

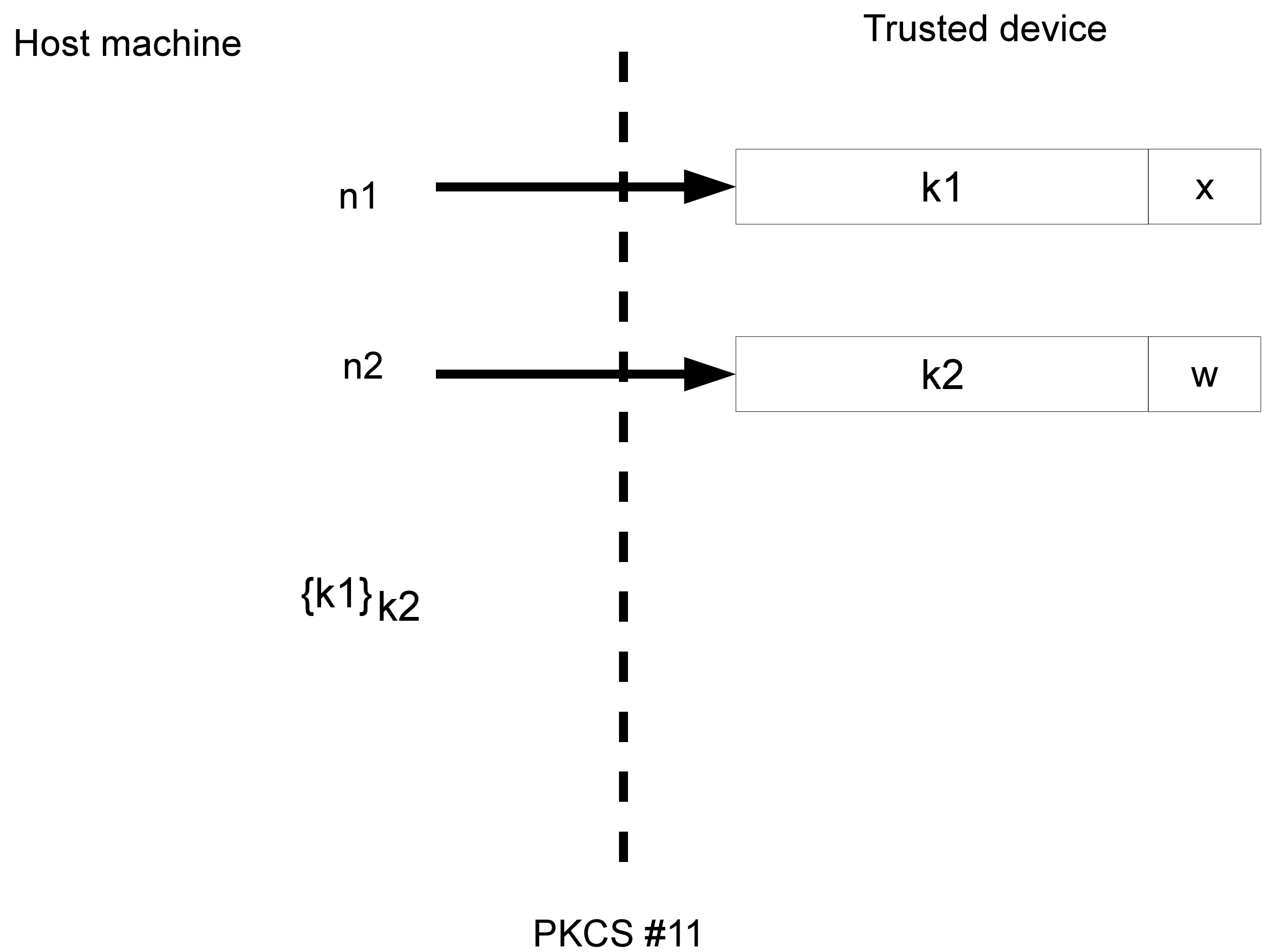
$$h(x_1, y_1), h(x_2, y_2);\ \mathrm{wrap}(x_1),\ \mathrm{extract}(x_2) \quad \rightarrow \quad \{y_2\}_{y_1}$$

Unwrap :

$$h(x_2, y_2), \{y_1\}_{y_2};\ \mathrm{unwrap}(x_2) \xrightarrow{\text{new } n_1} h(n_1, y_1);\ \mathrm{extract}(n_1),\ L$$

where $L =$

$\neg\mathrm{wrap}(n_1), \neg\mathrm{unwrap}(n_1), \neg\mathrm{encrypt}(n_1), \neg\mathrm{decrypt}(n_1), \neg\mathrm{sensitive}(n_1)$.


Host machine
Trusted device
n1
k1
x
n2
k2
w
{k1}k2
PKCS #11

Key Usage

Encrypt :

$$h(x_1, y_1), y_2;\ \text{encrypt}(x_1) \quad \rightarrow \quad \{y_2\}_{y_1}$$

Decrypt :

$$h(x_1, y_1), \{y_2\}_{y_1};\ \text{decrypt}(x_1) \quad \rightarrow \quad y_2$$

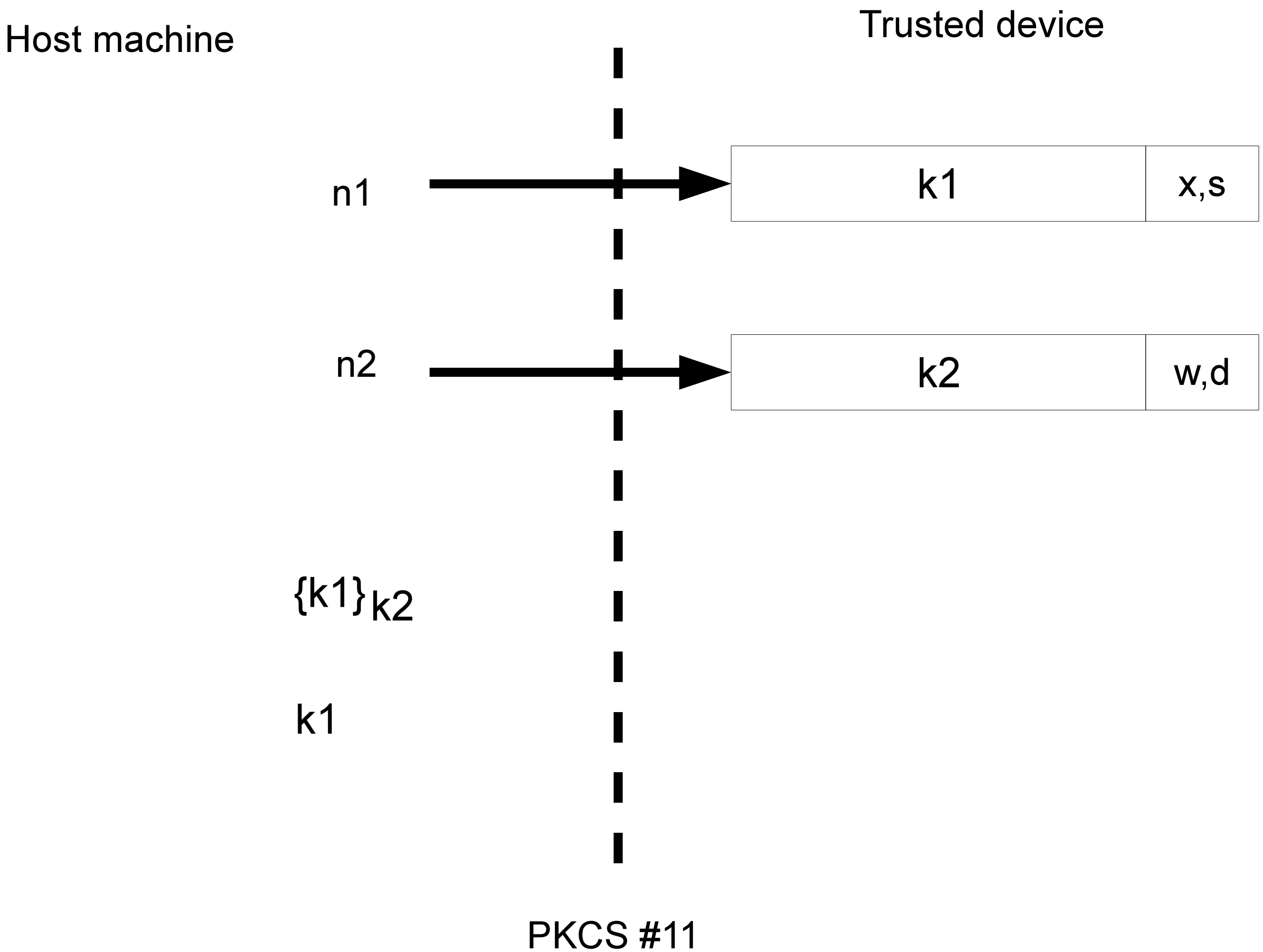
Key Separation Attack (Clulow, 2003)

Intruder knows: $\mathsf{h}(\mathsf{n}_1, \mathsf{k}_1)$, $\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2)$.

State: $\mathsf{wrap}(\mathsf{n}_2)$, $\mathsf{decrypt}(\mathsf{n}_2)$, $\mathsf{sensitive}(\mathsf{n}_1)$, $\mathsf{extract}(\mathsf{n}_1)$

Wrap: $\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2), \mathsf{h}(\mathsf{n}_1, \mathsf{k}_1) \rightarrow \{\mathsf{k}_1\}_{\mathsf{k}_2}$

Decrypt: $\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2), \{\mathsf{k}_1\}_{\mathsf{k}_2} \rightarrow \mathsf{k}_1$


Host machine
Trusted device
n1
k1
x,s
n2
k2
w,d
{k1}k2
k1
PKCS #11

Fix decrypt/wrap attack..

Fix decrypt/wrap attack..

$$\begin{array}{llll} \mathsf{Set\_Wrap:} & \mathsf{h}(\mathsf{x}_1,\mathsf{y}_1);\ \neg\mathsf{wrap}(\mathsf{x}_1),\neg\mathsf{decrypt}(\mathsf{x}_1) & \longrightarrow & \mathsf{wrap}(\mathsf{x}_1) \\ \mathsf{Set\_Decrypt:} & \mathsf{h}(\mathsf{x}_1,\mathsf{y}_1);\ \neg\mathsf{wrap}(\mathsf{x}_1),\neg\mathsf{decrypt}(\mathsf{x}_1) & \longrightarrow & \mathsf{decrypt}(\mathsf{x}_1) \end{array}$$

Fix decrypt/wrap attack..

$$\begin{array}{llcl} \mathsf{Set\_Wrap:} & \mathsf{h}(x_1, y_1);\ \neg\mathsf{wrap}(x_1), \neg\mathsf{decrypt}(x_1) & \longrightarrow & \mathsf{wrap}(x_1) \\ \mathsf{Set\_Decrypt:} & \mathsf{h}(x_1, y_1);\ \neg\mathsf{wrap}(x_1), \neg\mathsf{decrypt}(x_1) & \longrightarrow & \mathsf{decrypt}(x_1) \end{array}$$

~~Unset_Wrap~~

~~Unset_Decrypt~~

Another Attack

Intruder knows: $\mathsf{h}(\mathsf{n}_1, \mathsf{k}_1)$, $\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2)$, k_3

State: $\mathsf{sensitive}(\mathsf{n}_1), \mathsf{extract}(\mathsf{n}_1)$, $\mathsf{unwrap}(\mathsf{n}_2)$, $\mathsf{encrypt}(\mathsf{n}_2)$

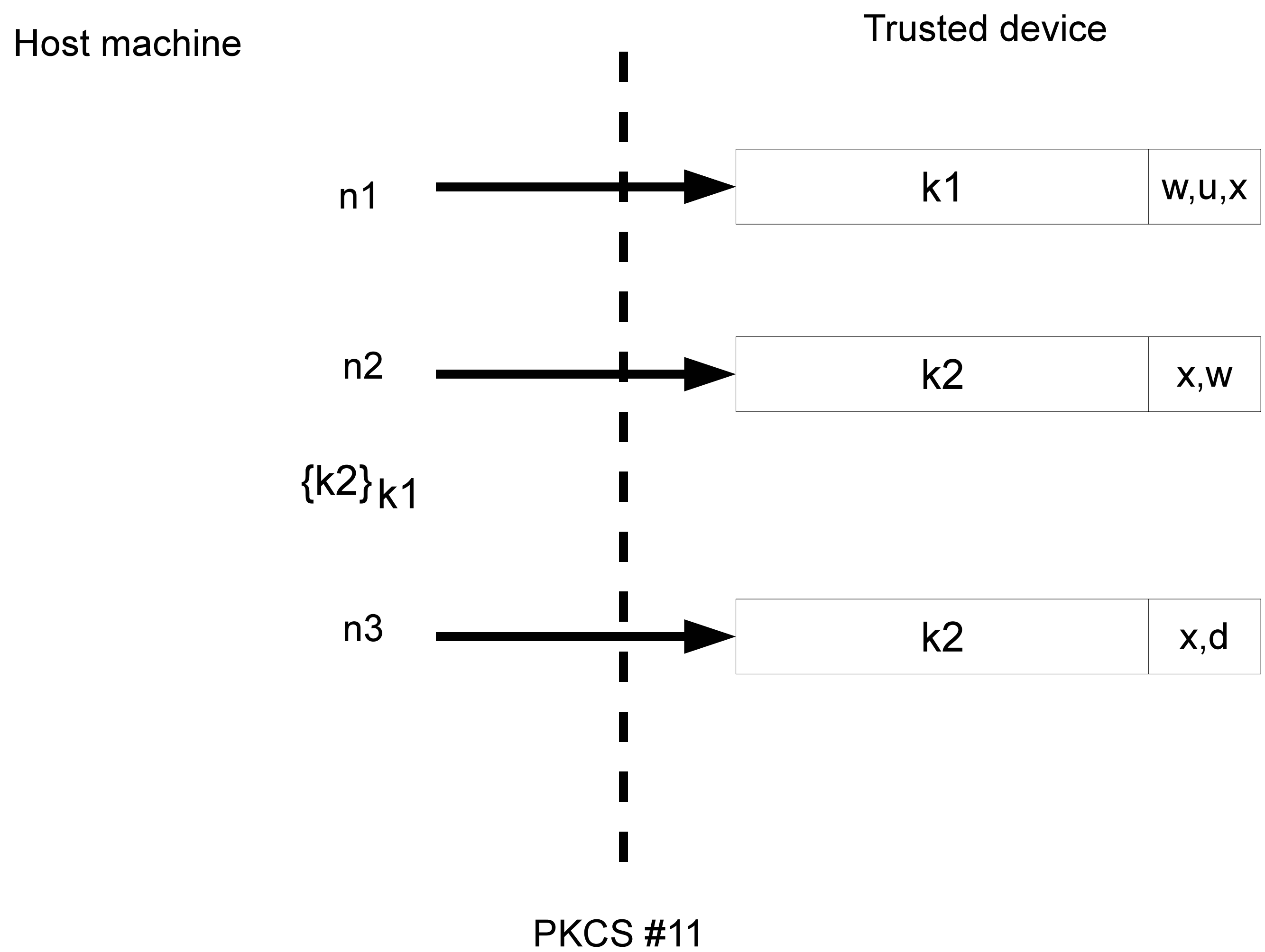
SEncrypt:	$\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2)$, k_3	$\rightarrow$	$\{\mathsf{k}_3\}_{\mathsf{k}_2}$
Unwrap:	$\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2)$, $\{\mathsf{k}_3\}_{\mathsf{k}_2}$	$\xrightarrow{\mathsf{new}\,\mathsf{n}_3}$	$\mathsf{h}(\mathsf{n}_3, \mathsf{k}_3)$
Set_wrap:	$\mathsf{h}(\mathsf{n}_3, \mathsf{k}_3)$	$\rightarrow$	$\mathsf{wrap}(\mathsf{n}_3)$
Wrap:	$\mathsf{h}(\mathsf{n}_3, \mathsf{k}_3)$, $\mathsf{h}(\mathsf{n}_1, \mathsf{k}_1)$	$\rightarrow$	$\{\mathsf{k}_1\}_{\mathsf{k}_3}$
Intruder:	$\{\mathsf{k}_1\}_{\mathsf{k}_3}$, k_3	$\rightarrow$	k_1

Fix decrypt/wrap, encrypt/unwrap..

Intruder knows: $\mathsf{h}(\mathsf{n}_1, \mathsf{k}_1)$, $\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2)$, k_3

State: $\mathsf{sensitive}(\mathsf{n}_1), \mathsf{extract}(\mathsf{n}_1)$, $\mathsf{extract}(\mathsf{n}_2)$

Set_wrap:	$\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2)$	$\rightarrow$	$;\mathsf{wrap}(\mathsf{n}_2)$
Set_wrap:	$\mathsf{h}(\mathsf{n}_1, \mathsf{k}_1)$	$\rightarrow$	$;\mathsf{wrap}(\mathsf{n}_1)$
Wrap:	$\mathsf{h}(\mathsf{n}_1, \mathsf{k}_1), \mathsf{h}(\mathsf{n}_2, \mathsf{k}_2)$	$\rightarrow$	$\{\mathsf{k}_2\}_{\mathsf{k}_1}$
Set_unwrap:	$\mathsf{h}(\mathsf{n}_1, \mathsf{k}_1)$	$\rightarrow$	$;\mathsf{unwrap}(\mathsf{n}_1)$
Unwrap:	$\mathsf{h}(\mathsf{n}_1, \mathsf{k}_1), \{\mathsf{k}_2\}_{\mathsf{k}_1}$	$\xrightarrow{\mathsf{new}\,\mathsf{n}_3}$	$\mathsf{h}(\mathsf{n}_3, \mathsf{k}_2)$
Wrap:	$\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2), \mathsf{h}(\mathsf{n}_1, \mathsf{k}_1)$	$\rightarrow$	$\{\mathsf{k}_1\}_{\mathsf{k}_2}$
Set_decrypt:	$\mathsf{h}(\mathsf{n}_3, \mathsf{k}_2)$	$\rightarrow$	$;\mathsf{decrypt}(\mathsf{n}_3)$
Decrypt:	$\mathsf{h}(\mathsf{n}_3, \mathsf{k}_2), \{\mathsf{k}_1\}_{\mathsf{k}_2}$	$\rightarrow$	k_1


Host machine
Trusted device
n1
k1
w,u,x
n2
k2
x,w
{k2}k1
n3
k2
x,d
PKCS #11

From Formal Models to Real Tokens

From Formal Models to Real Tokens

We published results in a formal model at CSF 2008.

Reaction from manufacturers was muted

From Formal Models to Real Tokens

We published results in a formal model at CSF 2008.

Reaction from manufacturers was muted

Decided to construct toolset to query functionality of real token, build model, find attacks, and execute candidate attacks directly on token.

From Formal Models to Real Tokens

We published results in a formal model at CSF 2008.

Reaction from manufacturers was muted

Decided to construct toolset to query functionality of real token, build model, find attacks, and execute candidate attacks directly on token.

Discovered real tokens differ from our CSF model:

- Most disable set_attribute, use *templates* instead

From Formal Models to Real Tokens

We published results in a formal model at CSF 2008.

Reaction from manufacturers was muted

Decided to construct toolset to query functionality of real token, build model, find attacks, and execute candidate attacks directly on token.

Discovered real tokens differ from our CSF model:

- Most disable set_attribute, use *templates* instead
- Include `CreateObject`

From Formal Models to Real Tokens

We published results in a formal model at CSF 2008.

Reaction from manufacturers was muted

Decided to construct toolset to query functionality of real token, build model, find attacks, and execute candidate attacks directly on token.

Discovered real tokens differ from our CSF model:

- Most disable set_attribute, use *templates* instead
- Include `CreateObject`
- Some don't follow the standard in critical ways

Some model fragments

$$\text{KeyGenerate}: \quad ;\ \text{template}(\mathcal{A}, X, \mathcal{G}) \xrightarrow{\text{new n,k}} h(n,k); \mathcal{A}(n,X)$$

$$\text{KeyPairGenerate}: \quad ;\ \text{template}(\mathcal{A}, X, \mathcal{G}) \xrightarrow{\text{new n,s}} h(n,s), \text{pub}(s); \mathcal{A}(n,X)$$

Unwrap (sym/sym) :

$$h(x, y_2), \{\!|y_1|\!\}_{y_2};\ \text{unwrap}(x, \top),\ \text{template}(\mathcal{A}, X, \mathcal{U}) \xrightarrow{\text{new } n_1} h(n_1, y_1);\ \mathcal{A}(n_1, X)$$

$$\text{CreateObject}: \quad x;\ \text{template}(\mathcal{A}, X, \mathcal{C}) \xrightarrow{\text{new n}} h(n,x);\ \mathcal{A}(n,X)$$

Abstractions for Proof (based on Fröschle & Steel WITS '09)

$$\mathsf{KeyGenerate}: \quad ;\ \mathsf{template_i}(\mathcal{A}, \mathsf{X}, \mathcal{G}) \quad \rightarrow \quad \mathsf{h}(\mathsf{n_i}, \mathsf{k_i}); \mathcal{A}(\mathsf{n_i}, \mathsf{X})$$

$$\mathsf{KeyPairGenerate}: \quad ;\ \mathsf{template_j}(\mathcal{A}, \mathsf{X}, \mathcal{G}) \quad \rightarrow \quad \mathsf{h}(\mathsf{n_j}, \mathsf{s_j}), \mathsf{pub}(\mathsf{s_j}); \mathcal{A}(\mathsf{n_j}, \mathsf{X})$$

$\mathsf{Unwrap}\,(\mathsf{sym}/\mathsf{sym}):$

$$\begin{array}{lll} \mathsf{h}(\mathsf{x}, \mathsf{y_2}), \{\!|\mathsf{y_1}|\!\}_{\mathsf{y_2}};\ \mathsf{unwrap}(\mathsf{x}, \top), & \rightarrow & \mathsf{h}(\mathsf{n_l}, \mathsf{y_1}); \\ \mathsf{template_l}(\mathcal{A}, \mathsf{X}, \mathcal{U}) & & \mathcal{A}(\mathsf{n_l}, \mathsf{X}) \end{array}$$

$$\mathsf{CreateObject}: \quad \mathsf{x};\ \mathsf{template_m}(\mathcal{A}, \mathsf{X}, \mathcal{C}) \quad \rightarrow \quad \mathsf{h}(\mathsf{n_m}, \mathsf{x});\ \mathcal{A}(\mathsf{n_m}, \mathsf{X})$$

Results

Results

Tested on 18 devices from 11 different manufacturers

Of these 9 are vulnerable to at least one attack

The other 9 do not support wrap for sensitive keys

Manufacturers have been informed, reaction mixed

Full results to appear later in 2010

OpencryptokiX

IBM Opencryptoki is a library including a software token

Vulnerable to many attacks (but it’s a software token)

OpencryptokiX

IBM Opencryptoki is a library including a software token

Vulnerable to many attacks (but it’s a software token)

We have coded two fixed versions

- one implements config from Fröschle & Steel WITS ’09
- one is a new fix with no new crypto mechanisms

  Uses a carefully chosen set of templates

OpencryptokiX

IBM Opencryptoki is a library including a software token

Vulnerable to many attacks (but it's a software token)

We have coded two fixed versions

- one implements config from Fröschle & Steel WITS '09
- one is a new fix with no new crypto mechanisms

  Uses a carefully chosen set of templates

Available to download from

http://secgroup.ext.dsi.unive.it/cryptokix

Details to appear at ASA-4 (FLoC '10 workshop)

A New Hope

Proposals for new APIs by Cachin and Chandran (CSF ’09), Cortier and Steel (ESORICS ’09).

A New Hope

Proposals for new APIs by Cachin and Chandran (CSF ’09), Cortier and Steel (ESORICS ’09).

– CC is for a single central server with a log, CS is for distributed tokens

A New Hope

Proposals for new APIs by Cachin and Chandran (CSF ’09), Cortier and Steel (ESORICS ’09).

– CC is for a single central server with a log, CS is for distributed tokens

– CC has a proof in the standard model of crypto, CS a symbolic proof.

A New Hope

Proposals for new APIs by Cachin and Chandran (CSF ’09), Cortier and Steel (ESORICS ’09).

– CC is for a single central server with a log, CS is for distributed tokens

– CC has a proof in the standard model of crypto, CS a symbolic proof.

Standards processes trying to set new APIs

– OASIS Key Management Interoperability Protocol

– IEEE Security in Storage Working Group

– PKCS#11 2.30 (no improvement)

Cachin-Chandran API

- Assume only one key server, many users, log of all operations
- Keys created with no attributes. Owner of key can set permissions
- Conflicts are checked by looking in the log, e.g. ’if this key has been used by any user for wrapping, do not allow it to be used for decryption’
- Also calculates dependencies between keys

+ very flexible, - fails immediately if a key is compromised, or if distributed over several servers, ‘proof’ a little odd

Cortier-Steel API

- Assume distributed tokens, one for each user
- Strict hierarchy of wrap/unwrap and encrypt/decrypt keys
- Keys created with attributes that cannot be changed in future
- Key attributes include names of other users key can be shared with
- All encryptions tagged with key/user information

+ strong security properties, robust to loss of keys, no central log required

- not as flexible as Cachin proposal

More on Key Management APIs

S. Delaune, S. Kremer and G. Steel. *Formal Analysis of PKCS#11 and Proprietary Extensions*. To appear in JCS 2010

V. Cortier and G. Steel. *A Generic API for Symmetric Key Management*. In ESORICS '09.

C. Chachin and N. Chandran. *A Secure Cryptographic Token Interface*. In CSF-22.

S. Fröschle and G. Steel. *Analysis of PKCS#11 APIs with Unbounded Fresh Data*, ARSPA-WITS '09.

OASIS www.oasis-open.org/committees/kmip

IEEE 1619 siswg.net

ASA-4, July 21, http://www.lsv.ens-cachan.fr/~steel/asa4