Formal Analysis of Security APIs

Graham Steel

INRIA & LSV, ENS de Cachan

Host machine

Trusted device

Security API

PIN Processing APIs

Photo: redspotted/Flickr

Verizon Breach Report 2008

Released April 2009

Verizon Breach Report 2008

Released April 2009

“While statistically not a large percentage of our overall caseload in 2008, attacks against PIN information represent individual data-theft cases having the largest aggregate exposure in terms of unique records,”

“In other words, PIN-based attacks and many of the very large compromises from the past year go hand in hand.”

Verizon Breach Report 2008

Released April 2009

“While statistically not a large percentage of our overall caseload in 2008, attacks against PIN information represent individual data-theft cases having the largest aggregate exposure in terms of unique records,”

“In other words, PIN-based attacks and many of the very large compromises from the past year go hand in hand.”

“We’re seeing entirely new attacks that a year ago were thought to be only academically possible,”

Verizon Breach Report 2008

Released April 2009

“While statistically not a large percentage of our overall caseload in 2008, attacks against PIN information represent individual data-theft cases having the largest aggregate exposure in terms of unique records,”

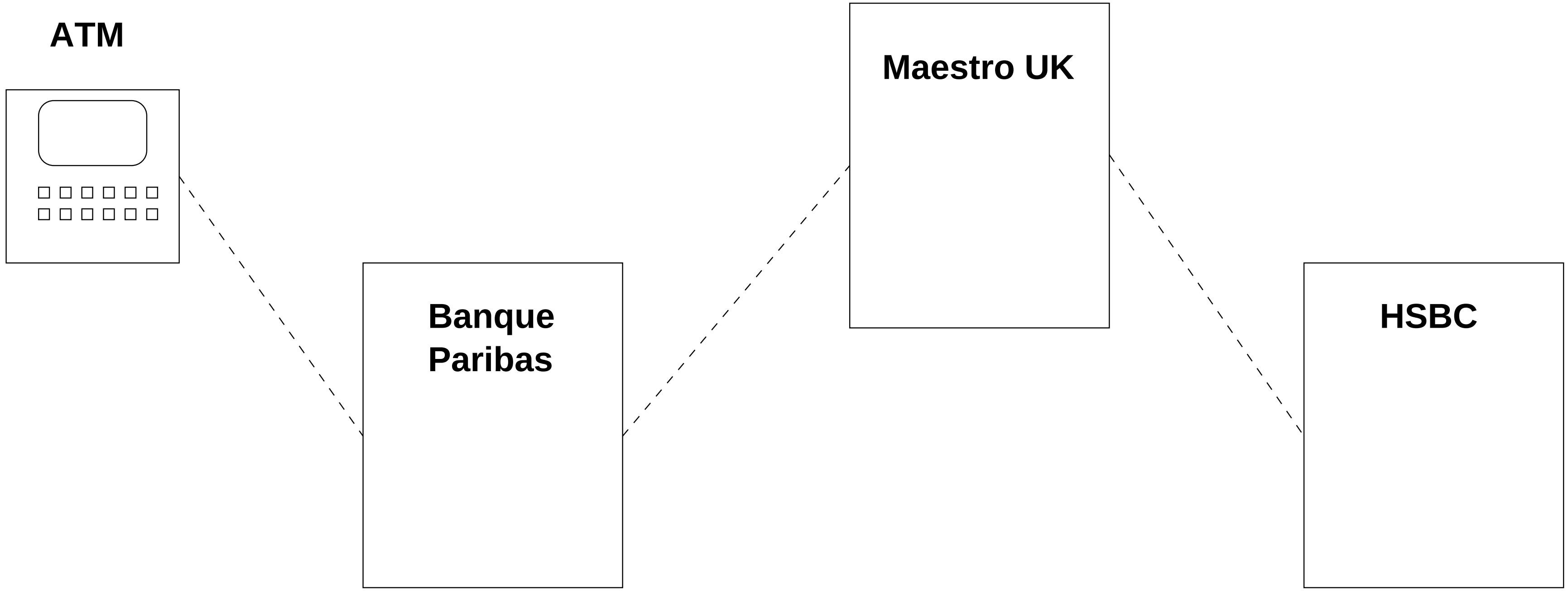
“In other words, PIN-based attacks and many of the very large compromises from the past year go hand in hand.”

“We’re seeing entirely new attacks that a year ago were thought to be only academically possible,”

“What we see now is people going right to the source [..] and stealing the encrypted PIN blocks and using complex ways to un-encrypt the PIN blocks.”

(Quotes from Wired Magazine interview with report author, Bryan Sartin)

Cash Machine Network



HSMs

- Manufacturers include IBM, VISA, nCipher, Thales, Utimaco, HP
- Cost around $10 000

Deriving a PIN: IBM 3624 Method

IPIN derived by:

Encode account number (PAN) as 0000AAAAAAAAAAAA

Deriving a PIN: IBM 3624 Method

IPIN derived by:

Encode account number (PAN) as 0000AAAAAAAAAAAA

3DES encrypt under a PDK (PIN Derivation Key)

Deriving a PIN: IBM 3624 Method

IPIN derived by:

Encode account number (PAN) as 0000AAAAAAAAAAAA

3DES encrypt under a PDK (PIN Derivation Key)

Take 4 leftmost hexadecimal digits of result

Deriving a PIN: IBM 3624 Method

IPIN derived by:

Encode account number (PAN) as 0000AAAAAAAAAAAA

3DES encrypt under a PDK (PIN Derivation Key)

Take 4 leftmost hexadecimal digits of result

Decimalise using a mapping table ('dectab')

```
0123456789ABCDEF
0123456789012345
```

Deriving a PIN: IBM 3624 Method

IPIN derived by:

Encode account number (PAN) as 0000AAAAAAAAAAAA

3DES encrypt under a PDK (PIN Derivation Key)

Take 4 leftmost hexadecimal digits of result

Decimalise using a mapping table ('dectab')

```
0123456789ABCDEF
0123456789012345
```

PIN = IPIN + Offset (modulo 10 each digit)

PIN Processing API

Verify PIN:

$\{\mathsf{PIN}\}_{\mathsf{K}}, \mathsf{PAN}, \mathsf{Dectab}$ $\longrightarrow$

Offset

yes/no $\longleftarrow$

K, PDK

PIN Processing API

Verify PIN:

$\{\mathsf{PIN}\}_{\mathsf{K}}, \mathsf{PAN}, \mathsf{Dectab}$ $\longrightarrow$

Offset

yes/no $\longleftarrow$

K, PDK

If host machine is attacked, PIN should remain secure (ANSI X7.8, ISO 9564 requirement)

Decimalisaton Table Attack (Clulow '02, Bond & Zeilinski '03)

Suppose in a hacked switch, an attacker has a set $\{\mathsf{PIN}\}_{\mathsf{K}}, \mathsf{PAN}, \mathsf{Dectab}, \mathsf{Offset}$ that verifies PIN is correct

Decimalisaton Table Attack (Clulow '02, Bond & Zeilinski '03)

Suppose in a hacked switch, an attacker has a set $\{\mathsf{PIN}\}_{\mathsf{K}}, \mathsf{PAN}, \mathsf{Dectab}, \mathsf{Offset}$ that verifies PIN is correct

Original Dectab

```
0123456789ABCDEF
0123456789012345
```

Dectab'

```
0123456789ABCDEF
1123456789112345
```

Decimalisaton Table Attack (Clulow '02, Bond & Zeilinski '03)

Suppose in a hacked switch, an attacker has a set $\{\mathsf{PIN}\}_\mathsf{K}, \mathsf{PAN}, \mathsf{Dectab}, \mathsf{Offset}$ that verifies PIN is correct

Original Dectab

```
0123456789ABCDEF
0123456789012345
```

Dectab'

```
0123456789ABCDEF
1123456789112345
```

Repeat verification command with Dectab'

Successful verification indicates no 0s in PIN

More dectab attack

To find the 0s, try changing the offset

Attacker set offset	Result from HSM	Knowledge of PIN
0001	Incorrect PIN	????
0010	Incorrect PIN	????
0100	Incorrect PIN	????
1000	Incorrect PIN	????
0011	Incorrect PIN	????
0101	Correct PIN	?0?0

AnaBlock (TCS 2006)

Take a customer configuration and an API spec. as input

AnaBlock (TCS 2006)

Take a customer configuration and an API spec. as input

Using CLP, generate tree of all possible attacks

AnaBlock (TCS 2006)

Take a customer configuration and an API spec. as input

Using CLP, generate tree of all possible attacks

Meta-logical predicates allow us to calculate transition probabilities

AnaBlock (TCS 2006)

Take a customer configuration and an API spec. as input

Using CLP, generate tree of all possible attacks

Meta-logical predicates allow us to calculate transition probabilities

Apply PRISM (Kwiatkowska et. al, 2004)

Get minimum expected number of steps to determine PIN

AnaBlock (TCS 2006)

Take a customer configuration and an API spec. as input

Using CLP, generate tree of all possible attacks

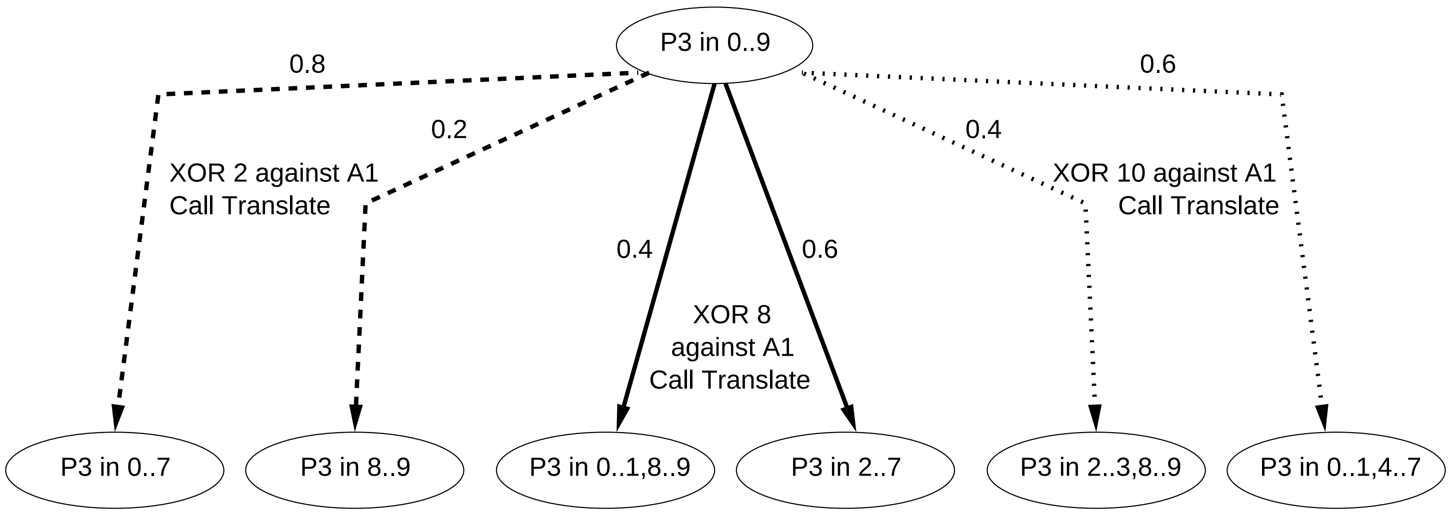
Meta-logical predicates allow us to calculate transition probabilities

Apply PRISM (Kwiatkowska et. al, 2004)

Get minimum expected number of steps to determine PIN

Generate tree for best attack

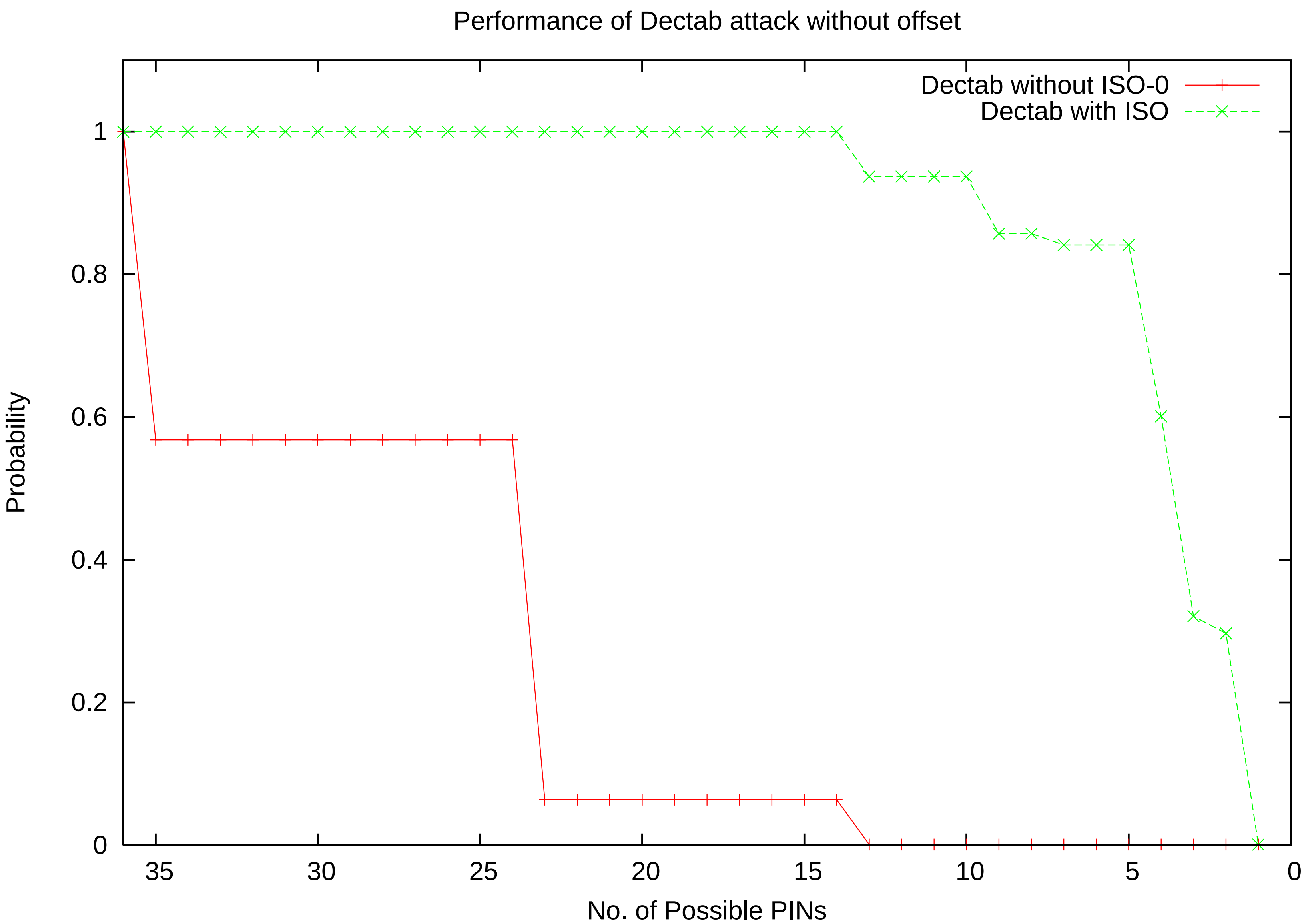
Attack Trees



Results from AnaBlock

No.	Attack	$E(Steps)$
(1)	ISO-0 (extended)	13.6
(2)	Dectab	16.145
(3)	Dectab & ISO (restricted)	15.275

No.	Attack	Range: 400	36	24	14	1
(4)	ISO-0 (restricted)	1	0	0	0	0
(5)	Dectab no offset	1	1	0.568	0.064	0.001
(6)	Dectab no offset & ISO-0 (restricted)	1	1	1	1	0.001



Performance of Dectab attack without offset
Dectab without ISO-0
Dectab with ISO
Probability
1
0.8
0.6
0.4
0.2
0
35
30
25
20
15
10
5
0
No. of Possible PINs

More PIN Cracking Attacks

- Dectab attacks
- Reformatting attacks
- Check value attack
- Calculate offset attack
- Competing verification algorithms attack

All require attacker to make ‘tweaked’ queries to HSM

Theory Behind Fix

Language based security

Theory Behind Fix

Language based security

- Multilevel view - high and low security

Theory Behind Fix

Language based security

- Multilevel view - high and low security
- Non-interference - no ‘flow’ from high to low

Theory Behind Fix

Language based security

- Multilevel view - high and low security
- Non-interference - no ‘flow’ from high to low
- Declassification - wrt a policy

Theory Behind Fix

Language based security

- Multilevel view - high and low security
- Non-interference - no ‘flow’ from high to low
- Declassification - wrt a policy
- Robustness - introduces integrity

Theory Behind Fix

Language based security

- Multilevel view - high and low security
- Non-interference - no ‘flow’ from high to low
- Declassification - wrt a policy
- Robustness - introduces integrity
- Endorsement - allows integrity to be raised

Theory Behind Fix

Language based security

- Multilevel view - high and low security
- Non-interference - no ‘flow’ from high to low
- Declassification - wrt a policy
- Robustness - introduces integrity
- Endorsement - allows integrity to be raised

We introduce cryptographically assured endorsement (ESORICS ’09) using MAC, and a ‘low cost’ version (NordSec ’09)

More PIN Processing

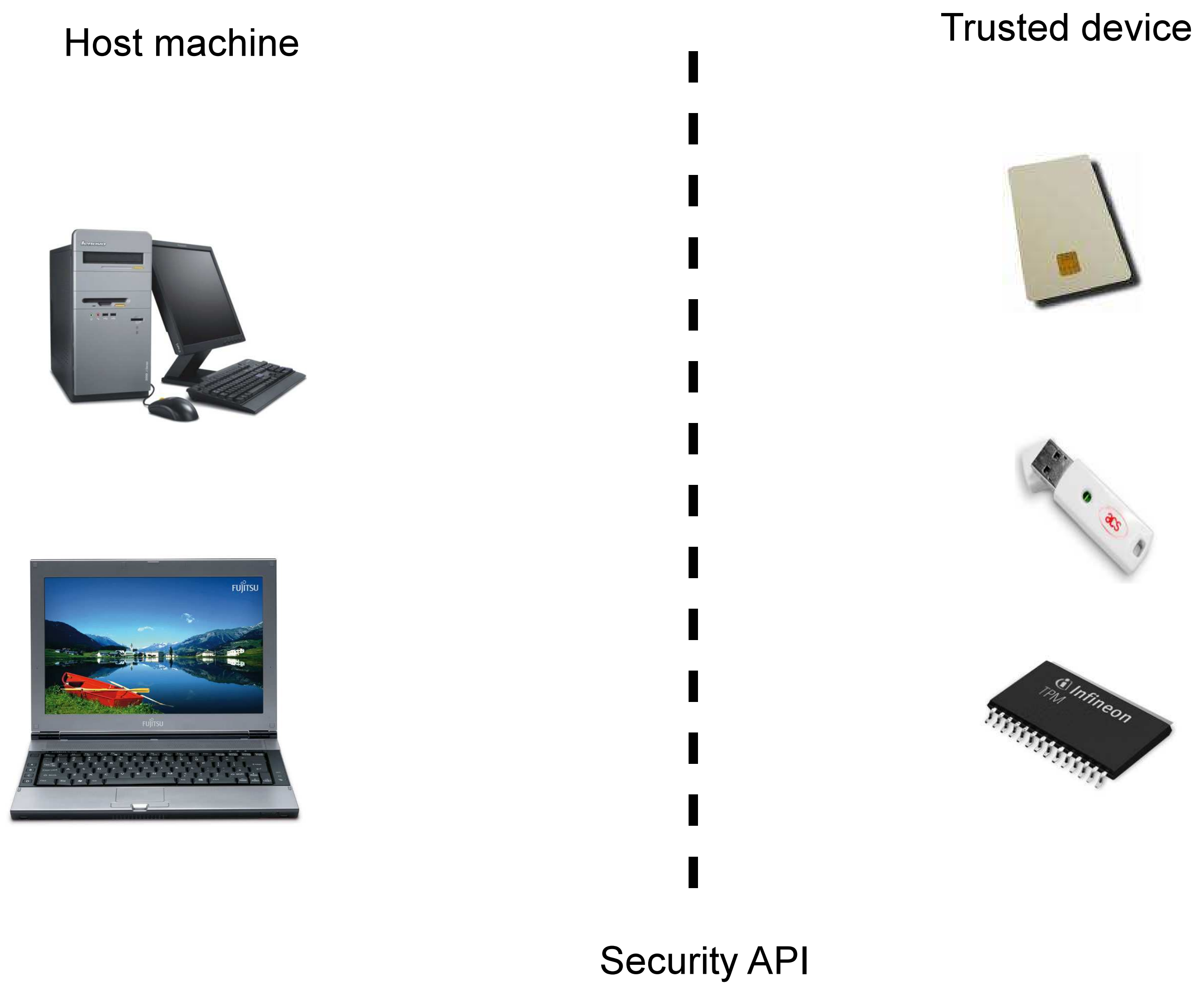
Wired Magazine, *PIN Crackers Nab Holy Grail of Bank Card Security* `http://www.wired.com/threatlevel/2009/04/pins/`

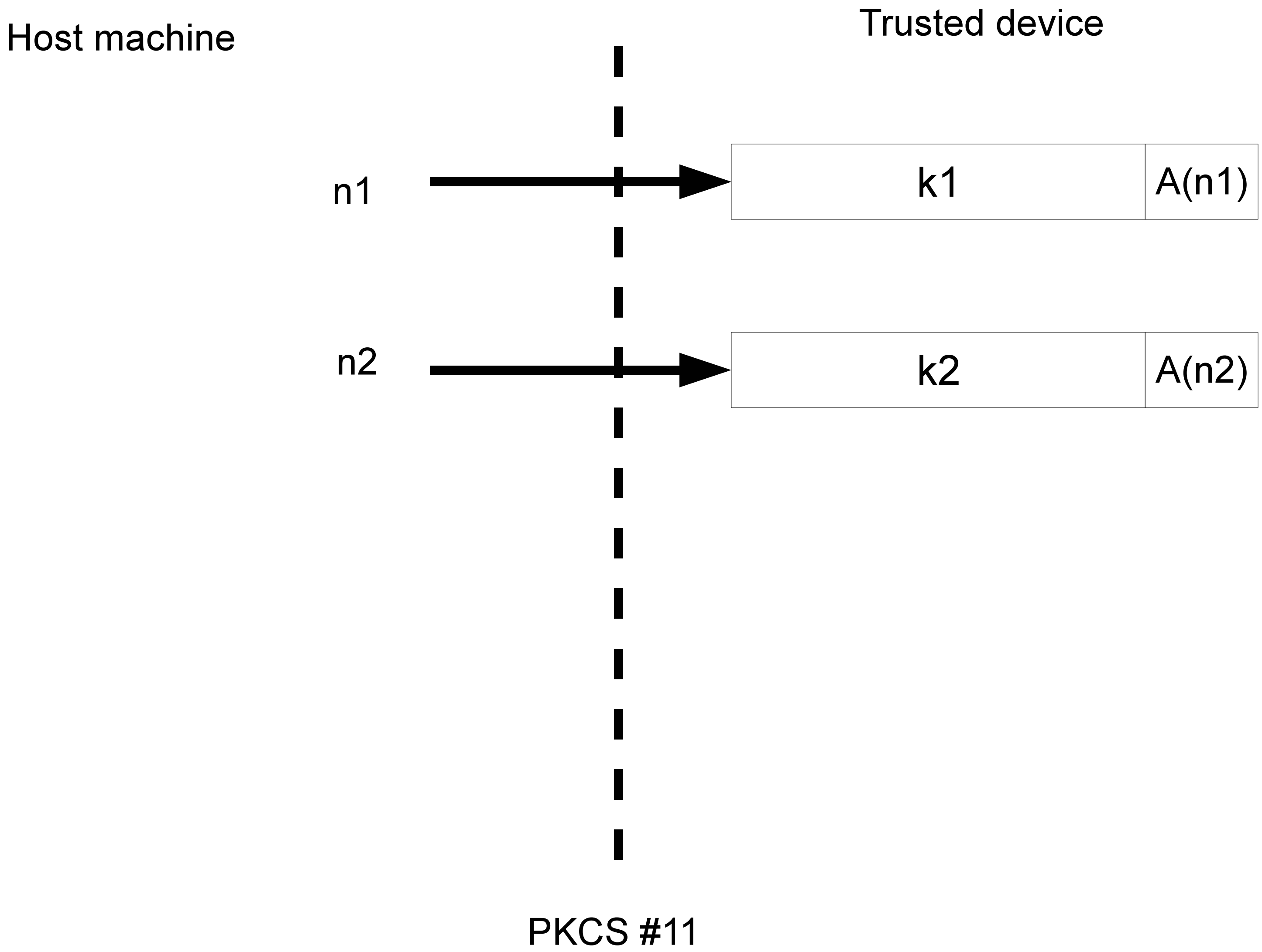
G. Steel. *Formal analysis of PIN block attacks*. Theoretical Computer Science 367(1-2), 2006.

R. Focardi, F. L. Luccio and G. Steel. *Blunting Differential Attacks on PIN Processing APIs*. In NordSec'09, LNCS 5838.

M. Centenaro, R. Focardi, F. L. Luccio and G. Steel. *Type-based Analysis of PIN Processing APIs*. In ESORICS'09, LNCS 5789

Mohammad Mannan, P.C. van Oorschot. *Reducing threats from flawed security APIs: The banking PIN case*, Computers & Security 28 (6), 2009.


Host machine
Trusted device
FUJITSU
Infineon
TPM
Security API


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

Key Management - 1

$$\mathsf{KeyGenerate:}$$

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

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

Key Management - 2

Wrap :

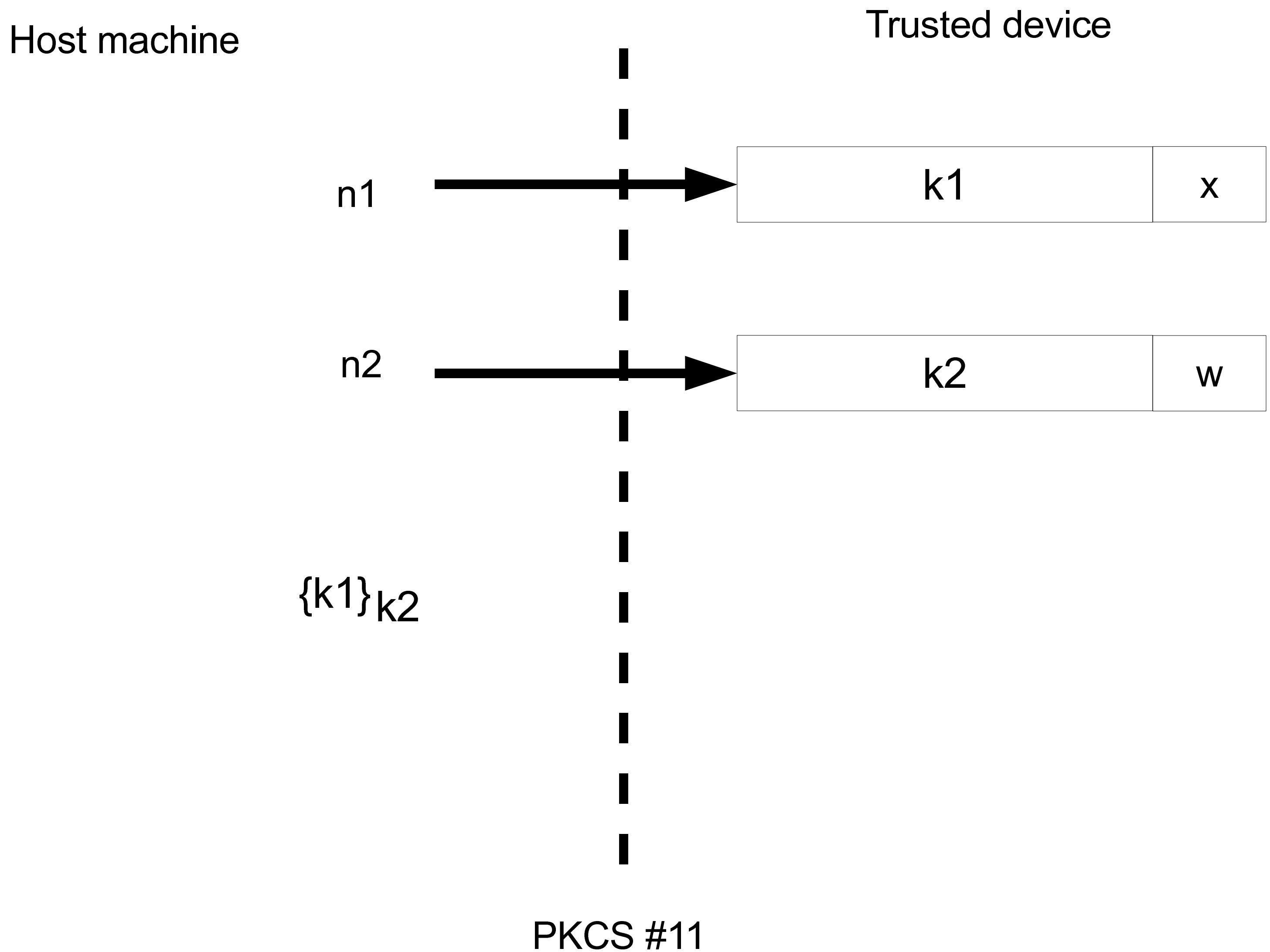
$$h(x_1, y_1), h(x_2, y_2); \text{wrap}(x_1), \text{extract}(x_2) \quad \rightarrow \quad \{y_2\}_{y_1}$$

Unwrap :

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

where $L =$

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



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

Key Management - 3

$$\begin{array}{lrcl}
\mathsf{Set_Wrap}: & h(x_1, y_1);\ \neg wrap(x_1) & \rightarrow & ;wrap(x_1) \\
\mathsf{Set_Encrypt}: & h(x_1, y_1);\ \neg encrypt(x_1) & \rightarrow & ;encrypt(x_1) \\
\vdots & & \vdots & \\
\mathsf{UnSet_Wrap}: & h(x_1, y_1);\ wrap(x_1) & \rightarrow & ;\neg wrap(x_1) \\
\mathsf{UnSet_Encrypt}: & h(x_1, y_1);\ encrypt(x_1) & \rightarrow & ;\neg encrypt(x_1) \\
\vdots & & \vdots &
\end{array}$$

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

Key Usage

$$\mathsf{Encrypt:}$$

$$\mathsf{h}(\mathsf{x}_1,\mathsf{y}_1),\mathsf{y}_2;\ \mathsf{encrypt}(\mathsf{x}_1) \quad \longrightarrow \quad \{\mathsf{y}_2\}_{\mathsf{y}_1}$$

$$\mathsf{Decrypt:}$$

$$\mathsf{h}(\mathsf{x}_1,\mathsf{y}_1),\{\mathsf{y}_2\}_{\mathsf{y}_1};\ \mathsf{decrypt}(\mathsf{x}_1) \quad \longrightarrow \quad \mathsf{y}_2$$

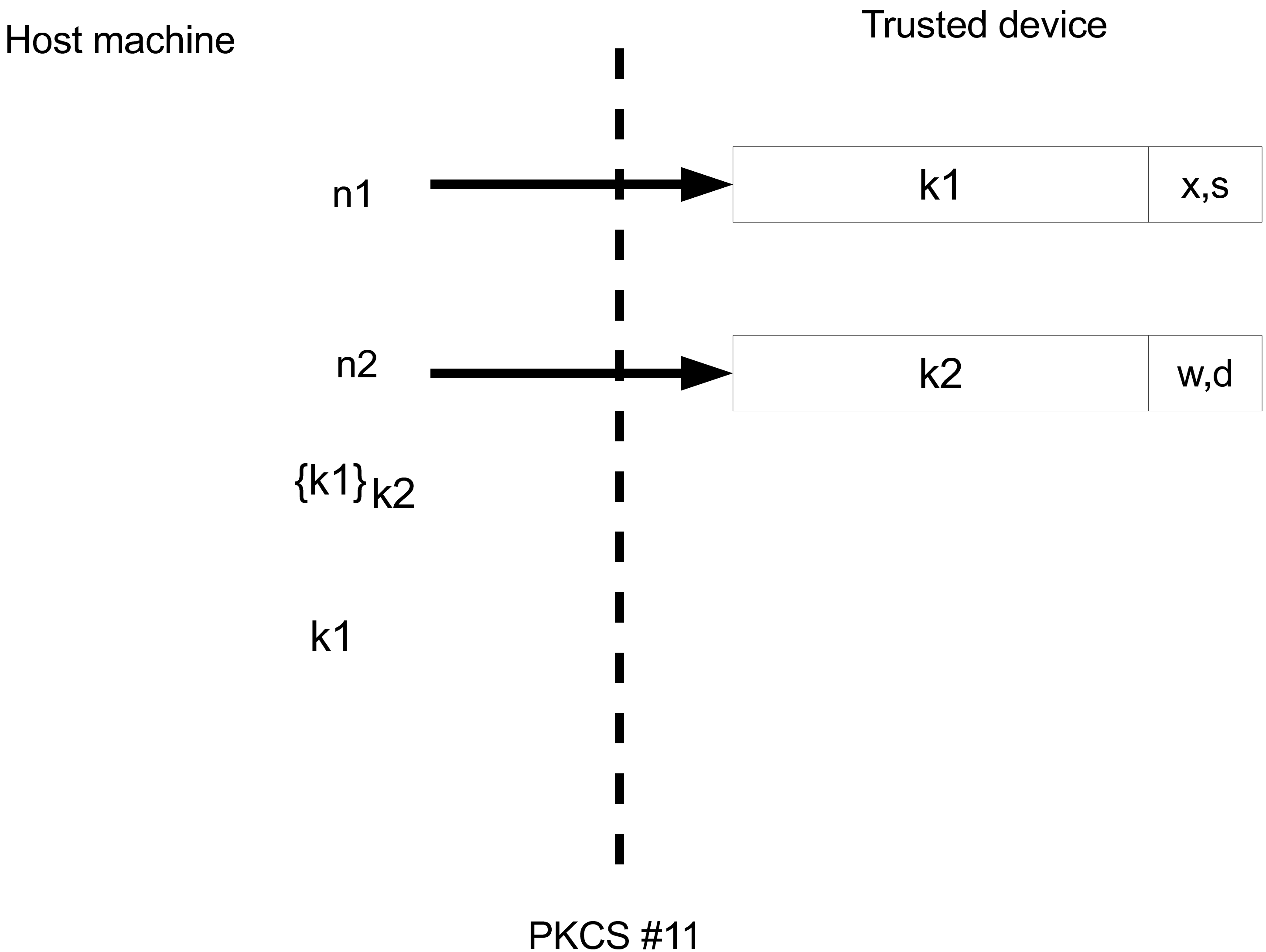
Key Separation Attack (Clulow, 2003)

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

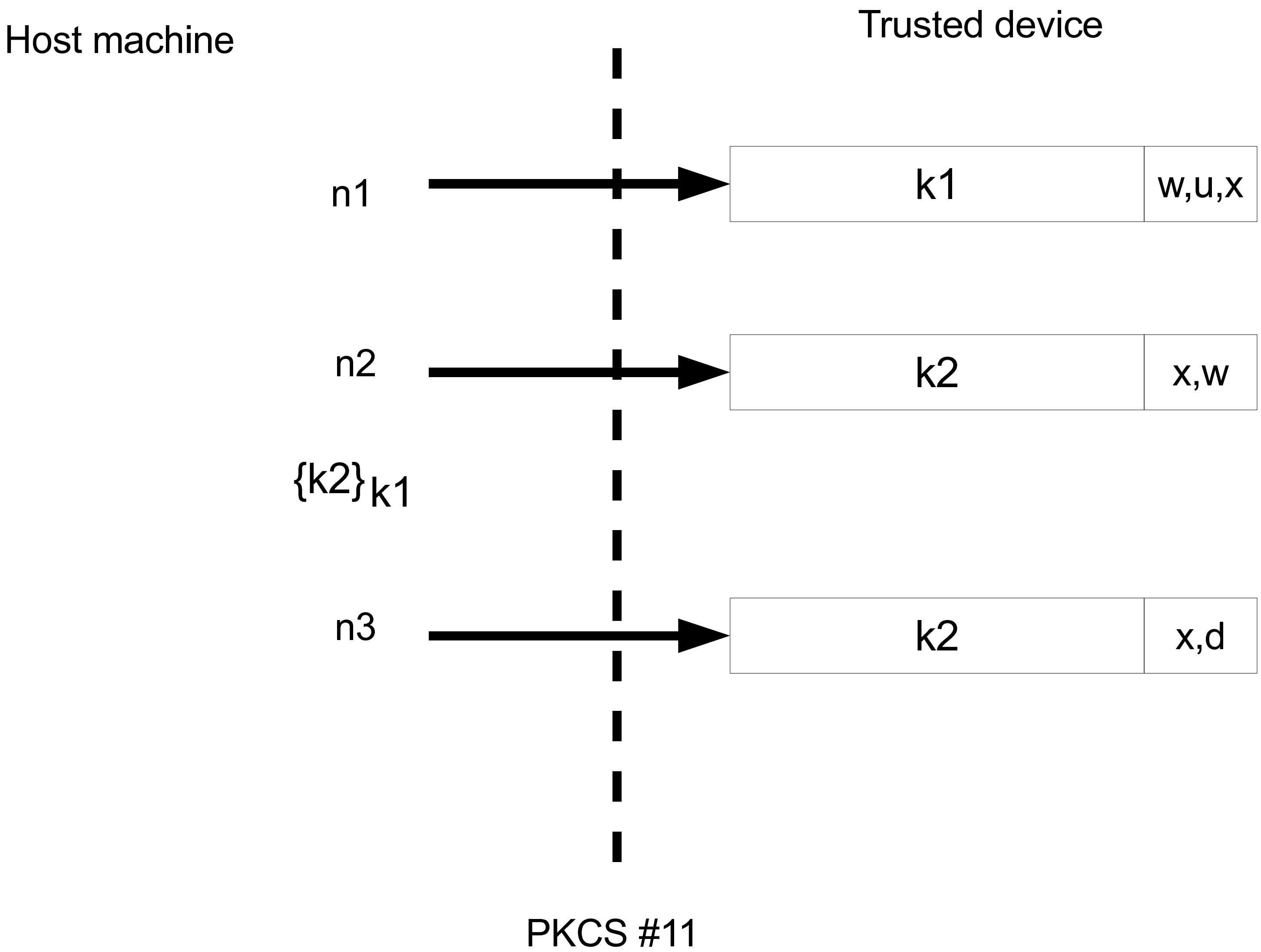

Host machine
Trusted device
n1
k1
x,s
n2
k2
w,d
{k1}k2
k1
PKCS #11

Re-import attack (DKS, 08)

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

Two kinds of problem

- A bad ‘attribute policy’

  One can set conflicting attributes for a key

- Policy not enforced

  By copying the key using wrap/unwrap, can ‘escape’ the policy

Two kinds of problem

- A bad ‘attribute policy’

  One can set conflicting attributes for a key

- Policy not enforced

  By copying the key using wrap/unwrap, can ‘escape’ the policy

Attack this problem by first formalising ‘attribute policy’

$$\text{KeyGenerate}: \quad \xrightarrow{\text{new } n_1, k_1} \quad h(n_1, k_1);\ L(n_1), \neg\text{extract}(n_1)$$

$$\text{Wrap}:$$

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

$$\text{Unwrap}:$$

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

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

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

$$\text{Set\_Encrypt}: \quad h(x_1, y_1);\ \neg\text{encrypt}(x_1) \quad \rightarrow \quad \text{encrypt}(x_1)$$

$$\text{UnSet\_Encrypt}: \quad h(x_1, y_1);\ \text{encrypt}(x_1) \quad \rightarrow \quad \neg\text{encrypt}(x_1)$$

$$\text{KeyGenerate}: \quad \xrightarrow{\text{new } n_1, k_1} \quad h(n_1, k_1);\ A(n_1)$$

$$\text{Wrap}:$$

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

$$\text{Unwrap}:$$

$$h(x_2, y_2), \{y_1\}_{y_2};\ \text{unwrap}(x_2) \quad \xrightarrow{\text{new } n_1} \quad h(n_1, y_1);\ A(n_1)$$

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

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

$$\text{Set\_Attribute\_Value}: \quad h(x_1, y_1);\ A_1(x_1) \quad \rightarrow \quad A_2(x_1)$$

Attribute Policy

An *attribute policy* is a finite directed graph $P = (S_P, \rightarrow_P)$ where S_P is the set of allowable object states, and $\rightarrow_P \subseteq S_P \times S_P$ is the set of allowable transitions between the object states.

Attribute Policy

An *attribute policy* is a finite directed graph $P = (S_P, \rightarrow_P)$ where S_P is the set of allowable object states, and $\rightarrow_P \subseteq S_P \times S_P$ is the set of allowable transitions between the object states.

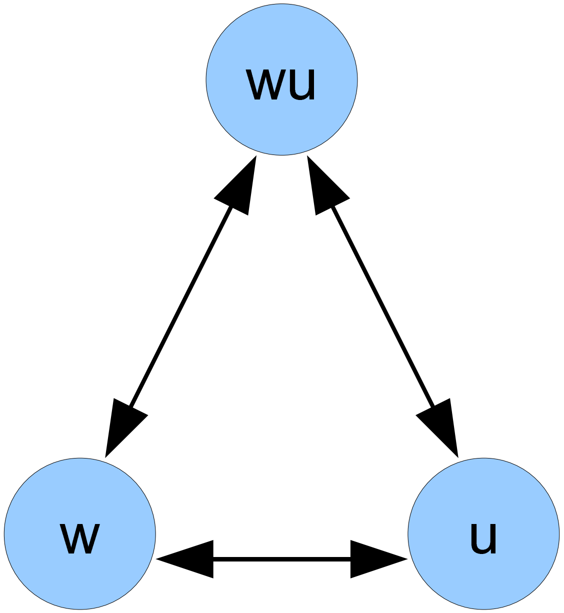
An attribute policy $P = (S, \rightarrow)$ is *complete* if P consists of a collection of disjoint, disconnected cliques, and for each clique C,
$c_0, c_1 \in C \Rightarrow c_0 \cup c_1 \in C$

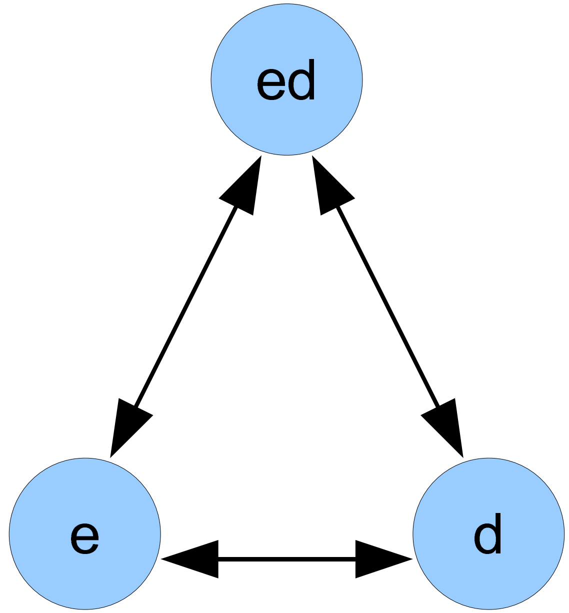
Attribute Policy

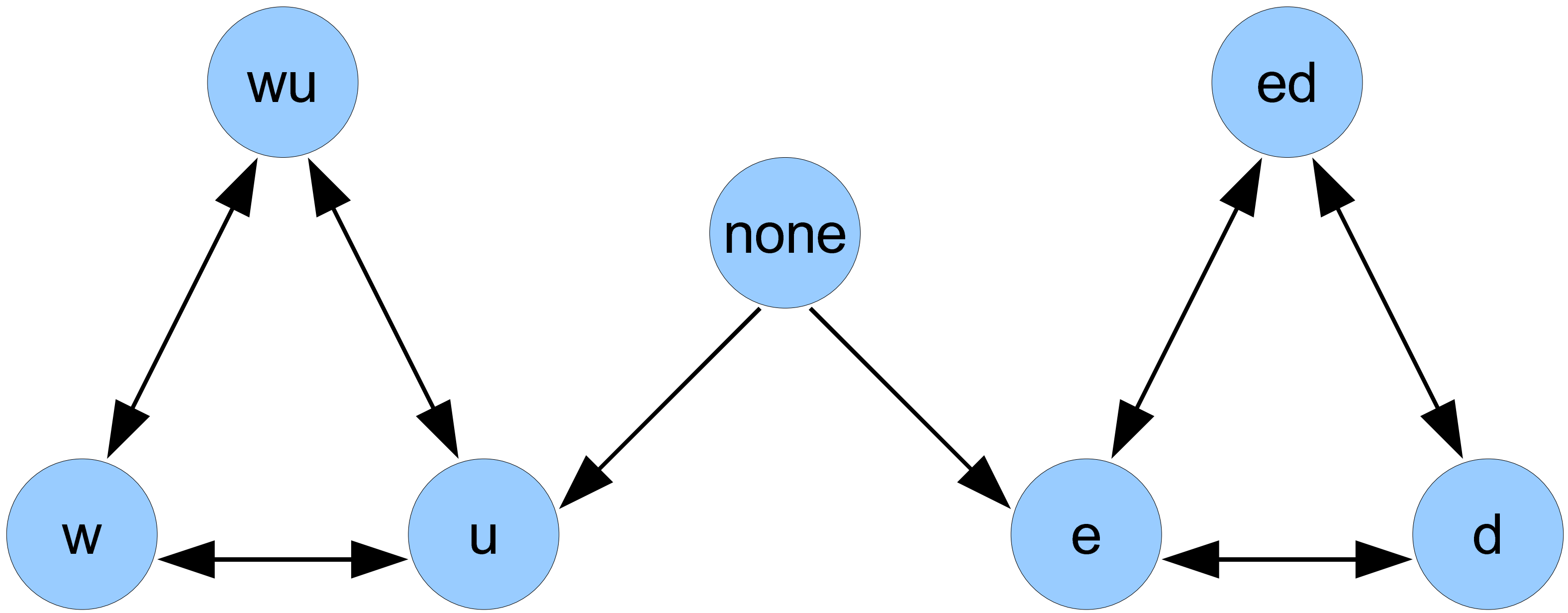
An *attribute policy* is a finite directed graph $P = (S_P, \rightarrow_P)$ where S_P is the set of allowable object states, and $\rightarrow_P \subseteq S_P \times S_P$ is the set of allowable transitions between the object states.

An attribute policy $P = (S, \rightarrow)$ is *complete* if P consists of a collection of disjoint, disconnected cliques, and for each clique C, $c_0, c_1 \in C \Rightarrow c_0 \cup c_1 \in C$

We insist on complete policies, assuming intruder can always copy keys.


wu
w
u


ed
e
d


wu
ed
none
w
u
e
d

Endpoints

We call the object states of S that are maximal in S with respect to set inclusion *end points* of P.

Theorem: Derivation in API with complete policy iff derivation in API with (static) endpoint policy

Bounds

Assume endpoint policies

Make series of simple transformations

Bounds

Assume endpoint policies

Make series of simple transformations

- Bound number of fresh keys to number of endpoints #ep

  - get the same key every time a particular endpoint is requested

Bounds

Assume endpoint policies

Make series of simple transformations

- Bound number of fresh keys to number of endpoints #ep

  - get the same key every time a particular endpoint is requested

- Bound number of handles to $(\#\mathrm{ep})^2$

  - for each key, get one handle for each endpoint

Bounds

Assume endpoint policies

Make series of simple transformations

- Bound number of fresh keys to number of endpoints #ep

  - get the same key every time a particular endpoint is requested

- Bound number of handles to $(\#\text{ep})^2$

  - for each key, get one handle for each endpoint

Intruder always starts with his own key

so require $\#\text{ep} + 1$ keys and $(\#\text{ep} + 1)^2$ handles

$$\mathsf{KeyGenerate:} \quad \xrightarrow{\mathsf{new}\ \mathsf{n}_1,\mathsf{k}_1} \quad \mathsf{h}(\mathsf{n}_1,\mathsf{k}_1);\mathsf{A}(\mathsf{n}_1)$$

$$\mathsf{Wrap:}$$

$$\mathsf{h}(\mathsf{x}_1,\mathsf{y}_1),\mathsf{h}(\mathsf{x}_2,\mathsf{y}_2);\ \mathsf{wrap}(\mathsf{x}_1),\mathsf{A}(\mathsf{x}_2) \quad \xrightarrow{\mathsf{new}\ \mathsf{m}_\mathsf{k}} \quad \mathsf{enc}(\mathsf{y}_2,\mathsf{y}_1),\mathsf{enc}(\mathsf{m}_\mathsf{k},\mathsf{y}_1)$$
$$\mathsf{hmac}_{\mathsf{m}_\mathsf{k}}(\mathsf{y}_2,\mathcal{A})$$

$$\mathsf{Unwrap:}$$

$$\mathsf{h}(\mathsf{x}_2,\mathsf{y}_2),\mathsf{enc}(\mathsf{y}_1,\mathsf{y}_2),\mathsf{enc}(\mathsf{x}_\mathsf{m},\mathsf{y}_2), \quad \xrightarrow{\mathsf{new}\ \mathsf{n}_1} \quad \mathsf{h}(\mathsf{n}_1,\mathsf{y}_1);\ \mathsf{A}(\mathsf{n}_1)$$
$$\mathsf{hmac}_{\mathsf{x}_\mathsf{m}}(\mathsf{y}_1,\mathcal{A});\ \mathsf{unwrap}(\mathsf{x}_2)$$

$$\begin{array}{llcl} \mathsf{Encrypt:} & \mathsf{h}(\mathsf{x}_1,\mathsf{y}_1),\mathsf{y}_2;\ \mathsf{encrypt}(\mathsf{x}_1) & \rightarrow & \mathsf{enc}(\mathsf{y}_2,\mathsf{y}_1) \\ \mathsf{Decrypt:} & \mathsf{h}(\mathsf{x}_1,\mathsf{y}_1),\mathsf{enc}(\mathsf{y}_2,\mathsf{y}_1);\ \mathsf{decrypt}(\mathsf{x}_1) & \rightarrow & \mathsf{y}_2 \end{array}$$

$P = (\{e,d,ed,w,u,wu\},\rightarrow)$ (where $\rightarrow$ makes the obvious cliques)

Model checking

We use SATMC from the AVISPA project.

Why?

- Can customize sort theory
- Can have protocols with loops
  – recent work by Roberto Carbone to detect fixpoints
- Good performance on previous API experiments

Model checking - 2

A *known key* is a key k such that the intruder knows the plaintext value k and the intruder has a handle $\mathrm{h}(\mathrm{n}, \mathrm{k})$.

Property 1 If an intruder starts with no known keys, he cannot obtain any known keys.

Verified for our API in 0.4 sec

Model checking - 2

A *known key* is a key k such that the intruder knows the plaintext value k and the intruder has a handle $h(n, k)$.

Property 1 If an intruder starts with no known keys, he cannot obtain any known keys.

Verified for our API in 0.4 sec

Property 2 If an intruder starts with a known key k_i with handle $h(n_i, k_i)$, and $ed(n_i)$ is true, then he cannot obtain any further known keys.

Attack

Lost session key attack

Initial knowledge: Handles $\mathsf{h}(\mathsf{n}_1, \mathsf{k}_1)$, $\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2)$, and $\mathsf{h}(\mathsf{n}_i, \mathsf{k}_i)$. Key k_i. Attributes $\mathsf{ed}(\mathsf{n}_1), \mathsf{wu}(\mathsf{n}_2), \mathsf{ed}(\mathsf{n}_i)$.

Trace:

Wrap: (ed)	$\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2)$, $\mathsf{h}(\mathsf{n}_i, \mathsf{k}_i) \rightarrow$
	$\{\mathsf{k}_i\}_{\mathsf{k}_2}$, $\{\mathsf{k}_3\}_{\mathsf{k}_2}$, $\mathsf{hmac}_{\mathsf{k}_3}(\mathsf{k}_i, \mathsf{ed})$
Unwrap: (wu)	$\mathsf{h}(\mathsf{n}_2, \mathsf{k}_2)$, $\{\mathsf{k}_i\}_{\mathsf{k}_2}$, $\{\mathsf{k}_i\}_{\mathsf{k}_2}$,
	$\mathsf{hmac}_{\mathsf{k}_i}(\mathsf{k}_i, \mathsf{wu}) \rightarrow \mathsf{h}(\mathsf{n}_2, \mathsf{k}_i)$
Wrap: (ed)	$\mathsf{h}(\mathsf{n}_2, \mathsf{k}_i)$, $\mathsf{h}(\mathsf{n}_1, \mathsf{k}_1) \rightarrow$
	$\{\mathsf{k}_1\}_{\mathsf{k}_i}$, $\{\mathsf{k}_3\}_{\mathsf{k}_i}$, $\mathsf{hmac}_{\mathsf{k}_3}(\mathsf{k}_1, \mathsf{ed})$
Decrypt:	k_i, $\{\mathsf{k}_1\}_{\mathsf{k}_i} \rightarrow \mathsf{k}_1$

Revised API

$\mathsf{Wrap:}$

$$\mathsf{h}(\mathsf{x}_1, \mathsf{y}_1), \mathsf{h}(\mathsf{x}_2, \mathsf{y}_2);\ \mathsf{wrap}(\mathsf{x}_1), \mathsf{A}(\mathsf{x}_2) \xrightarrow{\mathsf{new}\ \mathsf{m}_\mathsf{k}} \mathsf{enc}(\mathsf{y}_2, \mathsf{y}_1), \mathsf{enc}(\mathsf{m}_\mathsf{k}, \mathsf{y}_1)$$

$$\mathsf{hmac}_{\mathsf{m}_\mathsf{k}}(\mathsf{y}_2, \mathcal{A}, \mathsf{y}_1)$$

$\mathsf{Unwrap:}$

$$\mathsf{h}(\mathsf{x}_2, \mathsf{y}_2), \mathsf{enc}(\mathsf{y}_1, \mathsf{y}_2), \mathsf{enc}(\mathsf{x}_\mathsf{m}, \mathsf{y}_2), \xrightarrow{\mathsf{new}\ \mathsf{n}_1} \mathsf{h}(\mathsf{n}_1, \mathsf{y}_1);\ \mathsf{A}(\mathsf{n}_1)$$

$$\mathsf{hmac}_{\mathsf{x}_\mathsf{m}}(\mathsf{y}_1, \mathcal{A}, \mathsf{y}_2);\ \mathsf{unwrap}(\mathsf{x}_2)$$

Property 2 now verified by SATMC

Can also verify attribute policy is enforced

More Key Management APIs

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

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

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

V. Cortier, G. Keighren, and G. Steel. *Automatic analysis of the security of XOR-based key management schemes*. TACAS 2007.