Formal Analysis of Key Management APIs

Graham Steel

INRIA & LSV, ENS de Cachan

Host machine

Trusted device

Security API

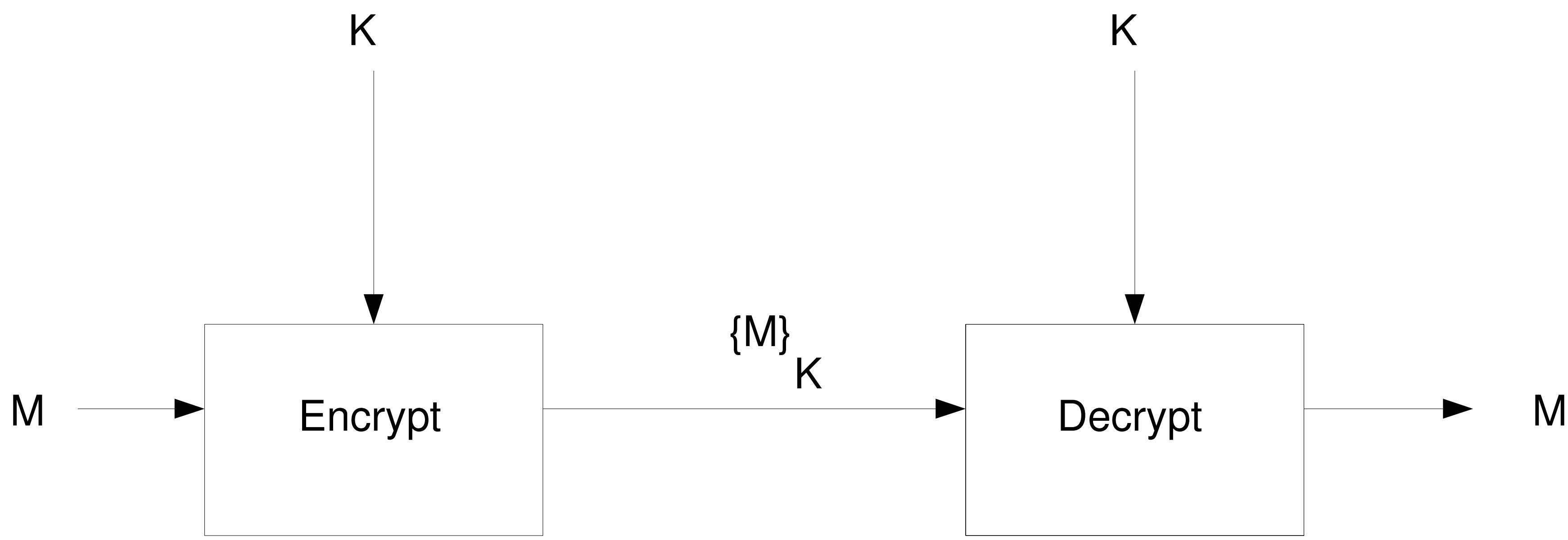
Cryptographic key management

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

Crypto Basics

We consider only symmetric key crypto



Problem is now the security of key K

Model

Signature $\Sigma ::= N, X, F, P$

Plain terms

$$\begin{array}{lcll} t, t_i & := & x & x \in X \\ & | & n & n \in N \\ & | & f(t_1, \ldots, t_n) & f \in F \end{array}$$

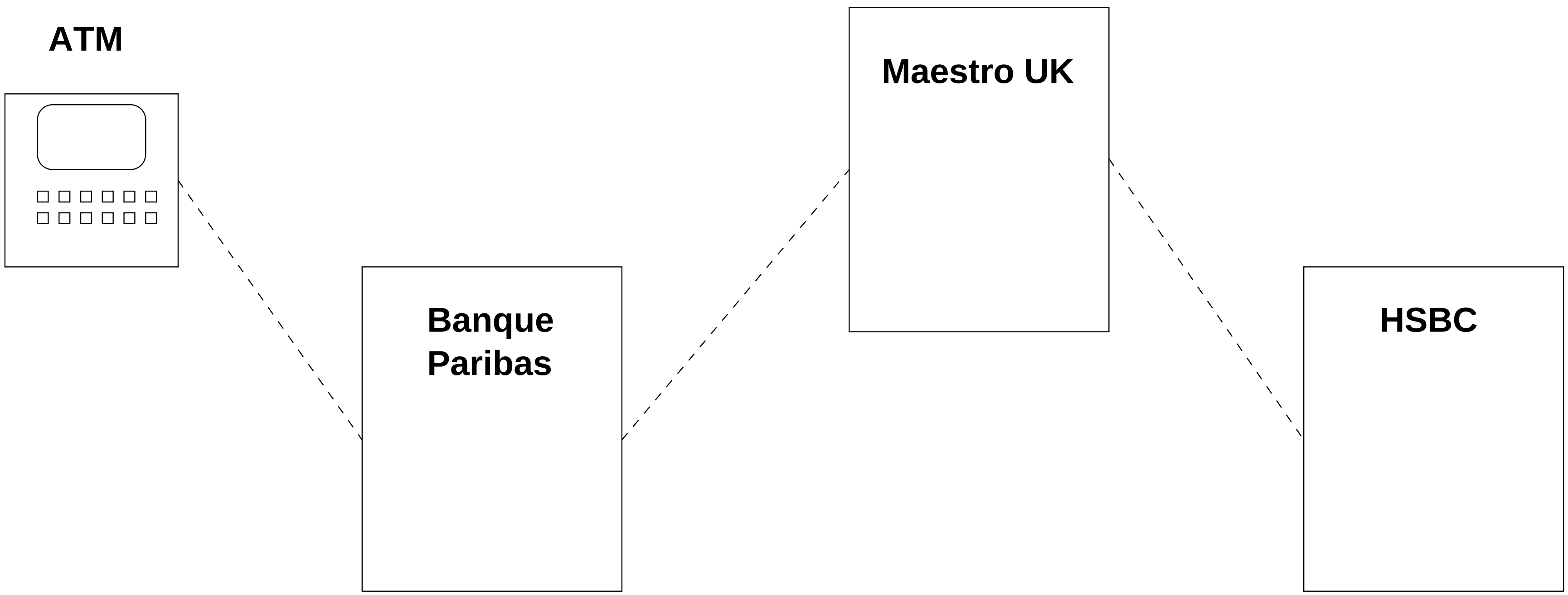
Facts

$$l = \{p(t, b) \mid p \in P, t \in T, b \in \{\top, \bot\}\}$$

Rules

$$T; L \xrightarrow{\mathsf{new}\, \tilde{n}} T'; L'$$

Cash Machine Network



HSMs

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

A Word About Your PIN

IPIN derived by:

Write account number (PAN) as 0000AAAAAAAAAAAA

Encrypt under a PIN Derivation Key (PDK)

$$\{\text{PAN}\}_{\text{PDK}} = \text{IPIN}$$

PIN = IPIN + Offset (modulo 10 each digit)

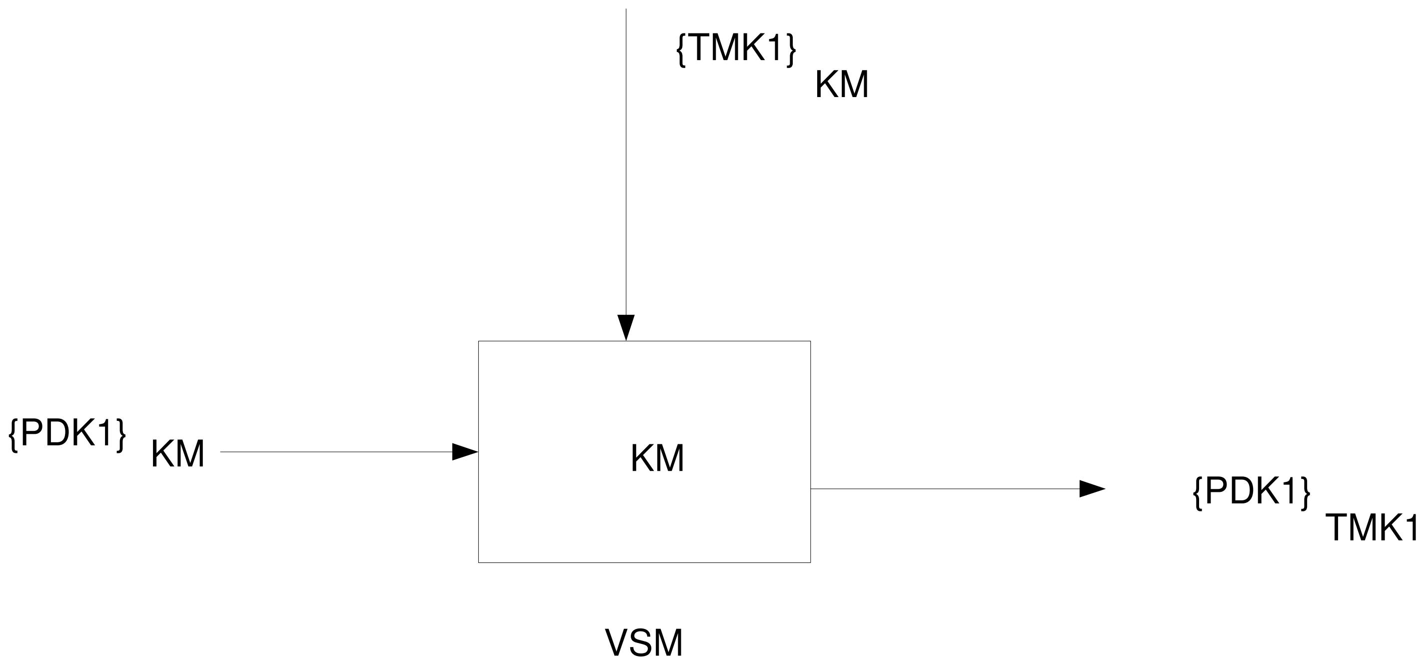
Offset NOT secure!

Master Key Scheme

Host machine	HSM
$\{ TMK1 \}_{KM}$	KM
$\{ PDK1 \}_{KM}$	

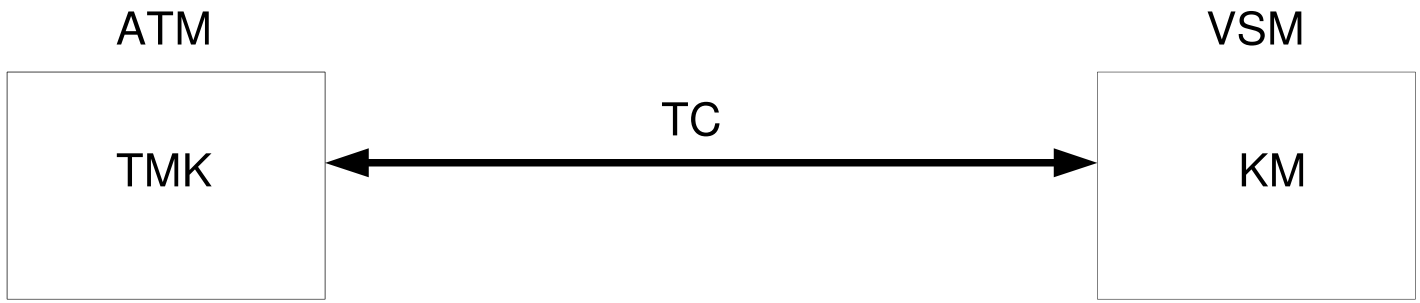
TMK = Terminal Master Key

Example: Send PDK to Terminal



$$\{\mathsf{PDK1}\}_{\mathsf{km}}, \{\mathsf{TMK1}\}_{\mathsf{km}} \rightarrow \{\mathsf{PDK1}\}_{\mathsf{TMK1}}$$

Terminal Comms (TC) Key



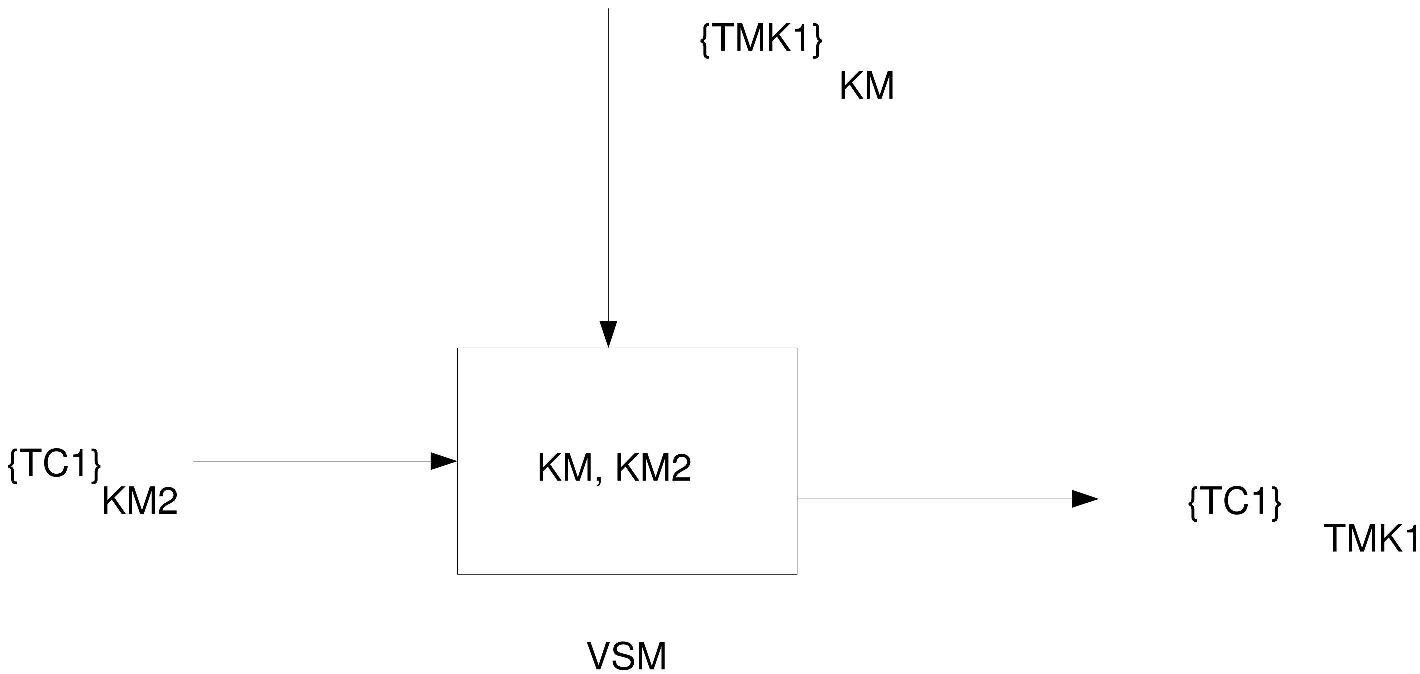
Managing Key Types

Host machine	VSM
$\{ TMK1 \}_{KM}$	KM
$\{ PDK1 \}_{KM}$	
	KM2
$\{ TC1 \}_{KM2}$	

Example: Enter TC key

$$TC \rightarrow \{TC\}_{km2}$$

Example: Send TC to Terminal

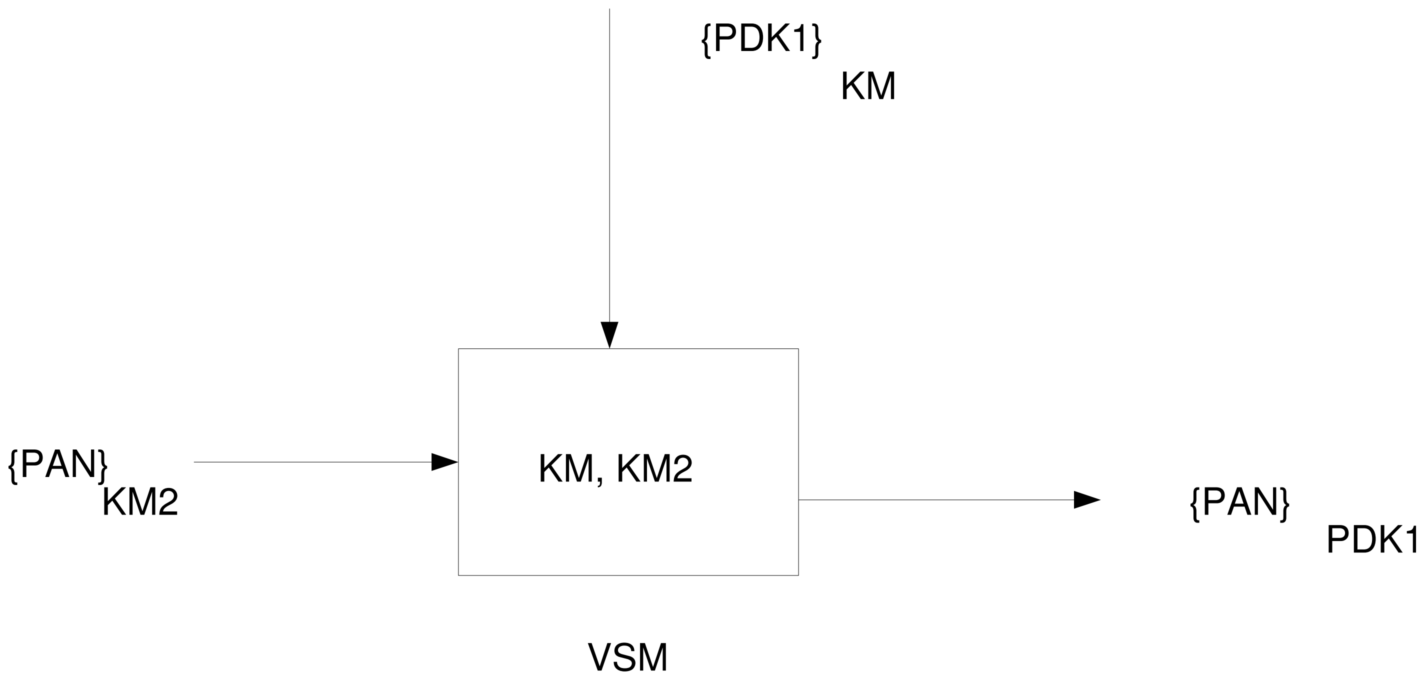


$$\{TC\}_{km2}, \{TMK1\}_{km} \rightarrow \{TC\}_{TMK1}$$

Attack - Step 1

$$\text{PAN} \longrightarrow \{\text{PAN}\}_{\text{km2}}$$

Attack - Step 2



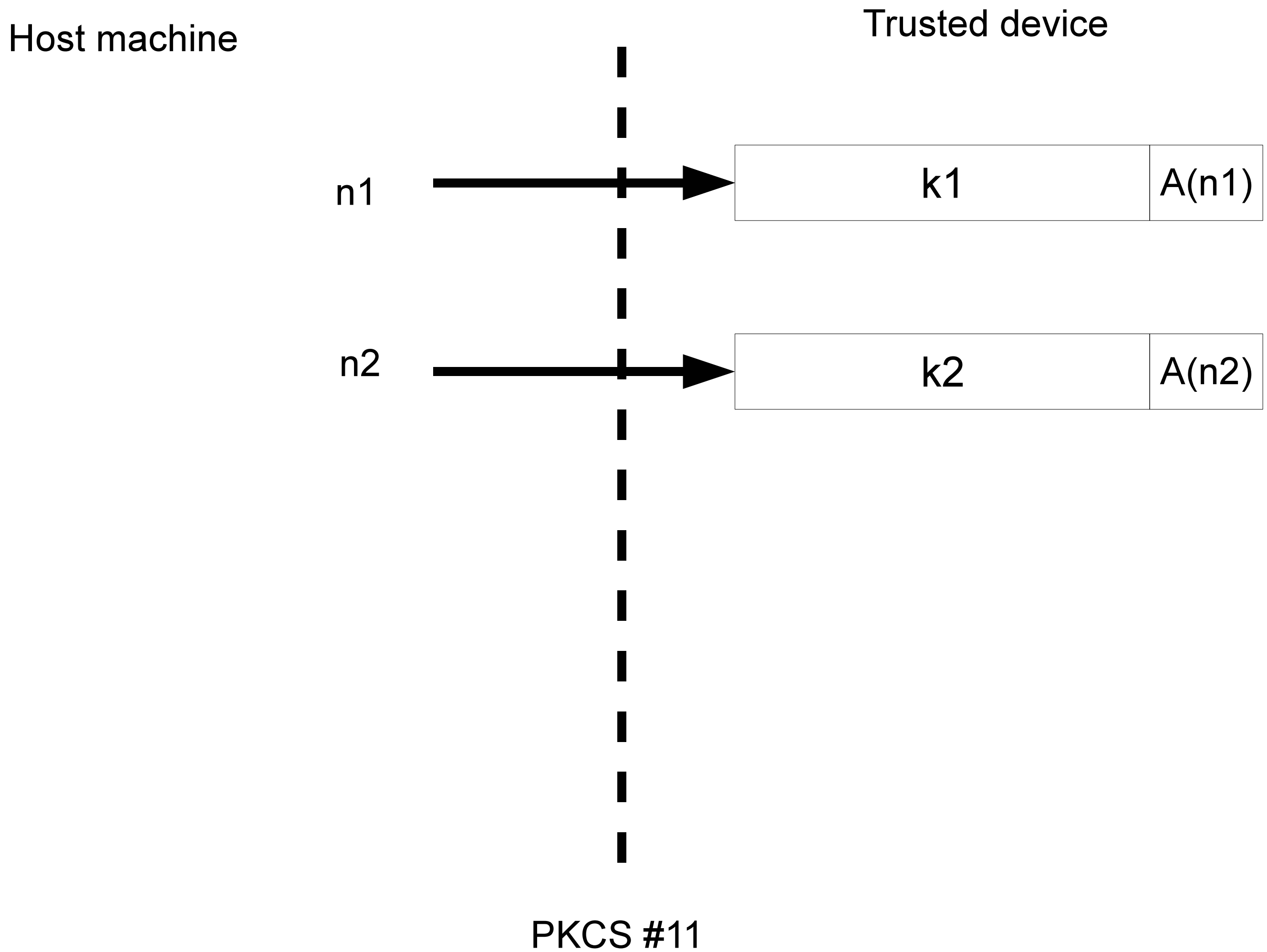
$$\{\text{PAN}\}_{\text{km2}}, \{\text{PDK1}\}_{\text{km}} \rightarrow \{\text{PAN}\}_{\text{PDK1}}$$

VSM - Formal Model

$$
\begin{array}{rcl}
\mathsf{X}, \mathsf{Y} & \longrightarrow & \{\mathsf{X}\}_{\mathsf{Y}} \\
\{\mathsf{X}\}_{\mathsf{Y}}, \mathsf{Y} & \longrightarrow & \mathsf{X} \\
 & \xrightarrow{\mathsf{new\,tmk}} & \{\mathsf{tmk}\}_{\mathsf{km}} \\
\mathsf{TC} & \longrightarrow & \{\mathsf{TC}\}_{\mathsf{km2}} \\
\{\mathsf{PDK}\}_{\mathsf{km}}, \{\mathsf{TMK}\}_{\mathsf{km}} & \longrightarrow & \{\mathsf{PDK}\}_{\mathsf{TMK}} \\
\{\mathsf{TC}\}_{\mathsf{km2}}, \{\mathsf{TMK}\}_{\mathsf{km}} & \longrightarrow & \{\mathsf{TC}\}_{\mathsf{TMK}}
\end{array}
$$

$I = \{\{\mathsf{pdk}\}_{\mathsf{km}}, \mathsf{pan}\}$, +8 more rules `SWV237,238` (`www.tptp.org`)

CASC at FLoC '10: 9/17 provers can find the attack, only E can find model


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

PKCS#11

Ubiquitous in authentication tokens, smartcards,...

RSA PKCS#11 is specified in a 400 page document

We consider here a core fragment of key management operations

Not included: signing, verification, certificate management, etc.

> $\mathsf{h}(\mathsf{n1}, \mathsf{k1})$ - a handle $\mathsf{n1}$ for key $\mathsf{k1}$ (h is a *private symbol*)
>
> $\mathsf{a1}(\mathsf{n1})$ - setting of attribute $\mathsf{a1}$ for handle $\mathsf{n1}$

We consider attributes:

$\mathsf{encrypt}(\mathsf{n}), \mathsf{decrypt}(\mathsf{n}), \mathsf{sensitive}(\mathsf{n})$
$\mathsf{extract}(\mathsf{n}), \mathsf{wrap}(\mathsf{n}), \mathsf{unwrap}(\mathsf{n})$

Key Management - 1

$$\mathsf{KeyGenerate}:$$

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

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

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

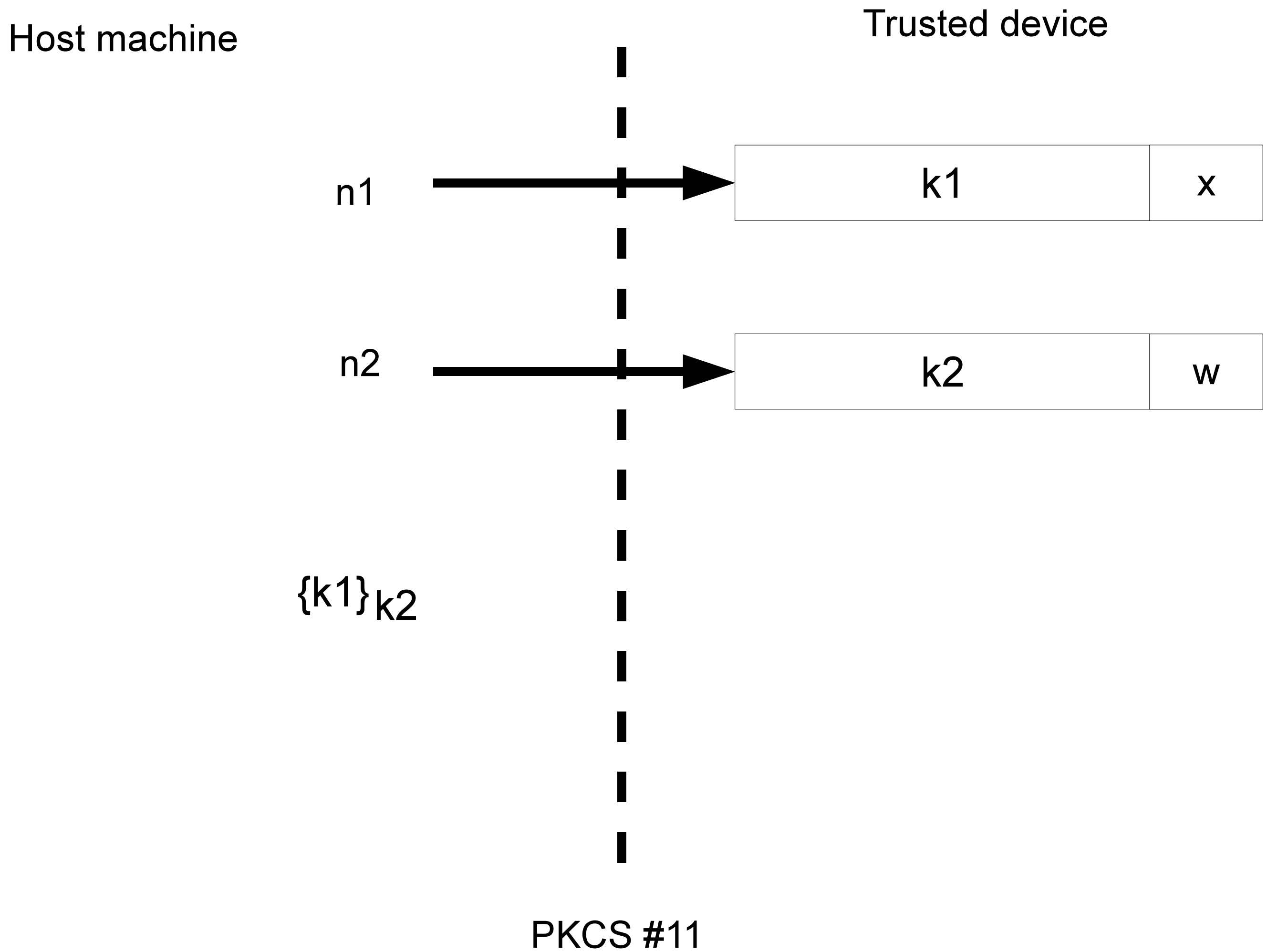
Wrap :

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

Unwrap :

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

where $L =$
$\neg\mathsf{wrap}(n_1), \neg\mathsf{unwrap}(n_1), \neg\mathsf{encrypt}(n_1), \neg\mathsf{decrypt}(n_1), \neg\mathsf{sensitive}(n_1).$


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

Key Usage

Encrypt :

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

Decrypt :

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

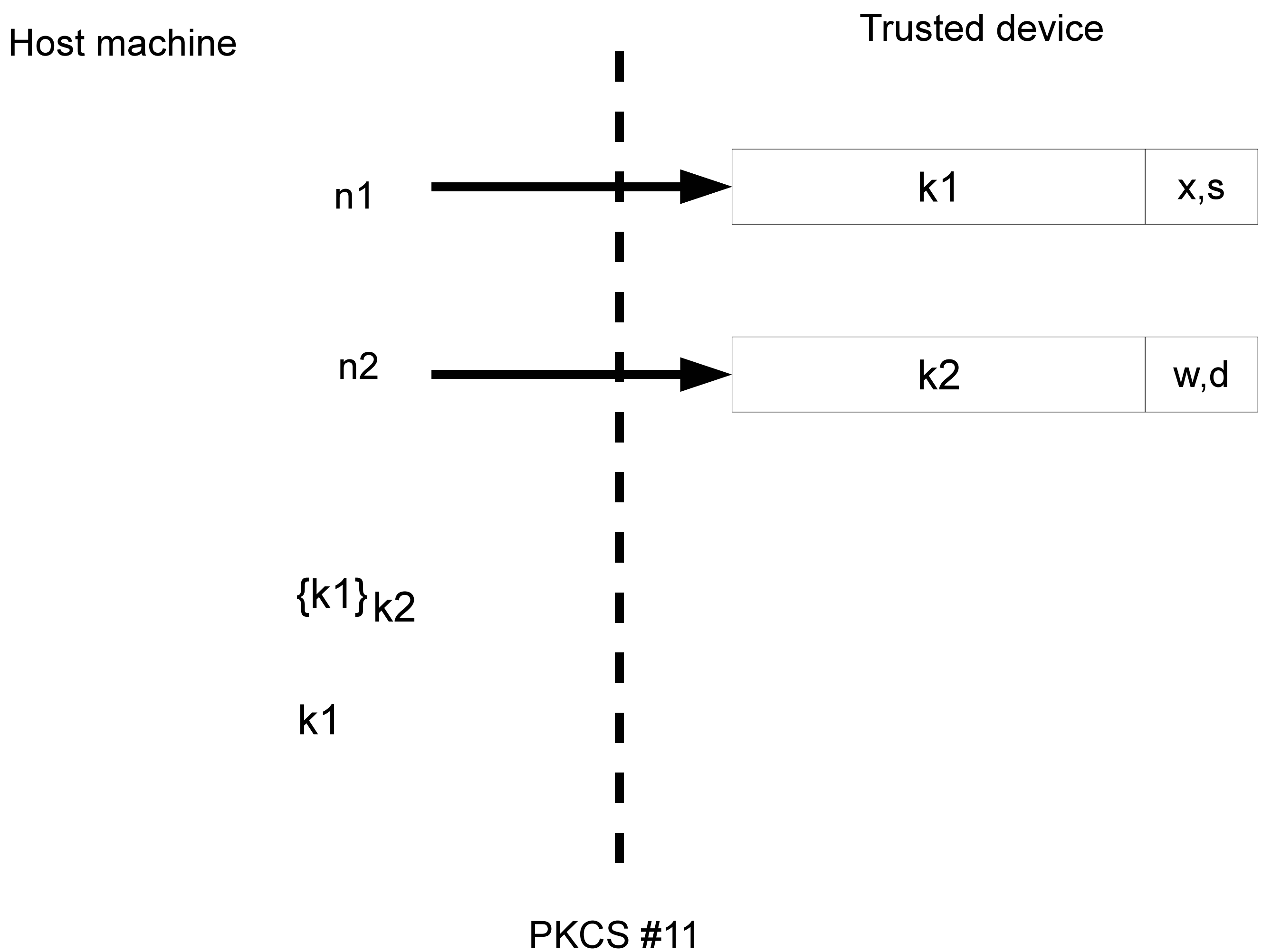
Key Separation Attack (Clulow, 2003)

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..

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

~~Unset_Wrap~~

~~Unset_Decrypt~~

Another Attack

Intruder knows: $h(n_1, k_1)$, $h(n_2, k_2)$, k_3

State: $sensitive(n_1), extract(n_1)$, $unwrap(n_2)$, $encrypt(n_2)$

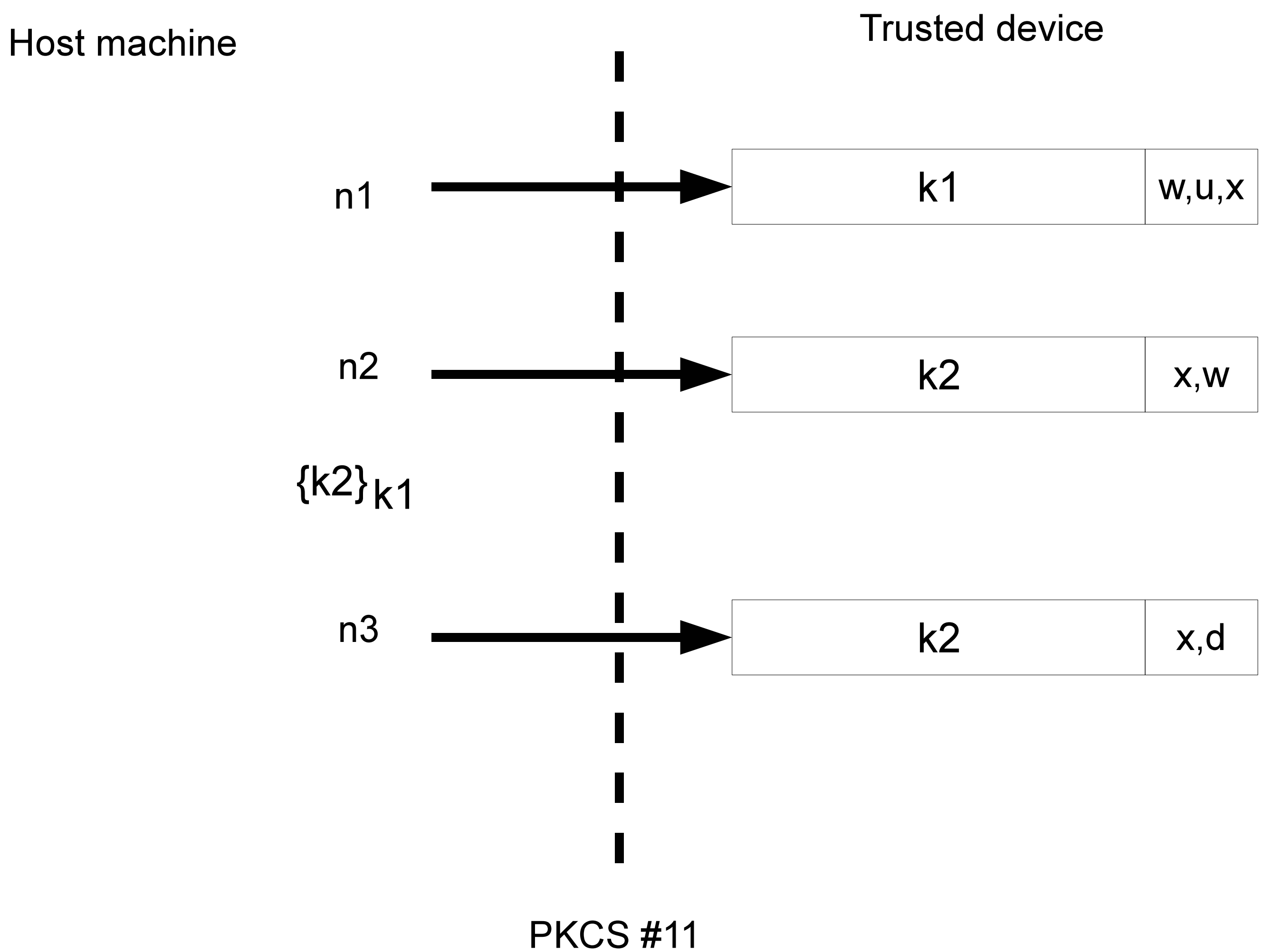
SEncrypt:	$h(n_2, k_2)$, k_3	$\longrightarrow$	$\{k_3\}_{k_2}$
Unwrap:	$h(n_2, k_2)$, $\{k_3\}_{k_2}$	$\xrightarrow{\text{new } n_3}$	$h(n_3, k_3)$
Set_wrap:	$h(n_3, k_3)$	$\longrightarrow$	$wrap(n_3)$
Wrap:	$h(n_3, k_3)$, $h(n_1, k_1)$	$\longrightarrow$	$\{k_1\}_{k_3}$
Intruder:	$\{k_1\}_{k_3}$, k_3	$\longrightarrow$	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: $\mathsf{sensitive}(\mathsf{n}_1), \mathsf{extract}(\mathsf{n}_1)$, $\mathsf{extract}(\mathsf{n}_2)$

$$
\begin{array}{lrcl}
\text{Set\_wrap:} & h(n_2, k_2) & \rightarrow & ;\mathsf{wrap}(n_2) \\
\text{Set\_wrap:} & h(n_1, k_1) & \rightarrow & ;\mathsf{wrap}(n_1) \\
\text{Wrap:} & h(n_1, k_1), h(n_2, k_2) & \rightarrow & \{k_2\}_{k_1} \\
\text{Set\_unwrap:} & h(n_1, k_1) & \rightarrow & ;\mathsf{unwrap}(n_1) \\
\text{Unwrap:} & h(n_1, k_1), \{k_2\}_{k_1} & \xrightarrow{\mathsf{new}\, n_3} & h(n_3, k_2) \\
\text{Wrap:} & h(n_2, k_2), h(n_1, k_1) & \rightarrow & \{k_1\}_{k_2} \\
\text{Set\_decrypt:} & h(n_3, k_2) & \rightarrow & ;\mathsf{decrypt}(n_3) \\
\text{Decrypt:} & h(n_3, k_2), \{k_1\}_{k_2} & \rightarrow & k_1
\end{array}
$$


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

Modes

$$
\begin{aligned}
\mathsf{h} &: \mathsf{Nonce} \times \mathsf{Key} \rightarrow \mathsf{Handle} \\
\mathsf{senc} &: \mathsf{Key} \times \mathsf{Key} \rightarrow \mathsf{Cipher} \\
\mathsf{aenc} &: \mathsf{Key} \times \mathsf{Key} \rightarrow \mathsf{Cipher} \\
\mathsf{pub} &: \mathsf{Seed} \rightarrow \mathsf{Key} \\
\mathsf{priv} &: \mathsf{Seed} \rightarrow \mathsf{Key} \\
\mathsf{a} &: \mathsf{Nonce} \rightarrow \mathsf{Attribute} \quad \text{for all } \mathsf{a} \in \mathcal{A} \\
\mathsf{x}_1, \mathsf{x}_2, \mathsf{n}_1, \mathsf{n}_2 &: \mathsf{Nonce} \\
\mathsf{y}_1, \mathsf{y}_2, \mathsf{k}_1, \mathsf{k}_2 &: \mathsf{Key} \\
\mathsf{z}, \mathsf{s} &: \mathsf{Seed}
\end{aligned}
$$

See Delaune, Kremer & S., *Formal Analysis of PKCS#11*, CSF '08

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’

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

$$\text{Wrap}:$$

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

$$\text{Unwrap}:$$

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

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

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

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

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

$$\vdots \qquad\qquad \vdots$$

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

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 :

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

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

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

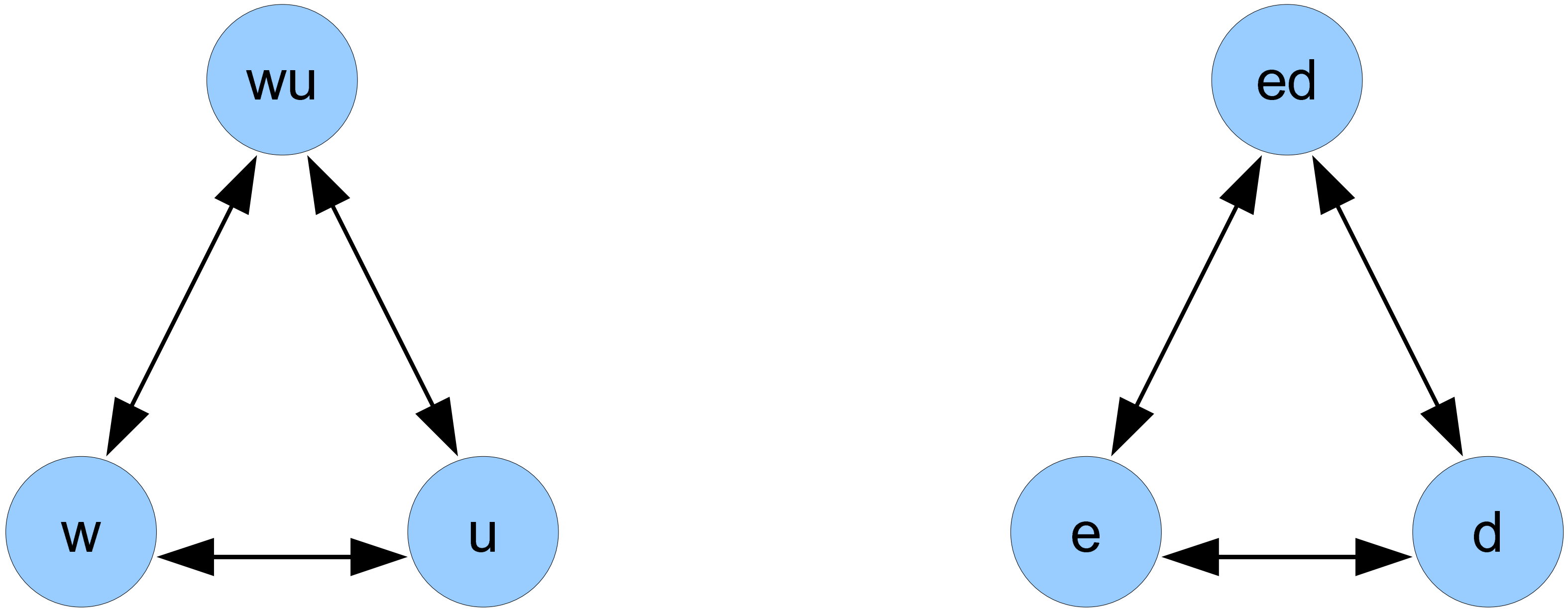
$$\text{Set\_Attribute\_Value}: \quad h(x_1, y_1);\ A_1(x_1) \quad \rightarrow \quad A_2(x_1)$$

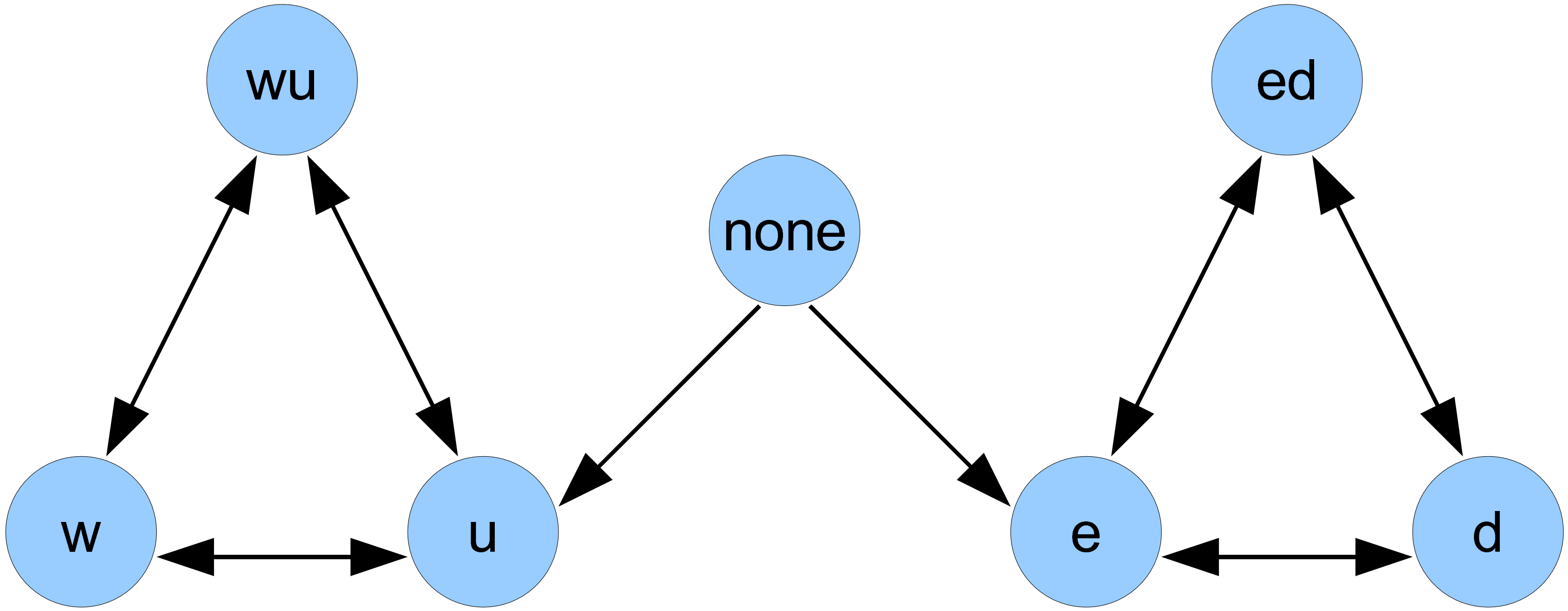
Attribute Policy

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.

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

- 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

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

Wrap :

$$h(x_1, y_1), h(x_2, y_2);\ \mathsf{wrap}(x_1), A(x_2) \quad \xrightarrow{\text{new } m_k} \quad \{y_2\}_{y_1}, \{m_k\}_{y_1}$$

$$\mathsf{hmac}_{m_k}(y_2, \mathcal{A})$$

Unwrap :

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

$$\mathsf{hmac}_{x_m}(y_1, \mathcal{A});\ \mathsf{unwrap}(x_2)$$

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

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

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

Model checking

Use SATMC (U. di Genova) to check formal model for attack

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 revised 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

$$\text{Wrap}:$$

$$h(x_1, y_1), h(x_2, y_2);\ wrap(x_1), A(x_2) \xrightarrow{\text{new } m_k} \{y_2\}_{y_1}, \{m_k\}_{y_1}$$

$$hmac_{m_k}(y_2, \mathcal{A}, y_1)$$

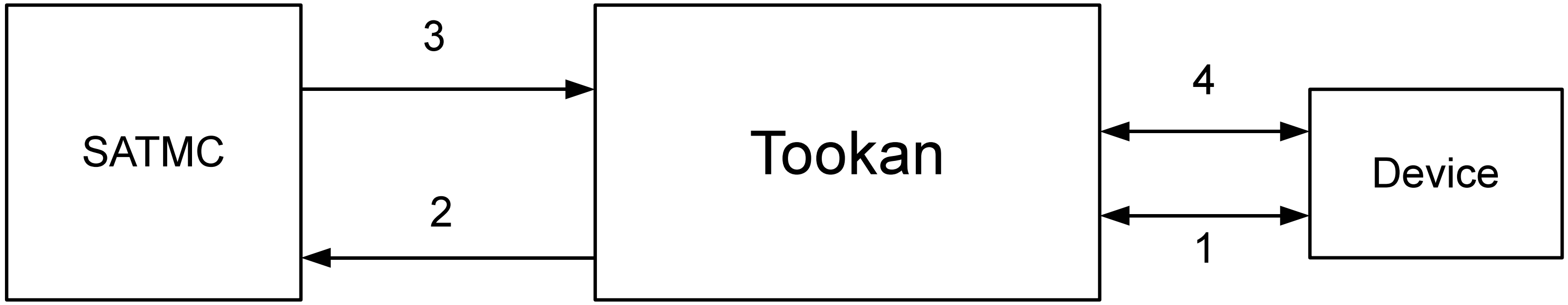
$$\text{Unwrap}:$$

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

$$hmac_{x_m}(y_1, \mathcal{A}, y_2);\ unwrap(x_2)$$

Property 2 now verified by SATMC

Can also verify attribute policy is enforced



See Bortolozzo, Centenaro, Focardi & S., *Attacking and Fixing PKCS#11 Security Tokens*, to appear at ACM CCS 2010.

	Device		Supported Functionality						Attacks found					
	Brand	Model	sym	asym	cobj	chan	w	ws	a1	a2	a3	a4	a5	mc
	XXXX	XXXX	✓	✓	✓	✓	✓	✓		✓	✓	✓		a3
	XXXX	XXXX	✓	✓	✓	✓	✓	✓	✓	✓				a1
	XXXX	XXXX	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	a3
	XXXX	XXXX		✓	✓									
	XXXX	XXXX		✓		✓								
USB	XXXX	XXXX	✓	✓	✓	✓	✓	✓		✓	✓	✓		a3
	XXXX	XXXX	✓	✓	✓		✓							
	XXXX	XXXX	✓	✓		✓								
	XXXX	XXXX	✓	✓	✓									
	XXXX	XXXX	✓	✓	✓	✓	✓	✓	✓	✓				a1
	XXXX	XXXX	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	a3
Card	XXXX	XXXX	✓	✓	✓		✓	✓		✓				a2
	XXXX	XXXX		✓		✓								
	XXXX	XXXX	✓	✓	✓									
	XXXX	XXXX	✓	✓	✓	✓								
	XXXX	XXXX	✓	✓	✓		✓					✓		a4
Soft	XXXX	XXXX	✓	✓		✓	✓	✓	✓	✓		✓		a1
	XXXX	XXXX	✓	✓	✓	✓	✓	✓	✓	✓		✓		a1

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

– possibility of unifying these proposals?

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

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, `http://www.lsv.ens-cachan.fr/~steel/asa4`

Interested? Internships + postdocs available, get in touch