Formal Analysis of Key Integrity in PKCS#11*

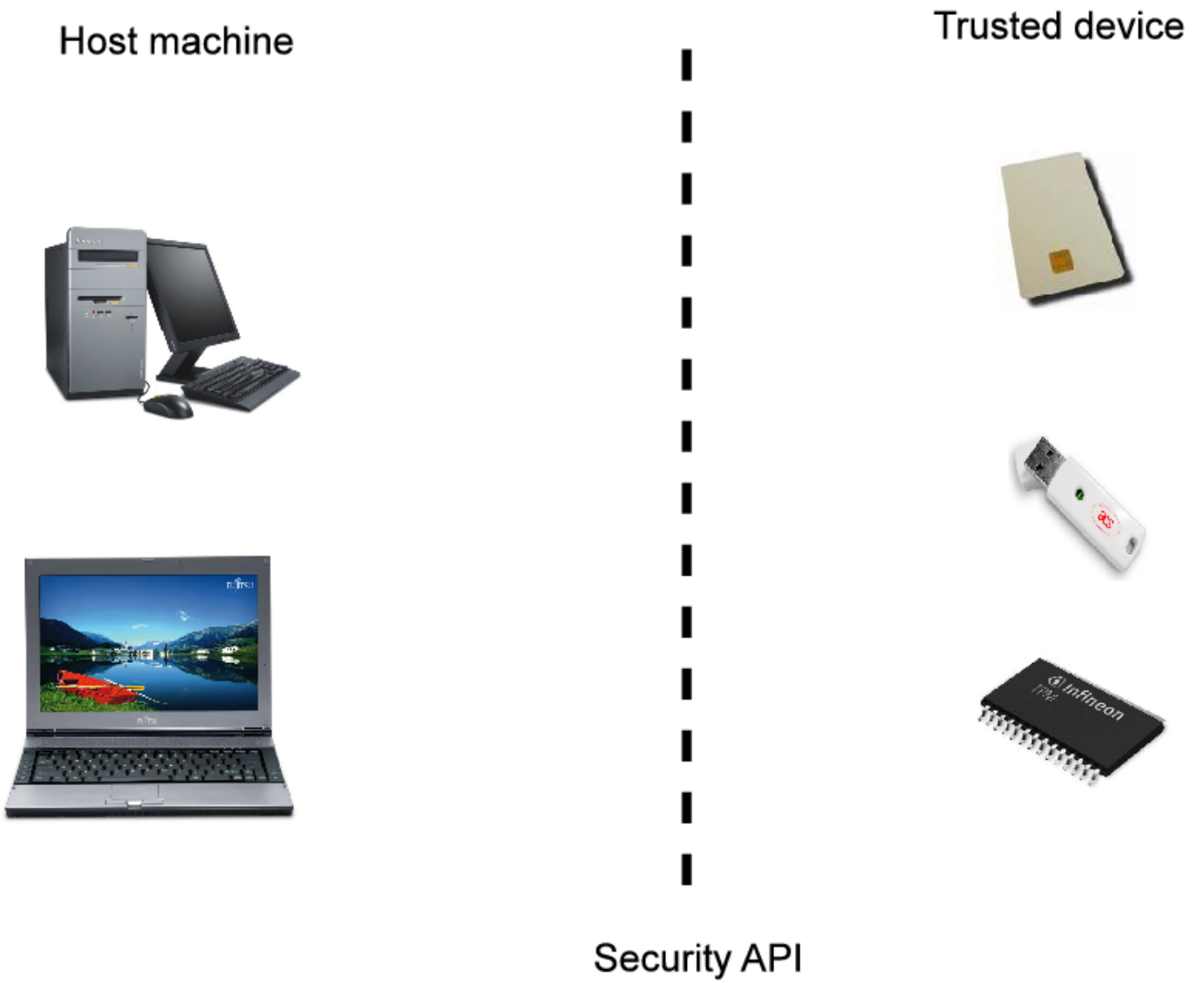
Andrea Falcone[1] Riccardo Focardi[1]

[1]Università Ca' Foscari di Venezia, Italy
focardi@dsi.unive.it

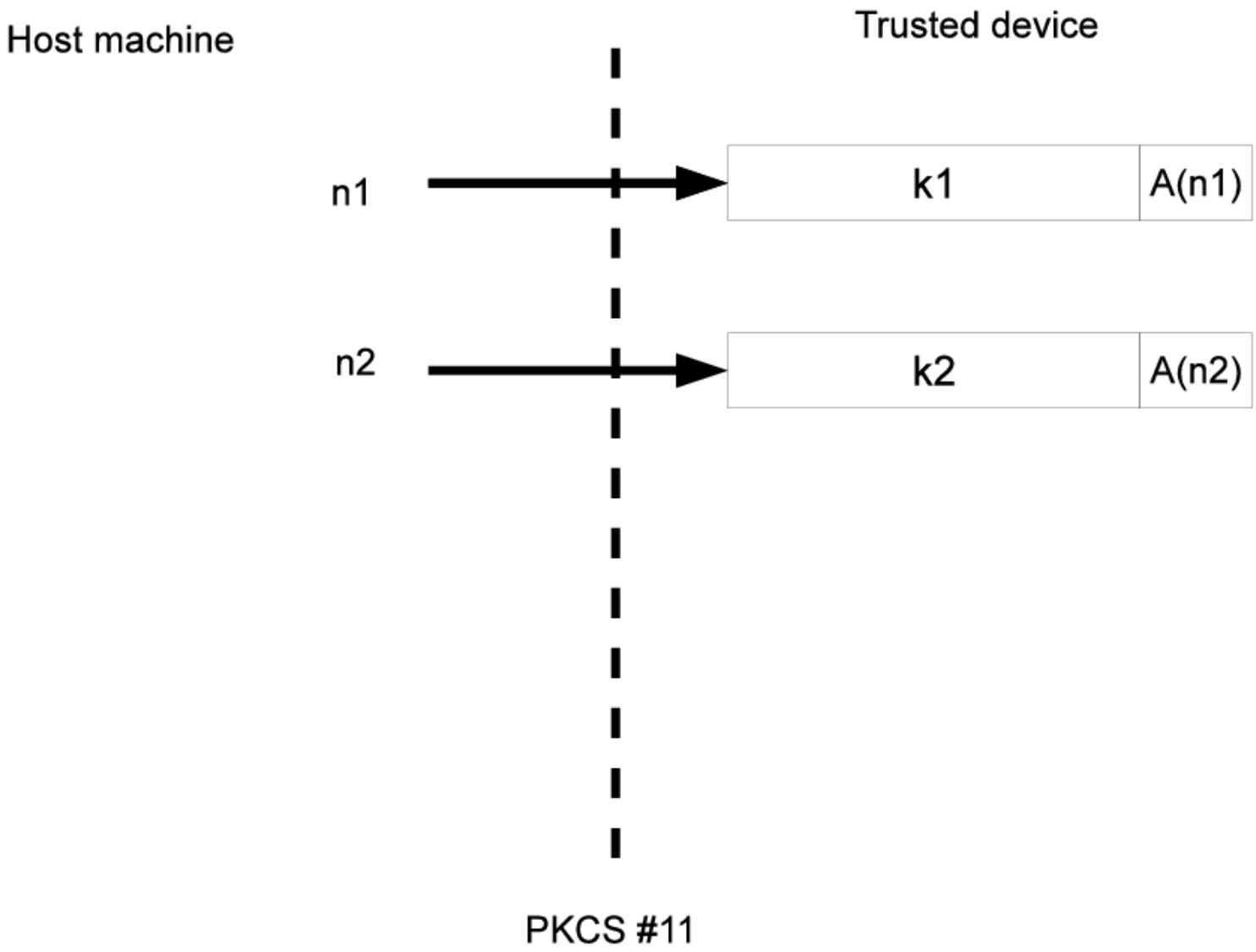
ARSPA-WITS'10
Paphos, Cyprus March 27-28, 2010

*Work partially supported by:
Miur'07 Project SOFT: "*Security Oriented Formal Techniques*"

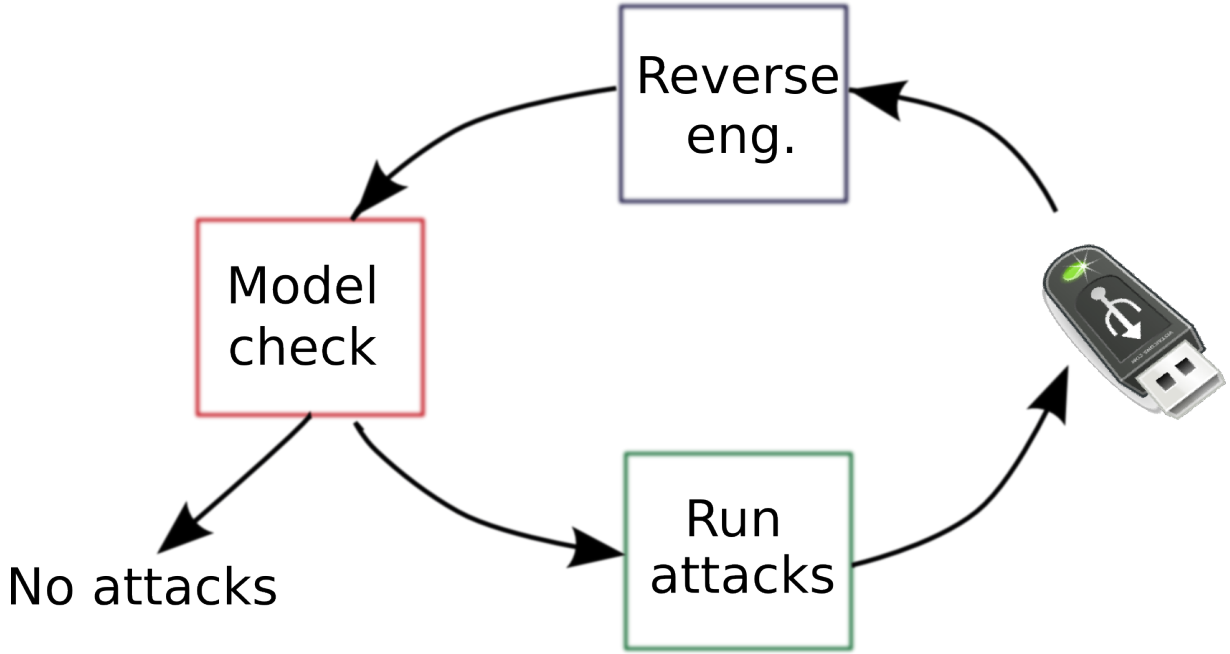
Security APIs



PKCS#11 API [RSA Security]

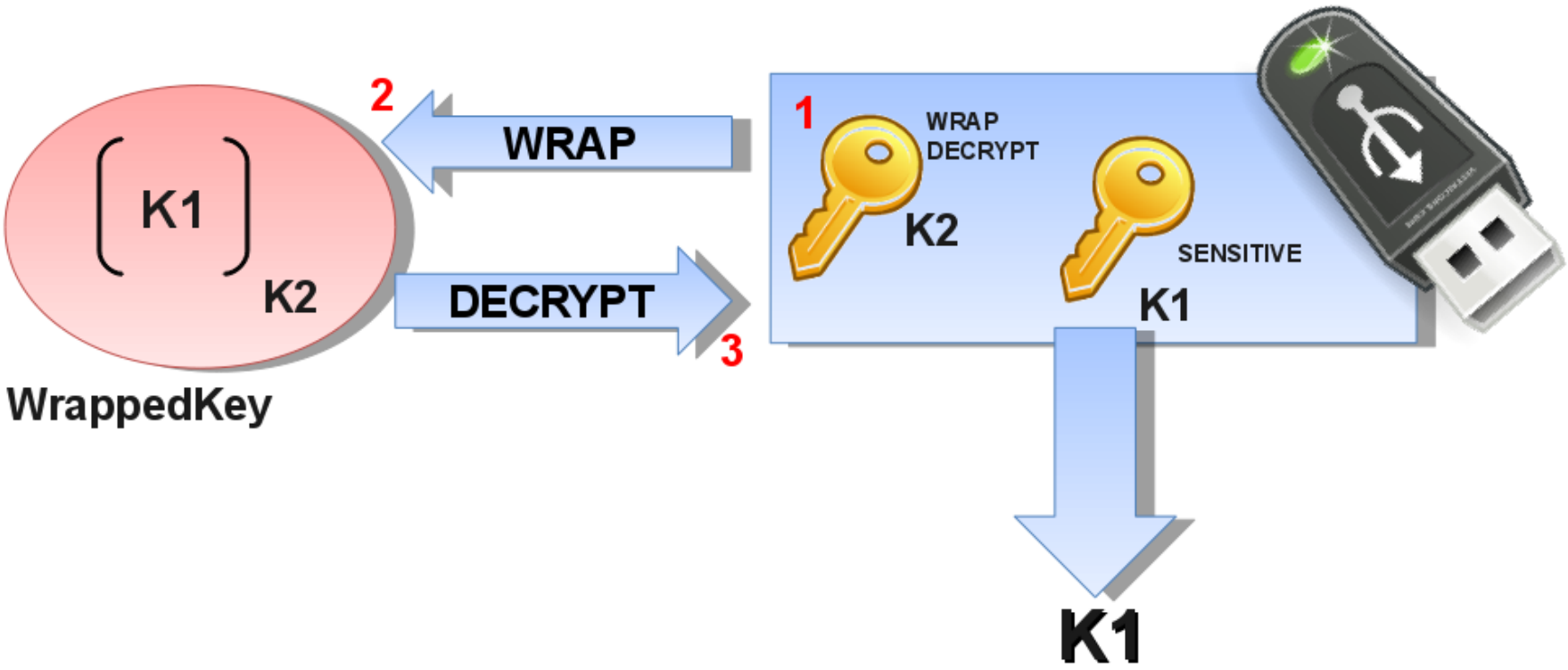


Analysis of real PKCS#11 devices: the 'big picture'



with
G. Steel (INRIA), M. Centenaro, M. Bortolozzo, C. Bozzato (Univ. Venice)

The Wrap-Decrypt attack [Clulow CHES'03]



ATTACK SEQUENCE:

1. n2 = KEY_GENERATE(WD)
2. WrappedKey = WRAP (n2, n1)
3. K1 = DECRYPT (WrappedKey, n2)

Attack scenario

1. The token is used on a public access point
2. the attacker steals the PIN and extracts some sensitive keys
3. any subsequent usage of such token keys is insecure

> *"... the PIN may be passed through the operating system. This can make it easy for a rogue application on the operating system to obtain the PIN ... Rogue applications and devices may also change the commands sent to the cryptographic device to obtain services other than what the application requested."*
> [RSA Security]

- PKCS#11 tokens should not be violated even when used on untrusted hosts

Formal analysis of PKCS#11 [Delaune, Kremer, Steel '08]

- Terms representing keys, ciphertexts, handles

$$k, \mathit{senc}(d, k), h(n, k)$$

- Rules $T; L \xrightarrow{\text{new } \tilde{n}} T'; L'$ representing API calls

$$h(x_1, y_1), y_2;\ \mathit{encrypt}(x_1) \rightarrow \mathit{senc}(y_2, y_1)$$

- Transitions $(S, V) \rightsquigarrow (S', V')$ representing API invocation

$$\langle\ \{h(n,k), d\}; \mathit{encrypt}(n)\ \rangle \rightsquigarrow \langle\ \{h(n,k), d, \mathit{senc}(d,k)\}; \mathit{encrypt}(n)\ \rangle$$

Wrap-Decrypt attack, formally

- Rules for key generation, wrap, decrypt:

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

$$h(x_1,y_1), h(x_2,y_2); wrap(x_1), extract(x_2) \longrightarrow senc(y_2,y_1)$$

$$h(x_1,y_1), senc(y_2,y_1); decrypt(x_1) \longrightarrow y_2$$

- We start from state $\langle \{h(n_1,k_1)\}, sensitive(n_1), extract(n_1) \rangle$
  - $\rightsquigarrow \langle \{h(n_1,k_1), h(n_2,k_2)\}, sensitive(n_1), extract(n_1), wrap(n_2), decrypt(n_2) \rangle$
  - $\rightsquigarrow \langle \{h(n_1,k_1), h(n_2,k_2), senc(k_1,k_2)\}, sensitive(n_1), extract(n_1), wrap(n_2), decrypt(n_2) \rangle$
  - $\rightsquigarrow \langle \{h(n_1,k_1), h(n_2,k_2), senc(k_1,k_2), k_1\}, sensitive(n_1), extract(n_1), wrap(n_2), decrypt(n_2) \rangle$

Key Integrity

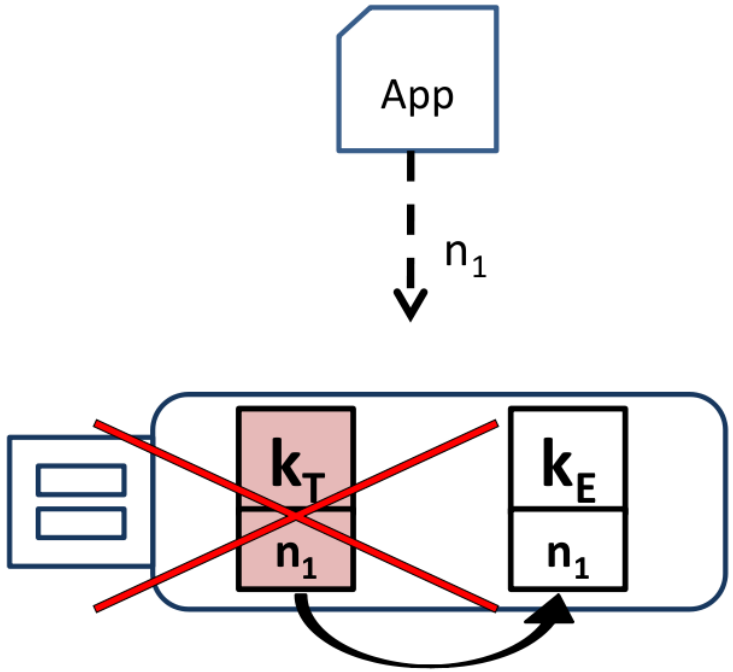
1. The token is used on a public access point
2. the attacker steals the PIN and replaces some sensitive key k
3. k might be subsequently used to:
   - encrypt sensitive data
   - wrap sensitive keys
   - sign secret data (attacker gets credit)
   - check signatures (impersonation)

- ... as critical as key confidentiality, not much discussed in PKCS#11:

"... CKA_CHECK_VALUE ... like a fingerprint, or checksum of the key ... intended to be used to cross-check symmetric keys against other systems where the same key is shared, and as a validity check after manual key entry or restore from backup. ... the attribute is optional"

Breaking key integrity

- Keys have *labels*
  - referred to by application
  - can be set, e.g., when a key is generated
- the attacker deletes user's key with label n_1
- then set n_1 to his own key
- subsequent accesses to n_1 will refer to attacker's key
- tested on real devices



New attacker capabilities

1. *overwriting* of keys in the device;
2. *interception* of messages sent on the network by the regular user;
3. *disconnection* from the system, interrupting the session with the device.

We thus model

- key integrity attacks
- scenarios where the attacker has a temporary access to the token

Extending the model

- New rules for overwriting keys.

$$h(x_1, y_2), senc(y_1, y_2); unwrap(x_1) \xrightarrow{\text{new } n} h(n, y_1); \mathcal{A}$$

has now the counterpart:

$$h(x_1, y_2), senc(y_1, y_2); unwrap(x_1) \xrightarrow{\text{used } n} h(n, y_1); \mathcal{A}$$

Example

i	$h(n_1, k_1), senc(k_3, k_2), h(n_2, k_2)$
i+1	$h(n_1, \mathbf{k_3}), senc(k_3, k_2), h(n_2, k_2)$

- separated knowledge and explicit message interception
- when disconnected, the only possible operations are Dolev-Yao:

$$\begin{aligned} x, y &\longrightarrow senc(x, y) \\ senc(x, y), y &\longrightarrow x \\ &\dots \end{aligned}$$

A complete key integrity attack

step	transition	σ	user knowledge	attacker knowledge
0	-	-	$d, h(t, k_t), h(i, k_i)$	$h(t, k_t), h(i, k_i), k_e$
1	encrypt	E	$d, h(t, k_t), h(i, k_i)$	$h(t, k_t), h(i, k_i), k_e,$ $\mathbf{senc}(\mathbf{k_e}, \mathbf{k_i})$
2	overwrite	E	$d, h(t, \mathbf{k_e}), h(i, k_i)$	$h(t, \mathbf{k_e}), h(i, k_i), k_e,$ $senc(k_e k_i)$
3	disconnect	-	$d, h(t, k_e), h(i, k_i)$	$k_e, senc(k_e k_i)$
4	encryption	T	$d, h(t, k_e), h(i, k_i),$ $\mathbf{senc}(\mathbf{d}, \mathbf{k_e})$	$k_e, senc(k_e k_i)$
5	Send	-	$d, h(t, k_e), h(i, k_i),$ $senc(d, k_e)$	$k_e, senc(k_e k_i),$ $\mathbf{senc}(\mathbf{d}, \mathbf{k_e})$
6	decryption (disconn.)	E	$d, h(t, k_e), h(i, k_i),$ $senc(d, k_e)$	$k_e, senc(k_e k_i),$ $senc(d, k_e), \mathbf{d}$

A simple fix

- The attribute *trusted* can only be set by the Security Officer
- IDEA: check that a key has *trusted* set before using it
- does not prevent overwriting but usage of overwritten keys

st.	transition	σ	user knowledge	attacker knowledge	$tr(t)$
0	-	-	$d, h(t, k_t), h(i, k_i)$	$h(t, k_t), h(i, k_i), k_e$	*true*
1	encryption	E	$d, h(t, k_t), h(i, k_i)$	$h(t, k_t), h(i, k_i), k_e,$ $\mathbf{senc}(\mathbf{k_e}, \mathbf{k_i})$	*true*
2	unwrap	E	$d, h(t, \mathbf{k_e}), h(i, k_i)$	$h(t, \mathbf{k_e}), h(i, k_i), k_e,$ $senc(k_e k_i)$	**false**
3	disconnect		$d, h(t, k_e), h(i, k_i)$	$k_e, senc(k_e k_i)$	*false*
4	encryption (**STOP**)	T	-	-	-

Conclusion

- PKCS#11 is irritatingly liberal, especially wrt key integrity
- we have found practical ways to overwrite token keys
- extended the DKS model to treat integrity and off-line attacks
- proposed a fix based on *trusted* keys
- NEW: extended the model checker of DKS (thanks to Graham)

Future work

- Investigate alternative, less restrictive, fixes
  - MAC-based
  - non-deletable keys
  - ...

References

Clulow, J.
On the security of PKCS#11.
In Proceedings of CHES'03.

Delaune, S. , Kremer, S., Steel, G.
Formal analysis of PKCS#11.
In Proceedings of CSF'08, June 2008.

RSA Security Inc.
PKCS #11 v.2.20: Cryptographic Token Interface Standard
June 2004

G. Steel,
Experiments: Key Integrity in PKCS#11
http://www.lsv.ens-cachan.fr/~steel/pkcs11/replacement.php

Fragment of model-checking output

Step 0: [sc_step_generate_sym_Key_ttttftt_1(...)]
Step 1: [sc_encrypt_sym_sym_1(...)]
Step 2: [sc_step_unwrap_key_sym_sym_ttttttt_1(...)]
Step 3: [sc_disconnect_1(...)]
Step 4: [sc_disconnected_send_1(...)]
Step 5: [decrypt_symmetric_key_1(...)]

Attacks Found: true
Stop Condition Reached: false
Formula statistics:
  Graph Construction Time: 22.54
  Graph Leveled Off: 5
...
Total Time: 39.92