

Analysis of Security APIs by Typing

Graham Steel

ongoing work with Riccardo Focardi, Matteo Centenaro,
Gavin Keighren and David Aspinall

INRIA & LSV, ENS de Cachan



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



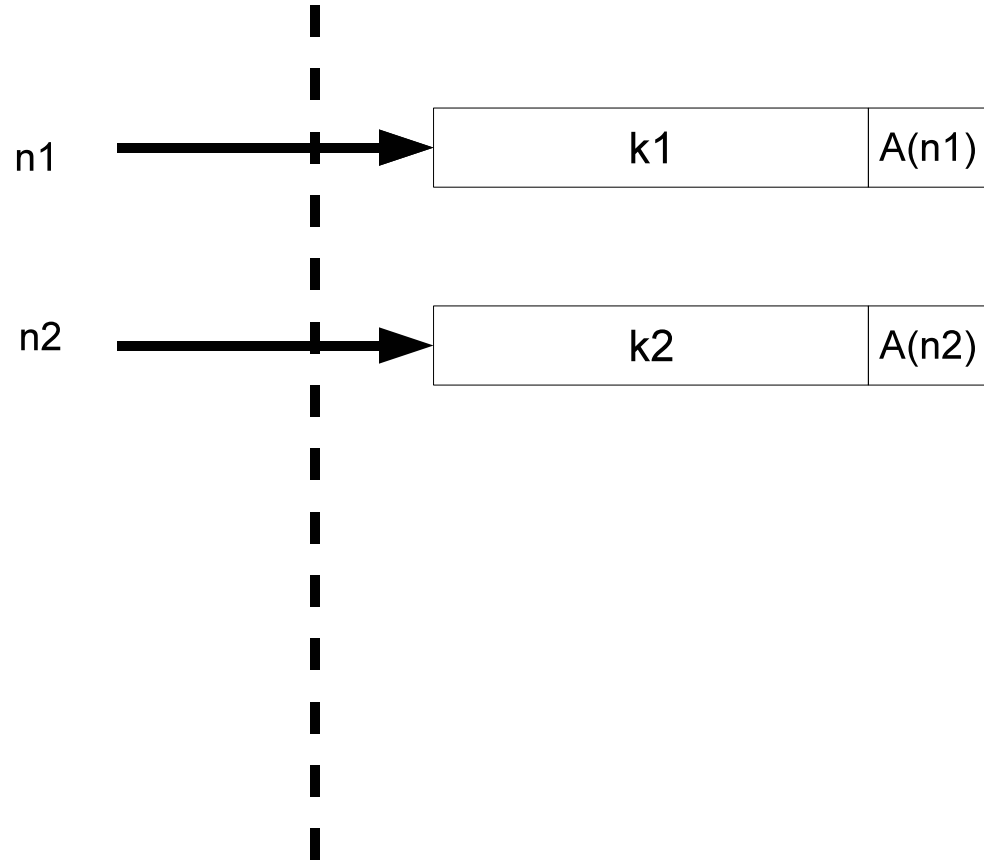
Trusted device



Security API

Host machine

Trusted device



PKCS #11

Host machine

Trusted device

n1



k1

x

n2



k2

w

$\{k1\}_{k2}$

PKCS #11

Key Separation Attack (Clulow, 2003)

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

State: $\text{wrap}(n_2)$, $\text{decrypt}(n_2)$, $\text{sensitive}(n_1)$, $\text{extract}(n_1)$

Wrap: $h(n_2, k_2), h(n_1, k_1) \rightarrow \{k_1\}_{k_2}$

Decrypt: $h(n_2, k_2), \{k_1\}_{k_2} \rightarrow k_1$

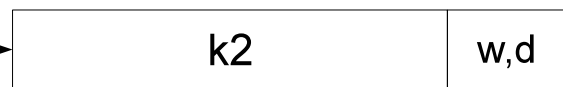
Host machine

Trusted device

n1



n2



{k1}k2

k1



PKCS #11

Verifying APIs

- We can propose improved versions of this PKCS#11 fragment and verify them in DY model using a model checker (Fröschle and Steel, WITS '09).

Verifying APIs

- We can propose improved versions of this PKCS#11 fragment and verify them in DY model using a model checker (Fröschle and Steel, WITS '09).
- But:
 - What properties are we verifying?
 - We needed powerful abstractions, but still are only verifying tiny fragment.
 - Ignored implementation

Verifying APIs

- We can propose improved versions of this PKCS#11 fragment and verify them in DY model using a model checker (Fröschle and Steel, WITS '09).
- But:
 - What properties are we verifying?
 - We needed powerful abstractions, but still are only verifying tiny fragment.
 - Ignored implementation

What kind of approach could address these shortcomings?

Security by Typing

very roughly..

- Define a nice semantic notion of security (typically some variant of *non-interference*)
- Define a type system s.t. all well-typed programs preserve this property

Should scale well because can reason locally

Semantic Security Notion for APIs

- Multilevel view - high and low confidentiality

Semantic Security Notion for APIs

- Multilevel view - high and low confidentiality
- Non-interference - no 'flow' from high to low

$$M_1 =_L M_2 \Rightarrow P(M_1) =_L P(M_2)$$

Semantic Security Notion for APIs

- Multilevel view - high and low confidentiality
- Non-interference - no 'flow' from high to low

$$M_1 =_L M_2 \Rightarrow P(M_1) =_L P(M_2)$$

Many, many, *many* variants

Semantic Security Notion for APIs

- Multilevel view - high and low confidentiality
- Non-interference - no 'flow' from high to low

$$M_1 =_L M_2 \Rightarrow P(M_1) =_L P(M_2)$$

Many, many, *many* variants

Specifically for APIs

- All API commands should take L inputs and return L
Handles are L pointers to H keys

Semantic Security Notion for APIs

- Multilevel view - high and low confidentiality
- Non-interference - no 'flow' from high to low

$$M_1 =_L M_2 \Rightarrow P(M_1) =_L P(M_2)$$

Many, many, *many* variants

Specifically for APIs

- All API commands should take L inputs and return L
Handles are L pointers to H keys
- We need to encrypt H with H, so need declassification
(also need to encrypt L with H, and encrypt keys)

Dealing with Crypto

$=_L$ becomes indistinguishability

Can give 'computational non-interference' semantics

Assuming 'Type 0' encryption, can use pattern notion of Abadi-Rogaway

(see Keighren's work)

An Inconvenient Truth

Security APIs often use deterministic crypto

(why?)

We would still like to analyse them

What semantic notion?

Example 1

loc *h0* *h1* *l0* *l1*

M1 *k1* *k1* *c* *c*

M2 *k1* *k1'* *c* *c*

P $l0 = \{l0\}_{h0^*}, l1 = \{l1\}_{h1^*}$

Example 1

$loc \quad h0 \quad h1 \quad l0 \quad l1$

$M1 \quad k1 \quad k1 \quad c \quad c$

$M2 \quad k1 \quad k1' \quad c \quad c$

$P \quad l0 = \{l0\}_{h0*}, l1 = \{l1\}_{h1*}$

$M1' \quad k1 \quad k1 \quad \{c\}_{k1} \quad \{c\}_{k1}$

$M2' \quad k1 \quad k1' \quad \{c\}_{k1} \quad \{c\}_{k1'}$

Example 2

loc *h0* *h1* *h2* *l0* *l1*

M1 *k1* *c1* *c2*

M2 *k1* *c1* *c1*

P $l0 = \{h1^*\}_{h0^*}, l1 = \{h2^*\}_{h0^*}$

Example 2

$loc \quad h0 \quad h1 \quad h2 \quad l0 \quad l1$

$M1 \quad k1 \quad c1 \quad c2$

$M2 \quad k1 \quad c1 \quad c1$

$P \quad l0 = \{h1^*\}_{h0^*}, l1 = \{h2^*\}_{h0^*}$

$M1' \quad k1 \quad c1 \quad c2 \quad \{c1\}_{k1} \quad \{c2\}_{k1}$

$M2' \quad k1 \quad c1 \quad c1 \quad \{c1\}_{k1} \quad \{c1\}_{k1}$

Conditions

Impose conditions on 'memory well-formedness' that restrict the cases our guarantee holds.

But want to avoid requiring $=_H$!

Map congruence classes across high memories:

'hidden values substitution' (see CentenaroFLS-ESORICS '09,
CentenaroF-WITS '10)

\square_v

Is this satisfactory?

Summary

Typing attractive approach for security API analysis

Nice semantic notion if probabilistic crypto used

More tricky for deterministic crypto

PUB

4th Analysis of Security APIs workshop (ASA-4)

Edinburgh, July 21 (day after FCC!)

(satellite of IEEE CSF)

<http://www.lsv.ens-cachan.fr/~steel/asa4>