

# Abstractions for Verifying Key Management APIs

Graham Steel

INRIA & LSV, ENS de Cachan



Host machine



Trusted device



Security API

## Formal Analysis of Key Management APIs

Model cryptography in a Dolev Yao style

(with equational theory perhaps)

Model API commands (and attacker) using (multiset) rewriting

(with state)

Security properties as secrecy properties

Over last 5 years, have verified versions of VSM, IBM CCA, PKCS#11, ...

## Limitations

Almost all previous verifications for **bounded** fresh keys, key handles, etc.

## Limitations

Almost all previous verifications for **bounded** fresh keys, key handles, etc.

Exception: (Fröschle & S. WITS '09) fragment of a particular configuration of PKCS#11

- used abstractions for generation of fresh material

## Limitations

Almost all previous verifications for **bounded** fresh keys, key handles, etc.

Exception: (Fröschle & S. WITS '09) fragment of a particular configuration of PKCS#11

- used abstractions for generation of fresh material

This talk: extending abstraction technique to a larger class of APIs

## Limitations

Almost all previous verifications for **bounded** fresh keys, key handles, etc.

Exception: (Fröschle & S. WITS '09) fragment of a particular configuration of PKCS#11

- used abstractions for generation of fresh material

This talk: extending abstraction technique to a larger class of APIs

Abstractions mostly simple, but will demonstrate efficacy on APIs of real devices.

# Model

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

Plain terms

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

Facts

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

Rules

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

## Example 0: VSM (stateless)

Visa Security Module API - for managing cash machine keys

$$\begin{array}{lcl} X, Y & \rightarrow & \{X\}_Y \\ \{X\}_Y, Y & \rightarrow & X \\ & \xrightarrow{\text{new tmk}} & \{\text{tmk}\}_{km} \\ TC & \rightarrow & \{TC\}_{km2} \\ \{PDK\}_{km}, \{TMK\}_{km} & \rightarrow & \{PDK\}_{TMK} \\ \{TC\}_{km2}, \{TMK\}_{km} & \rightarrow & \{TC\}_{TMK} \end{array}$$

$I = \{\{\text{pdk}\}_{km}, \text{pan}\}$ , +8 more rules SWV237, 238 ([www.tptp.org](http://www.tptp.org))

## Example 0: VSM (stateless)

Visa Security Module API - for managing cash machine keys

$$\begin{array}{lcl} X, Y & \rightarrow & \{X\}_Y \\ \{X\}_Y, Y & \rightarrow & X \\ & \xrightarrow{\text{new tmk}} & \{\text{tmk}\}_{km} \\ TC & \rightarrow & \{TC\}_{km2} \\ \{PDK\}_{km}, \{TMK\}_{km} & \rightarrow & \{PDK\}_{TMK} \\ \{TC\}_{km2}, \{TMK\}_{km} & \rightarrow & \{TC\}_{TMK} \end{array}$$

$I = \{\{pdk\}_{km}, \text{pan}\}$ , +8 more rules SWV237, 238 ([www.tptp.org](http://www.tptp.org))

% initial knowledge is just chosen to be 'typical'

Observation: no inequalities, no state. Easy abstraction.

## Example 1: IBM CCA

Used in 4758 series HSMs

Equational theory:  $\oplus$  used to manage key types

$$\{D\}_{km \oplus \text{data}}$$
$$\{I\}_{km \oplus \text{imp}}$$
$$\vdots$$

Attacks found by Bond and Anderson 2001

Four fixed verified configurations (Courant '06, Cortier Keighren S. '07)

(CKS configurations also treated by Küsters and Truderung '08,

Chen et al. '09, ...)

## Example 1: IBM CCA

Would like to abstract generation commands:

$$\begin{array}{l} \xrightarrow{\text{new d}} \{d\}_{km \oplus \text{data}} \\ \xrightarrow{\text{new kek}} \{\text{kek}\}_{km \oplus \text{exp}} \\ \vdots \end{array}$$

## Example 1: IBM CCA

Would like to abstract generation commands:

$$\begin{array}{l} \xrightarrow{\text{new d}} \{d\}_{km \oplus \text{data}} \\ \xrightarrow{\text{new kek}} \{\text{kek}\}_{km \oplus \text{exp}} \\ \vdots \end{array}$$

$$\{K\}_{KEK \oplus \text{TYPE}}, \text{TYPE}, \{\text{KEK}\}_{km \oplus \text{imp}} \rightarrow \{K\}_{km \oplus \text{TYPE}}$$

## Example 1: IBM CCA

Would like to abstract generation commands:

$$\begin{array}{l} \xrightarrow{\text{new d}} \{d\}_{km \oplus \text{data}} \\ \xrightarrow{\text{new kek}} \{\text{kek}\}_{km \oplus \text{exp}} \\ \vdots \end{array}$$

$$\{K\}_{\text{KEK} \oplus \text{TYPE}}, \text{TYPE}, \{\text{KEK}\}_{km \oplus \text{imp}} \rightarrow \{K\}_{km \oplus \text{TYPE}}$$

$$\{K\}_X, \text{TYPE}, \{\text{KEK}\}_{km \oplus \text{imp}} \rightarrow \{K\}_{km \oplus \text{TYPE}} \quad \text{IF } X \oplus \text{TYPE} \oplus \text{KEK} = 0$$

(S. CADE '05)

## **Example 2: PKCS#11 with static state**

RSA standard, ubiquitous in commercial devices

Modelled with mutable state (DKS CSF '08)

Recent experiments with Tookan (BCFS CCS '10) show real devices often fix state

## Example 2: PKCS#11 with static state

RSA standard, ubiquitous in commercial devices

Modelled with mutable state (DKS CSF '08)

Recent experiments with Tookan (BCFS CCS '10) show real devices often fix state

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

$a1(n1)$  - setting of Boolean attribute  $a1$  for handle  $n1$

## Example 2: PKCS#11 with static state

RSA standard, ubiquitous in commercial devices

Modelled with mutable state (DKS CSF '08)

Recent experiments with Tookan (BCFS CCS '10) show real devices often fix state

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

$a1(n1)$  - setting of Boolean attribute  $a1$  for handle  $n1$

We consider attributes:

$encrypt(n)$ ,  $decrypt(n)$ ,  $sensitive(n)$

$extract(n)$ ,  $wrap(n)$ ,  $unwrap(n)$

## PKCS#11 - 1

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

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

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

## PKCS#11 - 2

Wrap :

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

Unwrap (sym/sym) :

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

## Key Usage

Encrypt :

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

Decrypt :

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

## Abstraction

KeyGenerate :            ;     $\rightarrow$   $h(n_i, k_i); \mathcal{A}_\gamma(n_i, X), \mathcal{A}_\gamma \in \mathcal{G}$

KeyPairGenerate :    ;     $\rightarrow$   $h(n_j, s_j), \text{pub}(s_j); \mathcal{A}_\perp(n_j, X), \mathcal{A}_\perp \in \mathcal{G}$

Unwrap (sym/sym) :

$h(x, y_2), \{y_1\}_{y_2}; \text{unwrap}(x, \top), \rightarrow h(n_l, y_1);$   
 $\mathcal{A}_\uparrow(n_l, X), \mathcal{A}_\uparrow \in \mathcal{U}$

CreateObject :  $x; \rightarrow h(n_m, x); \mathcal{A}_{\uparrow\downarrow}(n_m, X), \mathcal{A}_{\uparrow\downarrow} \in \mathcal{C}$

### Example 3: 'Full' PKCS#11

Set\_Wrap :  $h(x_1, y_1); \neg \text{wrap}(x_1) \rightarrow ; \text{wrap}(x_1)$

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

⋮

⋮

UnSet\_Wrap :  $h(x_1, y_1); \text{wrap}(x_1) \rightarrow ; \neg \text{wrap}(x_1)$

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

⋮

⋮

### Example 3: 'Full' PKCS#11

Set\_Wrap :  $h(x_1, y_1); \neg \text{wrap}(x_1) \rightarrow ; \text{wrap}(x_1)$

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

⋮

⋮

UnSet\_Wrap :  $h(x_1, y_1); \text{wrap}(x_1) \rightarrow ; \neg \text{wrap}(x_1)$

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

⋮

⋮

First approach: infer abstraction from set/unset rules

### Example 3: 'Full' PKCS#11

Set\_Wrap :  $h(x_1, y_1); \neg \text{wrap}(x_1) \rightarrow ; \text{wrap}(x_1)$

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

⋮

⋮

UnSet\_Wrap :  $h(x_1, y_1); \text{wrap}(x_1) \rightarrow ; \neg \text{wrap}(x_1)$

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

⋮

⋮

First approach: infer abstraction from set/unset rules

Better approach: abstract as if static state, modified attributes

# Mechanising Models

Mutable state, protocol has loops

# Mechanising Models

Mutable state, protocol has loops

Use SATMC, encode directly at IF level

# Mechanising Models

Mutable state, protocol has loops

Use SATMC, encode directly at IF level

Fixpoint calculation in case graph levels off  
with secret term reachable in over-approximation

## **PKCS#11: Optimising Templates**

Problem: too many templates. SATMC fails to terminate

## **PKCS#11: Optimising Templates**

Problem: too many templates. SATMC fails to terminate

To make abstraction practical, remove some templates.

## PKCS#11: Optimising Templates

Problem: too many templates. SATMC fails to terminate

To make abstraction practical, remove some templates.

For each attribute construct LHS usage: +,-

For attributes with just +, delete templates where attribute is -, but only if corresponding + template is valid

etc.

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

## Conclusions

Simple but effective abstractions, implementation in Tookan

Verifications of unbounded models of several APIs

Future:

Make precise the class of models for which these abstractions are sound, complete proofs.

Expand to more complex APIs, more detailed cryptographic models.

Non-monotonic state (e.g. TPM)