

Concepts and Proofs for Configuring PKCS#11

Sibylle Fröschle¹ Nils Sommer²

¹University of Oldenburg
Germany

²MWR InfoSecurity
UK

September 15, 2011

Public Key Cryptographic Standard (PKCS) #11

Generic API for wide range of security modules.

Assumption: host system perhaps compromised!



Smartcard



eToken



Hardware Security Module



Alice

Public Key Cryptographic Standard (PKCS) #11

Generic API for wide range of security modules.

Assumption: host system perhaps compromised!



Smartcard



eToken



Hardware Security Module



API Attacks?

Can you trick a PKCS#11 token into revealing a secret key by some sequence of API commands?

API Attacks?

Can you trick a PKCS#11 token into revealing a secret key by some sequence of API commands?

It depends . . .

API Attacks?

Can you trick a PKCS#11 token into revealing a secret key by some sequence of API commands?

It depends ...

Many known attacks:

Clulow 03

Delaune et al. 08

On real tokens:

Sommer 09

Bortolozzo et al. 10

API Attacks?

Can you trick a PKCS#11 token into revealing a secret key by some sequence of API commands?

It depends ...

Many known attacks:

Clulow 03

Delaune et al. 08

On real tokens:

Sommer 09

Bortolozzo et al. 10

Smart configurations!

- ▶ use certain key types
- ▶ constrain functionality

Smart Configurations

FAST'10:

Keys that are generated with `extractable = false` are secure!
(against any logical attacks)

- ▶ Aladdin eToken
- ▶ Siemens HiPath Security
- ▶ Estonian ID



The pre-configured private key of each of these tokens is secure!

What if you need to export a critical key?

ATM Key Management: set up a key of the day to encrypt transactions to be sent from ATM 1 to Bank A



$\Leftarrow \{k\} =$



Is it possible to securely export a key?

What if you need to export a critical key?

ATM Key Management: set up a key of the day to encrypt transactions to be sent from ATM 1 to Bank A



$\Leftarrow \{k\} =$



Is it possible to securely export a key?

Yes!

Without constraints on the functionality!

Plan

1. Introduction
- 2. Export and Import of Keys in Cryptoki:
Problems and Features
3. Smart Configuration
4. Concepts
5. Proofs

Cryptoki

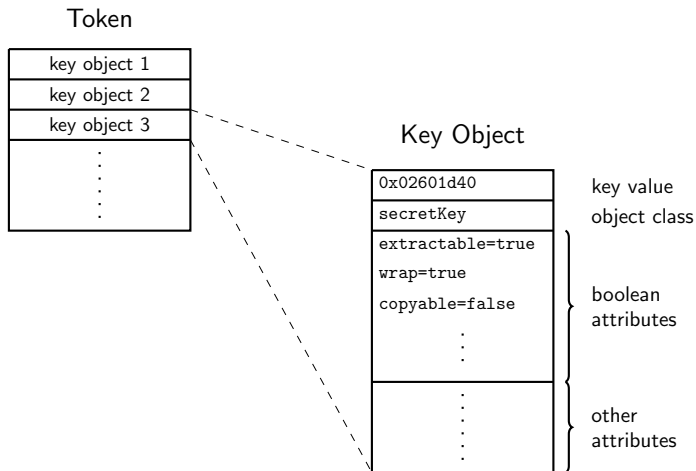
- ▶ Logical view:

Token

key object 1
key object 2
key object 3
⋮

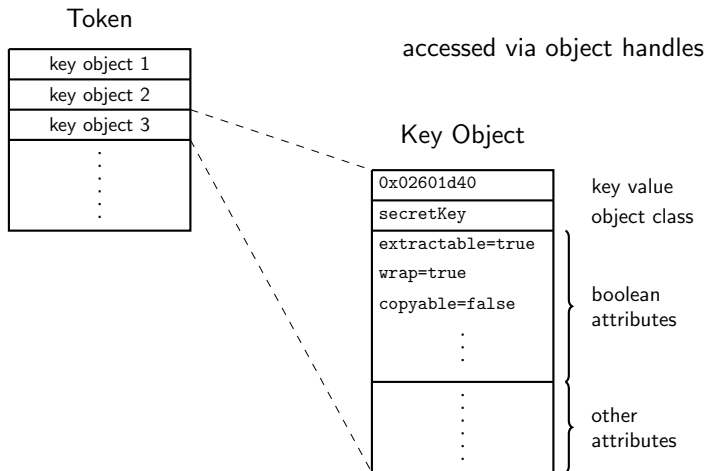
Cryptoki

► Logical view:



Cryptoki

► Logical view:



Basic Wrap and Unwrap of Keys

- ▶ Export key k wrapped under k_w

H \rightarrow T: `WrapKey` h_w h

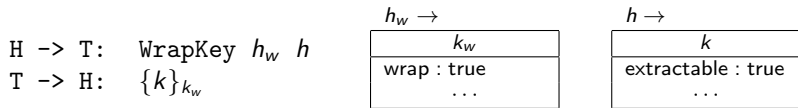
T \rightarrow H: $\{k\}_{k_w}$

$h_w \rightarrow$
k_w
<code>wrap : true</code>
...

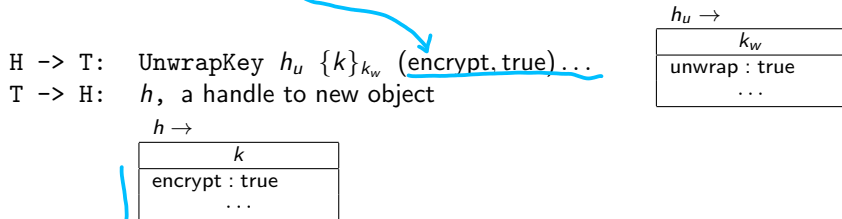
$h \rightarrow$
k
<code>extractable : true</code>
...

Basic Wrap and Unwrap of Keys

- ▶ Export key k wrapped under k_w



- ▶ Import key k wrapped under k_w with attributes as specified by attribute template t



Wrap/Unwrap: Problems and Features

- ▶ Problem 1

 - Cryptoki Solution 1: Wrap with Trusted

- ▶ Problem 2

 - Cryptoki Solution 2: Unwrap Templates

Problem 1

An extractable key can be wrapped by **any** wrap key.

$h \rightarrow$

k
extractable : true ...

Don't wrap me!
I don't trust you!

$h_w \rightarrow$

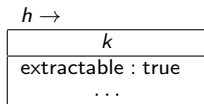
k_w
wrap : true ...

No chance!

A \rightarrow T: WrapKey h_w h

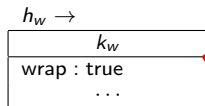
T \rightarrow A: $\{k\}_{k_w}$

Attack 1: Wrap with Trojan Key



Don't wrap me!
I don't trust you!

A \rightarrow T: CreateObject k_w (wrap, true)..
T \rightarrow A: h_w , handle to new object



Trojan key

No chance!

A \rightarrow T: WrapKey h_w h

T \rightarrow A: $\{k\}_{k_w}$

A \rightarrow A: ADecrypt k_w $\{k\}_{k_w}$

A \rightarrow A: k

Solution 1: Wrap with Trusted

1. If a key object has `wrapWTr = true` then it can only be wrapped by key objects with `trusted = true`.
2. Attribute `trusted` can only be set by the Security Officer.

$h \rightarrow$

k
extractable : true
wrapWTr : true
...

$h_w \rightarrow$

k_w
wrap : true
trusted : true
...

You are trusted!
You can wrap me!

H \rightarrow T: WrapKey h_w h
T \rightarrow H: $\{k\}_{k_w}$ ✓

Wrap/Unwrap: Problems and Features

- ✓ ▶ Problem 1
Cryptoki Solution 1: Wrap with Trusted
- ▶ Problem 2
Cryptoki Solution 2: Unwrap Templates

Problem 2

An unwrap key can be used to unwrap into a key object with **any** attributes.

Problem 2

An unwrap key can be used to unwrap into a key object with **any** attributes.

$\{k\}_{k_u}$

I can get unwrapped into an object with non-matching attributes y anyway!

$h_u \rightarrow$

k_u
unwrap : true
...

I'm special. I only want to unwrap keys into objects with attributes x .

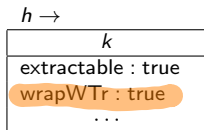
A \rightarrow T: UnwrapKey $h_u \{k\}_{k_u} y$

T \rightarrow A: h , a handle to the new object:

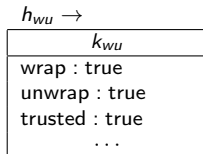
$h \rightarrow$

k
y

Attack 2: Downgrade



Only trusted keys
can wrap me!

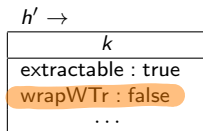


A \rightarrow T: WrapKey $h_{wu} h$

T \rightarrow A: $\{k\}_{k_{wu}}$

A \rightarrow T: UnwrapKey $h_{wu} \{k\}_{k_{wu}}$ (extractable, true)(wrapWTr, false) ...

T \rightarrow A: h' , a handle to new object:



Any key can wrap me!



Solution 2: Unwrap Templates

- ▶ If a key object has attribute `unwrapTempl` set to t then it can only unwrap into key objects that match t .

$\{k\}_{k_u}$

$h_u \rightarrow$

k_u
<code>unwrap : true</code> <code>unwrapTempl :</code> <code>(wrapWTr, true)</code> ...

I can only unwrap keys into objects with `wrapWTr` set to `true`.

H \rightarrow T: `UnwrapKey` h_u $\{k\}_{k_u}$ `(wrapWTr, true), ...`

T \rightarrow H: h , a handle to the new object:

$h \rightarrow$

k
<code>wrapWTr : true</code> ...



Wrap/Unwrap: Problems and Features

- ✓ ▶ Problem 1
Cryptoki Solution 1: Wrap with Trusted
- ✓ ▶ Problem 2
Cryptoki Solution 2: Unwrap Templates

Plan

1. Introduction
2. Export and Import of Keys in Cryptoki:
Problems and Features
- 3. Smart Configuration
4. Concepts
5. Proofs

What if you need to export a critical key?

ATM Key Management: set up a key of the day to encrypt transactions to be sent from ATM 1 to Bank A



$\Leftarrow \{k\} =$



Is it possible to securely export a key?

Yes!



Without constraints on the functionality!

Smart Configuration

Generate the key to be exported
like this:

$h \rightarrow$

k
sensitive : true
wrapWTr : true
...

Security Officer installs the wrap key
like this:

$h_w \rightarrow$

k_w
trusted : true
extractable : false
copyable : false
modifiable : false
decrypt : false
unwrapTempl : sensitive : true wrapWTr : true
...

Then k will always remain unknown to the intruder.

Plan

1. Introduction
2. Export and Import of Keys in Cryptoki:
Problems and Features
3. Smart Configuration
- 4. Concepts
5. Proofs

Concepts for Security APIs

1. How to talk concisely but informally about security properties and attacks?
2. How to find a plausibly secure configuration?
3. How to make a provably secure configuration plausible to practitioners?

It's all about the metadata attached to keys.
In Cryptoki: set of attributes

1. Types of Intention

Four-Axis (4AX) Model (Bond, Clulow, 2006)

Key 1	Key 2
A long-term key shared between Bank A and ATM 1 to securely transport a 'key of the day' from Bank A to ATM 1.	A working 'key of the day' to encrypt transactions to be sent from ATM 1 to Bank A
FORM: AES256	FORM: AES128
USE: secure transport of working keys	USE: to encrypt transactions from ATM 1 to Bank A
ROLE: for key export and import	ROLE: 1: to be exported 2: for data encryption and decryption
DOMAIN: Bank A and ATM 1	DOMAIN: 1: Bank A 2: Bank A and ATM 1

2. Concrete Types

PKCS#11 Type Information = Attributes

Key 1	Key 2
A long-term key shared between Bank A and ATM 1 to securely transport a 'key of the day' from Bank A to ATM 1.	A working 'key of the day' to encrypt transactions to be sent from ATM 1 to Bank A
CKA_CLASS = CK_SECRET_KEY CKA_KEY_TYPE = CKK_AES CKA_VALUE_LEN = 256 CKA_TRUSTED = TRUE CKA_EXTRACTABLE = FALSE CKA_MODIFIABLE = FALSE CKA_COPYABLE = FALSE CKA_WRAP = TRUE CKA_UNWRAP = TRUE CKA_DECRYPT = FALSE CKA_ENCRYPT = FALSE	CKA_CLASS = CK_SECRET_KEY CKA_KEY_TYPE = CKK_AES CKA_VALUE_LEN = 128 CKA_SENSITIVE = TRUE CKA_WRAP_WITH_TRUSTED = TRUE % initial set-up of roles: CKA_EXTRACTABLE = TRUE

3. Security Properties

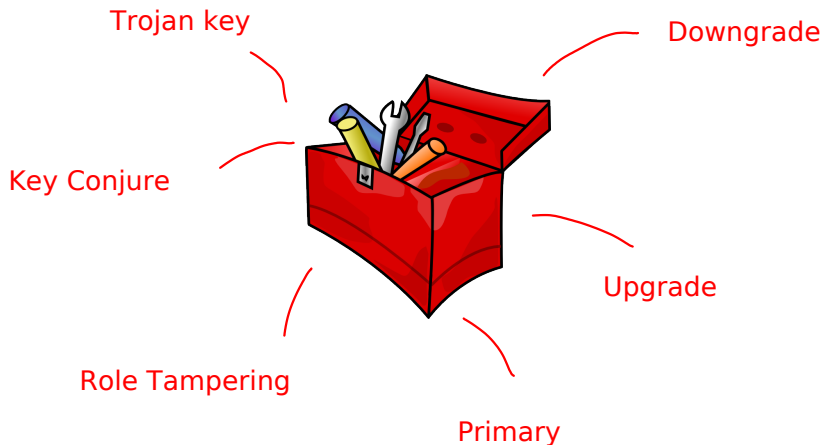
Origin-Secrecy of x : Every key k that originates securely with type x will always remain unknown to the intruder.

Secrecy of x : Every key k that is available with type x has always been and will always remain unknown to the intruder.

Lineage of x : Every key k that is available with type x has always and in all instances been of type x , and securely originated with type x .

Altogether 8 Properties.

4. The Goal-Oriented Attacker's Toolkit



4. The Goal-Oriented Attacker's Toolkit

Type of Attack	Goal of the Attack	Violated Property
Key Conjure	Obtain an unknown key with type x and role y onto the token	Absence of Role y for x
<u>Trojan Key</u>	Obtain a chosen (hence, known) key with type x and role y onto the token	<u>Lineage</u> and Secrecy of x , Absence of Role y for x
Role Tampering	Obtain an unknown key with type x enabled for role y	Absence of Role y for x
Downgrade	Obtain an unknown key with type x , available with a type that gives less protection	Preservation of x
<u>Upgrade</u>	Obtain an unknown key with type x available with a type that gives more protection	<u>Lineage</u> of x
Primary	Obtain an unknown key with type x in plaintext	Origin-Secrecy of x , Secrecy of x

How to find a plausibly secure configuration?

1. Formulate the intended key types in the 4AX Model.
2. Specify security properties for the intended key types.
3. Map the intended key types to concrete key types of the security API to be used, say Cryptoki.
4. Using the attacker's toolkit analyse whether the security properties are violated.
5. Refine the configuration and security properties accordingly.

Now prove formally!

Plan

1. Introduction
2. Export and Import of Keys in Cryptoki:
Problems and Features
3. Smart Configuration
4. Concepts
- 5. Proofs

Approach

'Security-APIs are like protocols
and they are also like programs'

M New

- complete* {
1. Model the data structures of Cryptoki as abstract data types
CASL (Common Algebraic Specification Language)
 2. Specify the commands and their policies of Cryptoki as conditional rewrite system over the data types
 3. Formulate axioms in FOLTL with past operators
 4. Tableau proof method for backwards analysis
 - 5. Simulation equivalence to transfer results

(Fröschle, Sommer FAST'10), (Fröschle, Sommer 11, full draft)

Final Remarks

- ▶ Smart configurations can make PKCS#11 tokens secure!
Even for full implementations!
- ▶ To make them usable:
Need Security Profiles for the various use cases.
- ▶ Need more concepts and 'language' for security API analysis!
Here only a start.