Analysis of Security APIs (part II)

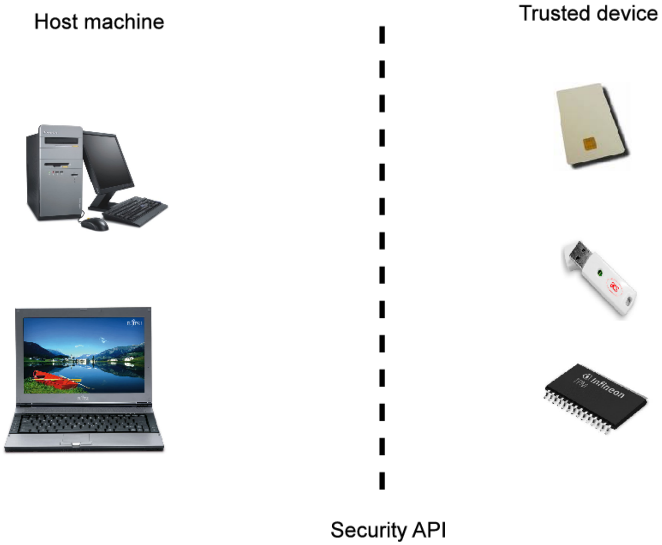
Riccardo Focardi

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

http://www.dsi.unive.it/~focardi
http://secgroup.ext.dsi.unive.it/

FOSAD 2010
Bertinoro, Italy, September 6-11, 2010

Security APIs



Example 1: Hardware Security Module (HSM)

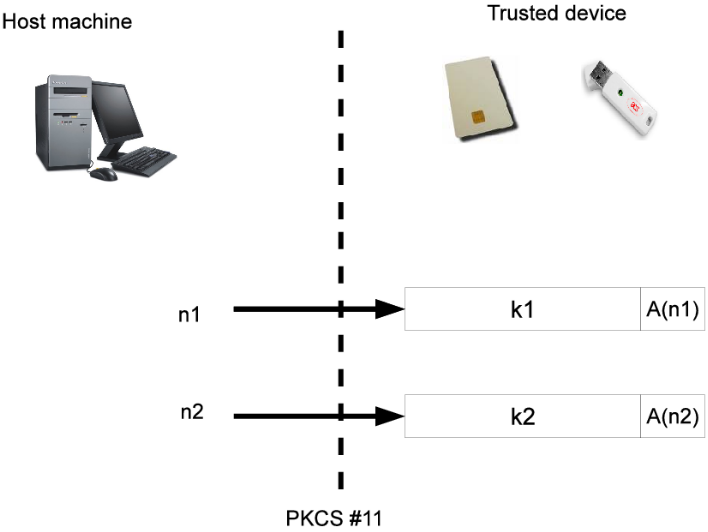
- Used in the ATM Bank network
- Tamper resistant
- Security API for
  - Managing cryptographic keys
  - Decrypting/re-encrypting the PIN
  - Checking the validity of the PIN

Example 1: Hardware Security Module (HSM)

- Used in the ATM Bank network
- Tamper resistant
- Security API for
  - Managing cryptographic keys
  - Decrypting/re-encrypting the PIN
  - Checking the validity of the PIN

... but still, attacks are possible

Example 2: PKCS#11 API for tokens/smarcards

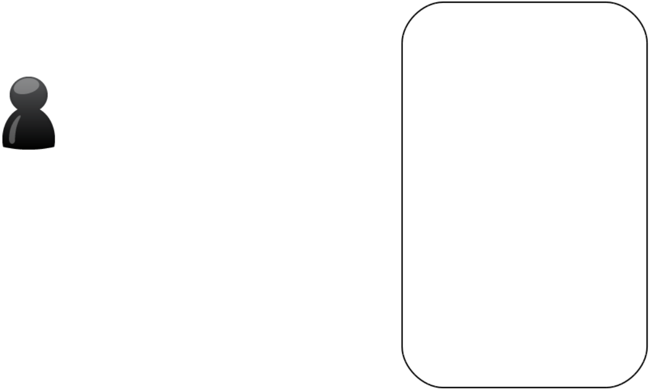


Outline of the course

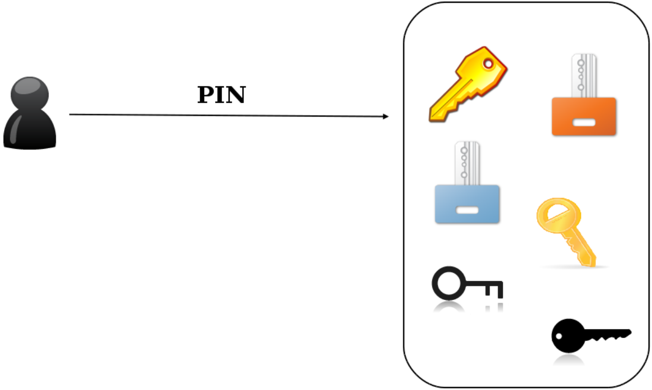
- ✓ Yesterday: PIN processing APIs
  - ✓ Attacks to guess bank PINs
  - ✓ Best strategies to break PINs
  - ✓ Language-based analysis and fixes

- Today: PKCS#11 devices
  - Attacks to compromise a sensitive key
  - A formal model of PKCS#11
  - How to secure PKCS#11: a software token
  - Tookan: Analysis of real tokens

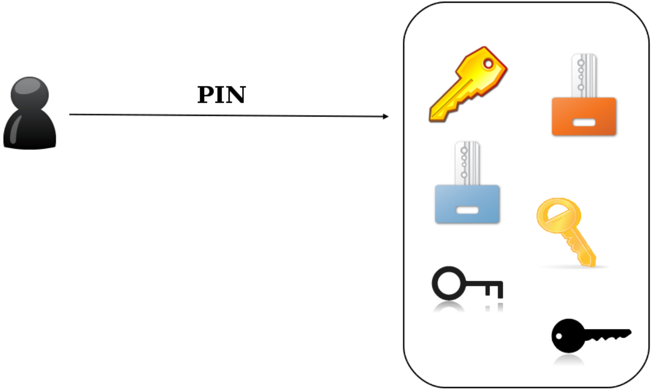
PKCS#11, an overview



PKCS#11, an overview

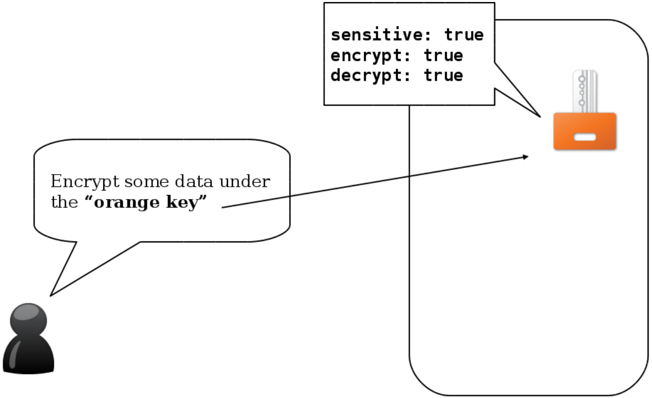


PKCS#11, an overview



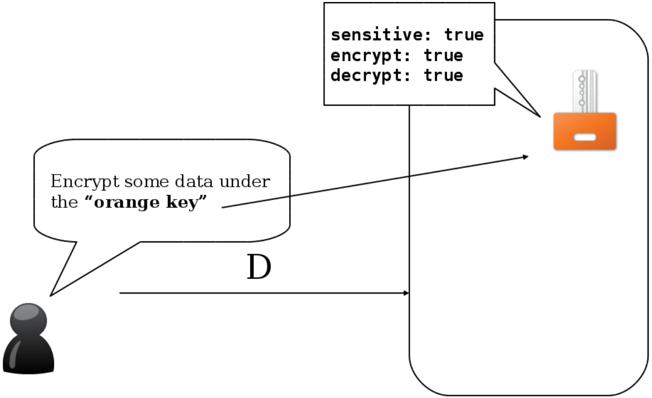
- The PIN is a 'second-layer' protection: Security of keys should not depend on PIN confidentiality

PKCS#11 keys and cryptographic operations



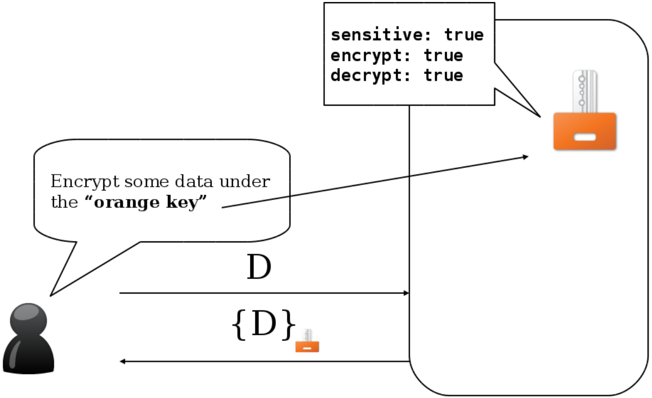
- Keys have *attributes* and are referenced via *handles*
- APIs for *cryptographic operations*

PKCS#11 keys and cryptographic operations



- Keys have *attributes* and are referenced via *handles*
- APIs for *cryptographic operations*

PKCS#11 keys and cryptographic operations



- Keys have *attributes* and are referenced via *handles*
- APIs for *cryptographic operations*

Security of keys

Confidentiality of sensitive keys

- Sensitive keys should never be accessible as plaintext outside the device

Attack scenario

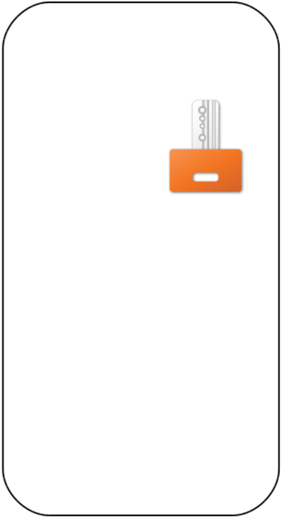
1. The token is used on a public access point
2. the attacker steals the PIN and extracts some sensitive keys
3. any subsequent usage of such token keys is insecure

"... 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 ... " [RSA Security]

- PKCS#11 sensitive keys should not be violated even when used on untrusted hosts and even if the PIN has been disclosed

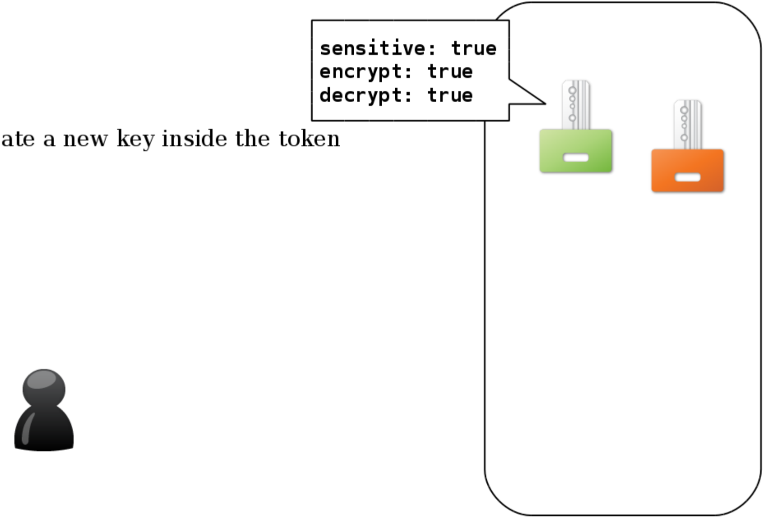
PKCS#11 key management

Create a new key inside the token



PKCS#11 key management

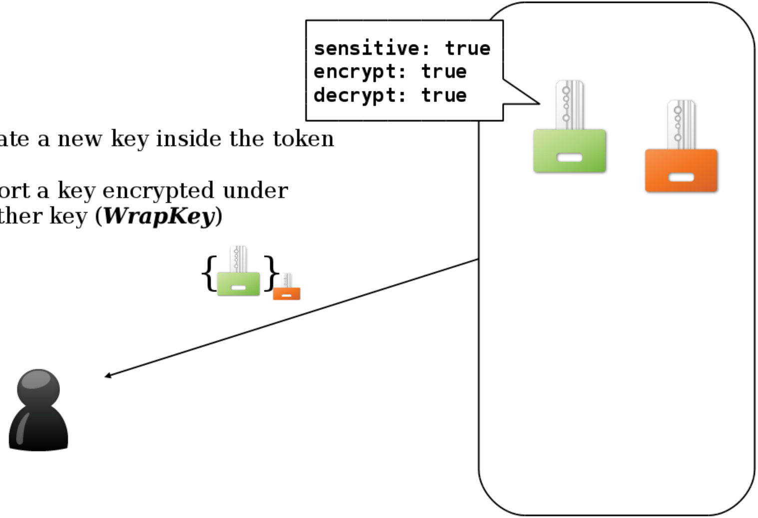
Create a new key inside the token



PKCS#11 key management

Create a new key inside the token

Export a key encrypted under another key (***WrapKey***)

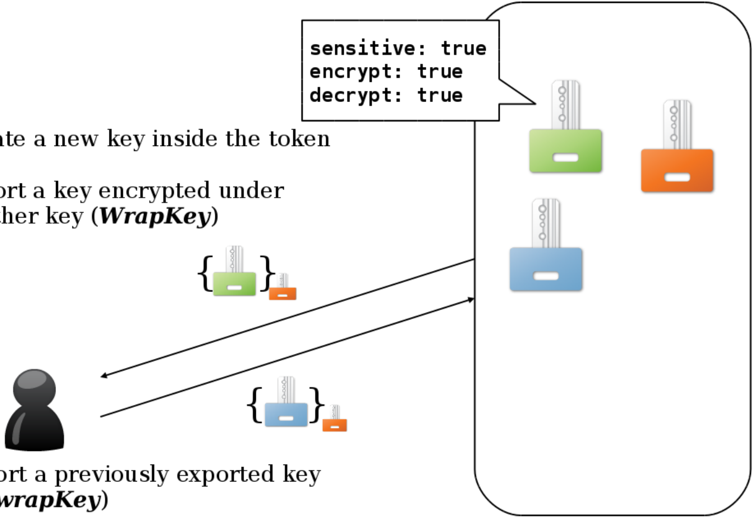


PKCS#11 key management

Create a new key inside the token

Export a key encrypted under another key (***WrapKey***)

Import a previously exported key (***UnwrapKey***)

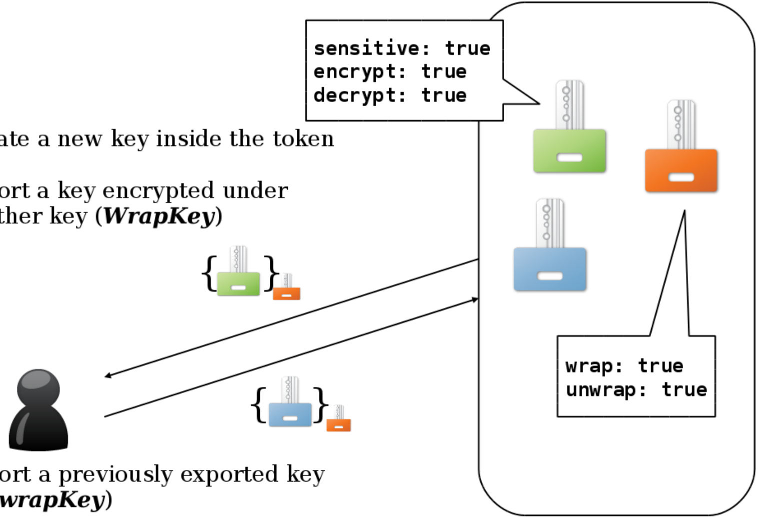


PKCS#11 key management

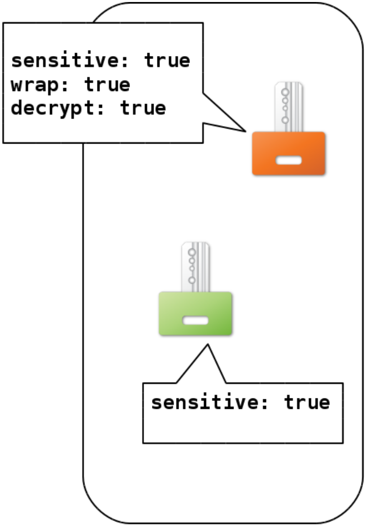
Create a new key inside the token

Export a key encrypted under another key (***WrapKey***)

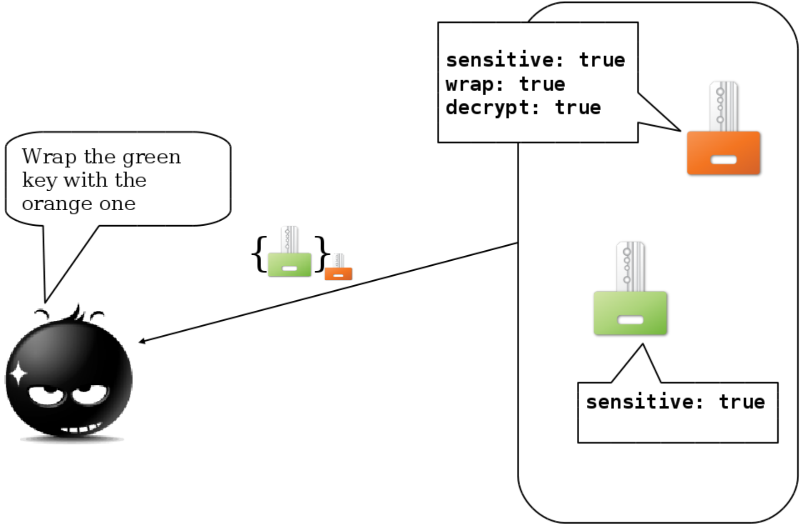
Import a previously exported key (***UnwrapKey***)



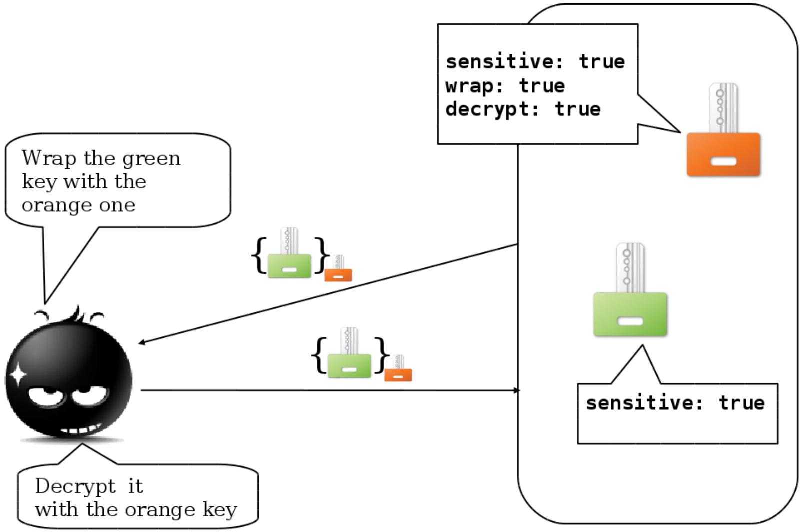
A simple API-level attack [Clulow CHES'03]



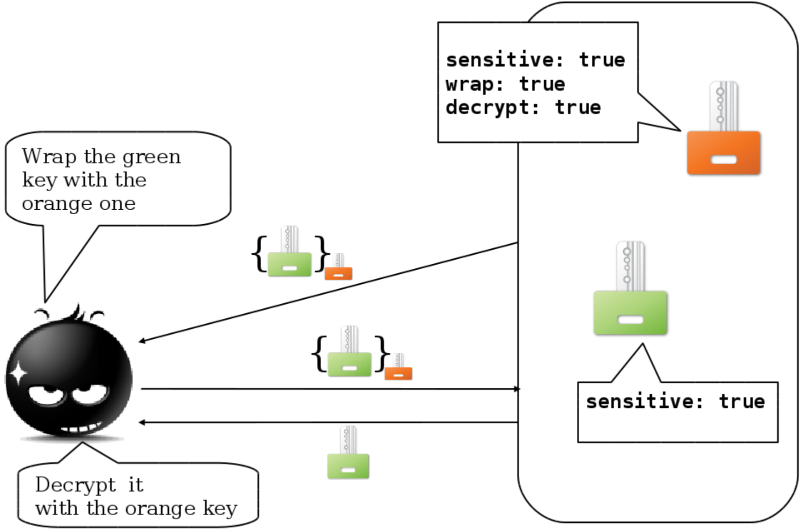
A simple API-level attack [Clulow CHES'03]



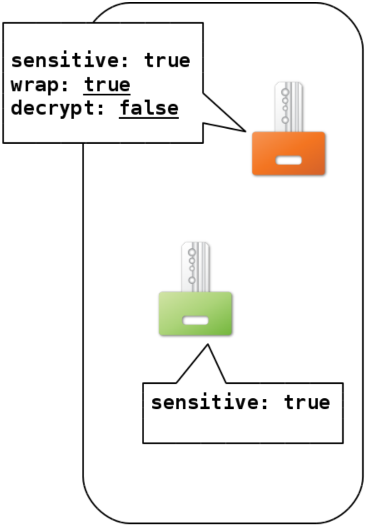
A simple API-level attack [Clulow CHES'03]



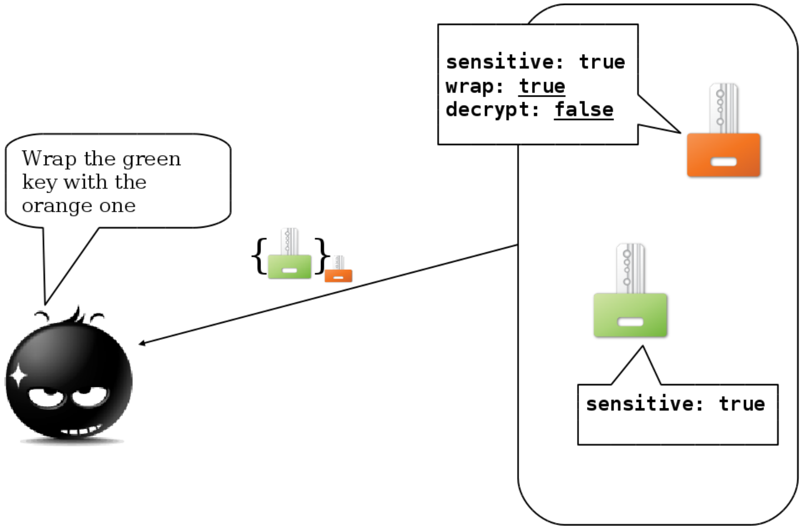
A simple API-level attack [Clulow CHES'03]



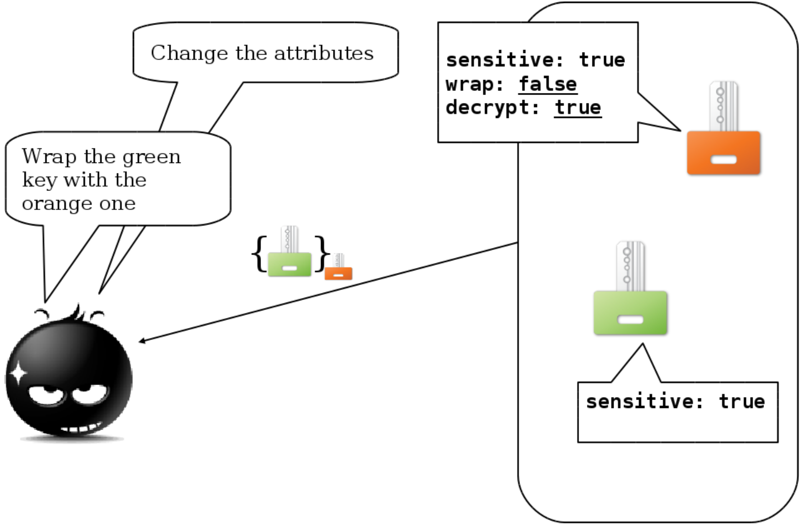
Key separation: forbid wrap and decrypt on the same key



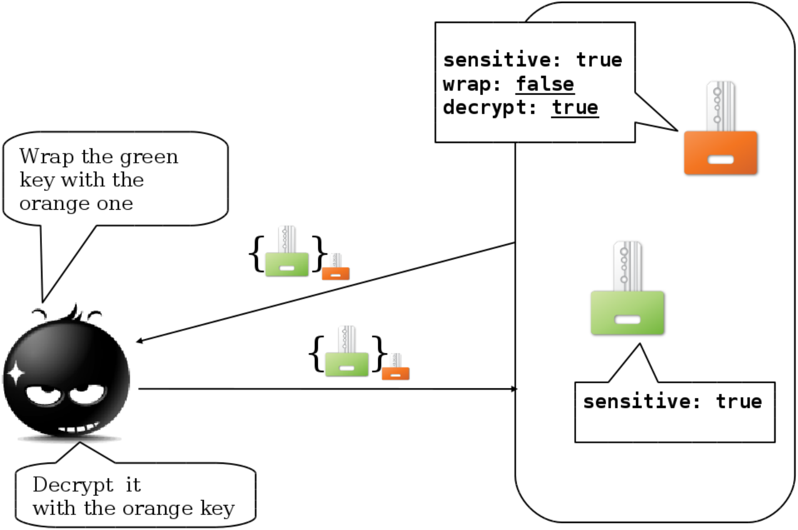
Key separation: forbid wrap and decrypt on the same key



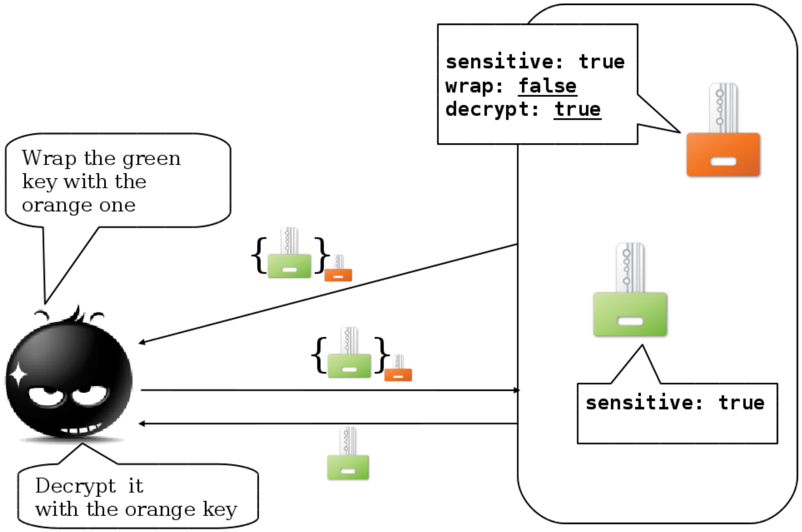
Key separation: forbid wrap and decrypt on the same key



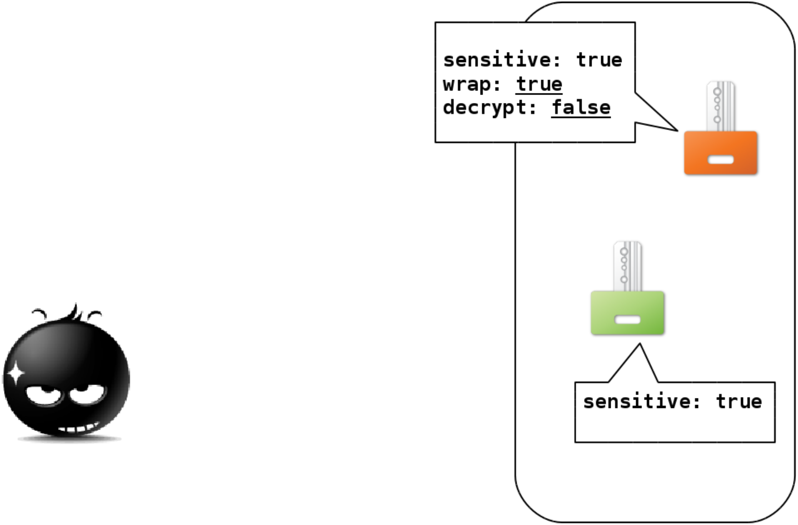
Key separation: forbid wrap and decrypt on the same key



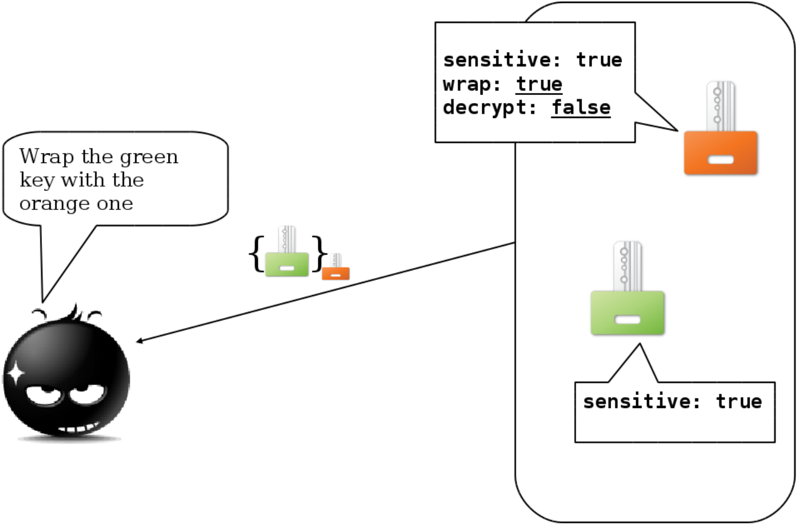
Key separation: forbid wrap and decrypt on the same key



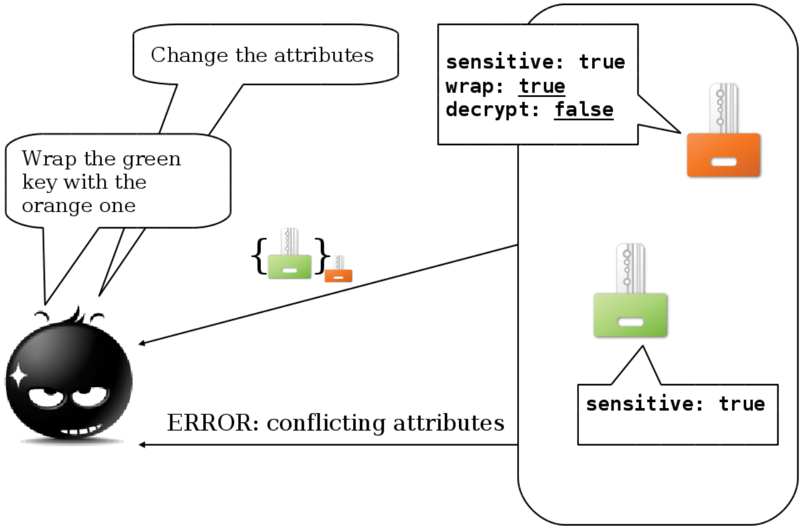
Well ... make attributes 'sticky on'



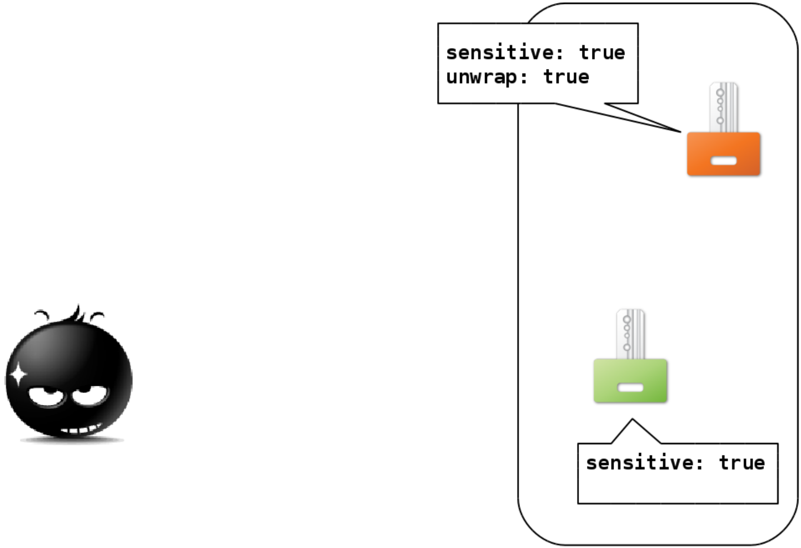
Well ... make attributes 'sticky on'



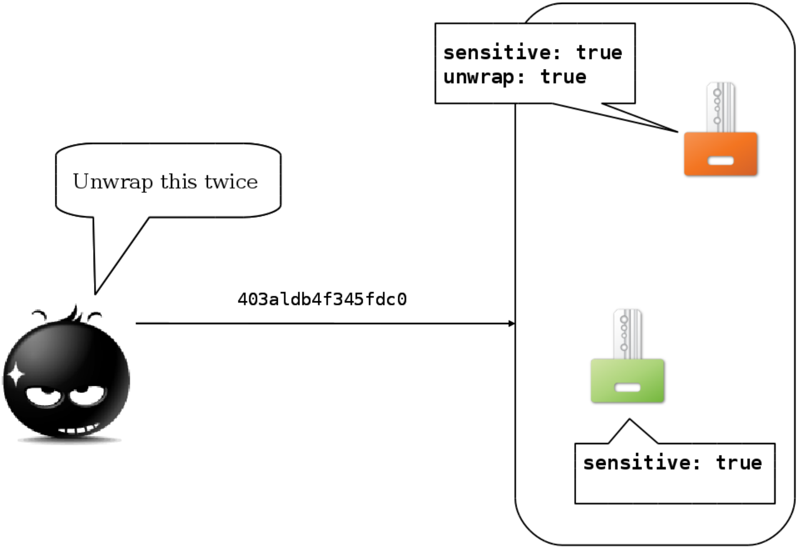
Well ... make attributes 'sticky on'



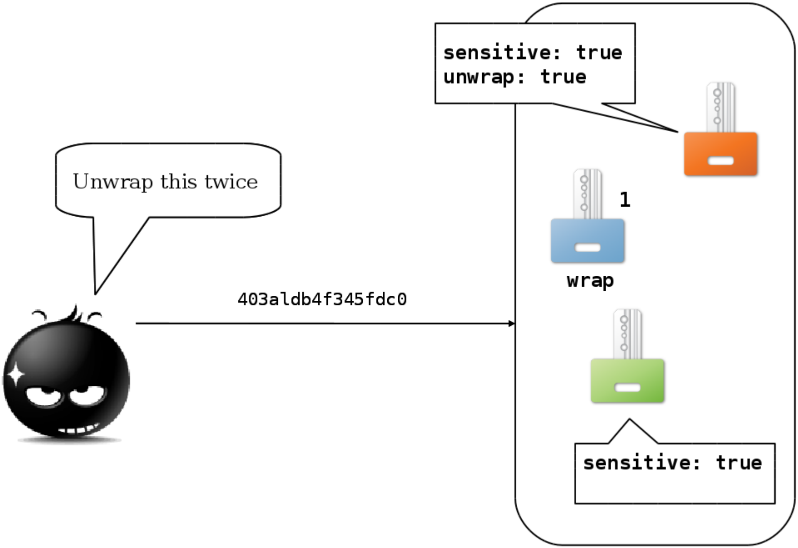
But still ...



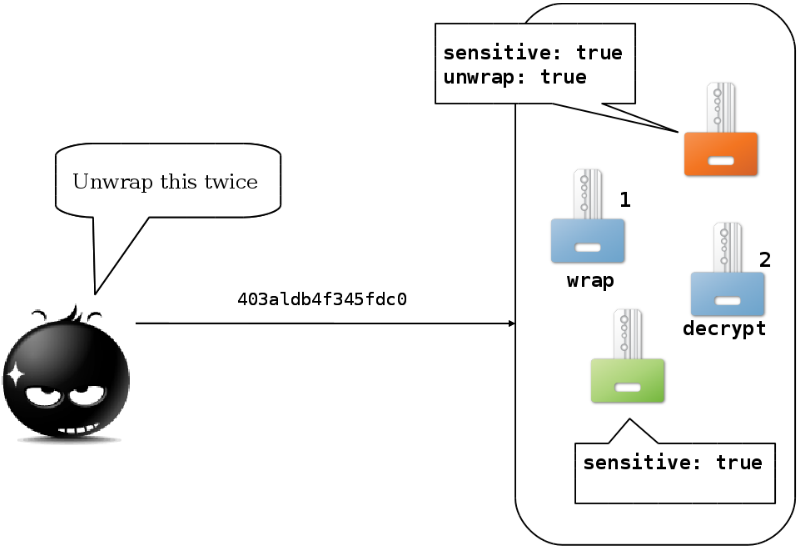
But still ...


Unwrap this twice
403a1db4f345fdc0
sensitive: true
unwrap: true
sensitive: true

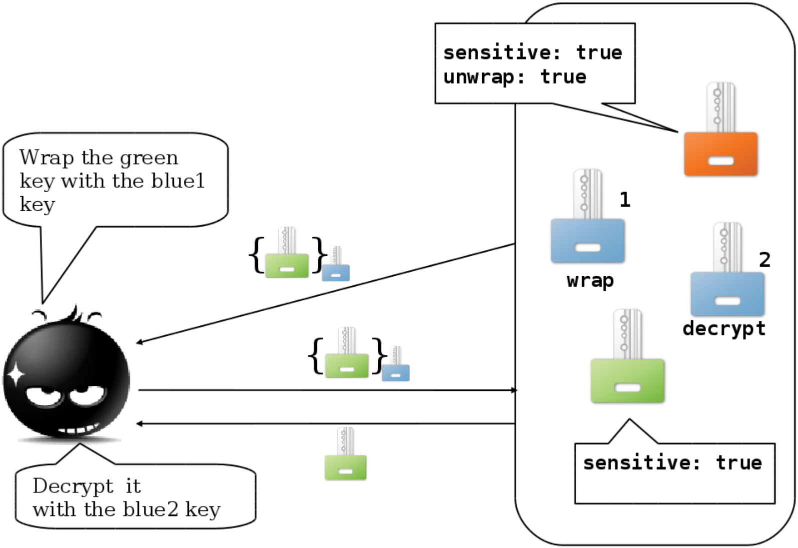
But still ...


Unwrap this twice
403a1db4f345fdc0
sensitive: true
unwrap: true
1
wrap
sensitive: true

But still ...


Unwrap this twice
403a1db4f345fdc0
sensitive: true
unwrap: true
1
wrap
2
decrypt
sensitive: true

But still ...


sensitive: true
unwrap: true
Wrap the green key with the blue1 key
1
wrap
2
decrypt
sensitive: true
Decrypt it with the blue2 key

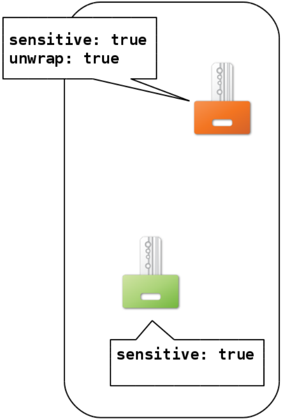
Now what?

Now what?

- check if two instances of the same key have different attributes
- is this of any help?

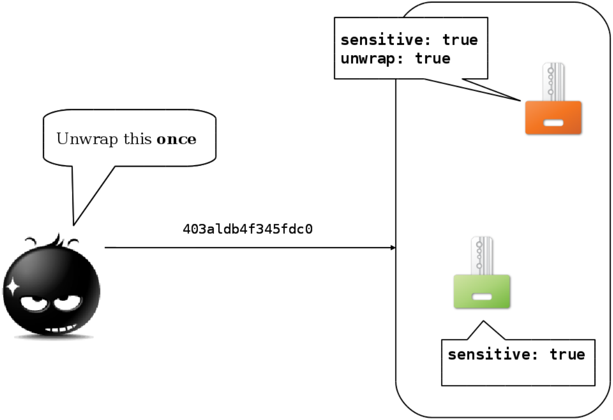
Now what?

- check if two instances of the same key have different attributes
  - is this of any help?



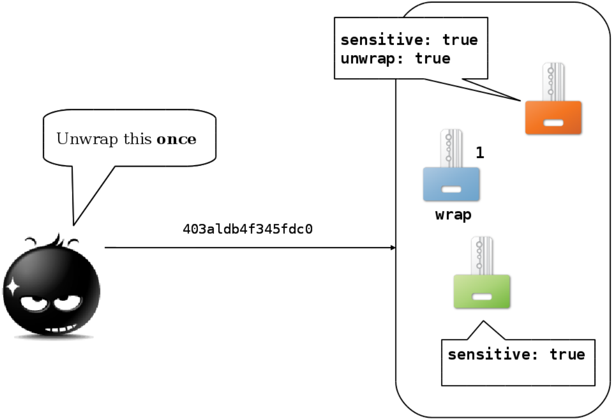
Now what?

- check if two instances of the same key have different attributes
  - is this of any help?



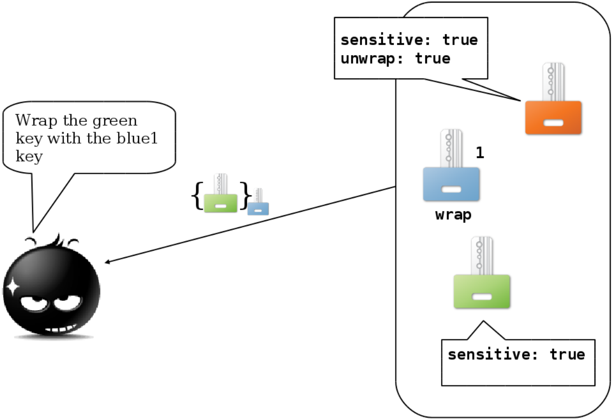
Now what?

- check if two instances of the same key have different attributes
  - is this of any help?



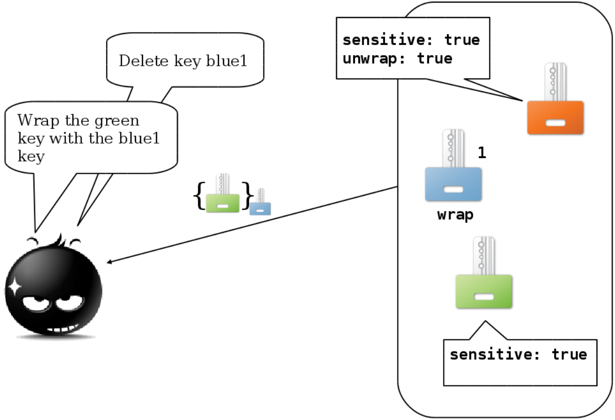
Now what?

- check if two instances of the same key have different attributes
  - is this of any help?



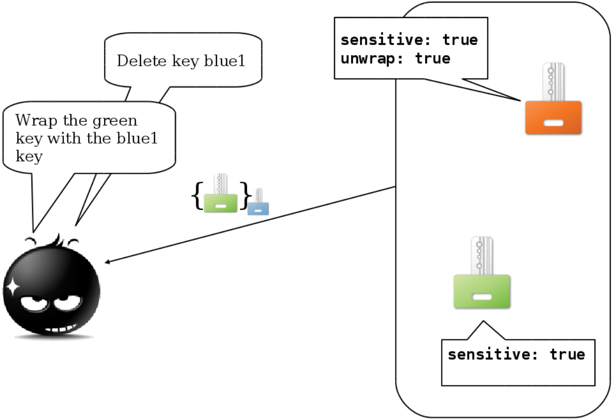
Now what?

- check if two instances of the same key have different attributes
  - is this of any help?



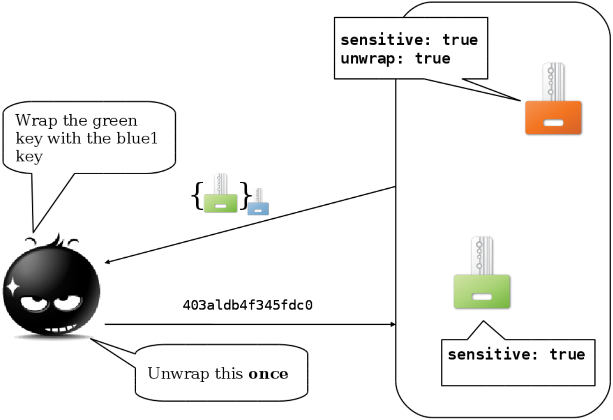
Now what?

- check if two instances of the same key have different attributes
  - is this of any help?



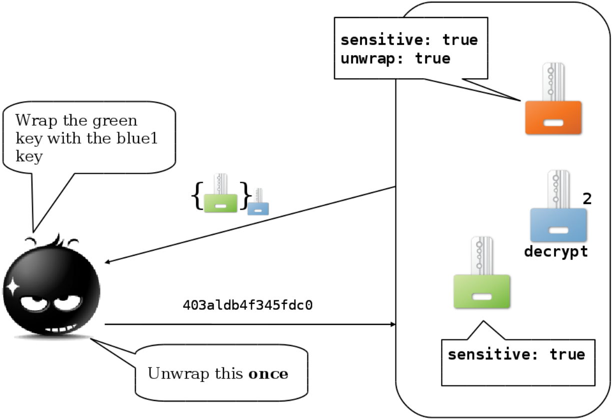
Now what?

- check if two instances of the same key have different attributes
  - is this of any help?



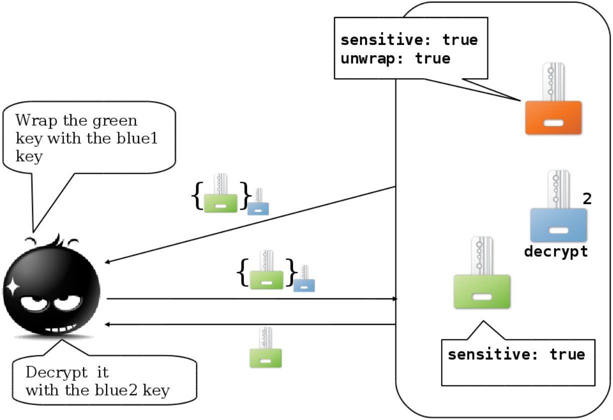
Now what?

- check if two instances of the same key have different attributes
  - is this of any help?



Now what?

- check if two instances of the same key have different attributes
  - is this of any help?



Wrapping format

- keep track of key template when wrapping it
- check that it corresponds when unwrapping

Wrapping format

- keep track of key template when wrapping it
- check that it corresponds when unwrapping

Compute a CBC-MAC of the wrapped key together with its relevant attributes

$$\mathrm{MAC}_{k_m}(\{k_1\}_{k_2}, \mathrm{sensitive}, \mathrm{wrap}, \mathrm{unwrap}, \ldots)$$

and give it as output together with $\{k_1\}_{k_2}$

- if the MAC does not correspond the key is not imported

Wrapping format

- keep track of key template when wrapping it
- check that it corresponds when unwrapping

Compute a CBC-MAC of the wrapped key together with its relevant attributes

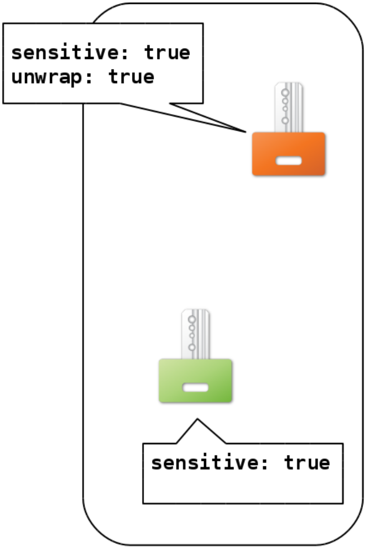
$$\mathrm{MAC}_{k_m}(\{k_1\}_{k_2}, \mathrm{sensitive}, \mathrm{wrap}, \mathrm{unwrap}, \ldots)$$

and give it as output together with $\{k_1\}_{k_2}$

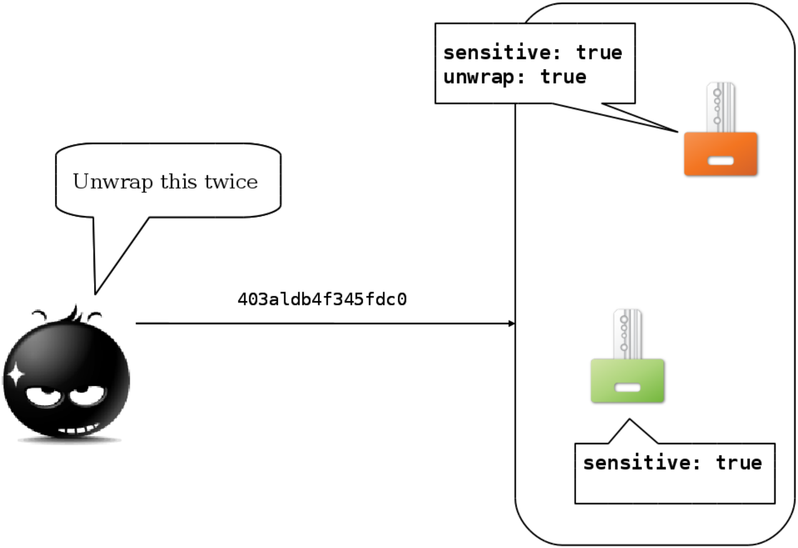
- if the MAC does not correspond the key is not imported

Note: k_m can be derived from k_2, e.g., by encrypting some constant

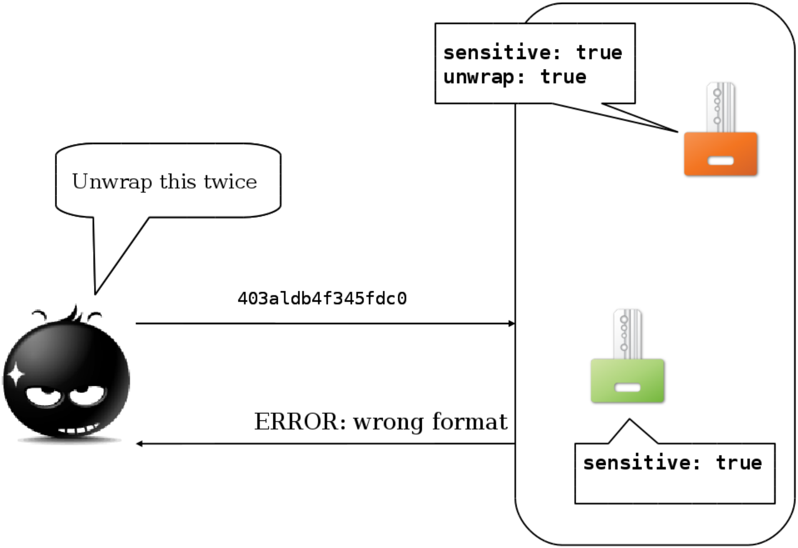
Unwrap of arbitrary data is prevented



Unwrap of arbitrary data is prevented



Unwrap of arbitrary data is prevented



Summary: Attribute policies and wrapping formats

Sticky

Once an attribute is set (unset), it may not be unset (set).
Read-only attributes can be thought as both sticky on and off.

Conflicting

Pairs of attributes that cannot be simultaneously set.
(not in the PKCS#11 documentation)

Tied

Attributes whose value is tied (changing one also changes the other)

Wrapping format

Keep track of relevant attributes when wrapping, and check they are the same when unwrapping

Never use the same thing for different purposes

- buffalo buffalo buffalo buffalo buffalo buffalo buffalo buffalo

Never use the same thing for different purposes

- buffalo buffalo buffalo buffalo buffalo buffalo buffalo buffalo
- Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo

Never use the same thing for different purposes

- buffalo buffalo buffalo buffalo buffalo buffalo buffalo buffalo
- Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo
- Buffalo buffalo, Buffalo buffalo buffalo, buffalo Buffalo buffalo

Never use the same thing for different purposes

- buffalo buffalo buffalo buffalo buffalo buffalo buffalo buffalo
- Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo
- Buffalo buffalo, Buffalo buffalo buffalo, buffalo Buffalo buffalo
- THE buffalo FROM Buffalo WHO ARE buffaloed (indimidated) BY buffalo FROM Buffalo, buffalo buffalo FROM Buffalo

Summary: key-separation attacks

Wrap-decrypt

same key used for wrapping a sensitive key and then decrypting it

Wrap-decrypt with attribute change

even if wrap and decrypt are configured as *conflicting*, we can first set wrap and successively unset it to set decrypt

Wrap-decrypt with 'key aliases'

even if we set wrap and decrypt *sticky on*, we can import a key twice and give the two copies some conflicting attributes.

- We can prevent the last attack by adding a *wrapping format*
- More attacks, e.g. Unwrap-encrypt. Try this as an exercise.

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

- Terms representing keys, ciphertexts, handles

$$k, senc\,(d, k)\,, h(n, k)$$

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

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

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

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

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 $\langle\ \{h(n_1,k_1)\}, sensitive(n_1), extract(n_1)\ \rangle$
  - $\leadsto \langle\ \{h(n_1,k_1), h(n_2,k_2)\},$ $sensitive(n_1), extract(n_1), wrap(n_2), decrypt(n_2)\ \rangle$
  - $\leadsto \langle\ \{h(n_1,k_1), h(n_2,k_2), senc(k_1,k_2)\},$ $sensitive(n_1), extract(n_1), wrap(n_2), decrypt(n_2)\ \rangle$
  - $\leadsto \langle\ \{h(n_1,k_1), h(n_2,k_2), senc(k_1,k_2), k_1\},$ $sensitive(n_1), extract(n_1), wrap(n_2), decrypt(n_2)\ \rangle$

The DKS model for symmetric keys

$$\xrightarrow{\text{new } n, k_1} \quad h(n, k_1); \neg extract(n), \mathcal{L}$$

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

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

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

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

$$h(x_1, y_1); \neg wrap(x_1) \longrightarrow wrap(x_1)$$

$$\ldots \quad \ldots \quad \ldots$$

$$h(x_1, y_1); wrap(x_1) \longrightarrow \neg wrap(x_1)$$

$$\ldots \quad \ldots \quad \ldots$$

- Similar rules for asymmetric keys

... plus 'Dolev-Yao'

$$\begin{array}{rcl} x, y & \longrightarrow & senc\,(x, y) \\ senc\,(x, y)\,, y & \longrightarrow & x \end{array}$$

What is this for? and why is it interesting?

... plus 'Dolev-Yao'

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

What is this for? and why is it interesting?

- Operations performed by the attacker *independently* of the device

... plus ‘Dolev-Yao’

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

What is this for? and why is it interesting?

- Operations performed by the attacker *independently* of the device
- Decrypting data encrypted with a broken key

... plus 'Dolev-Yao'

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

What is this for? and why is it interesting?

- Operations performed by the attacker *independently* of the device
- Decrypting data encrypted with a broken key
- Decrypting keys wrapped with a broken key

… plus 'Dolev-Yao'

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

What is this for? and why is it interesting?

- Operations performed by the attacker *independently* of the device
- Decrypting data encrypted with a broken key
- Decrypting keys wrapped with a broken key
- Wrapping keys with a broken key and import them in the device
- …

The model at work

Security as a reachability property

given an initial state $\langle T_0; L_0 \rangle$ and a set of sensitive keys S, is there a reduction $\langle T_0; L_0 \rangle \rightsquigarrow^* \langle T_n; L_n \rangle$ such that $S \cap T_n \neq \emptyset$?

The model at work

Security as a reachability property

given an initial state $\langle T_0; L_0 \rangle$ and a set of sensitive keys S, is there a reduction $\langle T_0; L_0 \rangle \rightsquigarrow^* \langle T_n; L_n \rangle$ such that $S \cap T_n \neq \emptyset$?

Exercise

Find the initial state and the reduction for the other two attacks. In doing so try to ‘patch’ the model with conflicting and sticky attributes.

The model at work

Security as a reachability property

given an initial state $\langle T_0; L_0 \rangle$ and a set of sensitive keys S, is there a reduction $\langle T_0; L_0 \rangle \rightsquigarrow^* \langle T_n; L_n \rangle$ such that $S \cap T_n \neq \emptyset$?

Exercise

Find the initial state and the reduction for the other two attacks. In doing so try to 'patch' the model with conflicting and sticky attributes.

- Automated check via NuSMV and SATMC. Known and new attacks found (plus new variants) [Delaune, Kremer, Steel CSF'08]

The model at work

Security as a reachability property

given an initial state $\langle T_0; L_0 \rangle$ and a set of sensitive keys S, is there a reduction $\langle T_0; L_0 \rangle \rightsquigarrow^* \langle T_n; L_n \rangle$ such that $S \cap T_n \neq \emptyset$?

Exercise

Find the initial state and the reduction for the other two attacks. In doing so try to 'patch' the model with conflicting and sticky attributes.

- Automated check via NuSMV and SATMC. Known and new attacks found (plus new variants) [Delaune, Kremer, Steel CSF'08]
- Model extensions for
    1. analyzing integrity issues [Falcone, Focardi, ARSPA-WITS'10]
    2. checking real devices [Bortolozzo, Centenaro, Focardi, Steel, CCS'10]

Key Integrity

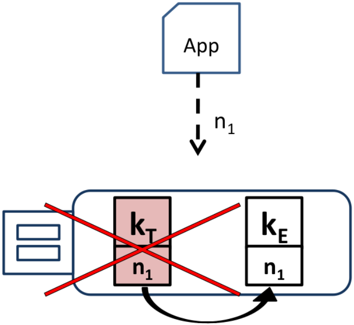
1. The token is used on a public access point
2. the attacker steals the PIN and replaces some sensitive key k
3. k might be subsequently used to:
   - encrypt sensitive data
   - wrap sensitive keys
   - sign secret data (attacker gets credit)
   - check signatures (impersonation)

- ... 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	$h(n_1, \mathbf{k_3}), senc(k_3, k_2), h(n_2, k_2)$

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

$$x, y \longrightarrow senc(x, y)$$
$$senc(x, y), y \longrightarrow x$$
$$\dots$$

A complete key integrity attack

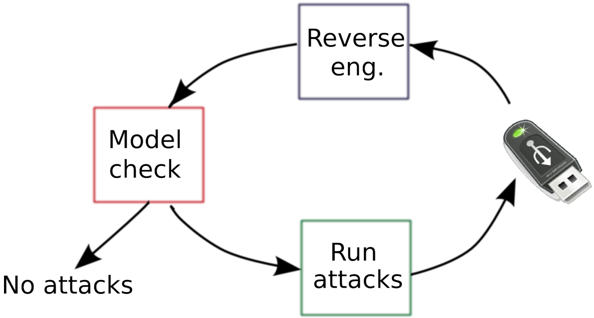
step	transition	σ	user knowledge	attacker knowledge
0	-	-	$d, h(t, k_t), h(i, k_i)$	$h(t, k_t), h(i, k_i), k_e$
1	encrypt	E	$d, h(t, k_t), h(i, k_i)$	$h(t, k_t), h(i, k_i), k_e$, $\mathbf{senc}(\mathbf{k_e}, \mathbf{k_i})$
2	overwrite	E	$d, h(t, \mathbf{k_e}), h(i, k_i)$	$h(t, \mathbf{k_e}), h(i, k_i), k_e$, $senc(k_e k_i)$
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	$d, h(t, k_e), h(i, k_i)$, $senc(d, k_e)$	$k_e, senc(k_e k_i)$, $senc(d, k_e)$, $\mathbf{d}$

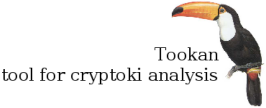
A (maybe too) simple fix

- 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	-	-	-

Analysis of real PKCS#11 devices [Bortolozzo, Centenaro, Focardi, Steel, CCS'10]





Why reverse engineering

- The standard does not say much about attribute policies
- We have noticed that some real devices prevent the attacks

start from the general model and refine it so to 'fit' the analysed device

Examples

Sticky: try to set on and then off an attribute

Conflicts: try to create a key with two attributes set

Tied: try to change one attribute and observe the others

API: check which functionalities are implemented

- not complete but works well on the 17(+) devices we have tested

An example of reverse engineering

```
# KEY TYPES
supports_symmetric_keys(true);
supports_asymmetric_keys(true);

# FUNCTIONS
functions('wrap', 'unwrap', 'encrypt', 'decrypt', 'create_object');

# MODES
wrap_modes('symmetric, sensitive / symmetric, sensitive',
          'symmetric, sensitive / symmetric, nonsensitive', ...);
unwrap_modes('symmetric, sensitive / symmetric, sensitive', ...);
encrypt_modes('symmetric, sensitive','symmetric, nonsensitive',..);
decrypt_modes('symmetric,sensitive', 'symmetric,nonsensitive', ..);

# ATTRIBUTES
attributes('sensitive', 'extract', 'wrap', 'unwrap',
           'encrypt', 'decrypt');
```

An example of reverse engineering

```
# SICKY ON / OFF ATTRIBUTES
sticky_on_asymmetric('sensitive');
sticky_off_asymmetric('extract');
sticky_on_symmetric('sensitive', 'never_extract');
sticky_off_symmetric('extract', 'never_extract');

# CONFLICTS ATTRIBUTES
conflict_symmetric('extract,never_extract');
conflict_asymmetric('extract,never_extract');

# TIED ATTRIBUTES
tied_symmetric('sensitive,always_sensitive');
tied_asymmetric('sensitive,always_sensitive');

# FLAGS
sensitive_prevents_read(true);
unextractable_prevents_read(false);
```

Model generation

- We refine the model by parametrizing the rules

Example: SetAttribute

DKS: The default rule for each attribute 'a' was

$$\mathrm{h}\,(x_1, y_1)\,;\neg\mathrm{a}\,(x_1) \;\rightarrow\; \mathrm{a}\,(x_1)$$

Tookan: We add constraints as follows

$$\mathrm{h}(\mathrm{x}_1, \mathrm{y}_1);\ \neg\mathrm{a}(\mathrm{x}_1), \neg\mathcal{A}^{\mathrm{conf(a)}}(\mathrm{x}_1) \;\rightarrow\; ;\ \mathrm{a}(\mathrm{x}_1), \mathcal{A}^{\mathrm{tied(a)}}(\mathrm{x}_1)$$

(with $\mathrm{a} \notin \texttt{sticky\_off\_attributes}$)

Let $\mathcal{A}^{\mathrm{conf(a)}} = \{a_1, \ldots, a_m\}$. Then $\mathcal{A}^{\mathrm{conf(a)}}(\mathrm{n})$ stands for $a_1(n), \ldots, a_m(n)$

Results of testing

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

Attacks	a1	wrap/decrypt attack based on symmetric keys
	a2	wrap/decrypt attack based on asymmetric keys
	a3	sensitive keys are directly readable
	a4	unextractable keys are directly readable (forbidden by the standard)
	a5	sensitive/unextractable keys can be changed into nonsensitive/extractable

CryptokiX

- CryptokiX is a fiXed software token based on openCryptoki [Bortolozzo, Centenaro, Focardi, Steel, ASA'10]
- Available at http://secgroup.ext.dsi.unive.it/CryptokiX
- Its security is configurable by selectively enabling different patches
  - Conflicts `conflict_sym('wrap,decrypt', 'unwrap,encrypt');`
  - Sticky `sticky_on_sym('wrap','unwrap','encrypt','decrypt');`
  - Format the CBC-MAC-based wrapping format
- When all enabled, these patches prevent all the discussed attacks (not the one on key integrity)

CryptokiX - secure templates

- limit the set of admissible assignments for key attributes
- configurable for each PKCS#11 command: generate, unwrap, create
- **first** secure configuration of PKCS#11 that does not require new cryptographic mechanisms

Key generation

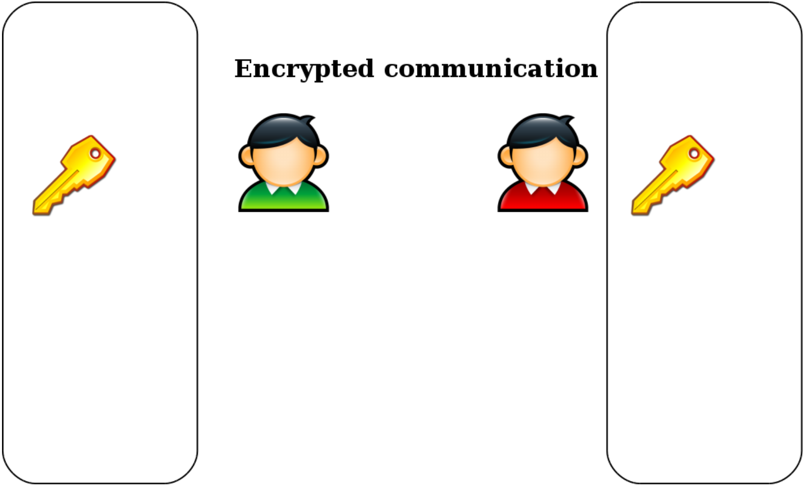
- Key encrypting keys: wrap and unwrap set
- Data keys: encrypt and decrypt set

Imported keys (unwrap and create)

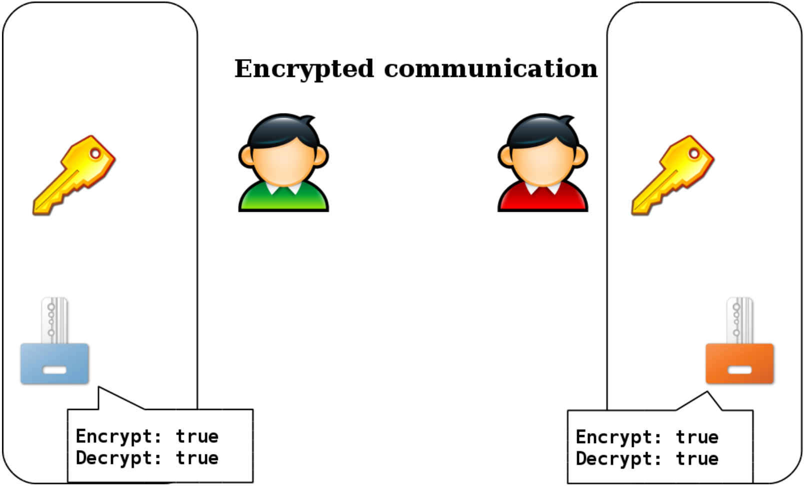
- unwrap,encrypt set and wrap,decrypt unset

- Attributes are not modifiable

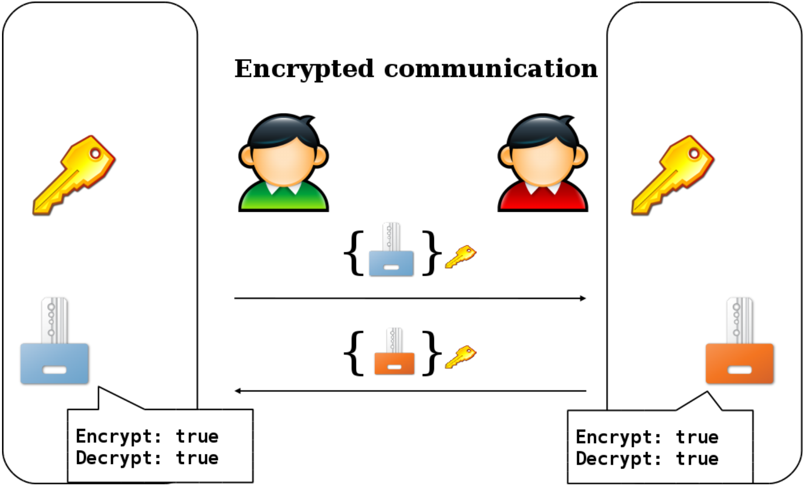
Secure templates: an example



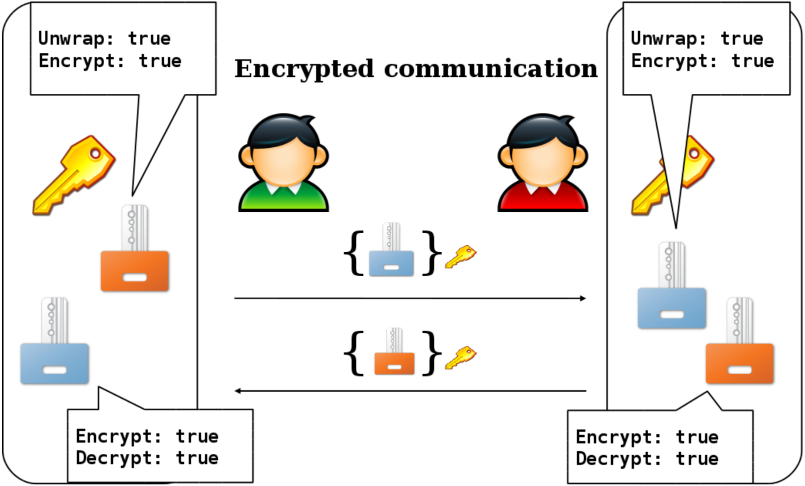
Secure templates: an example



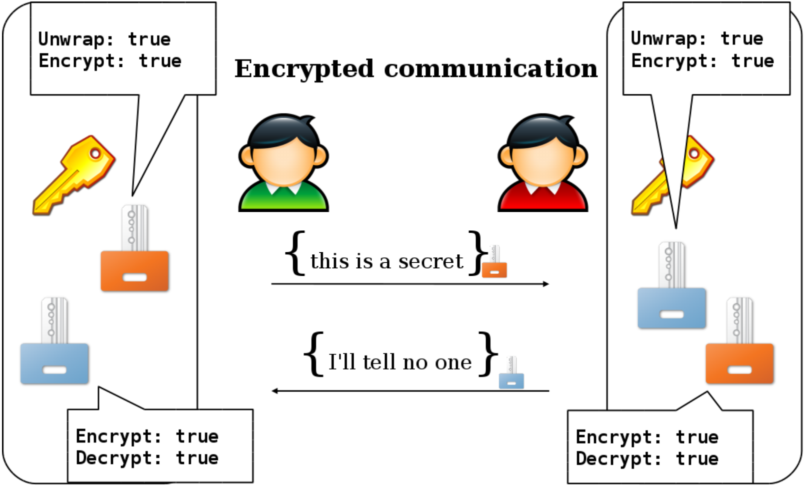
Secure templates: an example



Secure templates: an example



Secure templates: an example



Conclusion

- ✓ PKCS#11 is irritatingly liberal [RSA Security]
- ✓ Attacks to compromise a sensitive key and fixes [Clulow CHES'03][Delaune, Kremer, Steel CSF'08]
- ✓ A formal model of PKCS#11, with extension to integrity [Delaune, Kremer, Steel CSF'08][Falcone, Focardi, ARSPA-WITS'10]
- ✓ Tookan: Analysis of real tokens (disquieting results...) [Bortolozzo, Centenaro, Focardi, Steel, CCS'10]
- ✓ CryptokiX: A secure, fully fledge token can be realized in practice [Bortolozzo, Centenaro, Focardi, Steel, ASA'10]
  - Useful for educational purposes
  - Open-source: patches can be examined and extended by anyone

References

Bortolozzo M., Centenaro M., Focardi, R., Steel G.
Attacking and Fixing PKCS#11 Security Tokens.
In Proceedings of ACM CCS'10, October 2010, to appear.

Bortolozzo M., Centenaro M., Focardi, R., Steel G.
CryptokiX: a cryptographic software token with security fixes.
In Proceedings of ASA'10, July 2010.

V. Cortier and G. Steel.
A generic security API for symmetric key management on cryptographic devices.
In Proceedings of ESORICS'09, September 2009.

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

References

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

Falcone, A., Focardi R.
Formal Analysis of Key Integrity in PKCS#11.
In Proceedings of ARSPA-WITS'10, March 2010.

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