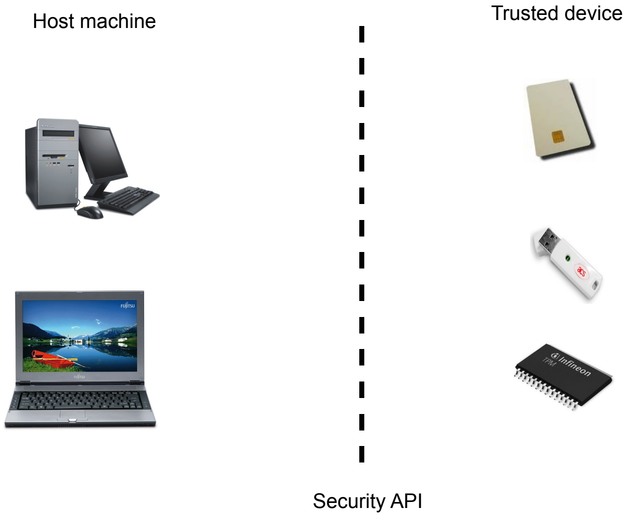
Introduction to Security APIs

Graham Steel

In this Lecture

- What is a Security API?
- Examples from Secure Hardware
- Formal tools and for analysis
- New security APIs

What is a security API?

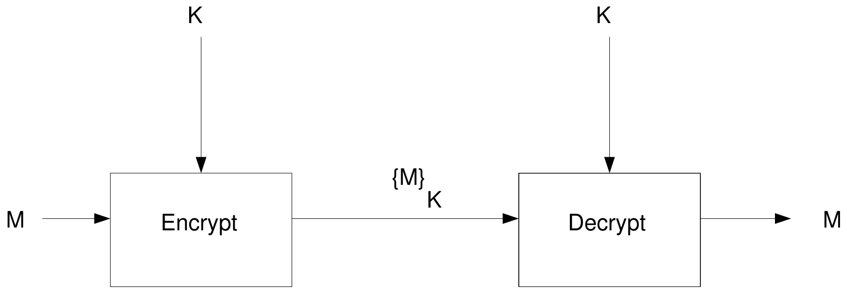


Security API Examples

Untrusted	Trusted
client application	tamper-resistant smartcard
host machine	Hardware Security Module (HSM)
web application	web service API
web application	web browser API
smartphone app	Sandbox APIs
process	hardened OS

Crypto 101

Symmetric key crypto



e.g. DES, 3DES, AES

Infineon
TPM

CAUTION
IBM

Secure Hardware History

Military:
WW2 Enigma machines
- captured machines used to help break codes
NSA devices with explosive tamper resistance
-
`http://www.nsa.gov/about/cryptologic_heritage/museum/`
Commercial:
IBM: Cryptoprocessors for mainframes
- tamper-resistant switches on case
ATMs (cash machines)
- Encrypted PIN Pads (EPPs) and Hardware Security Modules (HSMs)

Secure Hardware History - 2

Cryptographic Smartcards
- chip contains cryptoprocessor and keys in memory
- used in SIM cards, credit cards, ID cards, transport. . .

Authentication tokens
- generate One-Time Passwords, sometimes USB connection

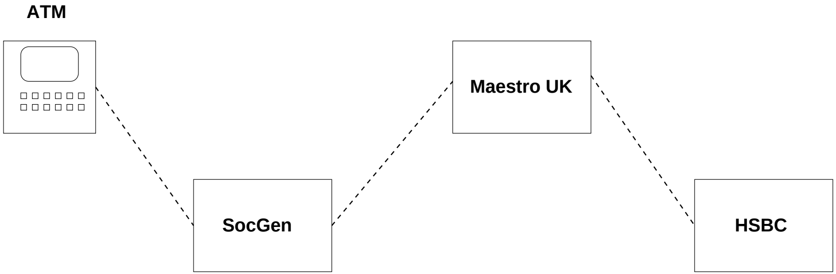
Trusted Platform Module (TPM)
- now standard in most PC laptops

The future..
- Secure Elements in mobile phones, cars,. . .

Example 1 - Cash Machine Network

- The ‘Killer app’ for cryptography according to Anderson (on alt.security, see e.g. `http://axion.physics.ubc.ca/atm.html`)
- Introduced in the UK in the late 1960s, first modern machines (with DES) in the 70s and 80s
- 2.2 million ATMs worldwide (`http://www.atmia.com/mig/globalatmclock/`)
- Network is now global and ubiquitous (at least in cities)

Simplified Network Schematic



HSMs

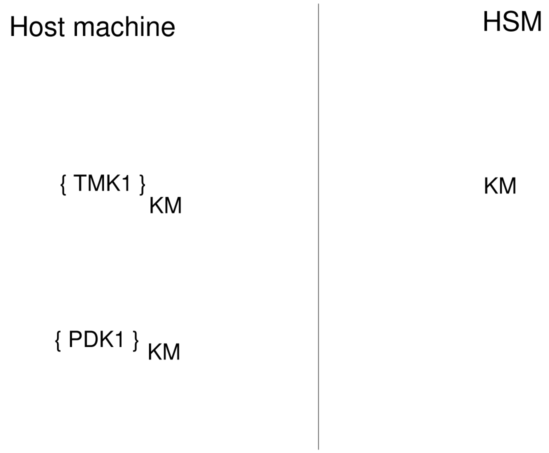
- Manufacturers include IBM, VISA, nCipher, Thales, Utimaco, HP
- Cost around $20 000

A Word About Your PIN

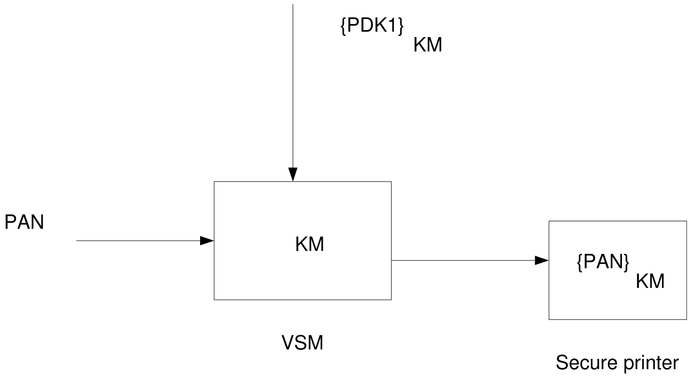
IBM method: IPIN derived by

Write account number (PAN) as 0000AAAAAAAAAAAA
3DES encrypt under a PDK (PIN Derivation Key),
decimalise first digits
PIN $=$ IPIN $+$ Offset (modulo 10 each digit)
Offset NOT secure!

API attack example: VSM (Bond, 2001)

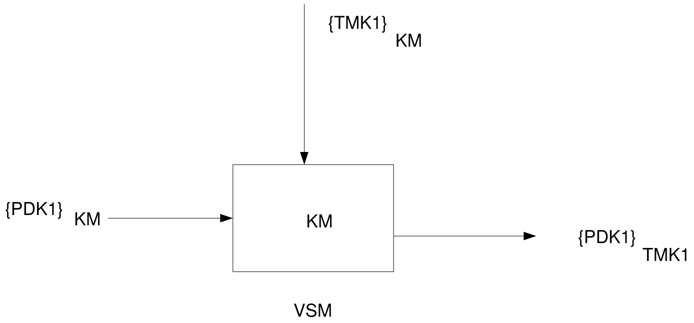


Example: Print Customer PIN



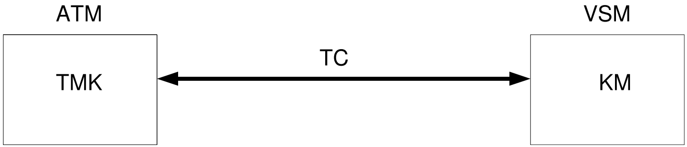
$$\begin{array}{llll} \text{Host} & \rightarrow & \text{HSM} & : \text{PAN}, \{\ \text{PDK1}\ \}_{\mathrm{Km}} \\ \text{HSM} & \rightarrow & \text{Printer} & : \{\ \text{PAN}\ \}_{\mathrm{PDK1}} \end{array}$$

Example: Send PDK to Terminal



$$\begin{array}{cccl} \text{Host} & \rightarrow & \text{HSM} & : \{\text{PDK1}\}_{\mathrm{Km}}, \{\text{TMK1}\}_{\mathrm{Km}} \\ \text{HSM} & \rightarrow & \text{Host} & : \{\text{PDK1}\}_{\mathrm{TMK1}} \end{array}$$

Terminal Comms Key



Managing Key Types

Host machine

$\{ \text{TMK1} \}_{\text{KM}}$

$\{ \text{PDK1} \}_{\text{KM}}$

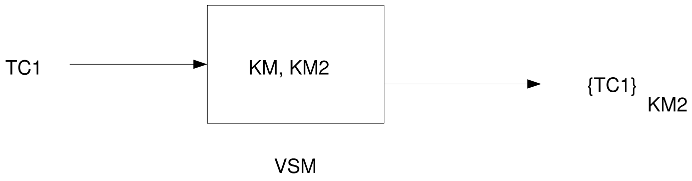
$\{ \text{TC1} \}_{\text{KM2}}$

VSM

KM

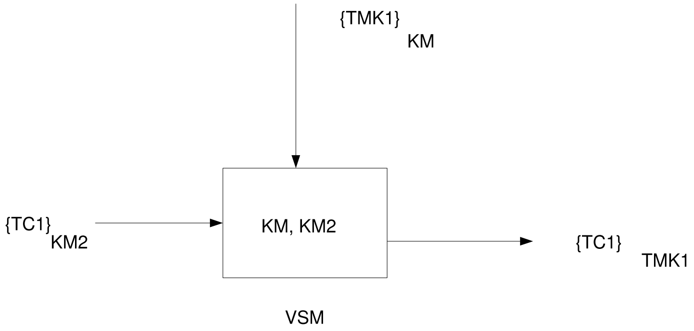
KM2

Example: Enter TC key



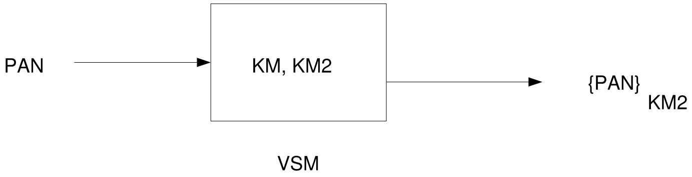
$$\begin{array}{llll} \text{Host} & \rightarrow & \text{HSM} & : & \text{TC} \\ \text{HSM} & \rightarrow & \text{Host} & : & \{\,\text{TC}\,\}_{\text{Km2}} \end{array}$$

Example: Send TC to Terminal



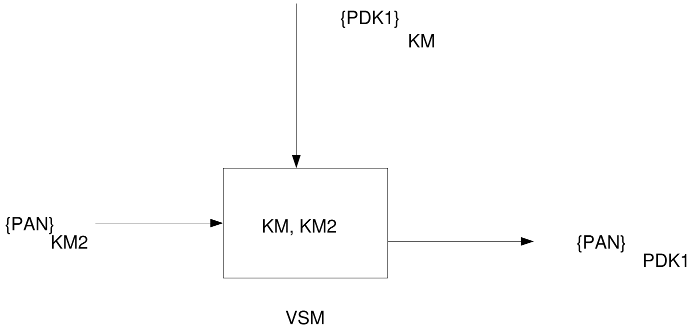
$$\begin{array}{lcccl} \text{Host} & \rightarrow & \text{HSM} & : & \{\ \text{TC}\ \}_{\text{Km2}},\ \{\ \text{TMK1}\ \}_{\text{Km}} \\ \text{HSM} & \rightarrow & \text{Host} & : & \{\ \text{TC}\ \}_{\text{TMK1}} \end{array}$$

Attack - Step 1



$$\begin{array}{lcll} \text{Spy} & \rightarrow & \text{HSM} & : \text{PAN} \\ \text{HSM} & \rightarrow & \text{Spy} & : \{\ \text{PAN}\ \}_{\text{Km2}} \end{array}$$

Attack - Step 2



$$\begin{array}{llll} \text{Spy} & \rightarrow & \text{HSM} & : \{ \text{PAN} \}_{\text{Km2}}, \{ \text{PDK1} \}_{\text{Km}} \\ \text{HSM} & \rightarrow & \text{Host} & : \{ \text{PAN} \}_{\text{PDK1}} \end{array}$$

Dolev-Yao Modelling

Following a seminal 1983 paper on formal analysis of security protocols:

Bitstrings modelled as terms in an abstract algebra
Cryptographic algorithms modelled as function on terms

Attacker controls network but cryptography is ‘perfect’

Attack modelled as derivation of secret term

Secrecy in general undecidable (see e.g. [Durgin et al. ’99])

Dolev-Yao Modelling 2

Atomic terms: $\mathsf{pdk1}, \mathsf{km}, \mathsf{km2}, \mathsf{pan}, \ldots$

Functions: $\{.\}_.$

Intruder rules:
e.g. $x, y \rightarrow \{x\}_y$

API rules:
e.g. $\{x\}_{km}, \{y\}_{km} \rightarrow \{x\}_y$

Semantics of model can be described by transition system

Attack Search

Start with ‘initial knowledge’ of intruder

e.g. $\mathsf{pan}, \{\mathsf{pdk1}\}_{\mathsf{km}}, \{\mathsf{tmk1}\}_{\mathsf{km}}$

apply rules
Note: intruder knowledge increases monotonically, no need to backtrack in search

Goal is to derive term $\{\mathsf{pan}\}_{\mathsf{pdk1}}$

Can automate search with model checker, theorem prover, or custom tool

Modelling With Horn Clauses

Form of rules:

$$P_1, \ldots, P_m \Rightarrow Q$$

can be written

$$\neg P_1 \vee \ldots \vee \neg P_n \vee Q$$

Note one positive literal (Q)
These restricted first-order clauses have a simpler theory and are easier to work with in practice.

Deduction with Horn clauses

Automated deduction usually uses (variants of) the *resolution* inference rule

$$\frac{\begin{array}{r}\neg P_1 \vee \ldots \vee \neg P_n \quad \vee \quad Q \\ \neg R_1 \vee \ldots \vee \neg R_n \quad \vee \quad S\end{array}}{(\neg P_1 \vee \ldots \vee \neg P_n \vee \neg R_2 \vee \ldots \vee \neg R_n \quad \vee \quad S)\theta}$$

where θ is most general unifier of R_1 and Q

Since resolution is refutation complete and first-order unification is decidable, we can search automatically for a proof of a negated conjecture (security property)

VSM Modelling

(only one predicate symbol $P(.)$, hence we omit it)

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

$+$ 7 more

VSM using First Order Theorem Provers

API and intruder modelled in 13 Horn clauses, 12 terms in initial knowledge

Appears as problem `SWV237` (`www.tptp.org`)

Attack found in $<$ 1 sec by several provers (Vampire, E,...)

VSM now has clear TC entry instruction disabled - problem `SWV238`

Only E can find a model (problem has no finite models)

Summary of First Half

- Security APIs as a paradigm
- Secure hardware needs a secure API
- Cash machine network and HSMs example
- APIs can be modelled in an abstract way following Dolev and Yao
- Tools such as First-Order Theorem Provers can be used to facilitate analysis

Next half

- A more complex secure hardware API:
  IBM 4758 CCA
- Modelling XOR
- More attacks
- Proving decidability of security for certain APIs
- Modern API examples

IBM 4758 CCA API

CCA Types - 1

The Common Cryptographic Architecture (CCA) API uses the same 'master key' architecture as the VSM

However, the (patented) type system is much richer

Before encrypting a working key, the master key is XORed against a 'control vector' indicating the type of the key

The control vectors are public values (they can be found in the programmers' manual), but the master key is secret

Control vectors can be composite, i.e. they may consist of a number of values XORed together

CCA Types - 2

Host machine

HSM

$\{ TC1 \}_{km + data}$

km

$\{ PDK1 \}_{km + pin}$

CCA API - Examples

Encrypt Data:

$$\begin{array}{llll} \text{Host} & \rightarrow & \text{HSM} & : \{ \text{d1} \}_{\text{km}\oplus\text{data}}, \text{message} \\ \text{HSM} & \rightarrow & \text{Host} & : \{ \text{message} \}_{\text{d1}} \end{array}$$

Verify PIN:

$$\begin{array}{llll} \text{Host} & \rightarrow & \text{HSM} & : \{ \text{PINBlock} \}_{\text{p1}}, \text{PAN}, \{ \text{pdk1} \}_{\text{km}\oplus\text{pin}}, \\ & & & \quad \text{OFFSET}, \{ \text{p1} \}_{\text{km}\oplus\text{ipinenc}} \\ \text{HSM} & \rightarrow & \text{Host} & : \text{yes/no} \end{array}$$

Bootstrapping

A common problem in the use of secure hardware

How to get the initial secrets onto the device (or encrypted by the device's master key) in a secure way?

A common solution is 'separation of duty': several members of staff are given individual parts of a secret.

Each individual part is worthless, so only collusion between several staff members can expose the secret.

Importing Key Parts

Separation of duty between e.g. 2 security officers

Key $k = k1 \oplus k2$

Host	$\rightarrow$	HSM	:	k1, TYPE
HSM	$\rightarrow$	Host	:	$\{ k1 \}_{km \oplus kp \oplus TYPE}$
Host	$\rightarrow$	HSM	:	$\{ k1 \}_{km \oplus kp \oplus TYPE}$, k2, TYPE
HSM	$\rightarrow$	Host	:	$\{ k1 \oplus k2 \}_{km \oplus TYPE}$

Usually used to import a ‘key encrypting key’ ($\{ KEK \}_{km \oplus imp}$)

Importing Encrypted Keys

Exported from another 4758 encrypted under KEK $\oplus$ TYPE

Key Import:

$$\begin{array}{lcll}
\text{Host} & \rightarrow & \text{HSM} & : \{\text{KEY1}\}_{\text{KEK}\oplus\text{TYPE}}, \text{TYPE}, \{\text{KEK}\}_{\text{km}\oplus\text{imp}} \\
\text{HSM} & \rightarrow & \text{Host} & : \{\text{KEY1}\}_{\text{km}\oplus\text{TYPE}}
\end{array}$$

Attack (Bond, 2001) (part 1)

PIN derivation key: $\{\ \text{pdk}\ \}_{\text{kek} \oplus \text{pin}}$

Have key part $\{\ \text{kek} \oplus \text{k2}\ \}_{\text{km} \oplus \text{imp} \oplus \text{kp}}$ for known k2

$$\begin{array}{llll} \text{Host} & \rightarrow & \text{HSM} & : & \{\ \text{kek} \oplus \text{k2}\ \}_{\text{km} \oplus \text{kp} \oplus \text{imp}},\ \text{k2} \oplus \text{pin} \oplus \text{data},\ \text{imp} \\ \text{HSM} & \rightarrow & \text{Host} & : & \{\text{kek} \oplus \text{pin} \oplus \text{data}\}_{\text{km} \oplus \text{imp}} \end{array}$$

Attack (Bond, 2001) (part 2)

Key Import

$$\begin{array}{llll} \text{Host} & \rightarrow & \text{HSM} & : \{\ \text{pdk}\ \}_{\text{kek}\oplus\text{pin}}, \text{data}, \\ & & & \{\ \text{kek} \oplus \text{pin} \oplus \text{data}\ \}_{\text{km}\oplus\text{imp}} \\ \text{HSM} & \rightarrow & \text{Host} & : \{\ \text{pdk}\ \}_{\text{km}\oplus\text{data}} \end{array}$$

Encrypt data

$$\begin{array}{llll} \text{Host} & \rightarrow & \text{HSM} & : \{\ \text{pdk}\ \}_{\text{km}\oplus\text{data}}, \text{pan} \\ \text{HSM} & \rightarrow & \text{Host} & : \{\ \text{pan}\ \}_{\text{pdk}}\ (= \text{PIN!}) \end{array}$$

Formal Modelling of the CCA

Encrypt data:

$$x, \{d1\}_{km \oplus data} \rightarrow \{x\}_{d1}$$

Extended intruder rules:

$$\begin{aligned} x, y &\rightarrow \{x\}_{y} \\ \{x\}_{y}, y &\rightarrow x \\ x, y &\rightarrow x \oplus y \end{aligned}$$

Also need

$$\begin{aligned} x \oplus (y \oplus z) &= (x \oplus y) \oplus z \\ x \oplus y &= y \oplus x \\ x \oplus x &= 0 \end{aligned}$$

Using FOTP

Problem `SWV234` in TPTP

11 rules, 15 items in knowledge set

However, only most recent version of Vampire can find the attack

‘Combinatorial explosion’ caused by XOR is the problem

Cannot verify fixes using TPTP provers (yet..)

Characterisation of API Class

Finite set of atoms (km, imp, data, pin, ...)

$$\text{XOR term} ::= \text{atom} \mid \text{atom} \oplus \text{XOR term}$$

$$\text{Encryption term} ::= \{\, \text{XOR term} \,\}_{\text{XOR term}}$$

$$\text{Well Formed Term} ::= \text{Encryption term} \mid \text{XOR term}$$

$$\text{Well Formed Rule} ::= \text{WFT}, \ldots, \text{WFT} \rightarrow \text{WFT}$$

Theorem

If:
R finite set of well-formed rules
S finite set of well-formed ground terms
u some ground well-formed term
Then:
$S \vdash_R u \iff S \vdash_R u$ using only well-formed terms.
Corollary:
The question of whether $S \vdash_R u$ is decidable

Proof Sketch

The theorem is proved by induction.
Conditions for API $\mathcal{R}$, initial knowledge S

- $\mathcal{R}$ contains the rule $x, y \to x \oplus y$;
- S contains 0 (the null element for XOR should always be known to an intruder).

Define function $t \mapsto \overline{t}$
e.g.
$\overline{\{a \oplus \{a\}_b\}_c \oplus \{c\}_b} = \{a\}_c \oplus \{c\}_b$.
Show that $t \mapsto \overline{t}$ preserves deducibility

Designing the Decision Procedure

Thanks to our theorem, we have a finite set of possible terms an attacker may learn

However, in a typical model (one key of each type) we may have 12 atoms (km, imp, data, pin, d1, p1, ...)

That means that there will be:
2^{12} possible unencrypted terms
2^{24} possible encrypted terms ($\{\,.\,\}_{.}$)

This is more than general purpose first-order theorem provers can deal with at present

Representation Change

Encode terms as integers by fixing an arbitrary ordering on the atoms:

		km	kp	kek	imp	exp	data	pin
kek $\oplus$ pin $\oplus$ data	$\rightarrow$	km	kp	kek	imp	exp	data	pin
19	$\leftarrow$	0	0	1	0	0	1	1

$\{$kek $\oplus$ pin $\oplus$ data$\}_{\text{km}\oplus\text{data}}$ is represented by

km	kp	kek	imp	exp	data	pin	km	kp	kek	imp	exp	data	pin
0	0	1	0	0	1	1	1	0	0	0	0	1	0

$$\downarrow$$

2498

Representation change - contd.

Each rule is now a partial function $f : \mathbb{N}^k \to \mathbb{N}$ for k inputs
e.g. $f_1 : x1, x2 \to x1 \oplus x2 \quad \equiv \quad x1, x2 \to x1 \oplus x2$

$$f_2 : [xkey|x], xtype, [xkek|q] \to [xkey|q \oplus xtype] \quad \mathrm{IF} \quad x = xkek \oplus xtype$$

Decision Procedure

Find the fixpoint:

1. Allocate sufficient memory for all possible terms
2. Store 1 in locations corresponding to initial knowledge, rest set to 0
3. Exhaustively apply each rule, setting newly discovered terms to 1
4. Repeat 3 until an attack no new terms are discovered

Some optimisations used (noting new terms)
Details in [Cortier, Keighren, Steel, TACAS'07]
Now we can verify fixes..

IBM Recommendations

Published in response to Bond's attacks

1. Use asymmetric key crypto for key import – 2 officer protocol to generate key pair at destination, transfer public key to source – PKA_SYMMETRIC_KEY_IMPORT command
2. More access control – security officers access fewer commands
3. Procedural controls to check entered key parts

2 and 3 verified in a few seconds, but 1 has a simple attack..

Attack on 1

$\{\mathsf{kek.IMP}\}_{\mathsf{PK}} \rightarrow \{\mathsf{kek}\}_{\mathsf{KM}\oplus\mathsf{IMP}}$ **PKA Symmetric Key Import**

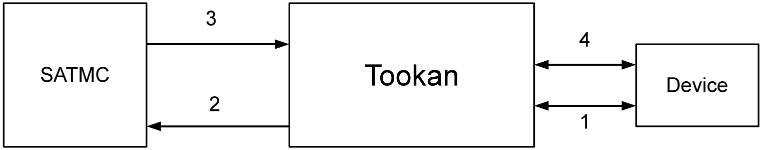
$\{\mathsf{k.EXP}\}_{\mathsf{PK}} \rightarrow \{\mathsf{k}\}_{\mathsf{KM}\oplus\mathsf{EXP}}$ **PKA Symmetric Key Import**

$\{\mathsf{pdk}\}_{\mathsf{kek}\oplus\mathsf{PIN}}, \mathsf{PIN}, \{\mathsf{kek}\}_{\mathsf{KM}\oplus\mathsf{IMP}} \rightarrow \{\mathsf{pdk}\}_{\mathsf{KM}\oplus\mathsf{PIN}}$ **Key Import**

$\{\mathsf{pdk}\}_{\mathsf{KM}\oplus\mathsf{PIN}}, \mathsf{PIN}, \{\mathsf{k}\}_{\mathsf{KM}\oplus\mathsf{EXP}} \rightarrow \{\mathsf{pdk}\}_{\mathsf{k}\oplus\mathsf{PIN}}$ **Key Export**

TOOKAN

‘Tool for cryptoKi Analysis’ `tookan.gforge.inria.fr`



1 = reverse engineering, 2= model sent to *model-checker* SATMC, 3 = attack found, 4 = attack executed on device

New Security APIs - 1

Privacy

- How to be sure an API satisfies a published privacy policy [May et. al CSFW '06]
- Smartmeters - how to be sure metering doesn't reveal privacy sensitive information [Molina-Markham et al, FC 2012]
- W3C Device APIs Working Group Privacy Rulesets

New Security APIs - 2

- OS APIs - model checking GRLinux policies [Bugliesi et al, CSF '12]
- Capsicum - model checking concurrency API vulnerabilities in the filesystem API [Watson & Anderson ASA'10]

Summary

- Security APIs critical part of system design
  - needs to be given attention
- Have seen attacks on VSM, CCA
- Formal analysis can find attacks and verify fixes
  - designing for easy analysis is a sound idea
- Theoretical results: Security in general undecidable, but can recover decidability in some cases
- Practical tools: Tools like Tookan can automatically test security of APIs
- Security API paradigm now spreading

Further reading

R. Anderson, *Security Engineering*, Wiley (2nd Ed.)

M. Bond and R. Anderson, *API Level Attacks on Embedded Systems*, IEEE Computer Magazine, 2001

D. Longley and S. Rigby, *An Automatic Search for Security Flaws in Key Management Schemes*, Computers and Security, 1992,

V. Cortier, G. Keighren and G. Steel, *Automatic Analysis of the Security of XOR-based Key Management Schemes*, TACAS '07

The Analysis of Security APIs Workshop, `http://www.lsv.ens-cachan.fr/~steel/asa/`