Defending the Bank with a Proof Assistant

J. Courant J.-F. Monin

VERIMAG
Grenoble, France

Workshop on Issues in the Theory of Security, 2006

Outline

What is a Security API?

API $=$ Application Programmer Interface

- Context : unsafe world accessing a secure application
- Aim: enforcing a security policy

Examples:

- RSA Laboratories Cryptographic Token Interface Standard (PKCS#11)
- Visa Security Module
- IBM 4758 cryptographic processor (used in cash-machines)

Outline

IBM 4758 Overview

- Tamper-resistant secure processor
- To be used as a PCI extension card plugged into a standard PC (typically in ATMs)
- Small memory
- Basically, stores only a master key KM.
- Storing a sensitive data x:
  - Encrypt it by $t \oplus KM$, with t describing the type of x
  - Keep it on the PC
- Types used for controlling acceptable operations
- Well-defined API: Common Cryptographic Architecture (CCA)

Excerpts from the CCA API

Importing a datum encrypted by an importation key

$$t, \{k\}_{IMP \oplus KM}, \{x\}_{t \oplus k} \rightarrow \{x\}_{t \oplus KM} \quad (1)$$

Excerpts from the CCA API

Importing a datum encrypted by an importation key

$$t, \{k\}_{IMP \oplus KM}, \{x\}_{t \oplus k} \longrightarrow \{x\}_{t \oplus KM} \qquad (1)$$

Encrypting/Decrypting applicative data

$$x, \{k\}_{DATA \oplus KM} \longrightarrow \{x\}_{k} \qquad (2)$$

$$\{x\}_{k}, \{k\}_{DATA \oplus KM} \longrightarrow x \qquad (3)$$

Excerpts from the CCA API

Importing a datum encrypted by an importation key

$$t, \{k\}_{IMP \oplus KM}, \{x\}_{t \oplus k} \rightarrow \{x\}_{t \oplus KM} \quad (1)$$

Encrypting/Decrypting applicative data

$$x, \{k\}_{DATA \oplus KM} \rightarrow \{x\}_k \quad (2)$$

$$\{x\}_k, \{k\}_{DATA \oplus KM} \rightarrow x \quad (3)$$

Adding to a key parts:

$$x, y, \{z\}_{x \oplus KP \oplus KM} \rightarrow \{z \oplus y\}_{x \oplus KP \oplus KM} \quad (4)$$

Excerpts from the CCA API

Importing a datum encrypted by an importation key

$$t, \{k\}_{IMP \oplus KM}, \{x\}_{t \oplus k} \rightarrow \{x\}_{t \oplus KM} \qquad (1)$$

Encrypting/Decrypting applicative data

$$x, \{k\}_{DATA \oplus KM} \rightarrow \{x\}_{k} \qquad (2)$$

$$\{x\}_{k}, \{k\}_{DATA \oplus KM} \rightarrow x \qquad (3)$$

Adding to a key parts:

$$x, y, \{z\}_{x \oplus KP \oplus KM} \rightarrow \{z \oplus y\}_{x \oplus KP \oplus KM} \qquad (4)$$

Importing a key part as a key:

$$x, y, \{z\}_{x \oplus KP \oplus KM} \rightarrow \{z \oplus y\}_{x \oplus KM} \qquad (5)$$

Outline

A security flaw of CCA

Different usages of $\oplus$

- Tagging encrypted values with types
- Building a secret key out of several pieces.

Bond & Anderson:

- Unauthorized type cast attack (2001)
- Can be found by running Otter, an automated theorem prover (2005)
- Proposed fix: replacing $\{x\}_{t \oplus k}$ by $\{x\}_{H(t,k)}$ (2001), with H a one-way function.

Would this fix secure CCA?

Outline

Modeling the problem

Data are first-order terms (Dolev-Yao model):

- Function symbols:
  - $\{.\}_.$, $\oplus$, H
  - Public constants : 0, $DATA$, IMP, PIN, K_3
  - Private constants : KM, P
- Predicate symbols: $=$, $known$.

API calls can be seen as propositions:

$$\forall t\ k\ \left(\begin{array}{l} \quad known(t) \\ \wedge known(\{k\}_{H(IMP,KM)}) \\ \wedge known(\{x\}_{H(t,k)}) \end{array}\right) \rightarrow known(\{x\}_{H(t,KM)})$$

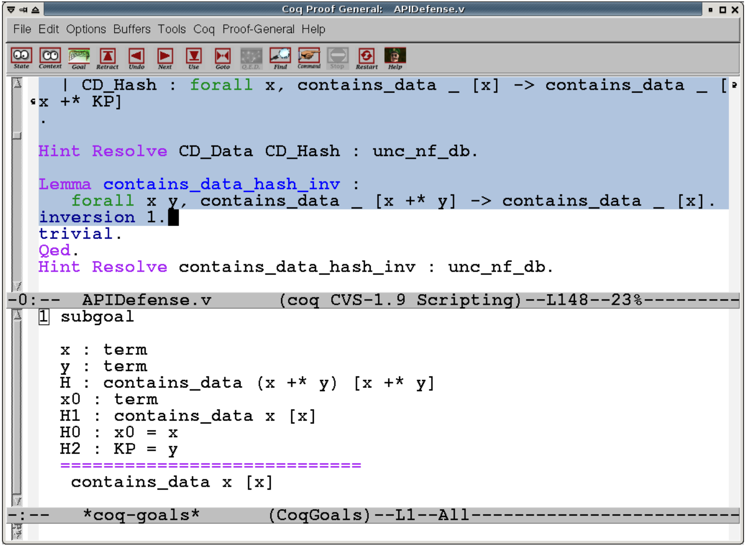
Outline

A case for the Coq proof assistant

- Secret leaks = finite seq. of steps leading to leaks
- No secret leaks = all sequences of steps are safe.
- First-order not enough: Induction needed.

The Coq proof assistant features:

- Inductive definitions of propositions, sets and proofs.
- Proofs partially automated (only).
- Interactive proof sessions (kind of nitpicking colleague you have to convince).
- Ability to record and replay proof sessions (proof scripts).
- Safety:
  - Solid metatheoretical foundations
  - Small kernel rechecking all proofs


Coq Proof General: APIDefense.v
File Edit Options Buffers Tools Coq Proof-General Help
| CD_Hash : forall x, contains_data _ [x] -> contains_data _ [x +* KP]
.
Hint Resolve CD_Data CD_Hash : unc_nf_db.
Lemma contains_data_hash_inv :
forall x y, contains_data _ [x +* y] -> contains_data _ [x].
inversion 1.
trivial.
Qed.
Hint Resolve contains_data_hash_inv : unc_nf_db.
-0:-- APIDefense.v (coq CVS-1.9 Scripting)--L148--23%--------
1 subgoal
x : term
y : term
H : contains_data (x +* y) [x +* y]
x0 : term
H1 : contains_data x [x]
H0 : x0 = x
H2 : KP = y
============================
contains_data x [x]
-:-- *coq-goals* (CoqGoals)--L1--All------------------------

Outline

Security APIs
- What is a Security API?
- The IBM 4758
- Our concerns

Our Contribution
- The model we used
- A case for the Coq proof assistant
- Sketch of the proof
- Lessons Learned
- Conclusion and Future Work

Modelization of *known*

Inductive proposition closed by :

Modelization of *known*

Inductive proposition closed by :

1. Initially known: $known(0), \ldots$

Modelization of *known*

Inductive proposition closed by :

1. Initially known: $known(0), \ldots$
2. Offline computations:
   $\forall x\ y\ known(x) \wedge known(y) \rightarrow known(\{x\}_y)$

Modelization of *known*

Inductive proposition closed by :

1. Initially known: $known(0), \ldots$
2. Offline computations:
   $\forall x\ y\ known(x) \wedge known(y) \rightarrow known(\{x\}_y)$
3. Algebraic reasoning:
   - $\forall x\ y\ x =_{\mathcal{T}} y \wedge known(x) \rightarrow known(y)$
   - algebraic laws for $\oplus$ (0 is neutral, commutativity, ...)

Modelization of *known*

Inductive proposition closed by :

1. Initially known: $known(0), \ldots$
2. Offline computations:
   $\forall x\ y\ known(x) \wedge known(y) \rightarrow known(\{x\}_y)$
3. Algebraic reasoning:
   - $\forall x\ y\ x =_{\mathcal{T}} y \wedge known(x) \rightarrow known(y)$
   - algebraic laws for $\oplus$ (0 is neutral, commutativity, ...)
4. CCA API calls:

$$\forall t\ k \left(\begin{array}{l} \ \ known(t) \\ \wedge known(\{k\}_{H(IMP,KM)}) \\ \wedge known(\{x\}_{H(t,k)}) \end{array} \right) \rightarrow known(\{x\}_{H(t,KM)})$$

Sketch of the proof

Introduction of inductive predicate *unc*. Intuitively:

$$unc(x) \stackrel{\text{def}}{=\!=} x \text{ can safely be revealed}$$

Example of a constructor :

$$\forall x\ y\ unc(x) \wedge unc(y) \rightarrow unc(\{x\}_y)$$

Requirements for *unc* :

- Decidable : $|\text{conclusion}| > |\text{size of premisses}|$ for all constructor
- $\forall x\ known(x) \rightarrow unc(x)$ (by induction over *known*)
- $\forall x\ y\ unc(x) \rightarrow \neg private(x)$

Outline

Security APIs
- What is a Security API?
- The IBM 4758
- Our concerns

Our Contribution
- The model we used
- A case for the Coq proof assistant
- Sketch of the proof
- Lessons Learned
- Conclusion and Future Work

Lessons Learned

Use of proof assistant proved invaluable:

- Right definition for *unc* difficult to find. Example of naively natural but wrong rule:

$$\forall x\ y\ unc(x) \rightarrow unc(\{x\}_y)$$

- Right definition found by trial and error.
- After a change, all proofs must be checked again
- Manually: tedious, errors likely to stay unnoticed
- Coq tells you which proofs do not pass any longer

Outline

Conclusions

- Coq increases your confidence in your proofs.
- Coq helps *finding* proofs (not just verifying them).
- Reasonning on algebraic properties (of $\oplus$) painful with Coq as well

Future works:

- More realistic modelization of the API
- Methodology and tools
  - Algebraic reasoning over $\oplus$ as an independent library
  - More automation in Coq
  - Full automation possible?
  - Provide a language for describing APIs and their properties
- Modeling computational properties