# Proving Secrecy Properties of an API

J. Courant J.-F. Monin

VERIMAG
Grenoble, France

ARTIST2 Workshop, May the 18th, 2006



Security APIs
What is a Security API?
Overview of the IBM 4758
Excerpts from the CCA API
A security flaw of CCA
Challenge

Certifying security properties
Modeling the problem
Where usual tools fail
Solutions
Modeling API calls
Sketch of the proof

Conclusions
Lessons Learned
Future Work

Appendix
A security flaw of CCA

# What is a Security API?

API = Application Programming 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)



Security APIs
What is a Security API?
Overview of the IBM 4758
Excerpts from the CCA API
A security flaw of CCA
Challenge
Certifying security properties
Modeling the problem
Where usual tools fail
Solutions
Modeling API calls
Sketch of the proof
Conclusions
Lessons Learned
Future Work
Appendix
A security flaw of CCA

# Overview of the IBM 4758

- Tamper-resistant secure processor
- On a PCI extension card plugged into a standard PC (typically in ATMs)
- Well-defined API: Common Cryptographic Architecture (CCA)
- Small memory
- Basically, stores only a master key $KM$.
- Storing a sensitive data $x$:
  - Request $\{x\}_{t \oplus KM}$, with $t$ describing the type of $x$
  - Keep it on the PC
- Knowledge of $\{x\}_{t \oplus KM}$ gives you
  - An "abstract" object $x$
  - Only the 4758 can apply operations on it
  - Allowed operations depend on type $t$



Security APIs
What is a Security API?
Overview of the IBM 4758
Excerpts from the CCA API
A security flaw of CCA
Challenge
Certifying security properties
Modeling the problem
Where usual tools fail
Solutions
Modeling API calls
Sketch of the proof
Conclusions
Lessons Learned
Future Work
Appendix
A security flaw of CCA

# Excerpts from the CCA API

Encrypting/Decrypting applicative data

$$x, \{k\}_{DATA \oplus KM} \to \{x\}_k \qquad (1)$$

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



Security APIs
What is a Security API?
Overview of the IBM 4758
Excerpts from the CCA API
A security flaw of CCA
Challenge
Certifying security properties
Modeling the problem
Where usual tools fail
Solutions
Modeling API calls
Sketch of the proof
Conclusions
Lessons Learned
Future Work
Appendix
A security flaw of CCA

# Excerpts from the CCA API

Encrypting/Decrypting applicative data

$$x, \{k\}_{DATA \oplus KM} \rightarrow \{x\}_k \quad (1)$$

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

Data of type *IMP* can be used as a key for importing data:

$$t, \{k\}_{IMP \oplus KM}, \{x\}_{t \oplus k} \rightarrow \{x\}_{t \oplus KM} \quad (3)$$



Security APIs
What is a Security API?
Overview of the IBM 4758
Excerpts from the CCA API
A security flaw of CCA
Challenge
Certifying security properties
Modeling the problem
Where usual tools fail
Solutions
Modeling API calls
Sketch of the proof
Conclusions
Lessons Learned
Future Work
Appendix
A security flaw of CCA

# 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 leading to a leakage (2001)
- Can be found by running Otter, an automated theorem prover (2005)
- Proposed fix (2001): replacing occurences of $x \oplus y$ by $H(x, y)$ with $H$ a one-way function. Conjectured to fix the flaw.



Security APIs
What is a Security API?
Overview of the IBM 4758
Excerpts from the CCA API
A security flaw of CCA
Challenge
Certifying security properties
Modeling the problem
Where usual tools fail
Solutions
Modeling API calls
Sketch of the proof
Conclusions
Lessons Learned
Future Work
Appendix
A security flaw of CCA

# Challenge

How can we prove their conjecture?
They claim:

- API are similar to cryptographic protocol
- But the surface of attack is much larger
- Hence it is doubtful one can apply usual cryptographic protocol tools



Security APIs
What is a Security API?
Overview of the IBM 4758
Excerpts from the CCA API
A security flaw of CCA
Challenge
Certifying security properties
Modeling the problem
Where usual tools fail
Solutions
Modeling API calls
Sketch of the proof
Conclusions
Lessons Learned
Future Work
Appendix
A security flaw of CCA

# Challenge

How can we prove their conjecture?
They claim:

- API are similar to cryptographic protocol
- But the surface of attack is much larger
- Hence it is doubtful one can apply usual cryptographic protocol tools

Challenging their claim:

- Modelize the problem as for cryptographic protocols
- See why usual tool fails
- Prove it anyway



Security APIs
What is a Security API?
Overview of the IBM 4758
Excerpts from the CCA API
A security flaw of CCA
Challenge
Certifying security properties
Modeling the problem
Where usual tools fail
Solutions
Modeling API calls
Sketch of the proof
Conclusions
Lessons Learned
Future Work
Appendix
A security flaw of CCA

# Modeling the problem

- Data are first-order terms (Dolev-Yao model)
- Only predicate symbol: *known*.

API as propositions:

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

Secrecy as logical entailment:

$r_1, \ldots, r_n \vdash t \iff$ a combination calls in $r_1, \ldots, r_n$ reveals $t$

Close to well-known cryptographic protocol modelizations.




Security APIs
What is a Security API?
Overview of the IBM 4758
Excerpts from the CCA API
A security flaw of CCA
Challenge

Certifying security properties
Modeling the problem
Where usual tools fail
Solutions
Modeling API calls
Sketch of the proof

Conclusions
Lessons Learned
Future Work

Appendix
A security flaw of CCA

# Where usual tools fail

All first-order automated theorem provers fail:

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

Hermes (VERIMAG) fails:

- Non atomic keys are essential here
- Hermes can not handle them

ProVerif (Bruno Blanchet, ENS) seems to fail:

- Encrypted data can be used as keys
- Composing rules leads to an ever growing pattern of chain $\{k_1\}_{IMP \oplus k_2}, \ldots, \{k_n\}_{IMP \oplus k_{n+1}}$
- ProVerif loops (it does not generalize the pattern)



Security APIs
What is a Security API?
Overview of the IBM 4758
Excerpts from the CCA API
A security flaw of CCA
Challenge
Certifying security properties
Modeling the problem
Where usual tools fail
Solutions
Modeling API calls
Sketch of the proof
Conclusions
Lessons Learned
Future Work
Appendix
A security flaw of CCA

# Solutions

- First solution: hand-written proof. Tedious and error-prone.
- Second solution: use a general-purpose proof assistant handling inductive proofs.
  Formalization done within the Coq proof assistant:
  - terms $\longrightarrow$ inductive type
  - *known* $\longrightarrow$ inductive predicate

  Advantages of Coq:
  - Expressive formalism (inductive types & proofs)
  - Dependable:
    - Metatheory under control
    - Small kernel (De Bruijn criterion)



Security APIs
What is a Security API?
Overview of the IBM 4758
Excerpts from the CCA API
A security flaw of CCA
Challenge
Certifying security properties
Modeling the problem
Where usual tools fail
Solutions
Modeling API calls
Sketch of the proof
Conclusions
Lessons Learned
Future Work
Appendix
A security flaw of CCA

# Modeling API calls

*known*: Inductive proposition closed by



Security APIs
What is a Security API?
Overview of the IBM 4758
Excerpts from the CCA API
A security flaw of CCA
Challenge
Certifying security properties
Modeling the problem
Where usual tools fail
Solutions
Modeling API calls
Sketch of the proof
Conclusions
Lessons Learned
Future Work
Appendix
A security flaw of CCA

# Modeling API calls

*known*: Inductive proposition closed by

1. Initially known: $known(DATA)$, ...



Security APIs
What is a Security API?
Overview of the IBM 4758
Excerpts from the CCA API
A security flaw of CCA
Challenge
Certifying security properties
Modeling the problem
Where usual tools fail
Solutions
Modeling API calls
Sketch of the proof
Conclusions
Lessons Learned
Future Work
Appendix
A security flaw of CCA

# Modeling API calls

*known*: Inductive proposition closed by

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



Security APIs
What is a Security API?
Overview of the IBM 4758
Excerpts from the CCA API
A security flaw of CCA
Challenge
Certifying security properties
Modeling the problem
Where usual tools fail
Solutions
Modeling API calls
Sketch of the proof
Conclusions
Lessons Learned
Future Work
Appendix
A security flaw of CCA

# Modeling API calls

*known*: Inductive proposition closed by

1. Initially known: $known(DATA), \ldots$
2. Offline computations:
   $\forall x\ y\ known(x) \wedge known(y) \rightarrow known(\{x\}_y)$
3. CCA API calls:

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



Security APIs
What is a Security API?
Overview of the IBM 4758
Excerpts from the CCA API
A security flaw of CCA
Challenge
Certifying security properties
Modeling the problem
Where usual tools fail
Solutions
Modeling API calls
Sketch of the proof
Conclusions
Lessons Learned
Future Work
Appendix
A security flaw of CCA

# Sketch of the proof

Introduction of inductive predicate *unc*. Intuitively:

$$unc(x) \stackrel{\text{def}}{=} \text{revealing } x \text{ is safe}$$

Example of a constructor :

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

Requirements for *unc* :

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



Security APIs
What is a Security API?
Overview of the IBM 4758
Excerpts from the CCA API
A security flaw of CCA
Challenge
Certifying security properties
Modeling the problem
Where usual tools fail
Solutions
Modeling API calls
Sketch of the proof
Conclusions
Lessons Learned
Future Work
Appendix
A security flaw of CCA

# Lessons Learned

- API modelization similar to cryptographic protocol modelization.
- But CCA API challenges existing automated tools.
- 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, Coq tells you which proofs do not pass any longer
  - Therefore Coq helps *finding* proofs (not just verifying them).



Security APIs
What is a Security API?
Overview of the IBM 4758
Excerpts from the CCA API
A security flaw of CCA
Challenge
Certifying security properties
Modeling the problem
Where usual tools fail
Solutions
Modeling API calls
Sketch of the proof
Conclusions
Lessons Learned
Future Work
Appendix
A security flaw of CCA

# Future Work

- More realistic modelization of the API
- Methodology and tools
  - More automation in Coq
  - Full automation possible?
  - Provide a language for describing APIs and their properties
- Modeling computational properties



Security APIs
What is a Security API?
Overview of the IBM 4758
Excerpts from the CCA API
A security flaw of CCA
Challenge
Certifying security properties
Modeling the problem
Where usual tools fail
Solutions
Modeling API calls
Sketch of the proof
Conclusions
Lessons Learned
Future Work
Appendix
A security flaw of CCA

# A security flaw of CCA: Background

- Key *KEK* used to import data from bank servers (type *IMP*)
- Build as three shares $K_1$, $K_2$ and $K_3$ st $KEK = K_1 \oplus K_2 \oplus K_3$.
- Secrecy of *KEK* supposed to resist interception of two of them.



Security APIs
What is a Security API?
Overview of the IBM 4758
Excerpts from the CCA API
A security flaw of CCA
Challenge
Certifying security properties
Modeling the problem
Where usual tools fail
Solutions
Modeling API calls
Sketch of the proof
Conclusions
Lessons Learned
Future Work
Appendix
A security flaw of CCA

# Security flaw of CCA: executive summary

Normal operation: one successively build

- $\{K_1\}_{IMP \oplus KP \oplus KM}$
- $\{K_1 \oplus K_2\}_{IMP \oplus KP \oplus KM}$
- $\{K_1 \oplus K_2 \oplus K_3\}_{IMP \oplus KM} = \{KEK\}_{IMP \oplus KM}$

Interception of $K_3$ dramatic:

- Apply “key part import completing” to $K_3 \oplus PIN \oplus DATA$
- Thus you get $\{KEK \oplus PIN \oplus DATA\}_{IMP \oplus KM}$
- When you get $\{P\}_{PIN \oplus KEK}$, pretend it has type *DATA*: $\{P\}_{PIN \oplus KEK} = \{P\}_{DATA \oplus (KEK \oplus PIN \oplus DATA)}$
- Import it and get $\{P\}_{DATA \oplus KM}$
- Key $P$ now an applicative data. Enjoy!



Security APIs
What is a Security API?
Overview of the IBM 4758
Excerpts from the CCA API
A security flaw of CCA
Challenge
Certifying security properties
Modeling the problem
Where usual tools fail
Solutions
Modeling API calls
Sketch of the proof
Conclusions
Lessons Learned
Future Work
Appendix
A security flaw of CCA