Analysis of the IBM CCA Security API Protocols in Maude-NPA

Antonio González-Burgueño[1] Sonia Santiago[1]
Santiago Escobar[1] Catherine Meadows[2] José Meseguer[3]

[1] Universitat Politècnica de València, Spain

[2] Naval Research Laboratory, Washington DC, USA

[3] University of Illinois at Urbana-Champaign, USA

What This Talk is About

- Applying automated crypto protocol analysis methods to Cryptographic Application Programmer Interfaces (Crypto APIs)
  - Functionality provided by secure device for applications that run on it
  - API enables applications to perform operations that they need to do
  - API must also prevent application from performing operations that it should not do
    - E.g., any operation that results in its getting a key in the clear
- API's are like crypto protocols
  - Rules for communicating using cryptography
- API's are like standards
  - Documentation focuses on interoperability and guidance for implementation
  - Reasons for security decisions often not recorded, or recorded incompletely
  - Makes it easier for problems to creep in
- How well can automated techniques developed for cryptographic protocol analysis work for API's?

What We Focus On

- Concentrate on a particular API, IBM 4758 Common Cryptographic Architecture (CCA)
  - Has been extensively analyzed, but still remains a challenge
  - Its extensive use of exclusive-or makes analysis vulnerable to state explosion
  - Has required development of special purpose techniques, augmented by manual input
- We apply a general purpose tool, Maude-NPA to the analysis of different versions of CCA
- Discuss results, and lessons learned
  - In particular, discuss directions for future research

Outline

1. Background on Formal Crypto Protocol Analysis
2. IBM CCA
3. Formal Analysis of CCA and CCA-Like Protocols
4. Maude-NPA Analysis of IBM CCA
5. Conclusions

Outline

1. Background on Formal Crypto Protocol Analysis
2. IBM CCA
3. Formal Analysis of CCA and CCA-Like Protocols
4. Maude-NPA Analysis of IBM CCA
5. Conclusions

"Dolev-Yao" Model for Automated Cryptographic Protocol Analysis

- Start with a term algebra representing messages sent, constructed from a signature of function symbols and variables
- For each role in the protocol, give a program describing how a principal executing that role sends and receives messages
- Give a set of inference rules describing the deductions an intruder can make
  - E.g. if intruder knows K and $e(K,M)$, can deduce M
- Assume that all messages go through intruder who can
  - Stop or redirect messages
  - Alter messages
  - Create messages from already sent messages using inference rules
- Decision problems for security NP-complete w. bounded sessions, undecidable with unbounded sessions
- Tools have been developed that behave well with respect to unbounded sessions in many cases

Dolev-Yao With Equational Theories

- Many cryptoalgorithms satisfy equations that can
    - Describe the necessary properties of the cryptoagorithms (e.g. $d(K, e(K, M)) = M$)
    - Describe the properties of the primitives the cryptosystems are based on (e.g. Abelian groups)
- This can be represented by adding an equational theory to the term algebra
- General complexity results still the same
- Some tools that support equational theories
    - Proverif: Rewrite rules (orientiable equations)
    - OTFMC: Rewrite rules, and rewrite rules + AC
    - Tamarin: Diffie-Hellman, bilinear maps
    - Maude-NPA: Theories of the form $(R \uplus Ax)$, where R is rewrite rules, Ax regular set of axioms (e.g. AC)

What Maude-NPA Is

- A tool to find or prove the absence of attacks on cryptographic protocols using backwards search
- Analyzes infinite state systems
    - Active intruder
    - No abstraction or approximation of nonces
    - Unbounded number of sessions
- Designed to support as wide a class of equational theories as possible
- Makes use of unification modulo equational theories and narrowing to support backwards search
- Unification problem modulo E: given s and t find all substitutions to the variables in them that make them equal modulo E

Outline

Overview of CCA

- Provides commands that use encrypted keys to perform operations such as encryption and decryption
  - Without giving application access to keys in the clear
- Master key stored in security module, and used to encrypt working keys, stored in host computer
  - PIN Keys: Used for cryptographic operations on PINS
  - Key Encryption Keys: used to encrypt other working keys during transfer between security modules
    - Two types: import and export
  - Key Generation Keys
  - Data Encryption Keys

CCA Commands and Description

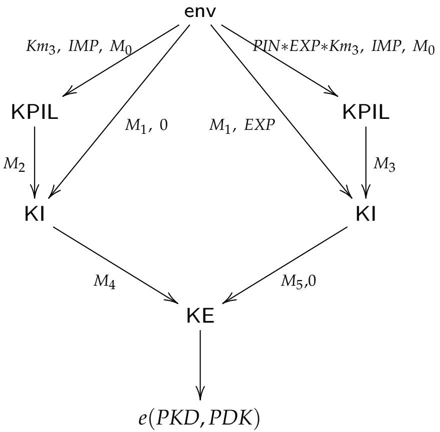
API command	**Description**
Encipher	X, $\{eK\}_{\{KM*DATA\}} \rightarrow \{X\}_{eK}$
Decipher	$\{X\}_{eK}$, $\{eK\}_{\{KM*DATA\}} \rightarrow X$
Key Export	$\{eK\}_{(KM*T\}}$, T, $\{ekek\}_{\{KM*EXP\}} \rightarrow \{eK\}_{(ekek*T)}$
Key Import	$\{eK\}_{(kek*T\}}$, T, $\{ekek\}_{\{KM*IMP\}} \rightarrow \{eK\}_{\{KM*T\}}$
Key Part Import First	km1, T $\rightarrow \{km1\}_{\{KM*KP*T\}}$
Key Part Import Middle	km2, $km1_{\{KM*KP*T\}}$, T $\rightarrow (km1 * km2)_{\{KM*KP*T\}}$
Key Part Import Last	km3, $km2_{\{KM*KP*T\}}$, T $\rightarrow (km2 * km3)_{\{KM*KP*T\}}$
Key Translate	$\{eK\}_{ekek1*T}$, T, $\{ekek1\}_{KM*IMP}$, $\{ekek2\}_{KM*EXP} \rightarrow \{eK\}_{(ekek2*T)}$
PKA Symmetric Key Import	$\{eK ; T\}_{PKA} \rightarrow \{eK\}_{KM*T}$

* stands for exclusive-or

Attack on CCA (Küsters and Truderung, 2011)

PKD a key, $kek = Km_1 * Km_2 * Km_3$,

M_0 and M_1 obtained by legitimate 3-part Key Import in which attacker contributed Km_3



$$M_0 = e(IMP * KP * KM, Km_1 * Km_2)$$

$$M_1 = e(PIN * kek, PDK)$$

$$M_2 = e(IMP * KM, PIN * kek)$$

$$M_3 = e(IMP * KM, PIN * EXP * kek)$$

$$M_4 = e(KM, PDK) \; M_5 = e(EXP * KM, PDK)$$

Outline

1. Background on Formal Crypto Protocol Analysis
2. IBM CCA
3. Formal Analysis of CCA and CCA-Like Protocols
4. Maude-NPA Analysis of IBM CCA
5. Conclusions

Past Work on Formal Analysis of CCA and CCA-like Protocols

- First attack (attacker can force conversion key type) found by Bond in 2000 as part of research project on formal analysis of crypto protocols
    - However, doesn't report use of automated techniques on CCA
- Decision procedures for protocols using exclusive-or
    - Chevalier et al. [LICS 2003], Comon-Lundh and Shmatikov [LICS 2003]
    - NP-complete for bounded session model (bound on number of calls to API in our case), undecidable for unbounded
- Cortier, Keighren, and Steel [TACAS 2007] develop exponential algorithm for checking a class of XOR-based key management schemes in unbounded session model, including CCA
    - Ran on CCA but completed analysis manually

Küsters and Truderung : XOR-Linear Protocols [J. Aut. Reasoning, 2011]

- XOR-linear protocols: protocols using XOR that can be analyzed using tools that don't support reasoning modulo AC, but do support reasoning modulo rewrite rules
    - Rewrite rule: an equation that can be given an orientation $\ell \rightarrow r$
    - An example: $d(K, e(K, M)) \rightarrow M$
- Küsters and Truderung convert CCA protocols to XOR-linear ones
- Then analyze using protocol analysis tool Proverif
    - Proverif supports large class of equational theories that can be oriented as rewrite rules
    - However, only limited support for AC, which can't be oriented
- XOR-linear conversion makes automated analysis possible, but still required
    - Hand conversion of CCA protocols to XOR-linear protocols
    - Hand verification that this conversion is sound and complete

Examples of XOR-linear Conversion

API command	Description
Key Part Import First	km1, T → $\{\text{km1}\}_{\{KM*KP*T\}}$
Key Part Import Middle	km2, $\text{km1}_{\{KM*KP*T\}}$, T → $(\text{km1} * \text{km2})_{\{KM*KP*T\}}$
Key Part Import Last	km3, $\text{km2}_{\{KM*KP*T\}}$, T → $(\text{km2} * \text{km3})_{\{KM*KP*T\}}$
Key Translate	$\{\text{eK}\}_{ekek1*T}$, T, $\{\text{ekek1}\}_{KM*IMP}$, $\{\text{ekek2}\}_{KM*EXP}$ → $\{\text{eK}\}_{(ekek2*T)}$

Table : Original specification of the protocol.

API command	Description
KPI-First + KPI-Add/Middle	→ $\{\text{km12}\}_{KM*KP*IMP}$
Key Part Import Last	x, T, KM * KP * T → $(\text{x})_{\{KM*T\}}$ x, IMP → $(\text{X} * \text{km12})_{\{KM*IMP\}}$
Key Translate	$\{\text{eK}\}_{ekek1*T}$, T, $\{\text{ekek1}\}_{KM*IMP}$ → transf(eK,T) transf(eK,T), $\{\text{ekek2}\}_{\{KM*EXP\}}$ → $\{\text{eK}\}_{(ekek2*T)}$

Table : Küesters and Truderung version

Why is This So Hard?

- From the complexity theory point of view, XOR-based APIs should be no more difficult to analyze than other types of protocols
  - NP-complete for bounded sessions, undecidable for unbounded
- But poses problems from a practical point of view
  - Large number of solutions to XOR-based unification problems
  - CCA doesn't do consistency and format checks like other protocols, leads to larger state space
- All this makes the problem harder

Outline

1. Background on Formal Crypto Protocol Analysis
2. IBM CCA
3. Formal Analysis of CCA and CCA-Like Protocols
4. Maude-NPA Analysis of IBM CCA
5. Conclusions

Our Approach to CCA: See What Can be Done With a General-Purpose Tool Supporting Equational Theories

- Specified the various CCA protocols, both original and XOR-linear versions in Maude-NPA
- Searched for state in which intruder learns `e(PDK,PDK)`
  - Once you get that, easy to use API to get `PDK`
- Made use of never patterns to guide search when necessary
  - Never patterns originally used to specify authentication, but can also be used to cut down search space
  - In general, sacrifice completeness
  - In practice can often avoid incompleteness or keep it within bounds

Versions of CCA We Analyzed, Looking for Küsters and Truderung Attack

- Some versions use access control, in those cases we assume attacker has access to only one role

CCA-0 Original version, vulnerable to attack

CCA-1 IBM added access control, no principal can access both `PKA Symmetric Key Import` and `Key Import`

  - CCA-1A (attacker has access to `Key Import`) and CCA-2A (atacker has access to `PKA Symmetric Key Import`

CCA-2 IBM adds role based access control

  - Five roles: A,B,C,D, and E
  - A and D don't have access to operations
  - Roles of Interest: CCA-2B, CCA-2C, and CCA-2E

Specifying Final States as Attack Patterns in Maude-NPA

```
eq ATTACK-STATE(1)
  = :: r ::
    [ nil, -(pk(b,a ; N)), +(pk(a,  N ; n(b,r))),  -(pk(b,n(b,r))) | nil ]
    ||  empty
    || nil
    || nil
    butNeverFoundAny *** for authentication
    (:: r' ::
    [ nil, +(pk(b,a ; N)), -(pk(a,  N ; n(b,r))) , +(pk(b,n(b,r)))| nil ]
    & S:StrandSet
    || K:IntruderKnowledge
    || M:SMsgList
    || G:GhostList)
  [nonexec] .
```

- An attack pattern specifies the form of an insecure state
- We want to show it's unreachable, or find a path to it (an attack)
- The first part gives the actions that should have happened
- The second part gives the actions that should not have happened

Using Never Patterns for Cutting Down Search Space

```
eq ATTACK-STATE(1)
  = :: r ::
    [ nil, -(pk(b,a ; N)), +(pk(a,  N ; n(b,r))),  -(pk(b,n(b,r))) | nil ]
    ||  empty
    || nil
    || nil
    butNeverFoundAny ***  for state space reduction
    S:StrandSet
    || (X ; W ; Y ; Z) inI , K:IntruderKnowledge
    || M:SMsgList
    || G:GhostList)
  [nonexec] .
```

- Maude-NPA will avoid all states in which the intruder learns a list of length four or greater
- May lose completeness, but if we believe intruder has no use for lists that long, can reduce time it takes to find attack

Two Types of Never Patterns That Provide Some Level of Guarantee

- Completeness-preserving
  - Unreachable attack pattern used as never pattern
- Attack-preserving
  - If we know a never pattern not needed in a particular attack, can use it as a never pattern and still find attack
- In CCA analysis used both completeness-preserving and attack-preserving never patterns
  - Minimized use of attack-preserving never patterns

Never Patterns Used in CCA Analysis

- Used same never patterns whenever we used them
- Generally, did not need them for XOR-linear protocols created by Küsters and Truderung, but did for others
- Completeness-preserving
  - `e(Key, KM * Msg) inI`
  - `e(IMP * KM, Type * Key) inI`
- Attack-preserving
  - `PDK inI`
  - Different forms of $(X * Y) \in \mathcal{I}$: $(Km1 * Y) \in \mathcal{I}$, $(Km2 * Y) \in \mathcal{I}$, $(PDK * Y) \in \mathcal{I}$, $(KM * Y) \in \mathcal{I}$, and $(Y * e(K, Y)) \in \mathcal{I}$ where `K` and `Y` are variables

Experimental Results

	Protocol	**States**	**Depth**	**Terminates**
1	CCA-0	291^*	7	Yes
2	CCA-0-XOR-linear	2495	7	Yes
3	CCA-1A	21^*	5	Yes
4	CCA-1B	48^*	6	Yes
5	CCA-1B-XOR-linear	1	2	Yes
6	CCA-2B	324^*	11	Yes
7	CCA-2C	131^*	6	No
8	CCA-2C-XOR-linear	105	4	No
9	CCA-2E	385^*	7	No

***This protocol analysis uses never patterns**

Discussion of Results

- In cases in which Maude-NPA failed to terminate, state explosion was *not* the reason
    - Rather, it was because Maude-NPA was taking a long time generating states
    - Further inspection shows Maude-NPA was discarding many states it generated due to application of its state space reduction techniques
    - Need way of applying state space reduction earlier in state generation process
- Maude-NPA on the most part did better with Küster-Truderung created XOR-linear protocols
    - Can a simplifying transformation that makes Maude-NPA analysis more tractable be developed?
    - Can it proved sound and complete and automated?

Outline

1. Background on Formal Crypto Protocol Analysis
2. IBM CCA
3. Formal Analysis of CCA and CCA-Like Protocols
4. Maude-NPA Analysis of IBM CCA
5. Conclusions

Conclusions

- Performed what is, to the best of our knowledge, the *first* analysis of an XOR-based crypto API using a general purpose crypto protocol analysis tool that supports reasoning about AC theories
- In doing so, performed a stress test on Maude-NPA
- Identified a number of bottlenecks and performance issues
- Introduced notion of *completeness-preserving* and *attack-preserving* never patterns