Computationally Sound Symbolic Protocol Analysis: Correspondence Theorems

Anupam Datta
CMU
Fall 2007-08
Protocol Analysis Techniques

Crypto Protocol Analysis

Formal Models
- Dolev-Yao (perfect cryptography)

Model Checking
- Murphi, AVISPA

Protocol Logics
- BAN, PCL

Process Calculi
- Applied II-calculus

Inductive Proofs
- Paulson, MSR

Computational Models
- Random oracle
- Probabilistic process calculi
- Probabilistic I/O automata
- Correspondence theorems
- Computational PCL
Cryptography: computational constructions

- Define actual algorithms operating on bit strings
- For example, RSA is defined as a triple of algorithms
  - Key generation: public key is (n,e), private key is d where n=pq for some large primes p,q; ed=1 mod (p-1)(q-1)
  - Encryption of m: $m^e \mod n$
  - Decryption of c: $c^d \mod n$

Formal model: abstract treatment of crypto

- Instead of defining cryptographic algorithms, simply say that they satisfy a certain set of properties
  - Given ciphertext, obtain plaintext if have exactly the right key
  - Cannot learn anything from ciphertext without the right key
Goal: Soundness of Formal Analysis

- Prove that properties assumed in the formal model are true for some cryptographic constructions
- Formal proofs must be sound in following sense:
  - For any attack in concrete (computational) model...
  - ...there is matching attack in abstract (formal) model
  - ...or else the concrete attack violates computational security of some cryptographic primitive
- If we don’t find an attack in the formal model, then no computational attack exists
  - More precisely, probability that a computational attack exists is negligible
Correspondence Theorems

- Protocol specification language
- Abstract execution model
- Security properties
- Soundness theorem
- Cryptographic security

Relation
Representative Papers

  - Symmetric encryption only, passive adversary

  - Public key encryption, signatures; active adversary

  - Public-key encryption, active adversary
Another Approach

High-level proof principles

PCL
- Syntax (Properties)
- Proof System (Proofs)

Computational PCL
- Syntax $\pm \Delta$
- Proof System $\pm \Delta$

Symbolic Model
- PCL Semantics
  (Meaning of formulas)

Cryptographic Model
- PCL Semantics
  (Meaning of formulas)

Soundness Theorem
(Induction)

[BPW, MW,...]

Unbounded # concurrent sessions

Polynomial # concurrent sessions
Representative Papers

  - Public-key encryption, active adversary

  - Signatures, key exchange, secure sessions

  - Symmetric and public key encryption; secrecy proof technique

  - Diffie-Hellman, symmetric and public key encryption, signatures.
Outline

- Define what it means for an encryption scheme to be secure against adaptive chosen-ciphertext attack in the multi-user setting
- Define a formal language for describing protocols
- Define concrete trace semantics for protocols
  - Actual execution traces of protocols obtained by instantiating nonces with bit strings, etc.
  - Traces include actions of the concrete adversary
- Show that almost every action of concrete adversary maps to an action of abstract adversary
  - This will follow from security of encryption scheme
Simple Protocol Language

◆ Pairing and encryption only

Term ::= Id | Identity
       Key | Public keys only
       Nonce |
       Pair | Ciphertext

Pair ::= (Term, Term)

Ciphertext ::= \{Term\}_Key

◆ Can write simple protocols with this syntax
  
  • Must describe valid computations of honest parties
    
    - For example, (B receives \{X\}_{pk(A)}, B sends \{X\}_{pk(B)}) is not valid because B can’t decrypt \{X\}_{pk(A)}
## Abstract vs. Concrete Execution

<table>
<thead>
<tr>
<th>Abstract execution</th>
<th>Concrete execution (with RSA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \rightarrow {N_a, A}_{pk(X)}$</td>
<td>$A \rightarrow \text{bits}(m_1^{ex} \mod n_x)$</td>
</tr>
<tr>
<td>$B \rightarrow {N_b, N_A}_{pk(A)}$</td>
<td>$B \rightarrow \text{bits}(m_2^{ea} \mod n_a)$</td>
</tr>
<tr>
<td>$A \rightarrow {N_b}_{pk(X)}$</td>
<td>$A \rightarrow \text{bits}(m_3^{ex} \mod n_x)$</td>
</tr>
</tbody>
</table>

- $(e_x, n_x)$ is responder’s RSA public key; $m_1$ is a number encoding a concatenation of a random bit string representing nonce $N_a$ and a fixed bit string representing A’s identity.
- $(e_a, n_a)$ is A’s RSA public key; $m_2$ is a number encoding a concatenation of a random bit string representing nonce $N_b$ and the bit string extracted from the message received by B.
- $(e_x, n_x)$ is responder’s RSA public key; $m_3$ is a number encoding the bit string extracted from the message received by A.
Abstract vs. Concrete Adversary

Abstract adversary
Fix some set of corrupt users $C$. Let $M$ be messages sent prior to some point in protocol execution.

- $M \in \text{know}(C,M)$
- If $t, t' \in \text{know}(C,M)$, then $(t, t') \in \text{know}(C,M)$
- If $(t, t') \in \text{know}(C,M)$, then $t, t' \in \text{know}(C,M)$
- If $t \in \text{know}(C,M)$, then $\{t\}_k \in \text{know}(C,M)$ for any key $k \in \text{Keys}$
  - Adversary can access any encryption oracle
- If $\{t\}_{k_i} \in \text{know}(C,M)$ and $A_i \in C$, then $t \in \text{know}(C,M)$
  - Adversary can decrypt only messages encrypted with public keys of corrupt users

Concrete adversary
Any polynomial-time algorithm (maybe probabilistic)
Equivalence of Concrete & Abstract

- Need to prove that concrete adversary cannot achieve more than the abstract adversary except with negligible probability
  - E.g., may guess secret key with negligible probability
- Show that almost any concrete trace is an implementation of some valid abstract trace
  - Concrete traces represent everything that the concrete adversary can achieve
  - If almost any concrete trace can be achieved by the abstract adversary, it is sufficient to look only at the abstract adversary when doing analysis
Concrete and Abstract Traces

- Representation function $R$ maps abstract symbols (nonces, keys, identities) to bit strings
  - Defines concrete “implementation” of abstract protocol
- Concrete trace is an implementation of an abstract trace if exists a representation function $R$ such that every message in concrete trace is a bit string instantiation of a message in abstract trace
  - Intuitively, concrete trace is an implementation if it is created by plugging bit strings in place of abstract symbols in the abstract trace
  - Denote implementation relation as $\leq$
How Can This Fail?

Suppose encryption scheme is malleable
- Can change encrypted message without decrypting
- Plain old RSA is malleable!

Abstract execution

\[ A \rightarrow \{N_a,A\}_{pk(B)} \]

Concrete execution (with RSA)

\[ A \rightarrow m^{eb} \mod n_b \]

\[ B \leftarrow \text{???} \]

\[ B \leftarrow m^{2eb} \mod n_b \]

Adversary intercepts and squares; B receives \( m^{2eb} \) instead of \( m^{eb} \)

There is no abstract operation corresponding to the action of concrete adversary!
Main Theorem

If the encryption scheme used in the protocol is IND-CCA secure, then for any concrete adversary $A_c$

$$\text{Prob}_{R_A, R_o} (\exists A_f : \text{traceset}(A_f, O^f) \leq \text{traceset}(A_c(R_A), O^c(R_o))) \geq 1 - \nu(\eta)$$

- Probability is taken over randomness of concrete adversary and encryption oracles (recall that concrete adversary may be probabilistic and public-key encryption must be probabilistic).

- There exists an abstract adversary such that...

- ...some abstract trace generated by abstract adversary interacting with abstract encryption and decryption oracles...

- is an implementation of...

- Every concrete trace generated by concrete adversary interacting with concrete encryption and decryption oracles...

- (note that concrete adversary and concrete encryption oracles are randomized)

- With overwhelming probability (i.e., dominated by any polynomial function of security parameter).
Proof Outline

1. For any (randomized) concrete adversary, construct the corresponding abstract adversary $A_f$
   - Abstract “model” of concrete adversary’s behavior
   - Guarantees $\text{traceset}(A_f, O_f) \leq \text{traceset}(A_c(R_A), O^c_c(R_O))$

2. Show that every action performed by constructed adversary is a valid action in the abstract model (with overwhelming probability)
   - Abstract adversary is only permitted to decrypt if he knows the right key (e.g., cannot gain partial info)
   - Prove: constructed adversary doesn’t do anything else
Constructing Abstract Adversary

1. Fix coin tosses (i.e., randomness) of honest participants and concrete adversary
2. This uniquely determines keys and nonces of honest participants
   - Use symbolic constants to give them names in abstract model
3. Every bitstring which is not an honest participant’s key or nonce is considered the adversary’s nonce and given some symbolic name

Must prove that every concrete action can be represented by a valid abstract action.
Reduction to IND-CCA

◆ Prove that every time concrete adversary does something that’s not permitted in abstract model, he breaks IND-CCA security of encryption
  • This can only happen with negligible probability
  • Therefore, with overwhelming probability every concrete action implements some valid abstract action

◆ We’ll prove this by constructing a simulator who runs the concrete adversary in a “box” and breaks IND-CCA exactly when the concrete adversary deviates from the abstract model
Overview of Reduction

CCA adversary interacts with encryption and decryption oracles according to rules of CCA game

Encryption and decryption oracles (CCA game)

CCA adversary interacts with encryption and decryption oracles as a subroutine

CCA adversary wins CCA game exactly when $A_f$ does something illegal in the abstract model

To do this, CCA adversary must be able to simulate protocol to $A_c$ (i.e., create an illusion for $A_c$ that he is interacting with the actual protocol)

Abstraction $A_f$

Concrete adversary $A_c$
What’s Illegal in Abstract Model?

- The only way in which constructed abstract adversary $A_f$ may violate rules of abstract model is by sending a term with some honest nonce $X$ that he could not have learned by abstract actions
  - $X$ cannot be learned from messages sent by honest parties by simple decryption and unpairing rules

- Case 1: If $A_f$ sends $X$ in plaintext, this means that concrete adversary managed to open encryption
  - Recall that $A_f$ is abstraction of concrete adversary $A_c$
  - If concrete adversary extracts a bitstring (abstracted as nonce $X$) from under encryption, he wins the CCA game
What if Encryption is Malleable?

Case 2: $A_f$ sends $\{t[X]\}_k$, but neither $t[X]$, nor $\{t[X]\}_k$ was previously sent by honest parties

- Adversary managed to take some encryption containing $X$ and convert it into another encryption containing $X$
- This is known as malleability. For example, with RSA can convert an encryption of $m$ into encryption of $m^2$

In this case, CCA adversary will win CCA game by using the concrete adversary’s ability to convert one encryption into another

- But to do this, CCA adversary must be able to simulate protocol execution to the concrete adversary
Winning the CCA Game (Simplified)

1. CCA adversary guesses nonce $X$ and picks two values $x_0$ and $x_1$
2. Every time $A_c$ asks for $\text{enc}_k(s(X))$, CCA adversary uses oracles to obtain $\text{enc}_k(s(x_b))$
3. When $A_c$ outputs $\text{enc}_k(t(x_b))$, CCA adversary submits it to decryption oracle
4. From $t(x_b)$ CCA adversary learns whether $x_0$ or $x_1$ is inside; outputs bit $b$ correctly

Ok, since the ciphertext was not previously created by encryption oracle
Summary

- If the encryption scheme used in the protocol is non-malleable under chosen-ciphertext attack, then the abstract model is sound.
  - If an attack is not discovered in the abstract model, a concrete attack may exist only with negligible probability.

- See this paper for a similar result with a formal definition of the operational semantics of the simulator.
In more detail...

- Abstract execution model and security properties
- Concrete execution model and security properties
- Relating abstract and concrete security properties
Semantics

- A protocol defines two interactive programs, one for the Initiator and one for the Responder;

\[
\begin{align*}
(1) \quad I & \rightarrow R : \{I, N_I\}_{K_R} \\
(2) \quad R & \rightarrow I : \{R, N_I, N_R\}_{K_I} \\
(3) \quad I & \rightarrow R : \{N_R\}_{K_R}
\end{align*}
\]

- The program of a party A running the NSL protocol with intended responder B:
  - Choose random \( N_A \), encrypt \((A, N_A)\) and send the ciphertext to B;
  - Receive a message C; decrypt C with the secret key of A; parse the plaintext as \((B, N_A, N_B)\);
  - If successful, send the encryption of \( N_B \) under the public key of B.
Semantics

• The local state of a party:
  - program counter
  - an assignment from protocol variables to values, e.g. for the NSL protocol

For the initiator A of a run between parties A, B the initial state: I=A, R=B, K_I=K_A, K_R=K_B, N_I=N_A, N_R=?

• The local state of the party changes by receiving/sending messages.
Execution Model

Environment $Env$ maintains the global state of all parties

- Adversary can create new sessions of the protocol: $\text{new}(A,B)$
- Adversary can send messages to any party running the protocol and obtains the message the party would normally output
Abstract Execution Model

(1) $I > R : \{I, N\}_{K_R}$
(2) $R > I : \{R, N_I, N_R\}_{K_I}$
(3) $I > R : \{N_R\}_{K_R}$

**Adv**$_A$

$\text{Env}_A$ maintains the global state of all parties;

- Message are terms from a term algebra
- Adversary is a Dolev-Yao
Abstract Execution Trace

- The interaction between each abstract adversary $\text{Adv}_A$ and the abstract environment $\text{Env}_A$ determines an execution trace.

- Denote by $\text{Tr}(\text{Adv}_A, \text{Env}_A)$ the trace $F_0, F_1, F_2, \ldots, F_n$, i.e. the sequence of the global state of the honest parties.
Abstract Security Properties

- An abstract security property is a predicate \( \text{Prop}_A \) on the set of all possible abstract execution traces.

- A protocol satisfies abstractly the security property \( \text{Prop}_A \) if for all Dolev-Yao adversaries \( \text{Adv}_A \)
  
  \[ \text{Tr}(\text{Adv}_A, \text{Env}_A) \in \text{Prop}_A \]
Concrete Execution Model

(1) $I \rightarrow R : \{I, N\}_{K_R}$
(2) $R \rightarrow I : \{R, N_I, N_R\}_{K_I}$
(3) $I \rightarrow R : \{N_R\}_{K_R}$

Adv$_C$ maintains the global state of all parties

- Messages are bit-strings
- Adversary is any probabilistic polynomial time Turing machine
Concrete Execution Model

Env\textsubscript{C} maintains the global state of all parties

<table>
<thead>
<tr>
<th>Party</th>
<th>I</th>
<th>R</th>
<th>Ni</th>
<th>Nr</th>
</tr>
</thead>
<tbody>
<tr>
<td>a/b Party a</td>
<td>a</td>
<td>b</td>
<td>n\textsubscript{A}</td>
<td>n\textsubscript{R}</td>
</tr>
<tr>
<td>c/a Party c</td>
<td>c</td>
<td>a</td>
<td>n\textsubscript{C}</td>
<td>n\textsubscript{R}</td>
</tr>
</tbody>
</table>

\( R_{adv} \) and  \( R_{env} \)
Fixing the randomness of the adversaries and of the parties, determines a unique execution trace

Denote by $\text{Tr}(\text{Adv}_C(R_{\text{adv}}),\text{Env}_C(R_{\text{env}}))$ the trace $C_0, C_1, ..., C_n$ determined by adversary $\text{Adv}_C$ using randomness $R_{\text{adv}}$ and environment $\text{Env}_C$ using randomness $R_{\text{env}}$. 

Concrete Execution Trace

- $C_0$
- $C_1$
- $C_n$

$R_{\text{adv}}$ $R_{\text{env}}$

$R_{\text{adv}}$ $R_{\text{env}}$

$R_{\text{adv}}$ $R_{\text{env}}$
Concrete Security Properties

- A concrete security property is a predicate $\text{Prop}_C$ on the set of all possible execution traces.

- A protocol satisfies security property $\text{Prop}_C$ if for any probabilistic polynomial-time adversaries $\text{Adv}_C$:

  $$\text{Tr}(\text{Adv}_C(R_{\text{adv}}),\text{Env}_C(R_{\text{env}})) \in \text{Prop}_C$$

  with overwhelming probability over the random choices $R_{\text{adv}}$ and $R_{\text{env}}$. 
Relating Abstract/Concrete Security Properties

- Fix $\text{Prop}_A$ and $\text{Prop}_C$, an abstract and a concrete security property
- $\text{Prop}_A < \text{Prop}_C$ if for all injective functions $F$, mapping formal values to appropriate bit-strings

$$F(\text{Prop}_A) \subseteq \text{Prop}_C$$

- “Mutual authentication” can be naturally specified in both frameworks, and a relation as the one above holds
THEOREM: Assume formal/computational security properties $\text{Prop}_A < \text{Prop}_C$ and a protocol implemented with an IND-CCA secure encryption scheme. If the abstract execution of the protocol is secure (with respect to $\text{Prop}_A$), then the concrete execution is secure (with respect to $\text{Prop}_C$).
Acknowledgements

Thanks to Vitaly Shmatikov and Bogdan Warinschi for providing a number of slides.