CS 395T

Computational Soundness of Formal Models

Bridging the Gap (Again)

- 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: axiomatic interpretation 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 axioms 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

State of the Art

- Abadi, Rogaway. “Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption)”. J. Cryptology, 2002.
  - Symmetric encryption only, passive adversary
- Micciancio, Warinschi. “Completeness Theorems for the Abadi-Rogaway Logic of Encrypted Expressions”. JCS 2004.
  - Fixes some bugs in the Abadi-Rogaway paper
  - Notion of “confusion-free” encryption
- Micciancio, Warinschi. “Soundness of Formal Encryption in the Presence of Active Adversaries”. TCC 2004.
  - Public-key encryption, active adversary

We’ll look at this paper today

Outline

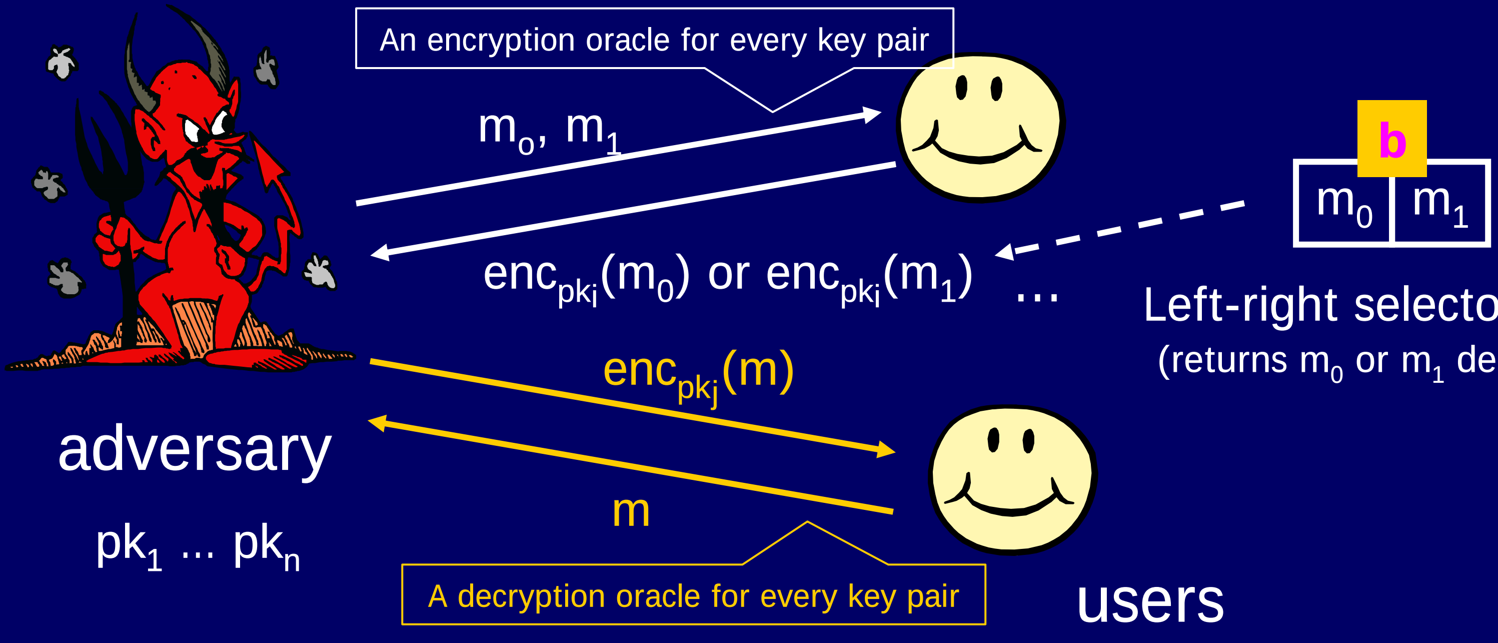
- Define what it means for an encryption scheme to be secure against adaptive chosen-ciphertext attack in the multi-user setting
- Define a formal language for describing protocols
- Define concrete trace semantics for protocols
  - Actual execution traces of protocols obtained by instantiating nonces 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

IND-CCA in Multi-User Setting

[Boldyreva, Bellare, Micali]

1. Generate n public-private key pairs (pk_i,sk_i); give public keys to adversary A.
2. A may send any two plaintexts m_0 and m_1 to any user. A receives $enc_{pk_i}(m_b)$.
3. A may send any c (<u>not received previously</u>). If c= $enc_{pk_j}(m)$, A receives m.
4. A wins if he guesses b correctly.

Encryption scheme is IND-CCA secure if this probability is within a negligible factor of ½


An encryption oracle for every key pair
m_0, m_1
$enc_{pk_i}(m_0)$ or $enc_{pk_i}(m_1)$
…
b
m_0 | m_1
Left-right selector LR(m_0,m_1,b)
(returns m_0 or m_1 depending on bit b)
$enc_{pk_j}(m)$
m
A decryption oracle for every key pair
adversary
pk_1 … pk_n
users

Simple Protocol Language

- Pairing and encryption only

Term ::=	Id |	Identity
	Key |	Public keys only
	Nonce |	
	Pair | Ciphertext	
Pair ::=	(Term, Term)	
Ciphertext ::=	$\{\text{Term}\}_{\text{Key}}$	

- 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

Abstract execution		Concrete execution (with RSA)	
A $\rightarrow$	$\{N_a, A\}_{pk(X)}$	A $\rightarrow$	bits($m_1^{e_x}$ mod n_x)
B $\rightarrow$	$\{N_b, N_A\}_{pk(A)}$	B $\rightarrow$	bits($m_2^{e_a}$ mod n_a)
A $\rightarrow$	$\{N_b\}_{pk(X)}$	A $\rightarrow$	bits($m_3^{e_x}$ mod n_x)

(e_x, n_x) is responder's RSA public key; m_1 is a number encoding a concatentation 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 concatentation 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

<u>Abstract adversary</u>

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

<u>Concrete adversary</u>

Any polynomial-time algorithm (maybe probabilistic)

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

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!

<u>Abstract execution</u>

$A \rightarrow \quad \{N_a, A\}_{pk(B)}$

$B \leftarrow \quad ???$

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

<u>Concrete execution (with RSA)</u>

$A \rightarrow \quad m^{e_b} \bmod n_b$

Adversary intercepts and squares; B receives m^{2e_b} instead of m^{e_b}

$B \leftarrow \quad m^{2e_b} \bmod n_b$

Main Theorem

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

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

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...

...<u>some abstract trace</u> generated by abstract adversary interacting with abstract encryption and decryption oracles

<u>is an implementation of</u>...

<u>Every concrete trace</u> generated by concrete adversary interacting with concrete encryption and decryption oracles...

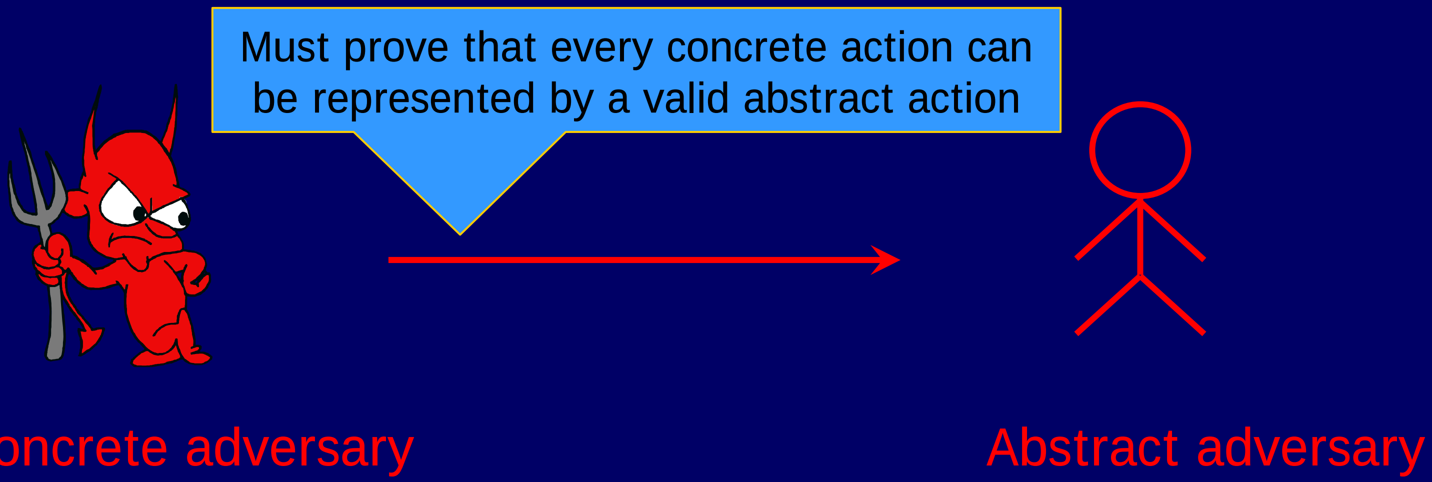
(note that concrete adversary and concrete encryption oracles are randomized)

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 traceset(A_f, O^f) $\leq$ traceset($A_c(R_A)$,$O^c(R_O)$)
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

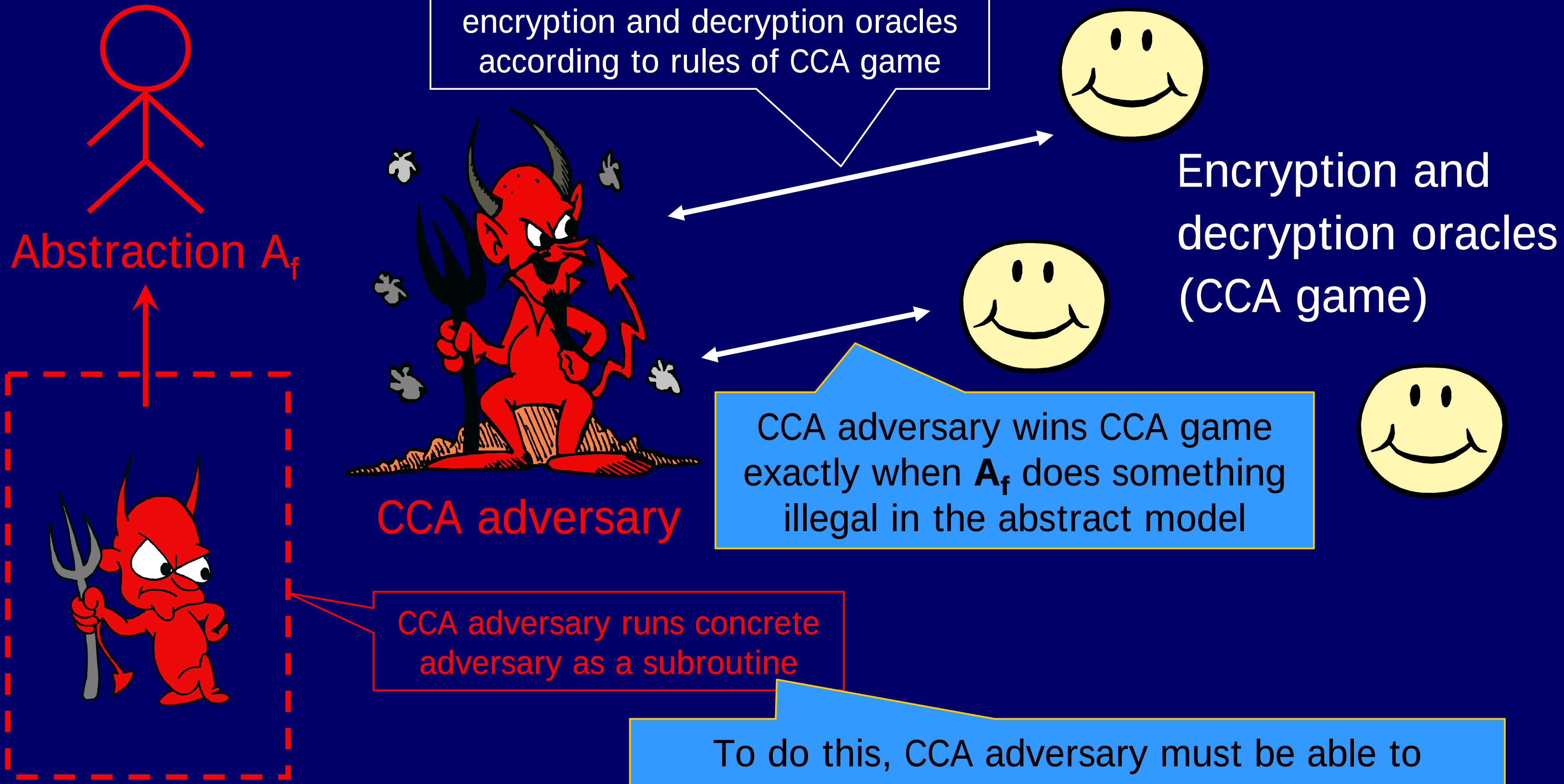

Must prove that every concrete action can be represented by a valid abstract action

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

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

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

CCA adversary runs concrete adversary as a subroutine

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)

What's Illegal in Abstract Model?

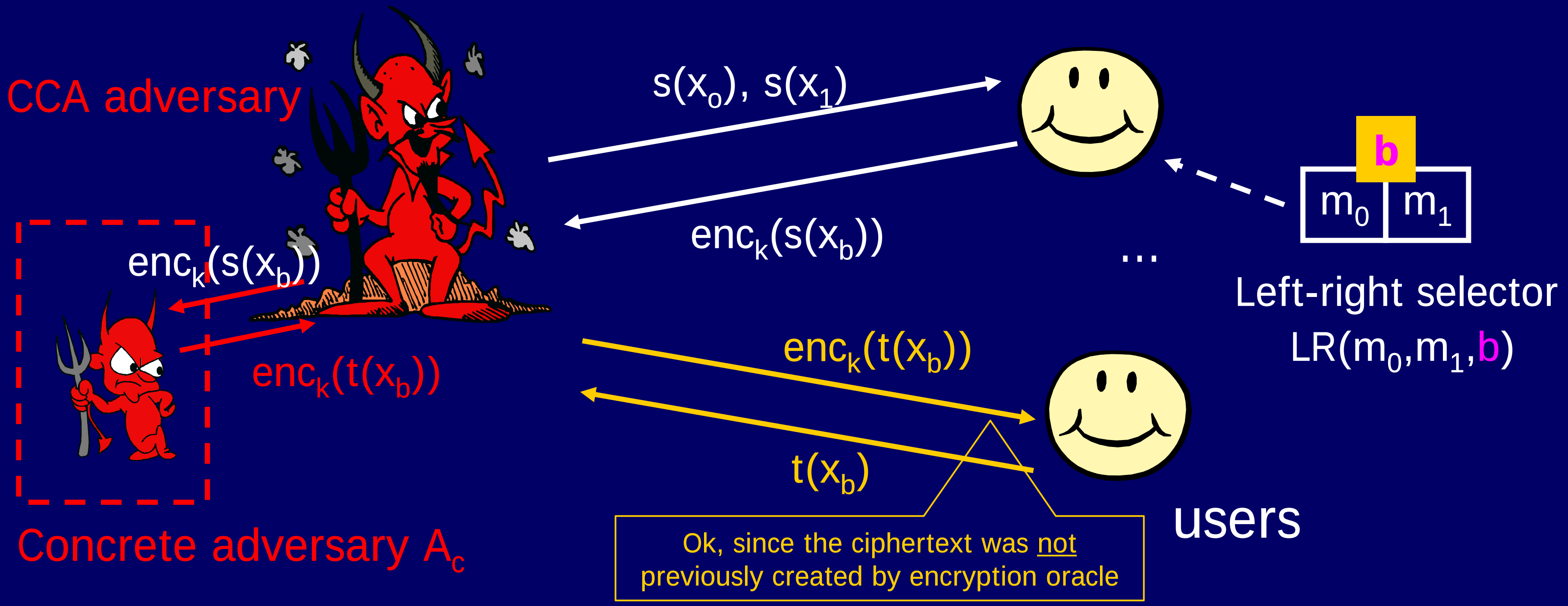
- The <u>only</u> way in which constructed abstract adversary A_f may violate rules of abstract model is by sending a term with some honest nonce X that he could not have learned by abstract actions
  - X cannot be learned from messages sent by honest parties by simple decryption and unpairing rules
- 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_o and x_1
2. Every time A_c asks for $enc_k(s(X))$, CCA adversary uses oracles to obtain $enc_k(s(x_b))$
3. When A_c outputs $enc_k(t(x_b))$, CCA adversary submits it to decryption oracle
4. From $t(x_b)$ CCA adversary learns whether x_o 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
- This result does not cover signatures, hashes, etc. (yet)