

Theory of Computation

Computational Complexity

And Cryptography

Complexity of Problems

Complexity of Problems

- Not all problems are made equal

Complexity of Problems

- Not all problems are made equal
- Decision problems of various levels of "complexity"

Complexity of Problems

- Not all problems are made equal
- Decision problems of various levels of "complexity"
 - Those that can be solved by DFAs (regular), by PDAs (context-free), by Turing Machines (decidable), and by no machines at all (undecidable)

Complexity of Problems

- Not all problems are made equal
- Decision problems of various levels of "complexity"
 - Those that can be solved by DFAs (regular), by PDAs (context-free), by Turing Machines (decidable), and by no machines at all (undecidable)
- Quantitative notions of complexity?

Complexity of Problems

- Not all problems are made equal
- Decision problems of various levels of "complexity"
 - Those that can be solved by DFAs (regular), by PDAs (context-free), by Turing Machines (decidable), and by no machines at all (undecidable)
- Quantitative notions of complexity?
 - For a regular language: e.g. minimum number of states in a DFA for deciding that language

Complexity of Problems

- Not all problems are made equal
- Decision problems of various levels of "complexity"
 - Those that can be solved by DFAs (regular), by PDAs (context-free), by Turing Machines (decidable), and by no machines at all (undecidable)
- Quantitative notions of complexity?
 - For a regular language: e.g. minimum number of states in a DFA for deciding that language
 - For a decidable language: e.g. minimum number of tape cells used by a TM; minimum number of time steps used by a TM

Complexity of Problems

- Not all problems are made equal
- Decision problems of various levels of "complexity"
 - Those that can be solved by DFAs (regular), by PDAs (context-free), by Turing Machines (decidable), and by no machines at all (undecidable)
- Quantitative notions of complexity?
 - For a regular language: e.g. minimum number of states in a DFA for deciding that language
 - For a decidable language: e.g. minimum number of tape cells used by a TM; minimum number of time steps used by a TM
 - As a function of $|x|$, the input size

Time-Complexity

Time-Complexity

- **Time-complexity of a language L** : minimum time taken by any TM deciding L on input x (as a function of $|x|$)

Time-Complexity

- **Time-complexity of a language L** : minimum time taken by any TM deciding L on input x (as a function of $|x|$)
 - Most natural and practically relevant complexity measure

Time-Complexity

- **Time-complexity of a language L** : minimum time taken by any TM deciding L on input x (as a function of $|x|$)
 - Most natural and practically relevant complexity measure
 - Reasonably robust against changes in the model (number of tapes, size of alphabet, RAM instead of TM)

Time-Complexity

- **Time-complexity of a language L** : minimum time taken by any TM deciding L on input x (as a function of $|x|$)
 - Most natural and practically relevant complexity measure
 - Reasonably robust against changes in the model (number of tapes, size of alphabet, RAM instead of TM)
 - **Time-hierarchy theorem**: for any decidable language L , there is another decidable language L' , with a (slightly) higher time-complexity

Time-Complexity

- **Time-complexity of a language L** : minimum time taken by any TM deciding L on input x (as a function of $|x|$)
 - Most natural and practically relevant complexity measure
 - Reasonably robust against changes in the model (number of tapes, size of alphabet, RAM instead of TM)
 - **Time-hierarchy theorem**: for any decidable language L , there is another decidable language L' , with a (slightly) higher time-complexity
 - The more time you can spend, the more problems you can solve

Time-Complexity

- **Time-complexity of a language L** : minimum time taken by any TM deciding L on input x (as a function of $|x|$)
 - Most natural and practically relevant complexity measure
 - Reasonably robust against changes in the model (number of tapes, size of alphabet, RAM instead of TM)
 - **Time-hierarchy theorem**: for any decidable language L , there is another decidable language L' , with a (slightly) higher time-complexity
 - The more time you can spend, the more problems you can solve
 - **Proved using diagonalization**

Decision Complexity vs. Verification Complexity

Decision Complexity vs. Verification Complexity

- **Solving a decision problem:** a program that accepts all x in L and rejects all x not in L

Decision Complexity vs. Verification Complexity

- **Solving a decision problem:** a program that accepts all x in L and rejects all x not in L
- **Verifying proof of membership:** a program that can be convinced that x is in L iff x is in L

Decision Complexity vs. Verification Complexity

- **Solving a decision problem:** a program that accepts all x in L and rejects all x not in L
- **Verifying proof of membership:** a program that can be convinced that x is in L iff x is in L
 - **Completeness:** for all x in L , there exists a proof that will pass the verification

Decision Complexity vs. Verification Complexity

- **Solving a decision problem:** a program that accepts all x in L and rejects all x not in L
- **Verifying proof of membership:** a program that can be convinced that x is in L iff x is in L
 - **Completeness:** for all x in L , there exists a proof that will pass the verification
 - **Soundness:** for any x not in L , there is no proof that will pass the verification

Decision Complexity vs. Verification Complexity

- **Solving a decision problem:** a program that accepts all x in L and rejects all x not in L
- **Verifying proof of membership:** a program that can be convinced that x is in L iff x is in L
 - **Completeness:** for all x in L , there exists a proof that will pass the verification
 - **Soundness:** for any x not in L , there is no proof that will pass the verification
- A complexity measure: **how efficient can the verification be?**

Non-deterministic Computation and Verification

Non-deterministic Computation and Verification

- An efficient non-deterministic decider iff an efficient proof verifier

Non-deterministic Computation and Verification

- An efficient non-deterministic decider iff an efficient proof verifier
- A verifier V from a non-deterministic decider M

Non-deterministic Computation and Verification

- An efficient non-deterministic decider iff an efficient proof verifier
- A verifier V from a non-deterministic decider M
 - Proof w that x is in L : choices made by the non-deterministic computation that leads to acceptance on input x

Non-deterministic Computation and Verification

- An efficient non-deterministic decider iff an efficient proof verifier
- A verifier V from a non-deterministic decider M
 - Proof w that x is in L : choices made by the non-deterministic computation that leads to acceptance on input x
 - Verified by simply running M on input x , using choices specified in w

Non-deterministic Computation and Verification

- An efficient non-deterministic decider iff an efficient proof verifier
- A verifier V from a non-deterministic decider M
 - Proof w that x is in L : choices made by the non-deterministic computation that leads to acceptance on input x
 - Verified by simply running M on input x , using choices specified in w
- A non-deterministic decider M from a proof-verifier

Non-deterministic Computation and Verification

- An efficient non-deterministic decider iff an efficient proof verifier
- A verifier V from a non-deterministic decider M
 - Proof w that x is in L : choices made by the non-deterministic computation that leads to acceptance on input x
 - Verified by simply running M on input x , using choices specified in w
- A non-deterministic decider M from a proof-verifier
 - Non-deterministically guess a proof w

Non-deterministic Computation and Verification

- An efficient non-deterministic decider iff an efficient proof verifier
- A verifier V from a non-deterministic decider M
 - Proof w that x is in L : choices made by the non-deterministic computation that leads to acceptance on input x
 - Verified by simply running M on input x , using choices specified in w
- A non-deterministic decider M from a proof-verifier
 - Non-deterministically guess a proof w
 - Run the verifier on (x,w)

Non-deterministic Computation and Verification

- An efficient non-deterministic decider iff an efficient proof verifier
- A verifier V from a non-deterministic decider M
 - Proof w that x is in L : choices made by the non-deterministic computation that leads to acceptance on input x
 - Verified by simply running M on input x , using choices specified in w
- A non-deterministic decider M from a proof-verifier
 - Non-deterministically guess a proof w
 - Run the verifier on (x,w)
- Verification complexity is essentially the same as non-deterministic complexity

Gap between Solving and Verifying

Gap between Solving and Verifying

- Having an efficient verifier need not mean having an efficient deterministic decider

Gap between Solving and Verifying

- Having an efficient verifier need not mean having an efficient deterministic decider
 - It is much easier to verify than to decide

Gap between Solving and Verifying

- Having an efficient verifier need not mean having an efficient deterministic decider
 - It is much easier to verify than to decide
 - As far as we know

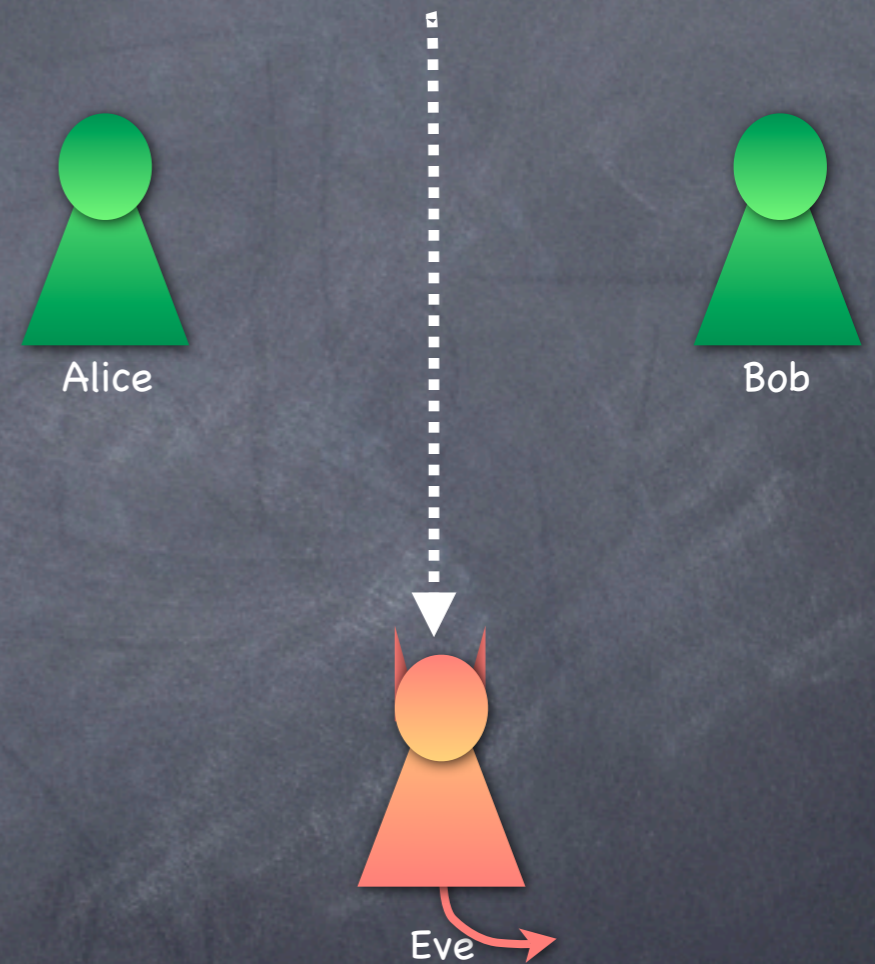
Gap between Solving and Verifying

- Having an efficient verifier need not mean having an efficient deterministic decider
 - It is much easier to verify than to decide
 - As far as we know
- Central problem in computational complexity theory: **prove this gap!**

Gap between Solving and Verifying

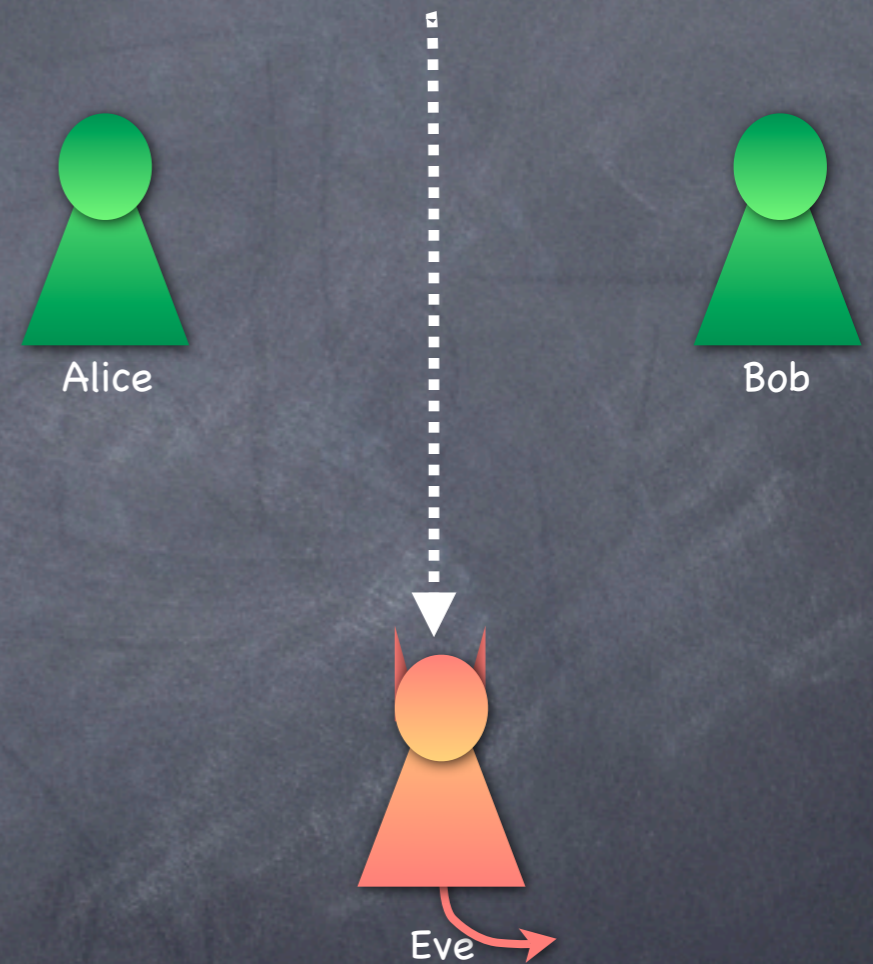
- Having an efficient verifier need not mean having an efficient deterministic decider
 - It is much easier to verify than to decide
 - As far as we know
- Central problem in computational complexity theory: **prove this gap!**
- Central to cryptography: if no such gap, virtually no modern cryptography

Public-Key Encryption



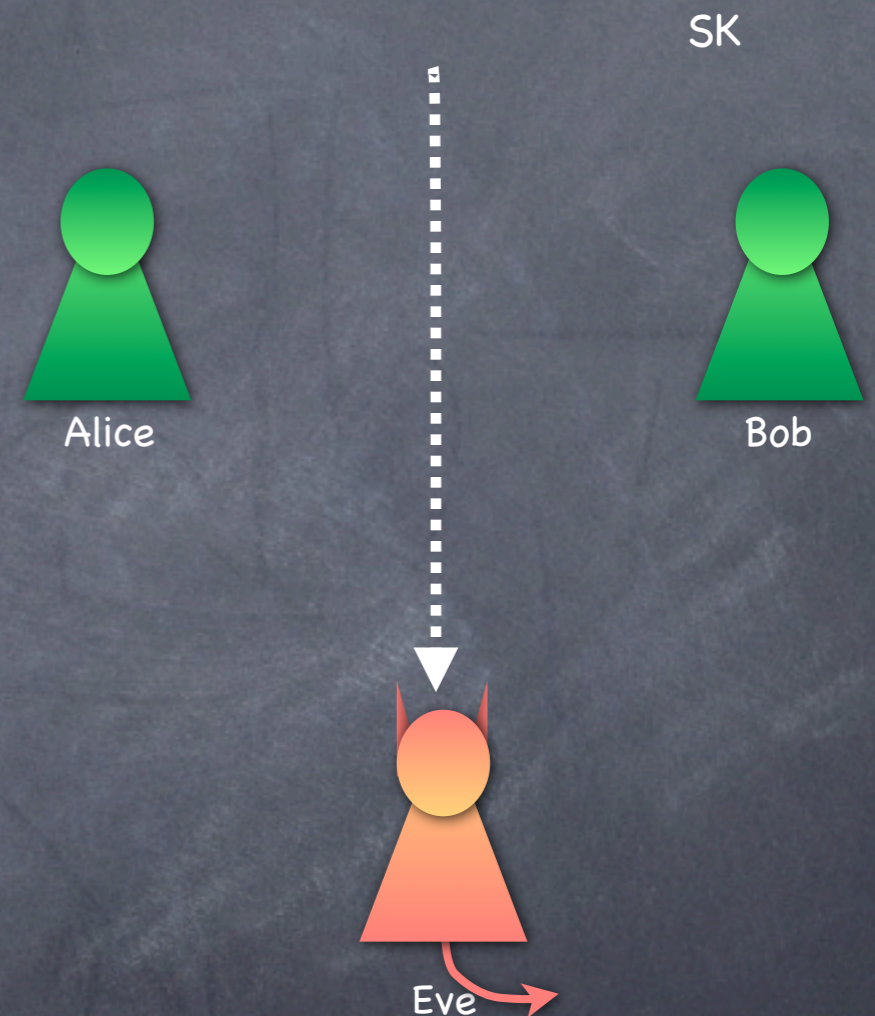
Public-Key Encryption

- Alice wants to send a private message to Bob



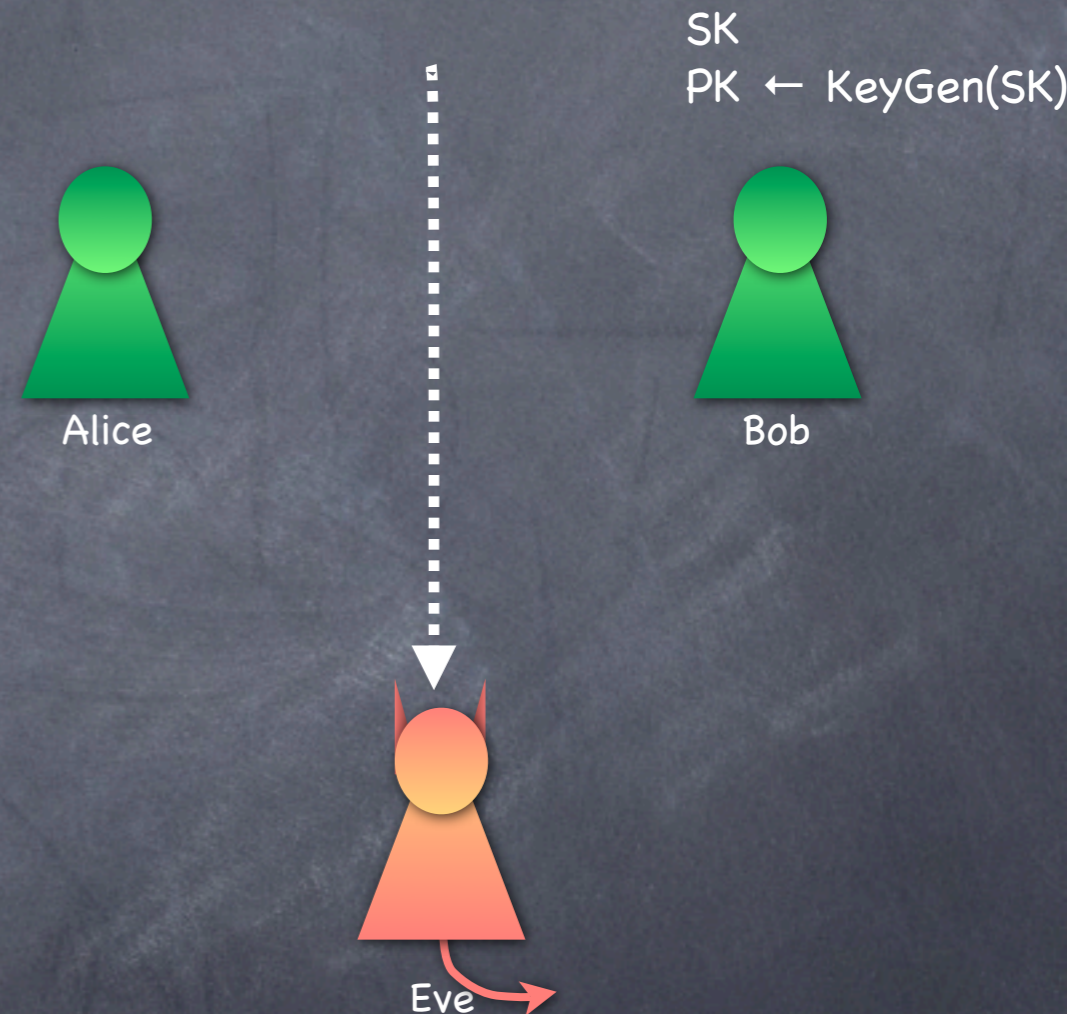
Public-Key Encryption

- Alice wants to send a private message to Bob
- Bob picks a long random string as his **Secret Key**



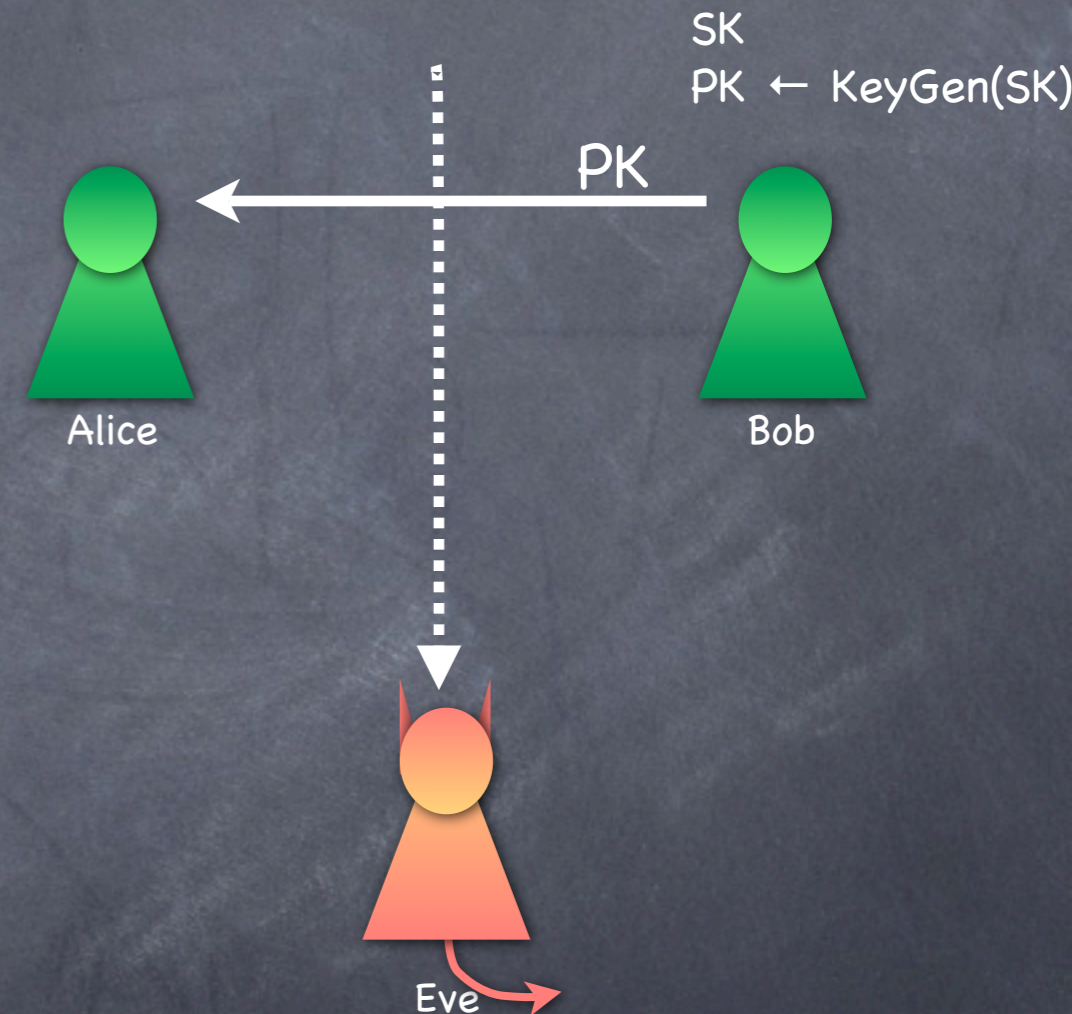
Public-Key Encryption

- Alice wants to send a private message to Bob
- Bob picks a long random string as his **Secret Key**
 - Runs a **Key Generation** program on it to generate a **Public Key**



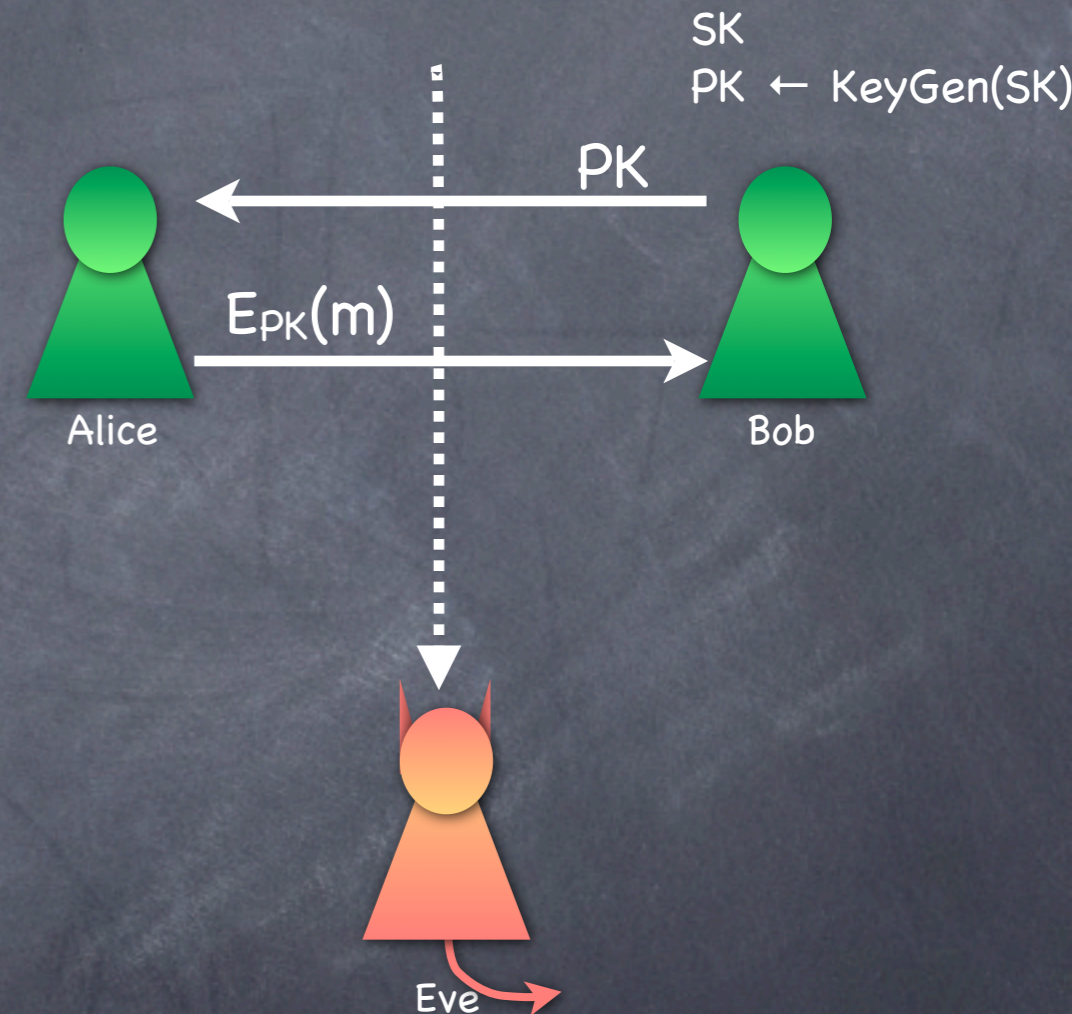
Public-Key Encryption

- Alice wants to send a private message to Bob
- Bob picks a long random string as his **Secret Key**
 - Runs a **Key Generation** program on it to generate a **Public Key**
 - Sends the Public Key to Alice (in the clear)



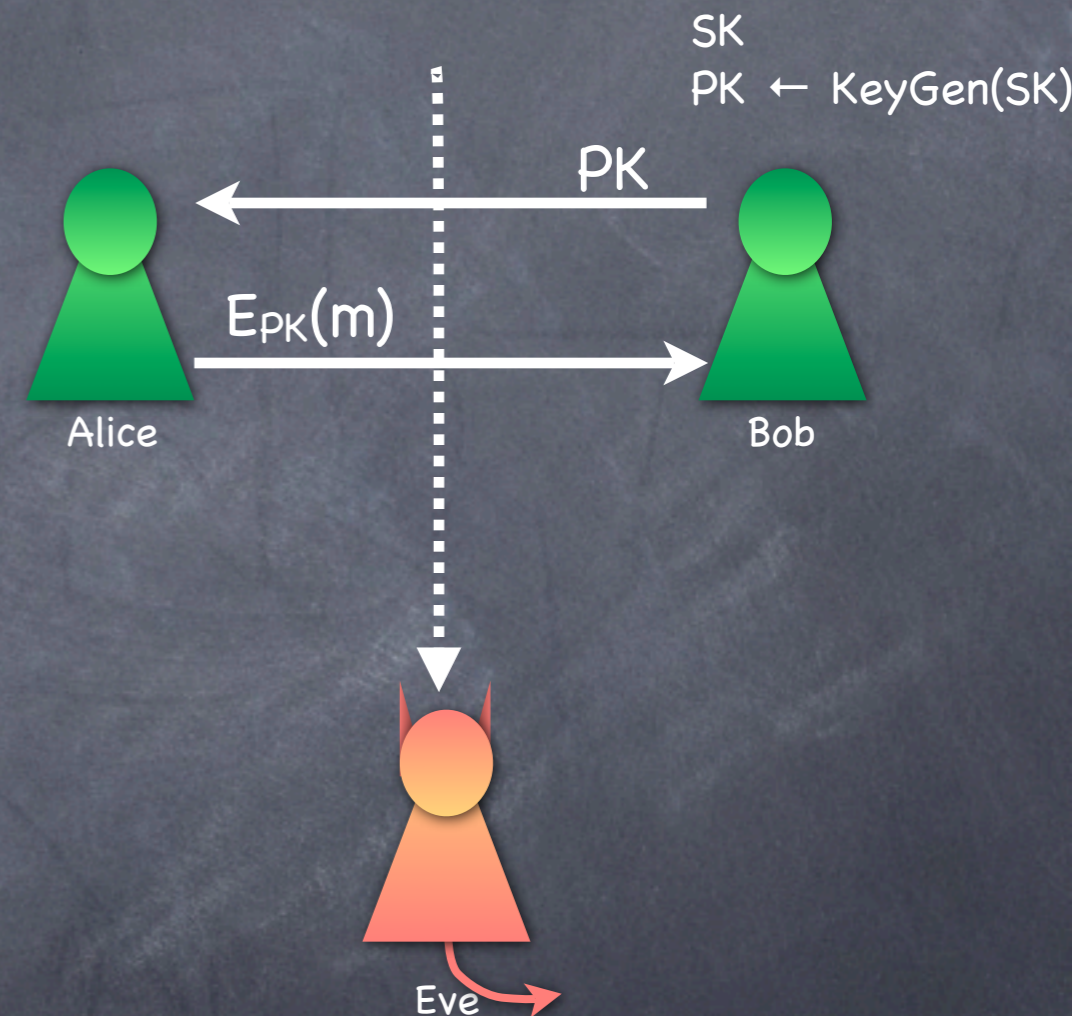
Public-Key Encryption

- Alice wants to send a private message to Bob
- Bob picks a long random string as his **Secret Key**
 - Runs a **Key Generation** program on it to generate a **Public Key**
 - Sends the Public Key to Alice (in the clear)
- Alice uses PK to **encrypt** her message and sends it to Bob. Bob **decrypts** it.



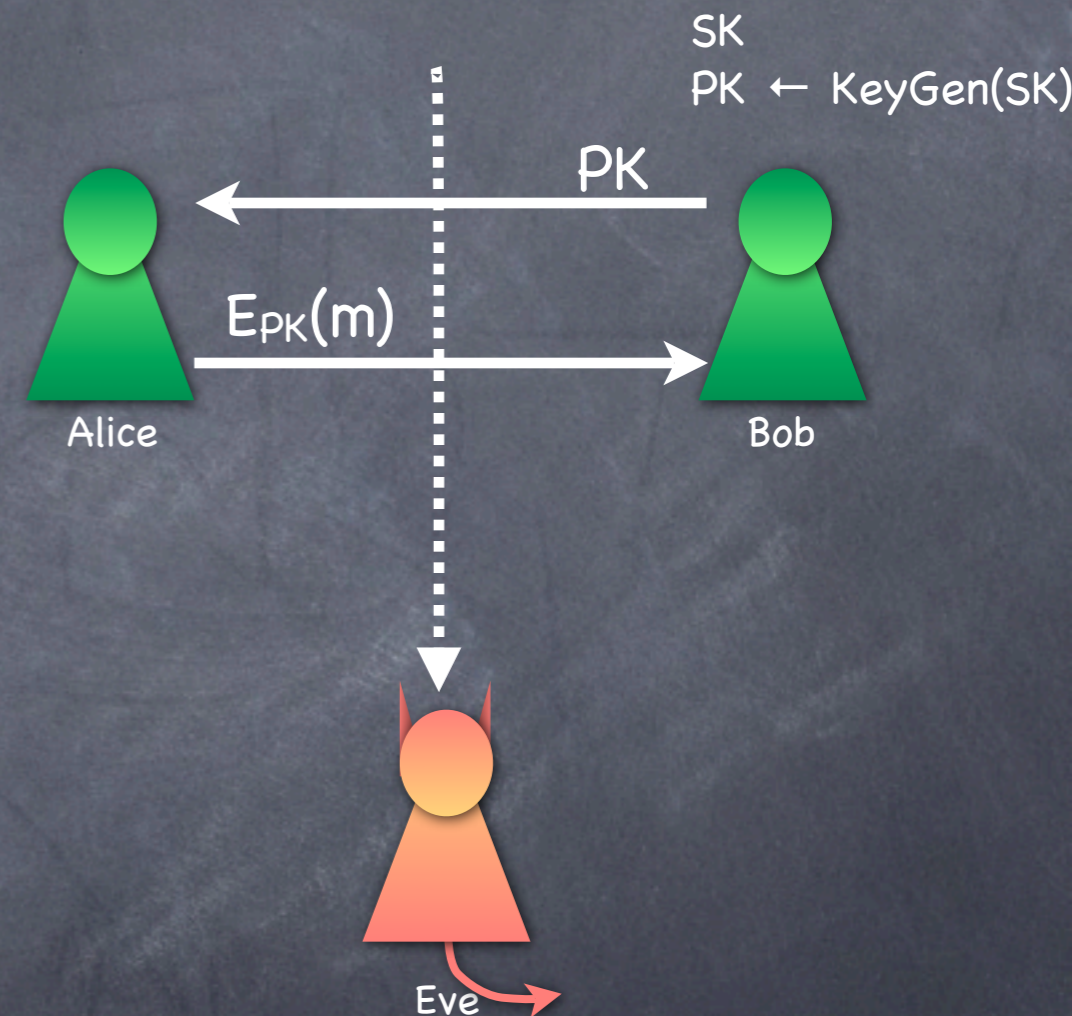
Public-Key Encryption

- Alice wants to send a private message to Bob
- Bob picks a long random string as his **Secret Key**
 - Runs a **Key Generation** program on it to generate a **Public Key**
 - Sends the Public Key to Alice (in the clear)
- Alice uses PK to **encrypt** her message and sends it to Bob. Bob **decrypts** it.
- If "solving as easy as verifying" then this is not secure!



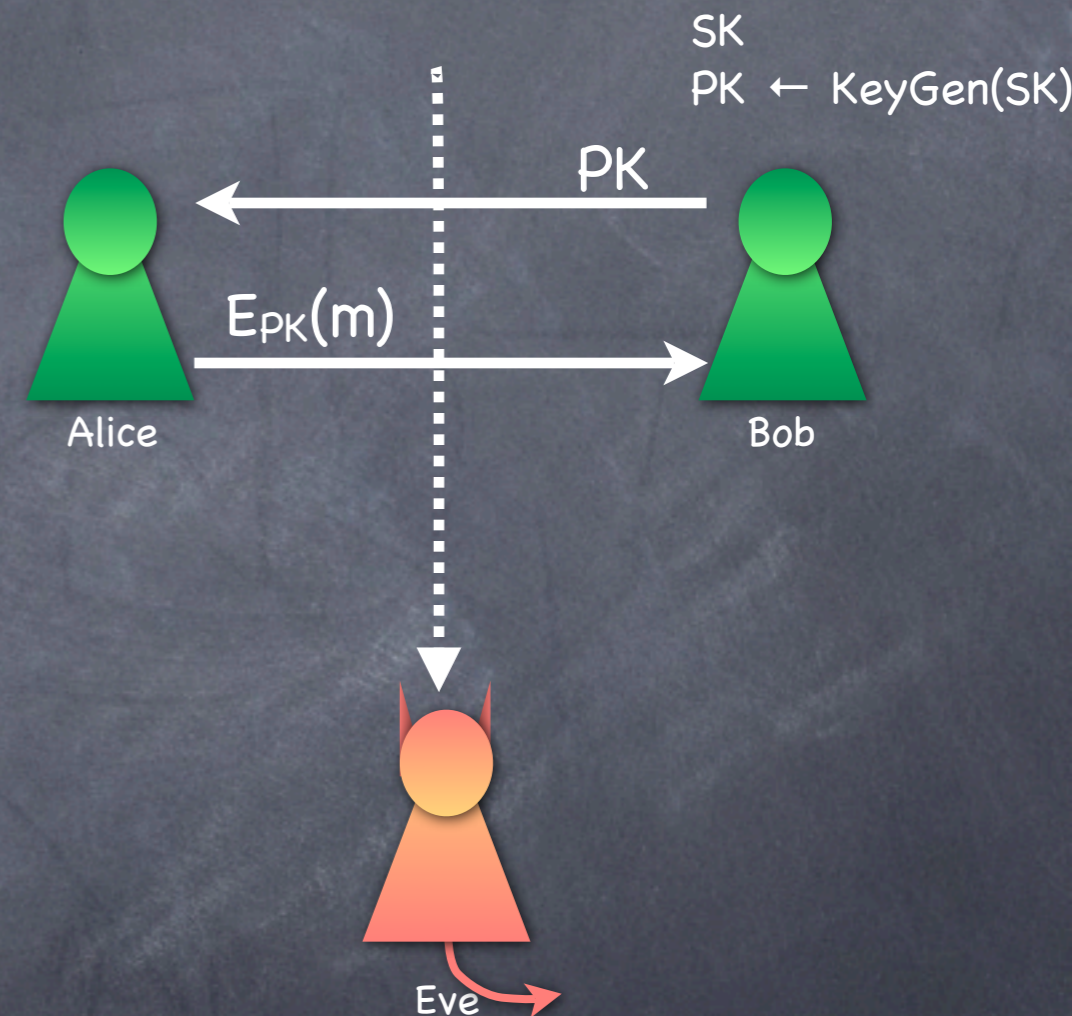
Public-Key Encryption

- Alice wants to send a private message to Bob
- Bob picks a long random string as his **Secret Key**
 - Runs a **Key Generation** program on it to generate a **Public Key**
 - Sends the Public Key to Alice (in the clear)
- Alice uses PK to **encrypt** her message and sends it to Bob. Bob **decrypts** it.
- If "solving as easy as verifying" then this is not secure!
 - Given PK, should be hard at least to solve for SK s.t. $PK = \text{KeyGen}(SK)$



Public-Key Encryption

- Alice wants to send a private message to Bob
- Bob picks a long random string as his **Secret Key**
 - Runs a **Key Generation** program on it to generate a **Public Key**
 - Sends the Public Key to Alice (in the clear)
- Alice uses PK to **encrypt** her message and sends it to Bob. Bob **decrypts** it.
- If "solving as easy as verifying" then this is not secure!
 - Given PK, should be hard at least to solve for SK s.t. $PK = \text{KeyGen}(SK)$
 - But given (PK,SK) it is easy to verify



Gap between Solving and Verifying

- Having an efficient verifier need not mean having an efficient deterministic decider
 - It is much easier to verify than to decide
 - As far as we know
- Central problem in computational complexity theory: **prove this gap!**
- Central to cryptography: if no such gap, virtually no modern cryptography

Gap between Solving and Verifying

- Having an efficient verifier need not mean having an efficient deterministic decider
 - It is much easier to verify than to decide
 - As far as we know
- Central problem in computational complexity theory: **prove this gap!**
- Central to cryptography: if no such gap, virtually no modern cryptography
- An instance of the gap: One-way functions

Gap between Solving and Verifying

- Having an efficient verifier need not mean having an efficient deterministic decider
 - It is much easier to verify than to decide
 - As far as we know
- Central problem in computational complexity theory: **prove this gap!**
- Central to cryptography: if no such gap, virtually no modern cryptography
- An instance of the gap: One-way functions
 - Valves of information: easy to compute, hard to invert

Commitment

Commitment

- An example use of one-way permutations

Commitment

- An example use of one-way permutations
- Analogy: Alice gives Bob a locked box, with the message inside. Later, to open, she sends the key to Bob.



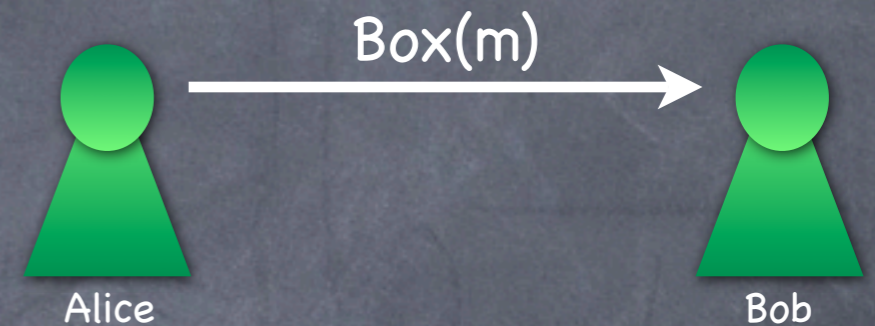
Alice



Bob

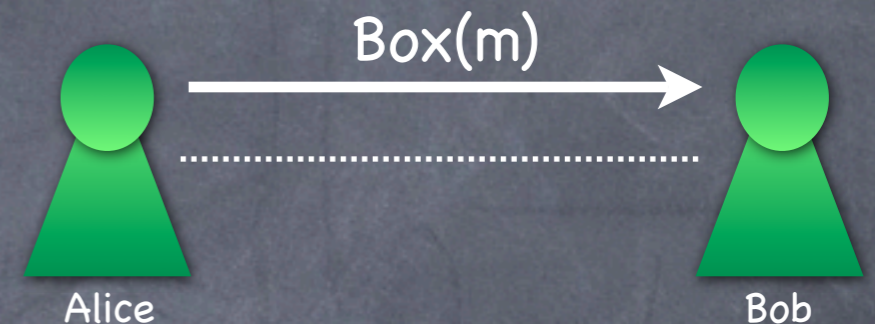
Commitment

- An example use of one-way permutations
- Analogy: Alice gives Bob a locked box, with the message inside. Later, to open, she sends the key to Bob.



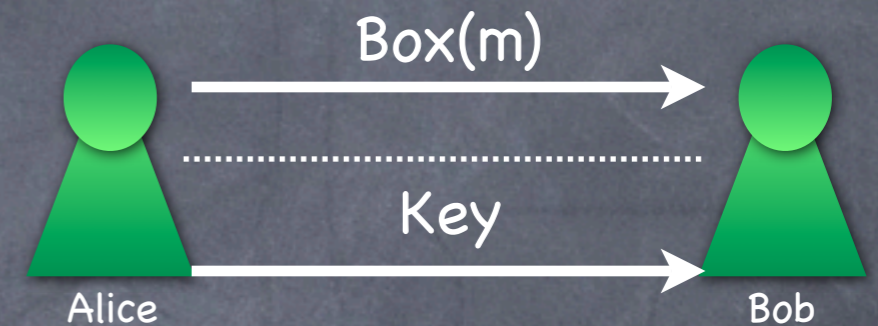
Commitment

- An example use of one-way permutations
- Analogy: Alice gives Bob a locked box, with the message inside. Later, to open, she sends the key to Bob.



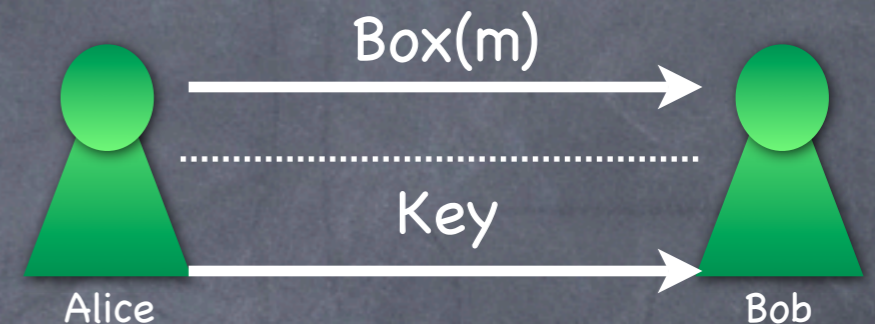
Commitment

- An example use of one-way permutations
- Analogy: Alice gives Bob a locked box, with the message inside. Later, to open, she sends the key to Bob.



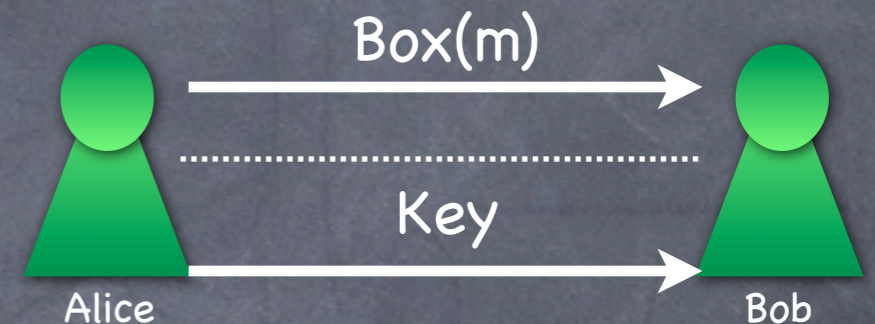
Commitment

- An example use of one-way permutations
- Analogy: Alice gives Bob a locked box, with the message inside. Later, to open, she sends the key to Bob.
- **Hiding:** Before opening, the message should be hidden from Bob



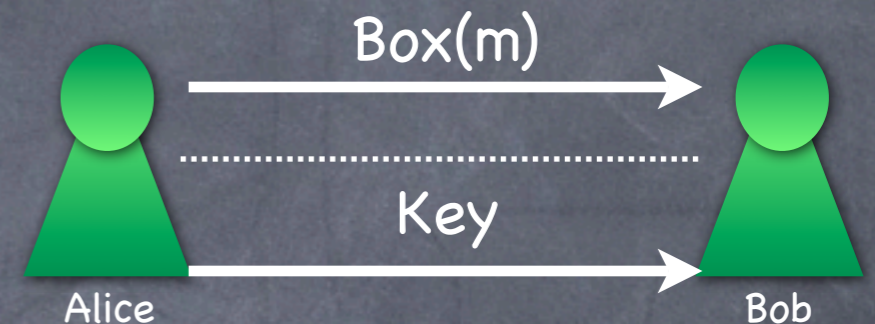
Commitment

- An example use of one-way permutations
- **Analogy:** Alice gives Bob a locked box, with the message inside. Later, to open, she sends the key to Bob.
- **Hiding:** Before opening, the message should be hidden from Bob
- **Binding:** On committing Alice is bound to a single message; she shouldn't be able to reveal to a different message



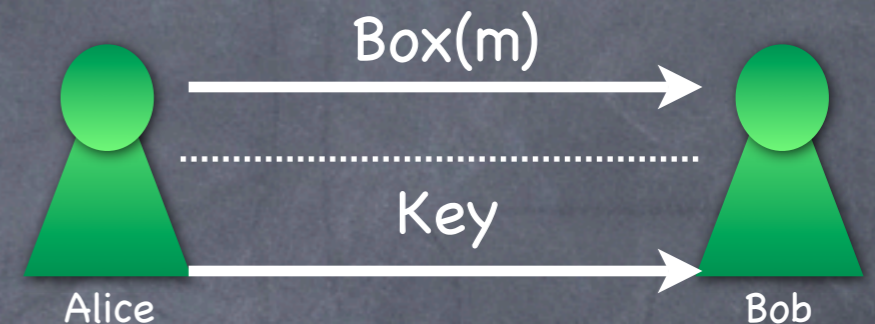
Commitment

- An example use of one-way permutations
- Analogy: Alice gives Bob a locked box, with the message inside. Later, to open, she sends the key to Bob.
 - **Hiding:** Before opening, the message should be hidden from Bob
 - **Binding:** On committing Alice is bound to a single message; she shouldn't be able to reveal to a different message
- A protocol: f be a one-way permutation and H a "hardcore" of f : given $f(w)$, $H(w)$ is completely hidden



Commitment

- An example use of one-way permutations
- Analogy: Alice gives Bob a locked box, with the message inside. Later, to open, she sends the key to Bob.

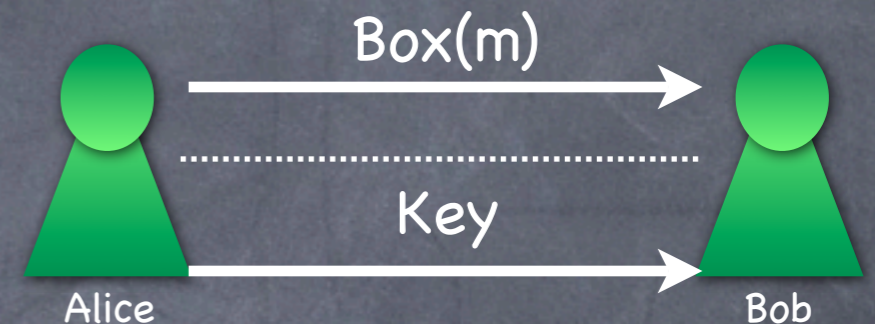


- **Hiding:** Before opening, the message should be hidden from Bob
- **Binding:** On committing Alice is bound to a single message; she shouldn't be able to reveal to a different message
- A protocol: f be a one-way permutation and H a "hardcore" of f : given $f(w)$, $H(w)$ is completely hidden



Commitment

- An example use of one-way permutations
- Analogy: Alice gives Bob a locked box, with the message inside. Later, to open, she sends the key to Bob.



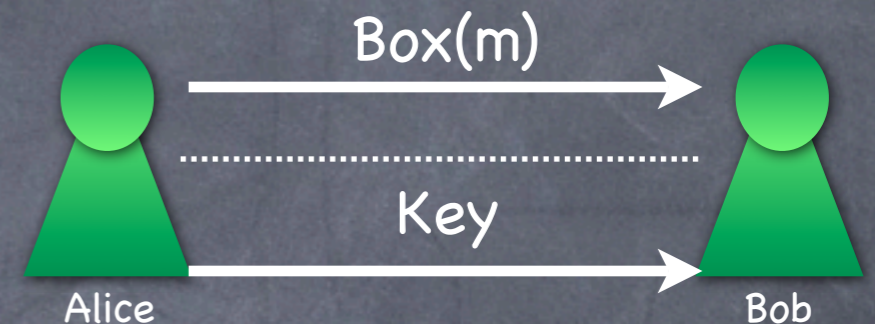
- **Hiding:** Before opening, the message should be hidden from Bob
- **Binding:** On committing Alice is bound to a single message; she shouldn't be able to reveal to a different message
- A protocol: f be a one-way permutation and H a "hardcore" of f : given $f(w)$, $H(w)$ is completely hidden



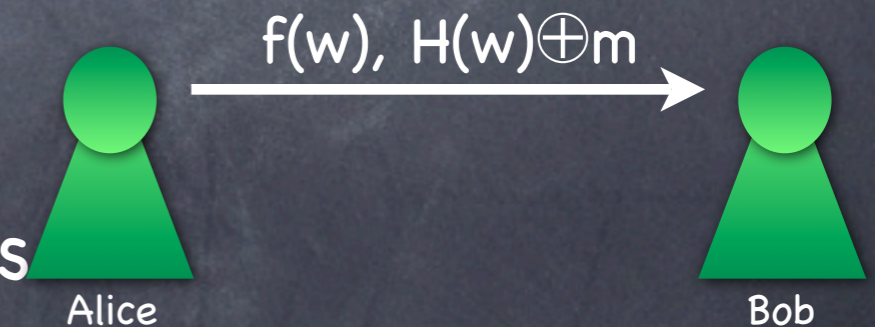
- $\text{Box}(m) = (f(w), H(w) \oplus m)$, and $\text{Key} = w$.

Commitment

- An example use of one-way permutations
- Analogy: Alice gives Bob a locked box, with the message inside. Later, to open, she sends the key to Bob.



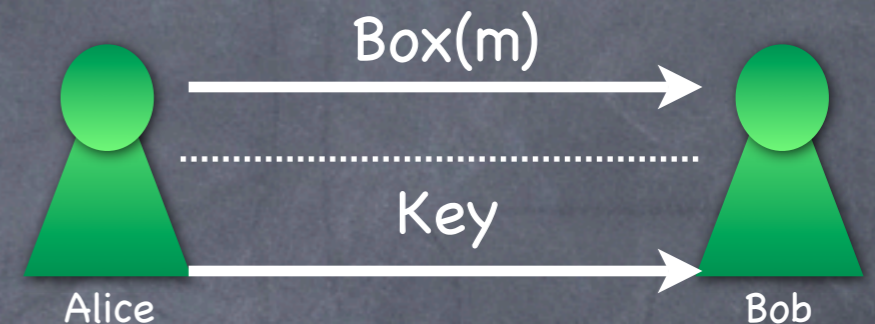
- **Hiding:** Before opening, the message should be hidden from Bob
- **Binding:** On committing Alice is bound to a single message; she shouldn't be able to reveal to a different message
- A protocol: f be a one-way permutation and H a "hardcore" of f : given $f(w)$, $H(w)$ is completely hidden



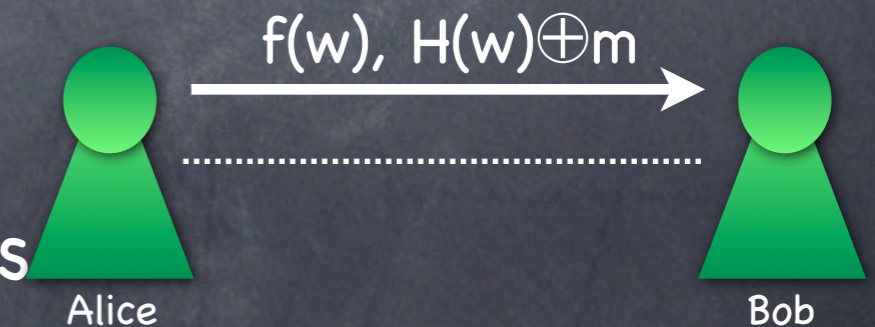
- $\text{Box}(m) = (f(w), H(w) \oplus m)$, and $\text{Key} = w$.

Commitment

- An example use of one-way permutations
- Analogy: Alice gives Bob a locked box, with the message inside. Later, to open, she sends the key to Bob.



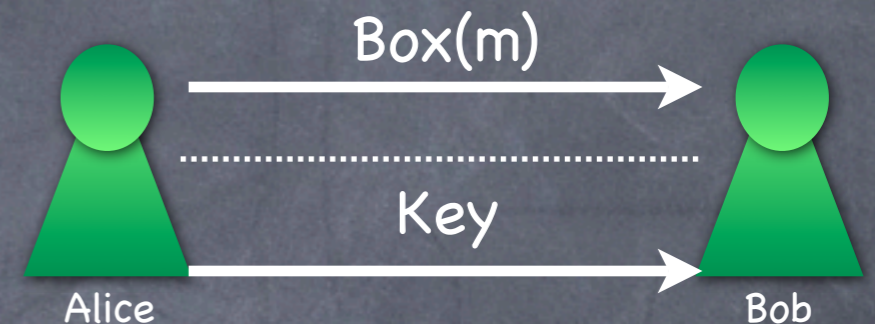
- **Hiding:** Before opening, the message should be hidden from Bob
- **Binding:** On committing Alice is bound to a single message; she shouldn't be able to reveal to a different message
- A protocol: f be a one-way permutation and H a "hardcore" of f : given $f(w)$, $H(w)$ is completely hidden



- $\text{Box}(m) = (f(w), H(w) \oplus m)$, and $\text{Key} = w$.

Commitment

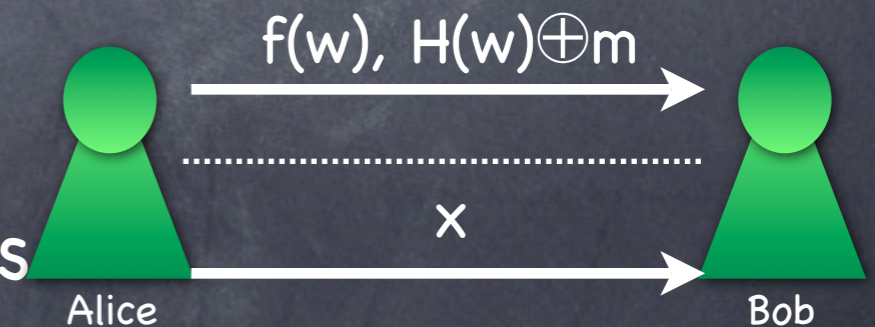
- An example use of one-way permutations
- Analogy: Alice gives Bob a locked box, with the message inside. Later, to open, she sends the key to Bob.



- **Hiding:** Before opening, the message should be hidden from Bob

- **Binding:** On committing Alice is bound to a single message; she shouldn't be able to reveal to a different message

- A protocol: f be a one-way permutation and H a "hardcore" of f : given $f(w)$, $H(w)$ is completely hidden



- $\text{Box}(m) = (f(w), H(w) \oplus m)$, and $\text{Key} = w$.

Interactive Proofs

Interactive Proofs

- A generalization of “proof”:

Interactive Proofs

- A generalization of “proof”:
 - An interaction wherein a **Prover** tries to convince **verifier** that x has some property

Interactive Proofs

- A generalization of “proof”:
- An interaction wherein a **Prover** tries to convince **verifier** that x has some property



Interactive Proofs

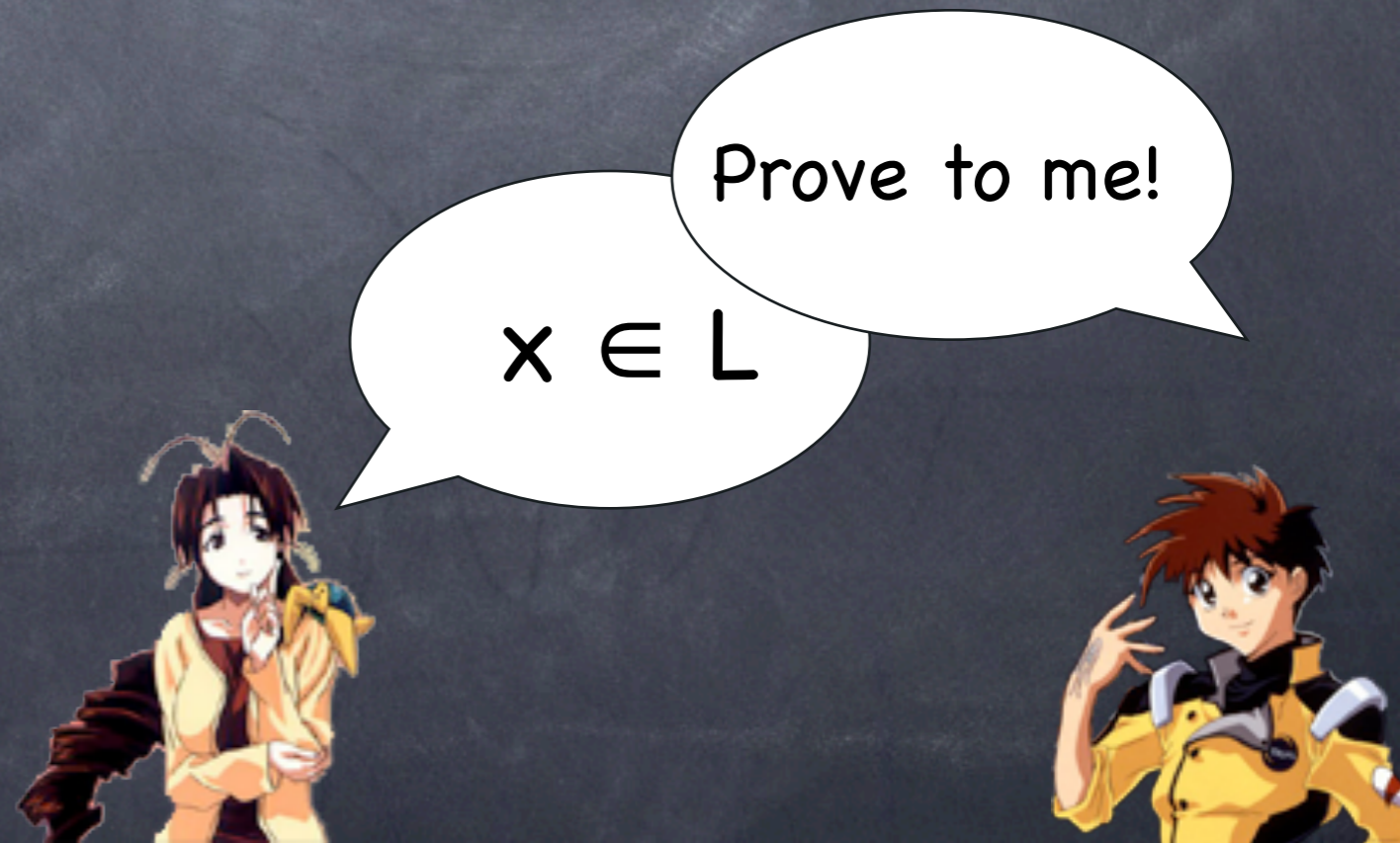
- A generalization of “proof”:
- An interaction wherein a **Prover** tries to convince **verifier** that x has some property



$x \in L$

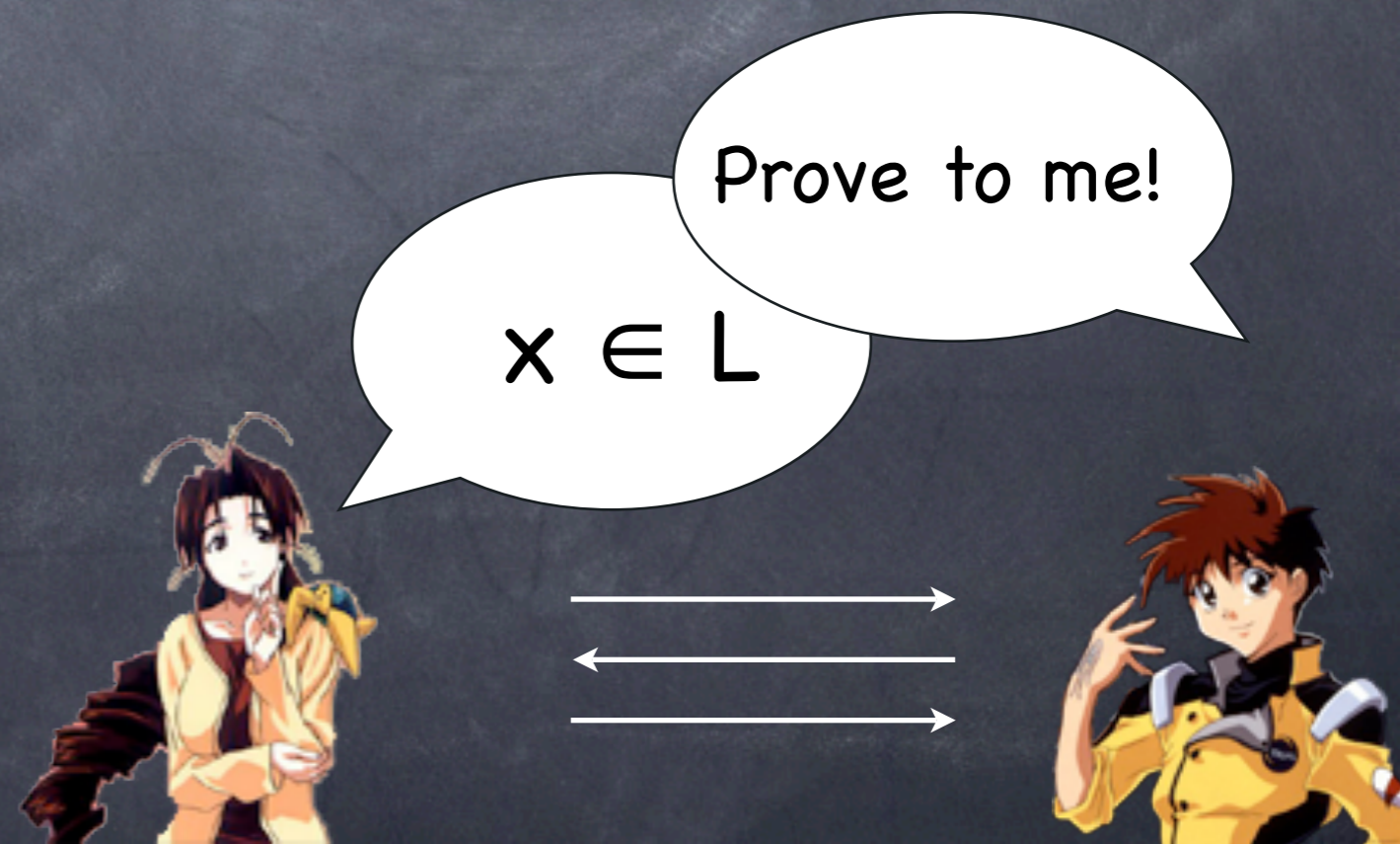
Interactive Proofs

- A generalization of “proof”:
- An interaction wherein a **Prover** tries to convince **verifier** that x has some property



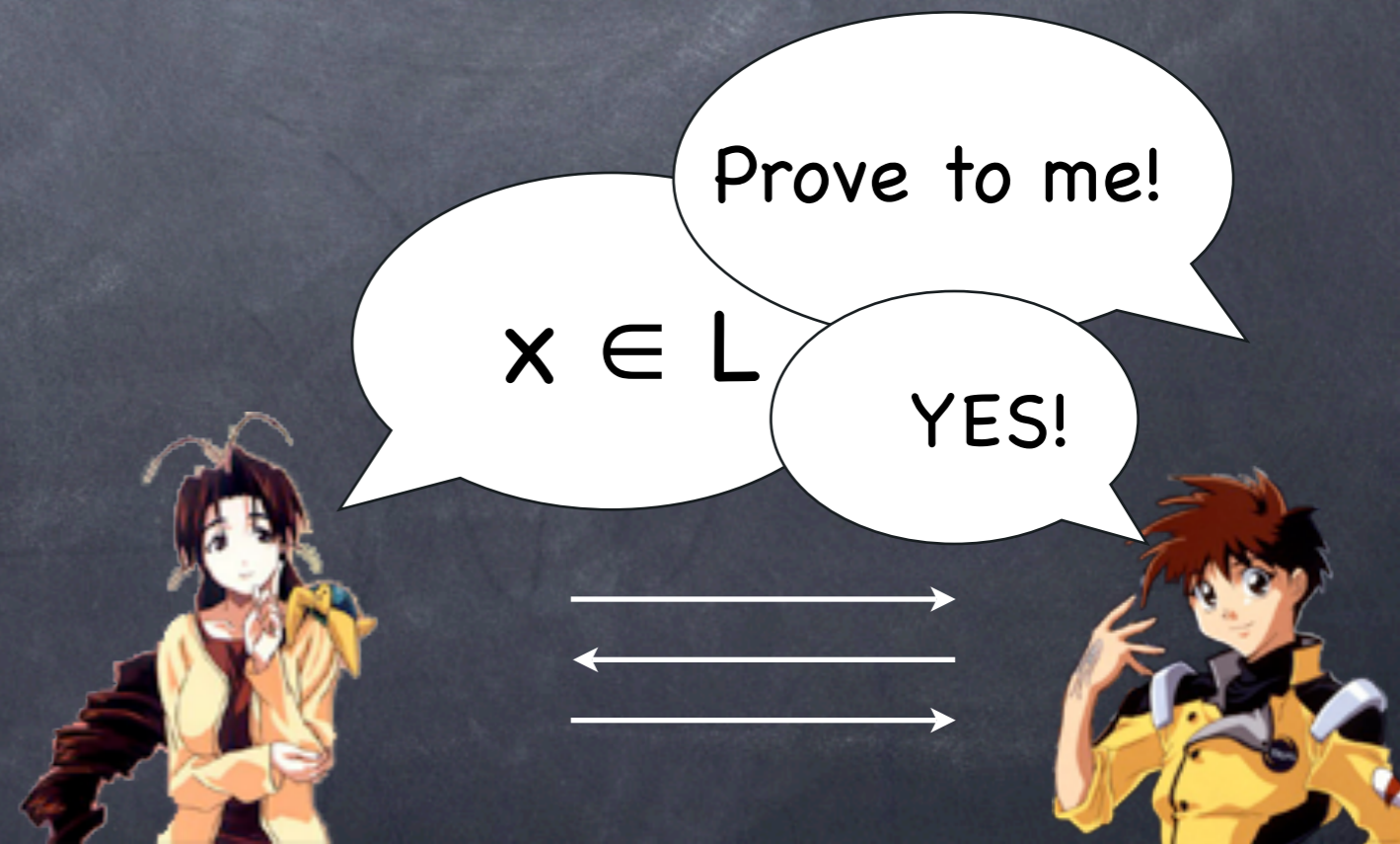
Interactive Proofs

- A generalization of “proof”:
- An interaction wherein a **Prover** tries to convince **verifier** that x has some property



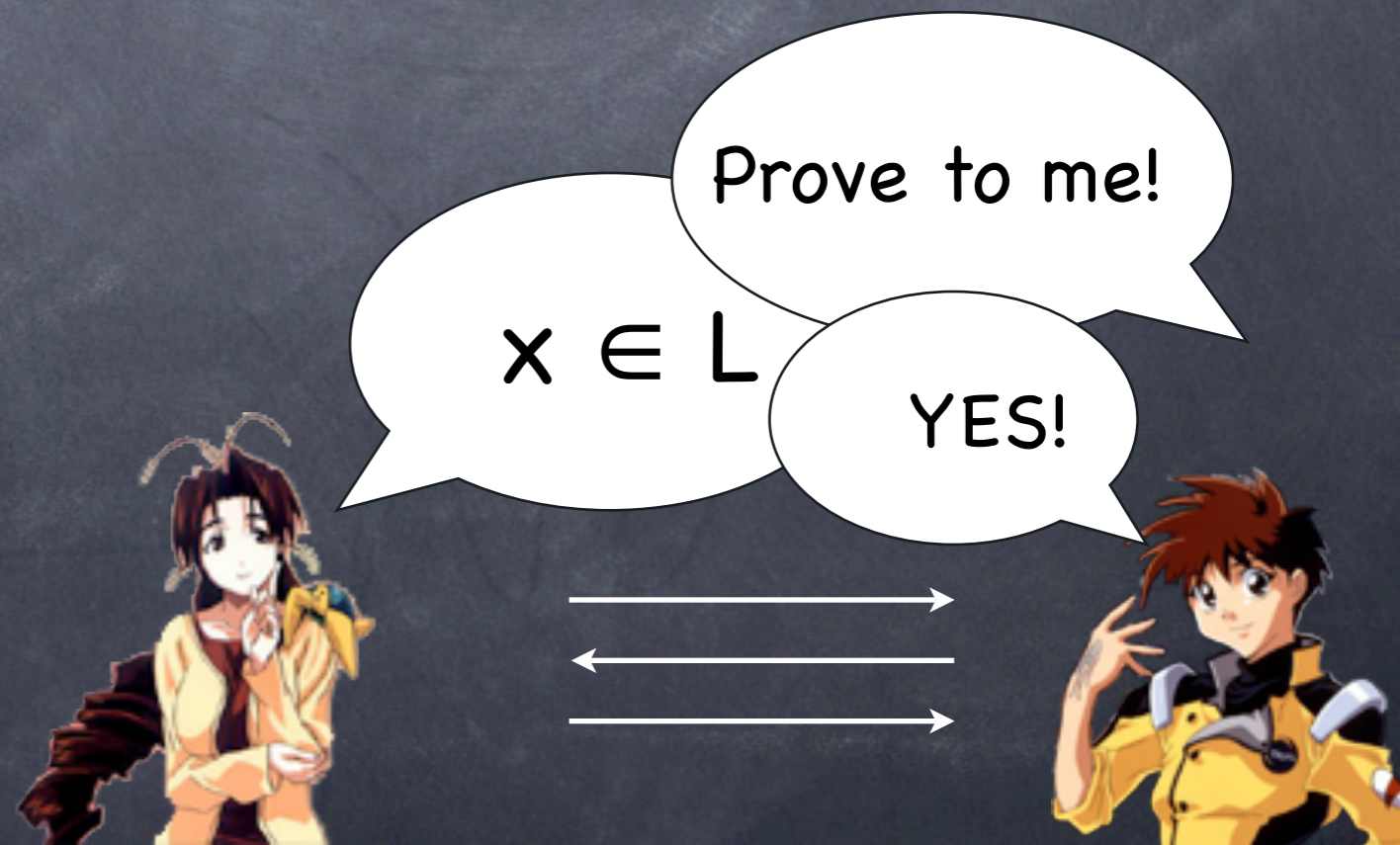
Interactive Proofs

- A generalization of “proof”:
- An interaction wherein a **Prover** tries to convince **verifier** that x has some property



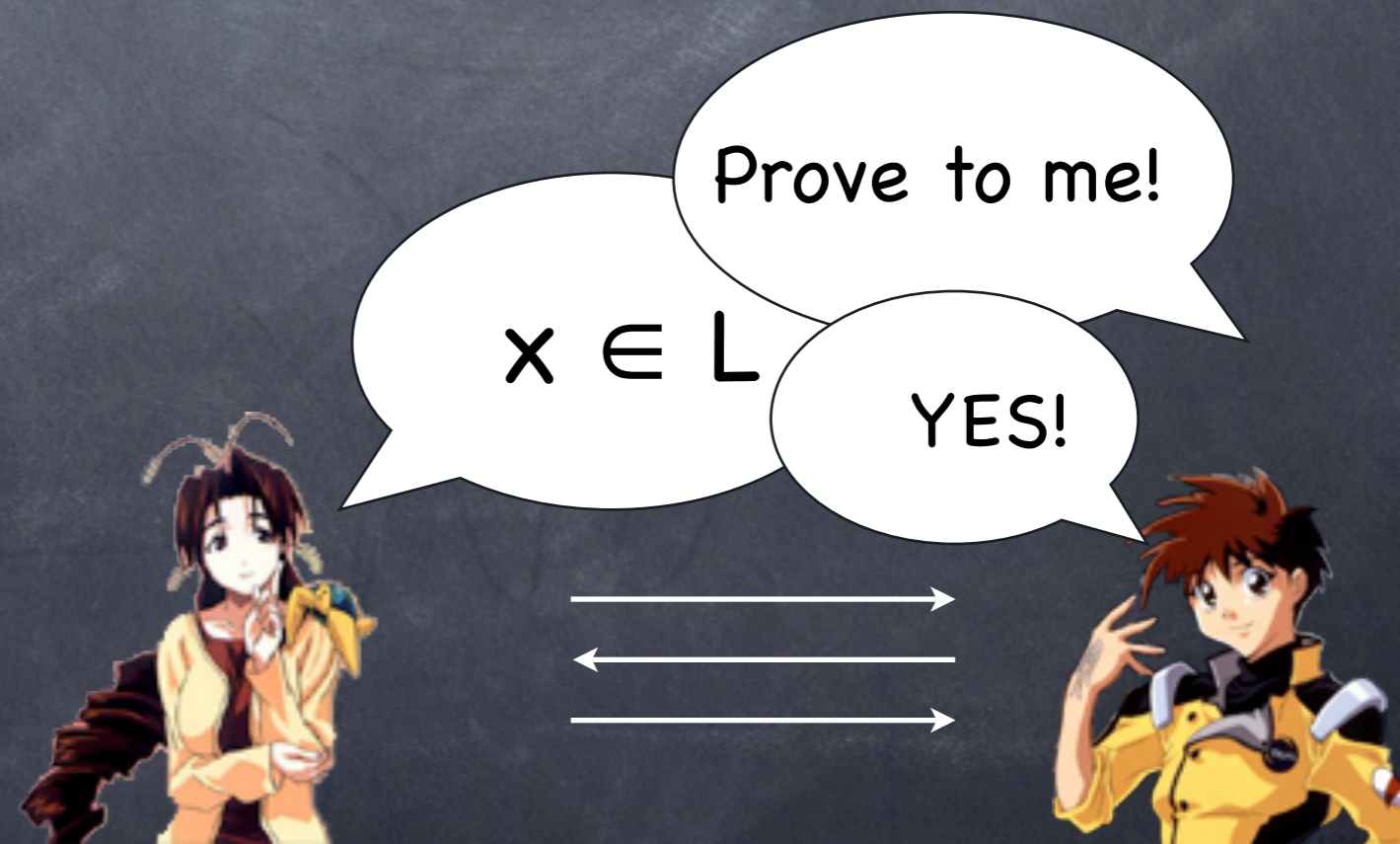
Interactive Proofs

- A generalization of “proof”:
- An interaction wherein a **Prover** tries to convince **verifier** that x has some property
 - i.e. that x is in language L



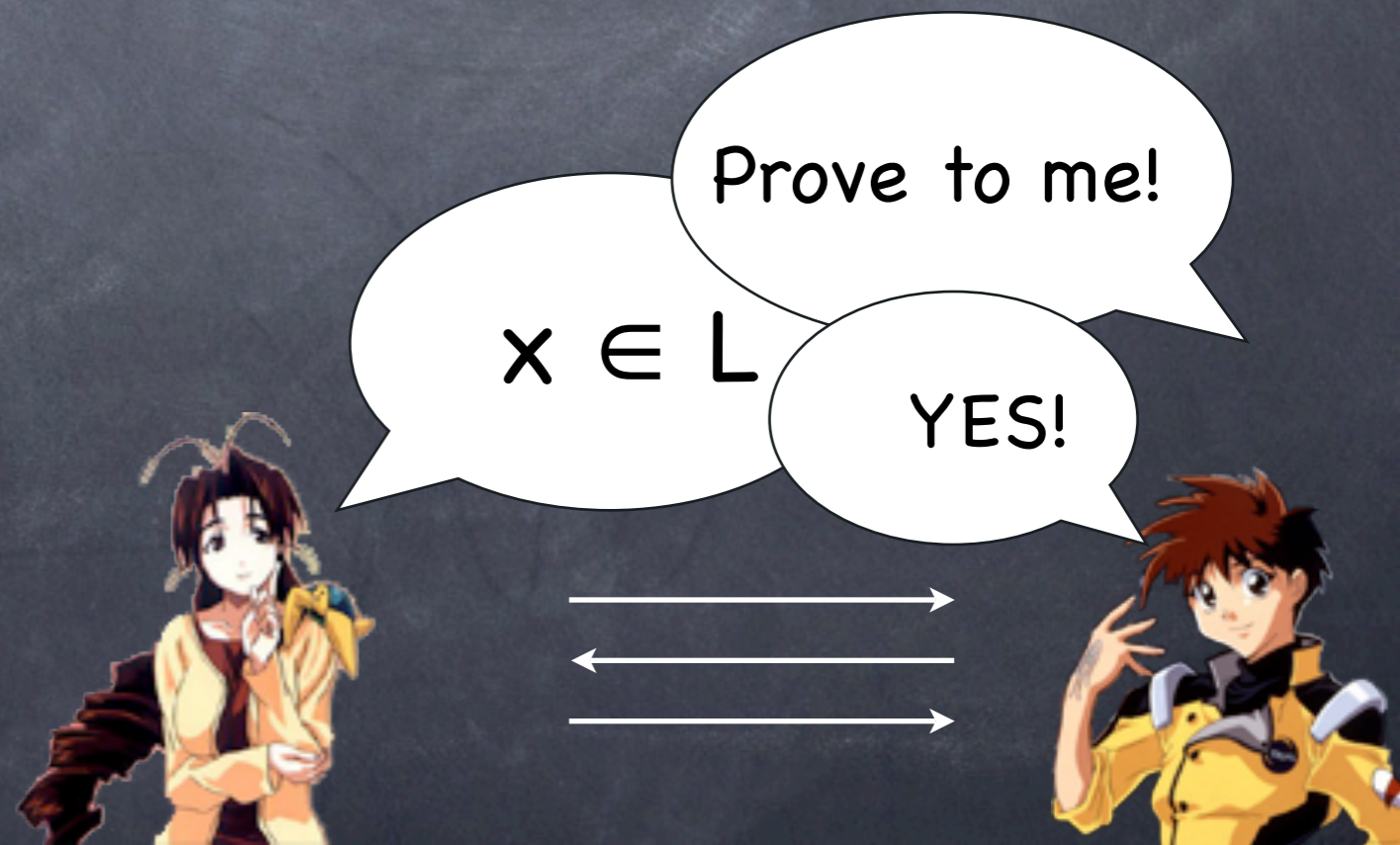
Interactive Proofs

- A generalization of “proof”:
 - An interaction wherein a **Prover** tries to convince **verifier** that x has some property
 - i.e. that x is in language L
- All powerful prover, computationally bounded verifier



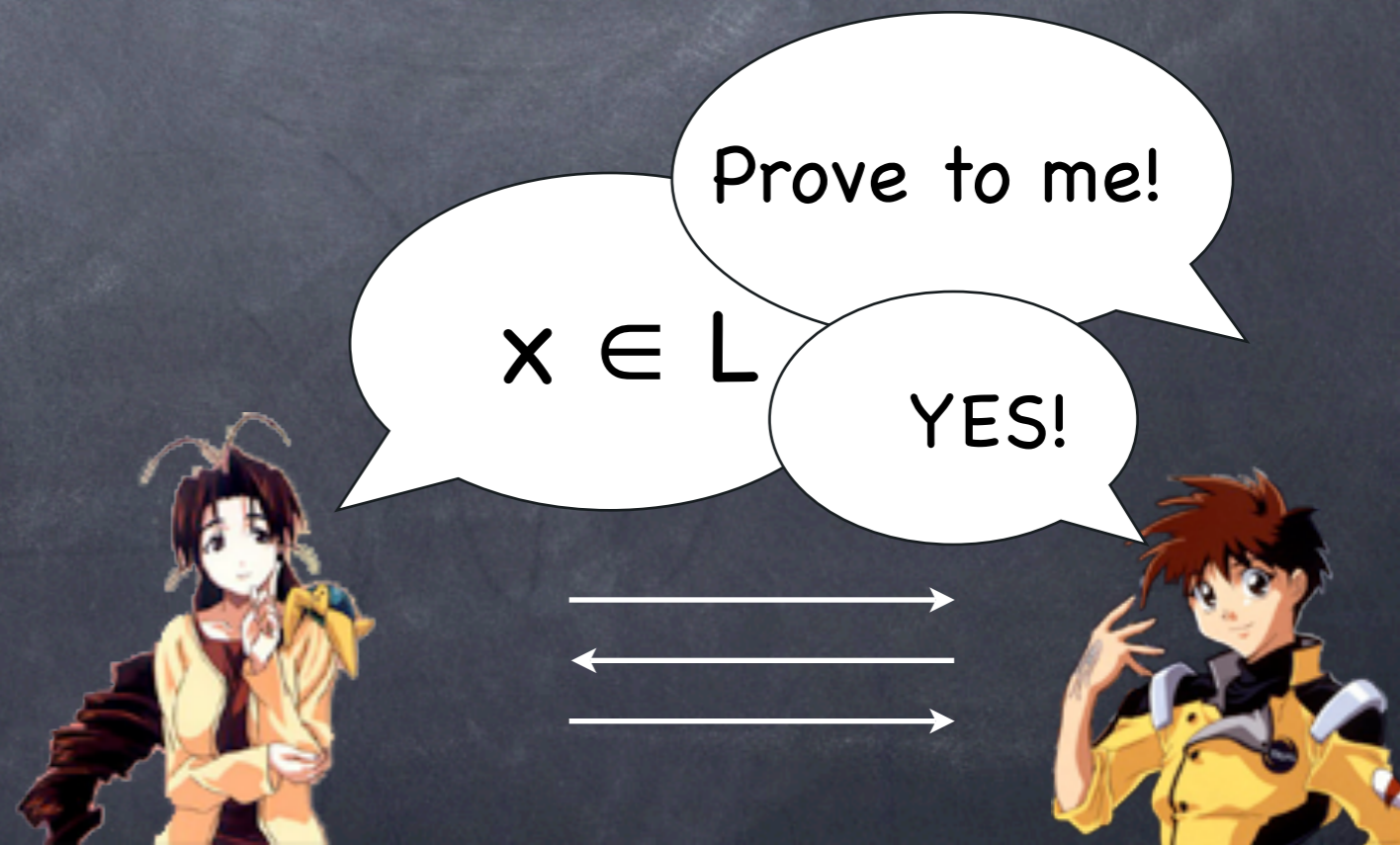
Interactive Proofs

- A generalization of “proof”:
 - An interaction wherein a **Prover** tries to convince **verifier** that x has some property
 - i.e. that x is in language L
 - All powerful prover, computationally bounded verifier
 - Verifier doesn't trust prover



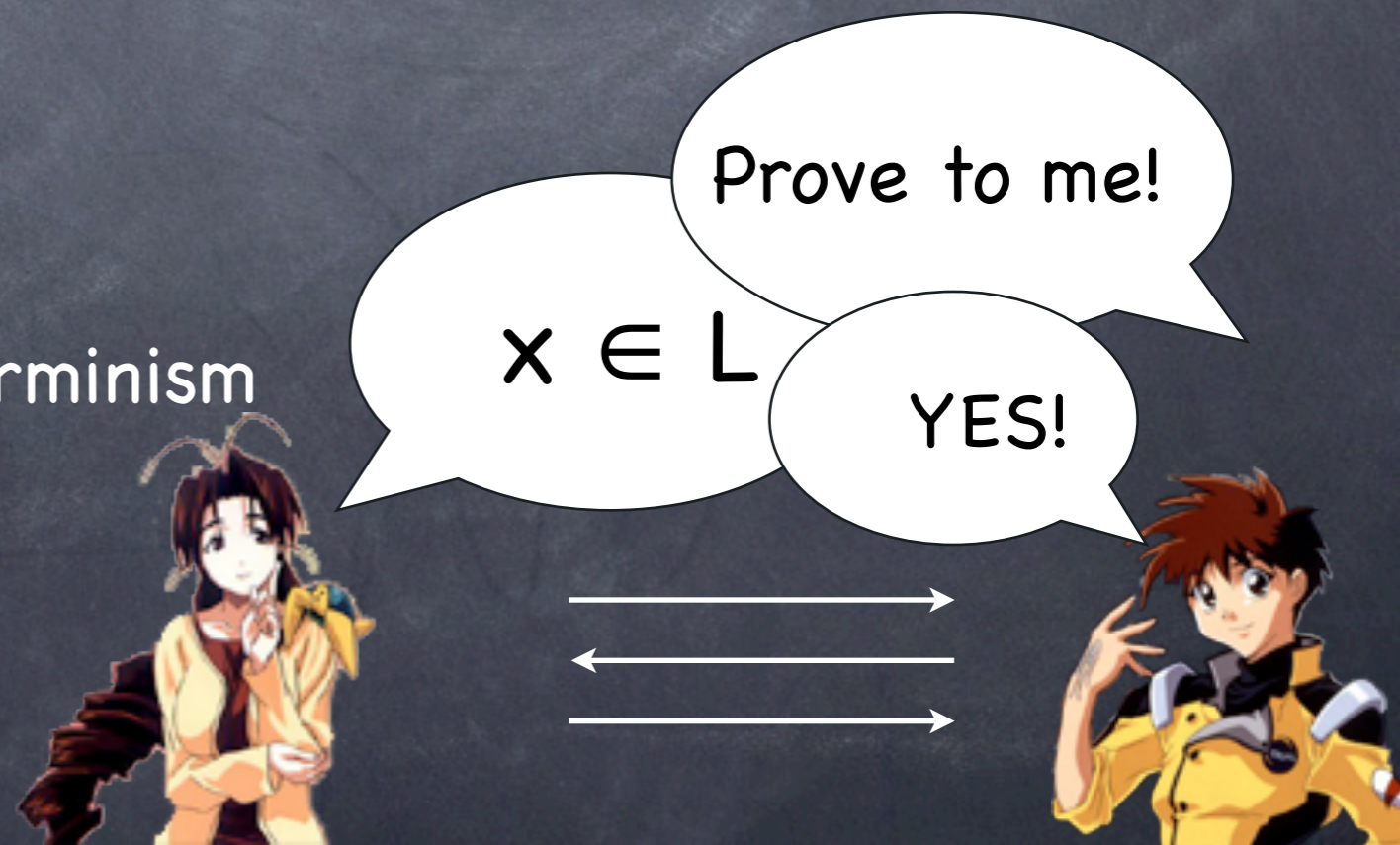
Interactive Proofs

- A generalization of “proof”:
 - An interaction wherein a **Prover** tries to convince **verifier** that x has some property
 - i.e. that x is in language L
 - All powerful prover, computationally bounded verifier
 - Verifier doesn't trust prover
 - Limits the power



Interactive Proofs

- A generalization of “proof”:
 - An interaction wherein a **Prover** tries to convince **verifier** that x has some property
 - i.e. that x is in language L
 - All powerful prover, computationally bounded verifier
 - Verifier doesn't trust prover
 - Limits the power
 - A generalization of non-determinism



Interactive Proofs



Interactive Proofs

- **Completeness**



Interactive Proofs

- **Completeness**

- If x in L , **honest Prover** should convince **honest Verifier**



Interactive Proofs

- **Completeness**
 - If x in L , **honest Prover** should convince **honest Verifier**
- **Soundness**



Interactive Proofs

- **Completeness**

- If x in L , **honest Prover** should convince **honest Verifier**

- **Soundness**

- If x not in L , **honest Verifier** won't accept **any purported proof**



Interactive Proofs

- **Completeness**

- If x in L , **honest Prover** should convince **honest Verifier**

- **Soundness**

- If x not in L , **honest Verifier** won't accept **any purported proof**



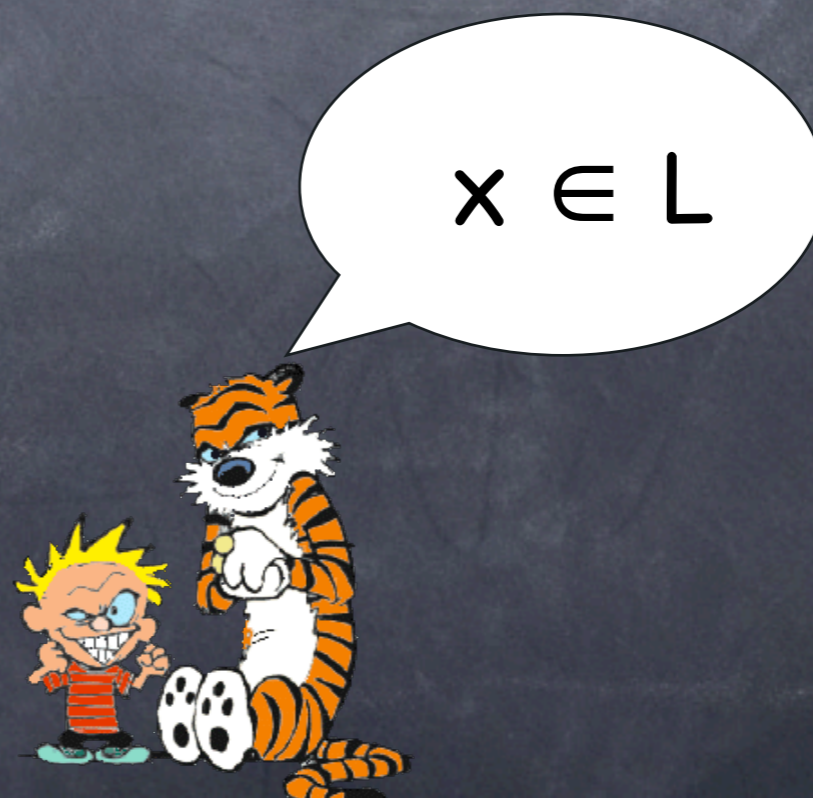
Interactive Proofs

- **Completeness**

- If x in L , **honest Prover** should convince **honest Verifier**

- **Soundness**

- If x not in L , **honest Verifier** won't accept **any purported proof**



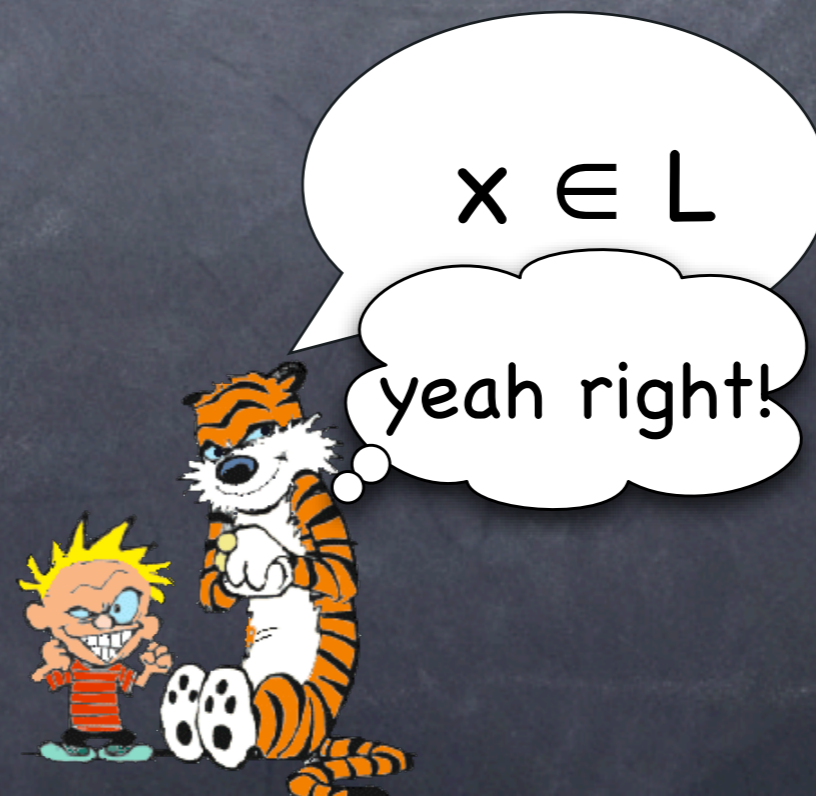
Interactive Proofs

- **Completeness**

- If x in L , **honest Prover** should convince **honest Verifier**

- **Soundness**

- If x not in L , **honest Verifier** won't accept **any** **purported proof**



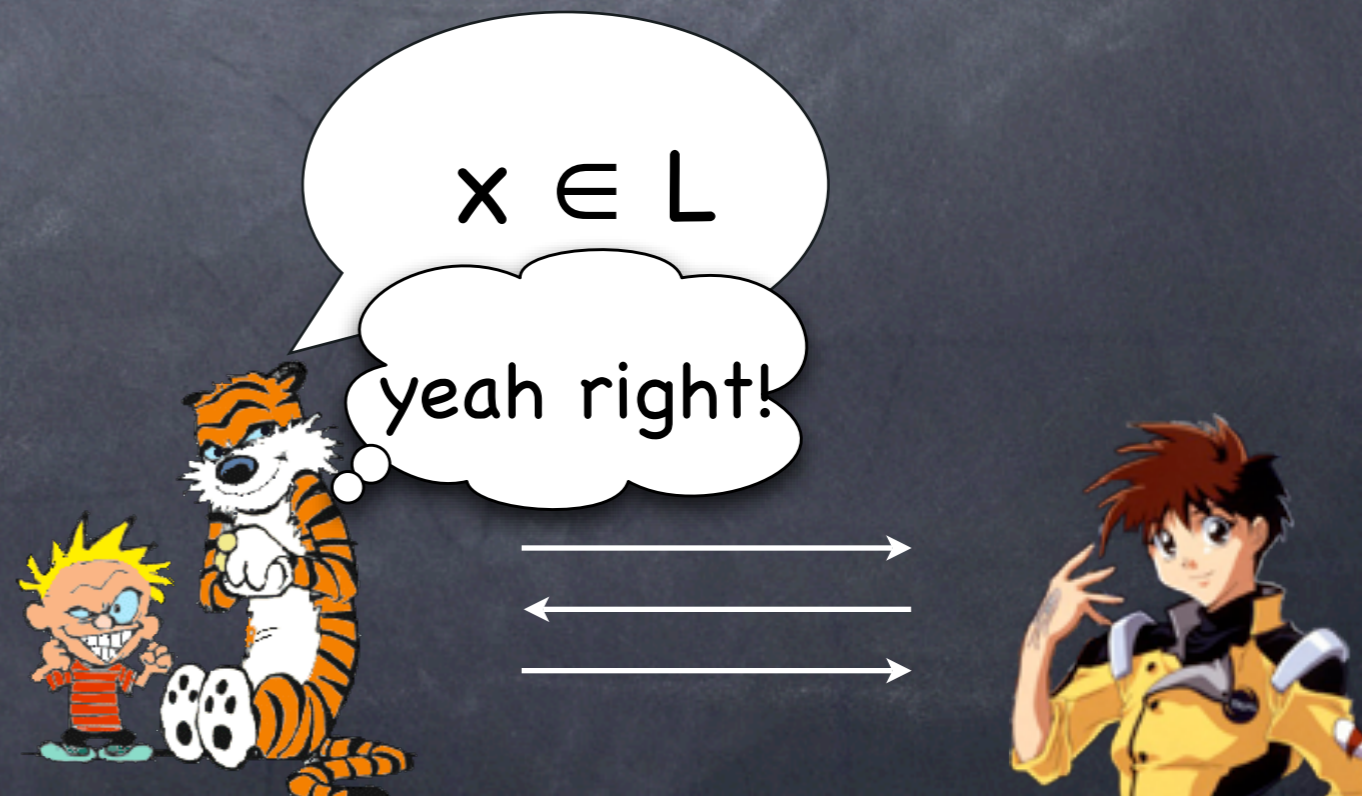
Interactive Proofs

- **Completeness**

- If x in L , **honest Prover** should convince **honest Verifier**

- **Soundness**

- If x not in L , **honest Verifier** won't accept **any** **purported proof**



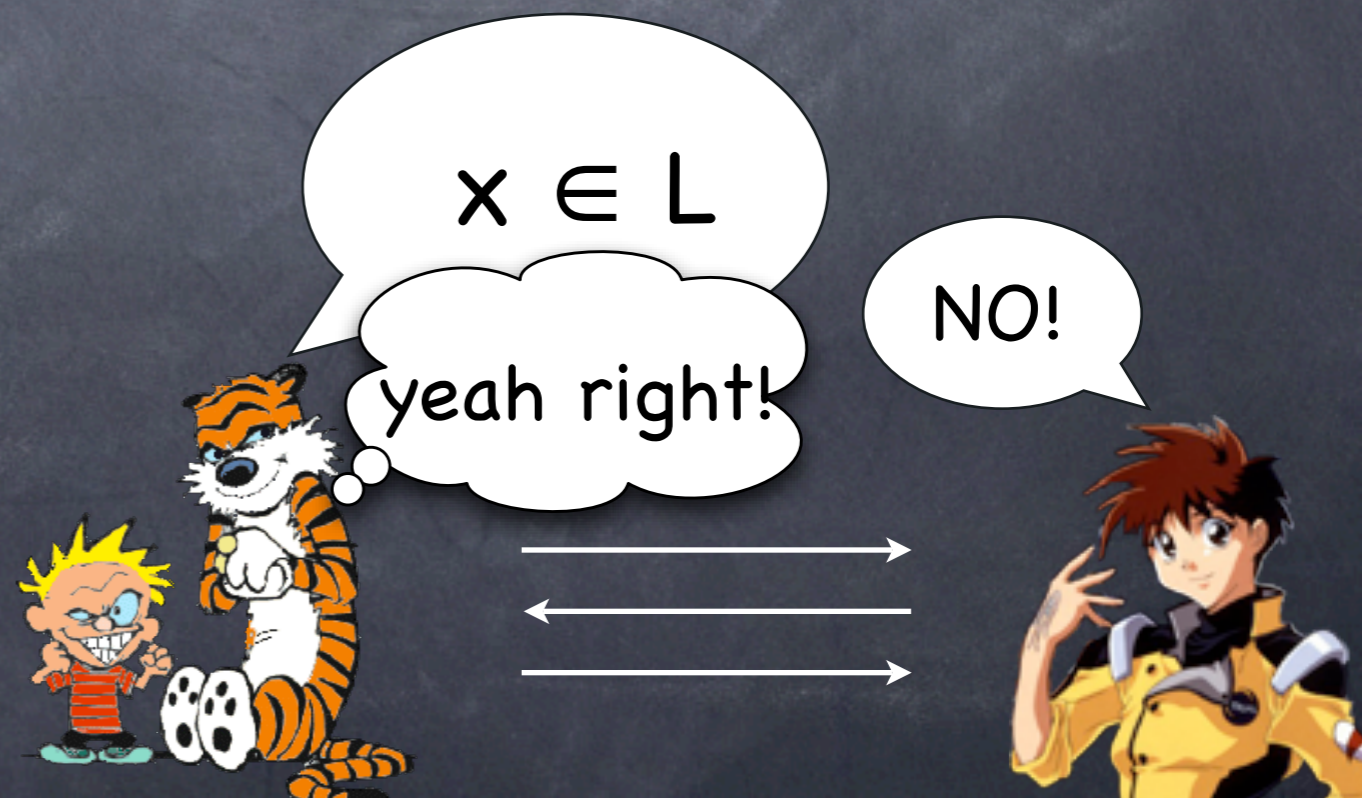
Interactive Proofs

- **Completeness**

- If x in L , **honest Prover** should convince **honest Verifier**

- **Soundness**

- If x not in L , **honest Verifier** won't accept **any** **purported proof**



An Example



An Example

- **Coke in bottle or can**



An Example

- **Coke in bottle or can**
- Prover claims: coke in bottle and coke in can are different



An Example

- **Coke in bottle or can**
- Prover claims: coke in bottle and coke in can are different
- IP protocol:



An Example

- **Coke in bottle or can**
- Prover claims: coke in bottle and coke in can are different
- IP protocol:



An Example

- **Coke in bottle or can**
- Prover claims: coke in bottle and coke in can are different
- IP protocol:

Pour into
from can or
bottle



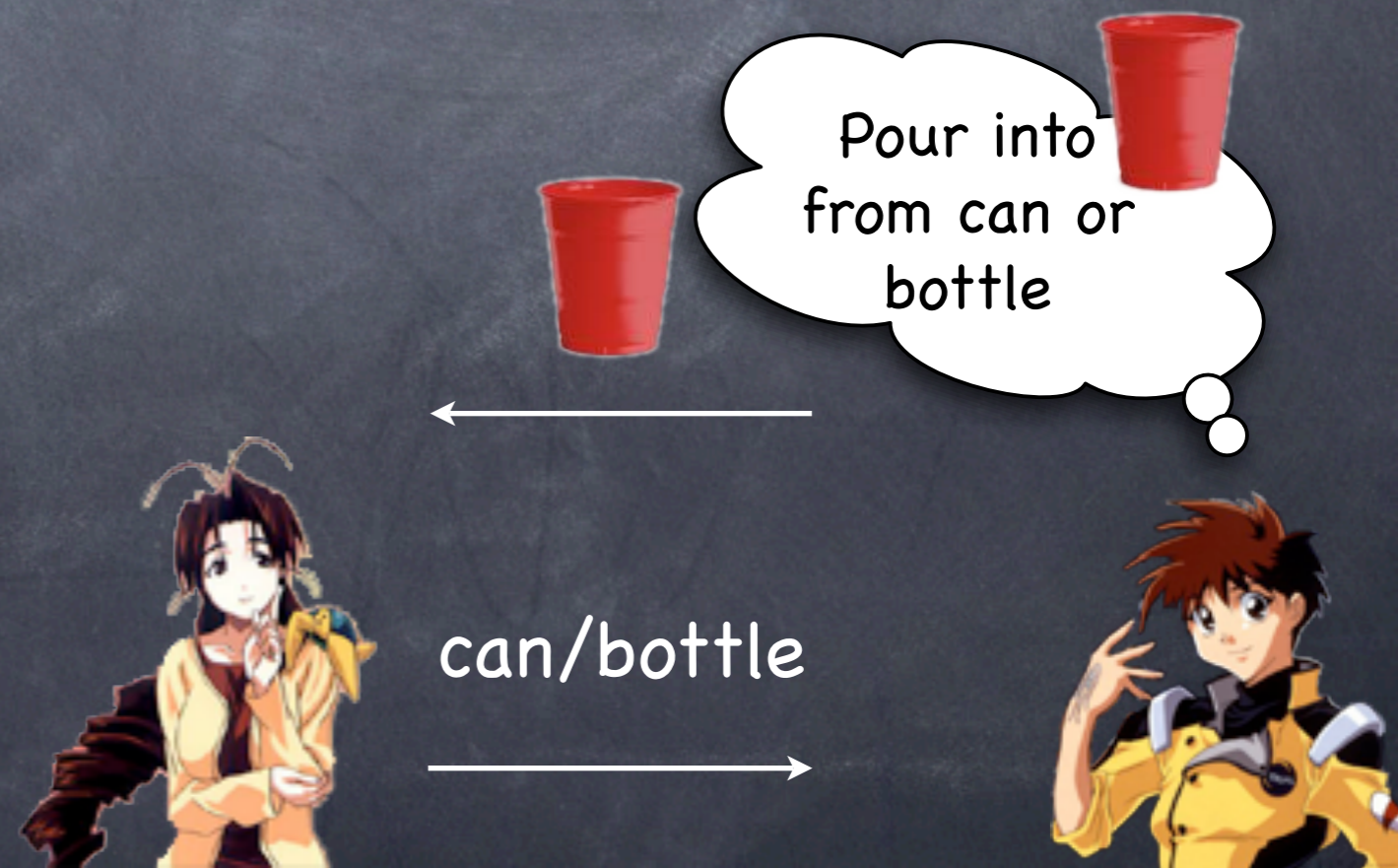
An Example

- **Coke in bottle or can**
- Prover claims: coke in bottle and coke in can are different
- IP protocol:



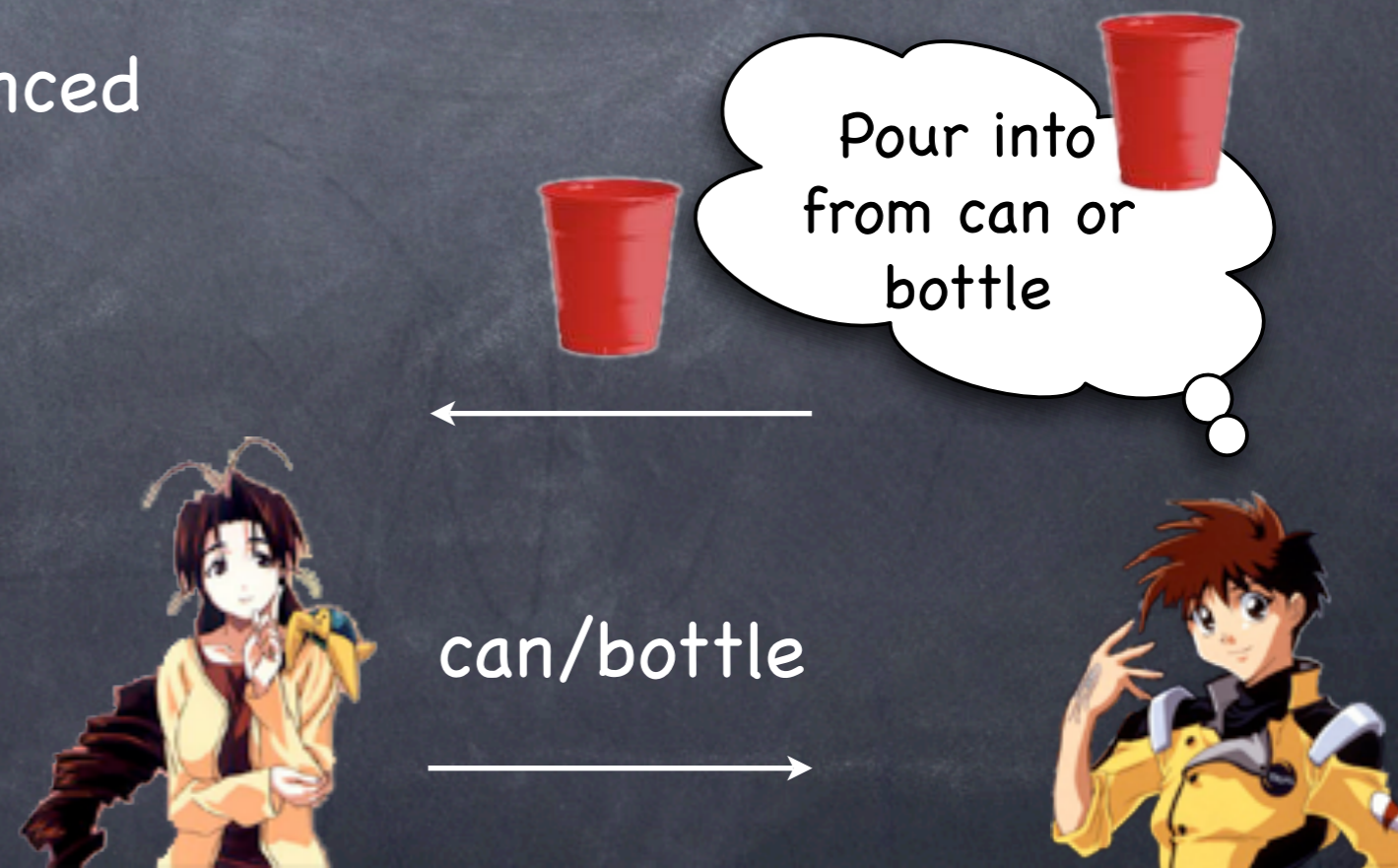
An Example

- **Coke in bottle or can**
 - Prover claims: coke in bottle and coke in can are different
- IP protocol:
 - prover tells whether cup was filled from can or bottle



An Example

- **Coke in bottle or can**
 - Prover claims: coke in bottle and coke in can are different
- IP protocol:
 - prover tells whether cup was filled from can or bottle
 - repeat till verifier is convinced



Proofs in Cryptography

Proofs in Cryptography

- Alice opening her commitment (by sending the key) is a (non-interactive) proof that $\text{Box}(m)$ contains m

Proofs in Cryptography

- Alice opening her commitment (by sending the key) is a (non-interactive) proof that $\text{Box}(m)$ contains m
- Sometimes Alice may want to not reveal x , but still convince Bob that $\text{Box}(m)$ contains m

Proofs in Cryptography

- Alice opening her commitment (by sending the key) is a (non-interactive) proof that $\text{Box}(m)$ contains m
- Sometimes Alice may want to not reveal x , but still convince Bob that $\text{Box}(m)$ contains m
 - e.g., $\text{Box}(m_1, m_2) = (f(w), H_1(w) \oplus m_1, H_2(w) \oplus m_2)$, and Alice wants to reveal only m_1

Proofs in Cryptography

- Alice opening her commitment (by sending the key) is a (non-interactive) proof that $\text{Box}(m)$ contains m
- Sometimes Alice may want to not reveal x , but still convince Bob that $\text{Box}(m)$ contains m
 - e.g., $\text{Box}(m_1, m_2) = (f(w), H_1(w) \oplus m_1, H_2(w) \oplus m_2)$, and Alice wants to reveal only m_1
 - i.e., if $\text{Box}(m_1, m_2) = (a, b, c)$, prove that **there exists w** such that $f(w)=a$ and $H_1(w) \oplus m_1 = b$

Proofs in Cryptography

- Alice opening her commitment (by sending the key) is a (non-interactive) proof that $\text{Box}(m)$ contains m
- Sometimes Alice may want to not reveal x , but still convince Bob that $\text{Box}(m)$ contains m
 - e.g., $\text{Box}(m_1, m_2) = (f(w), H_1(w) \oplus m_1, H_2(w) \oplus m_2)$, and Alice wants to reveal only m_1
 - i.e., if $\text{Box}(m_1, m_2) = (a, b, c)$, prove that **there exists w** such that $f(w)=a$ and $H_1(w) \oplus m_1 = b$
 - **Without revealing anything else about w**

Proofs in Cryptography

- Alice opening her commitment (by sending the key) is a (non-interactive) proof that $\text{Box}(m)$ contains m
- Sometimes Alice may want to not reveal x , but still convince Bob that $\text{Box}(m)$ contains m
 - e.g., $\text{Box}(m_1, m_2) = (f(w), H_1(w) \oplus m_1, H_2(w) \oplus m_2)$, and Alice wants to reveal only m_1
 - i.e., if $\text{Box}(m_1, m_2) = (a, b, c)$, prove that **there exists w** such that $f(w)=a$ and $H_1(w) \oplus m_1 = b$
 - **Without revealing anything else about w**
 - **Zero-Knowledge proof!**

Zero-Knowledge

Zero-Knowledge

- Zero-Knowledge is a general cryptographic principle

Zero-Knowledge

- Zero-Knowledge is a general cryptographic principle
 - Define exactly what each party is allowed to learn ideally; beyond that they should receive zero-knowledge

Zero-Knowledge

- Zero-Knowledge is a general cryptographic principle
 - Define exactly what each party is allowed to learn ideally; beyond that they should receive zero-knowledge
 - How to formalize this?

Zero-Knowledge

- Zero-Knowledge is a general cryptographic principle
 - Define exactly what each party is allowed to learn ideally; beyond that they should receive zero-knowledge
 - How to formalize this?
 - "Simulation"

Zero-Knowledge

- Zero-Knowledge is a general cryptographic principle
 - Define exactly what each party is allowed to learn ideally; beyond that they should receive zero-knowledge
 - How to formalize this?
 - "Simulation"
 - In an ideal setting the adversary can simulate by itself the experience of being in the actual protocol

Zero-Knowledge

- Zero-Knowledge is a general cryptographic principle
 - Define exactly what each party is allowed to learn ideally; beyond that they should receive zero-knowledge
 - How to formalize this?
 - "Simulation"
 - In an ideal setting the adversary can simulate by itself the experience of being in the actual protocol
 - So actual protocol is not leaking anything beyond the ideal setting

Zero-Knowledge

- Zero-Knowledge is a general cryptographic principle
 - Define exactly what each party is allowed to learn ideally; beyond that they should receive zero-knowledge
 - How to formalize this?
 - "Simulation"
 - In an ideal setting the adversary can simulate by itself the experience of being in the actual protocol
 - So actual protocol is not leaking anything beyond the ideal setting
 - e.g. Coke-protocol: verifier can simulate the correct answers by itself

Zero-Knowledge Proofs

- Zero-knowledge property:
 - Verifier's view could have been "simulated"



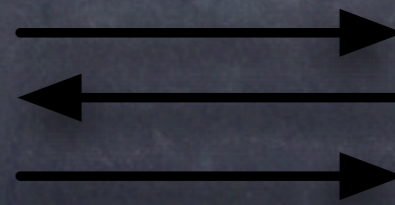
Zero-Knowledge Proofs

- Zero-knowledge property:
 - Verifier's view could have been "simulated"



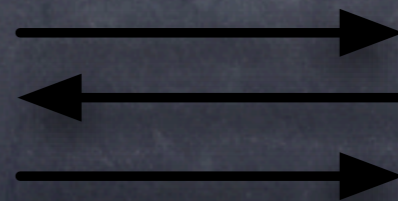
Zero-Knowledge Proofs

- Zero-knowledge property:
 - Verifier's view could have been "simulated"



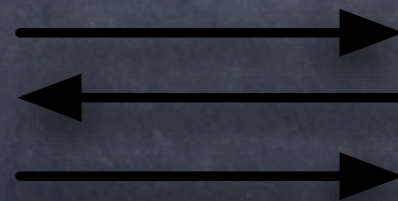
Zero-Knowledge Proofs

- Zero-knowledge property:
 - Verifier's view could have been "simulated"



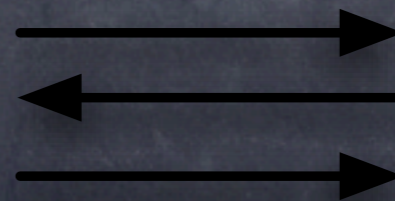
Zero-Knowledge Proofs

- Zero-knowledge property:
 - Verifier's view could have been "simulated"



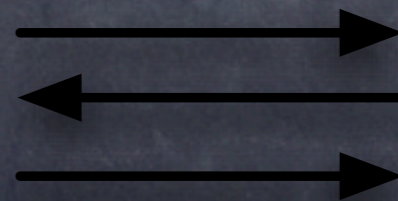
Zero-Knowledge Proofs

- Zero-knowledge property:
 - Verifier's view could have been "simulated"



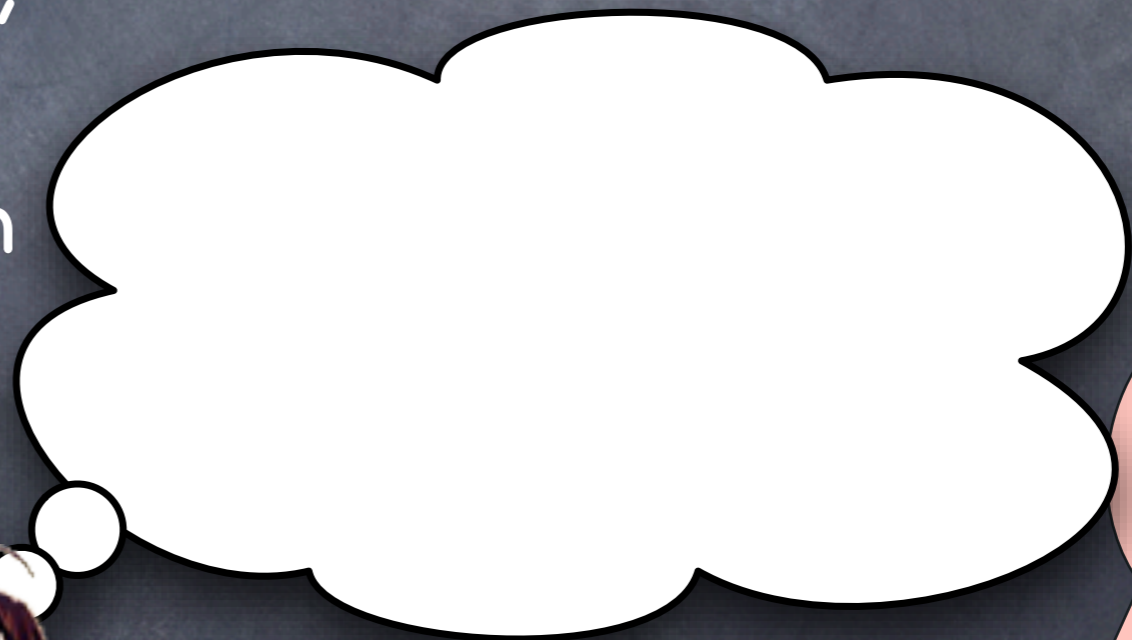
Zero-Knowledge Proofs

- Zero-knowledge property:
 - Verifier's view could have been "simulated"
- For every adversarial strategy in the protocol, there is a strategy in the "ideal world", which can be used to obtain the same view

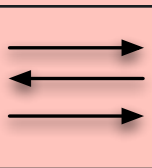


Zero-Knowledge Proofs

- Zero-knowledge property:
 - Verifier's view could have been "simulated"
- For every adversarial strategy in the protocol, there is a strategy in the "ideal world", which can be used to obtain the same view

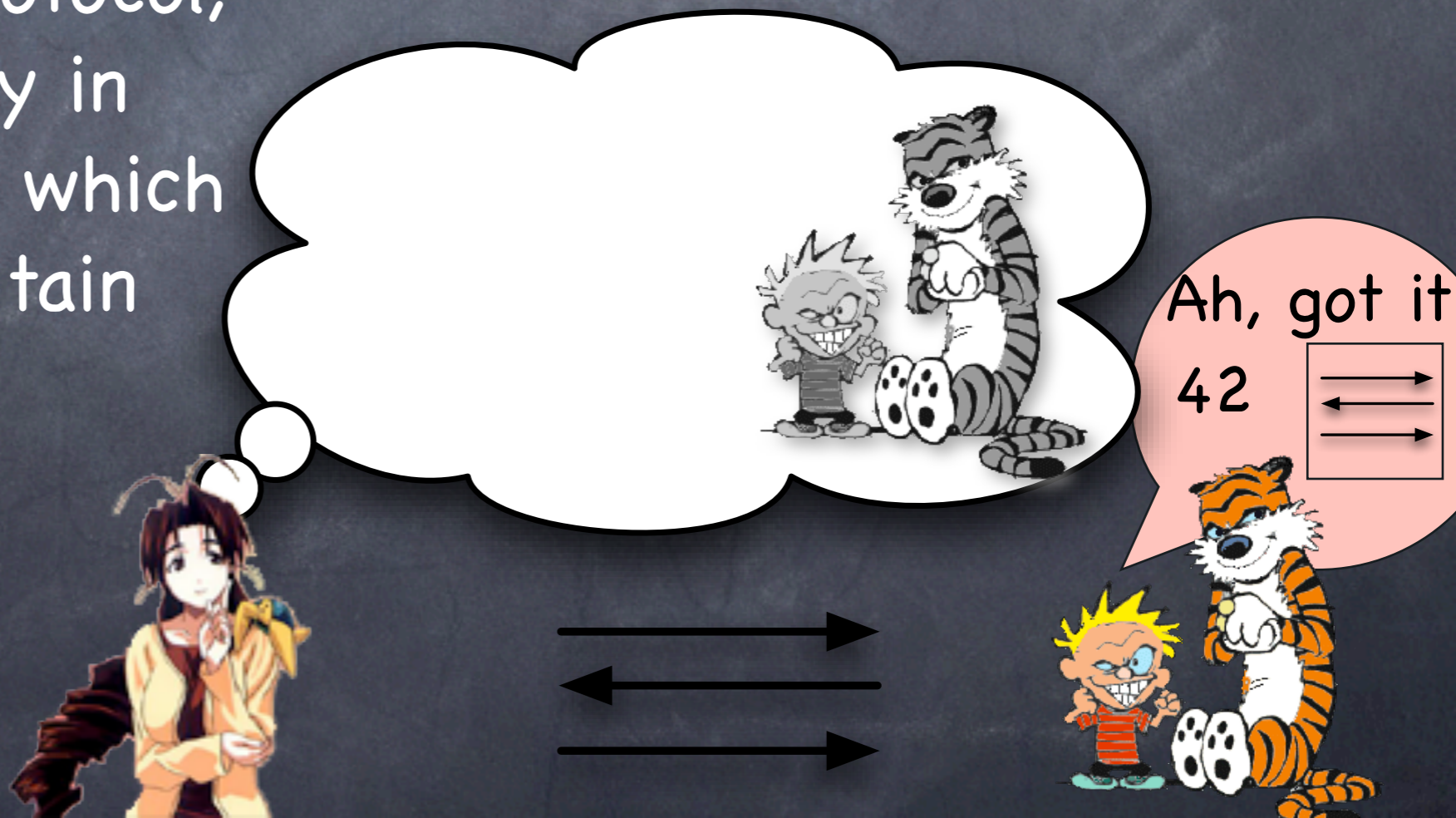


Ah, got it
42



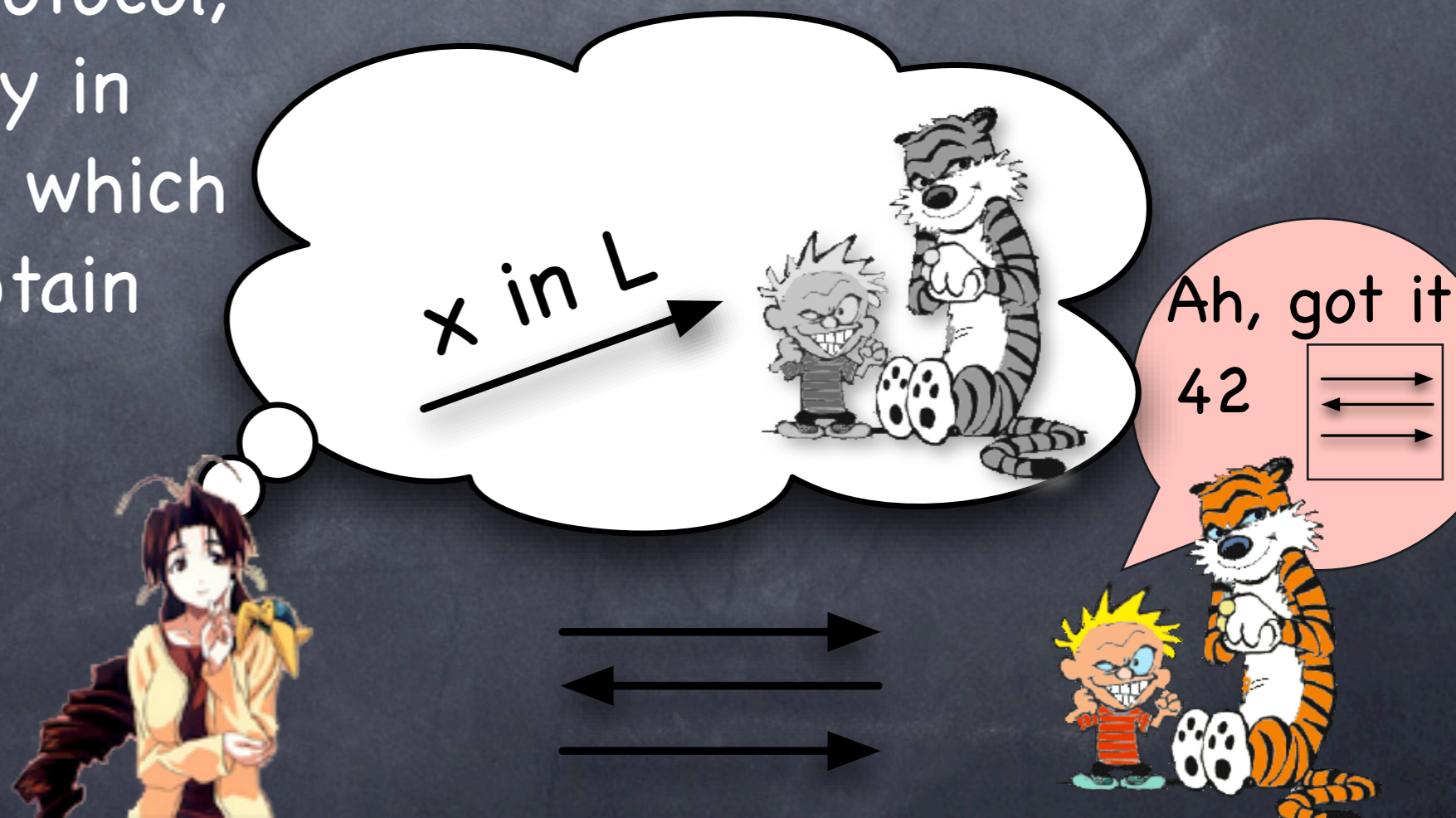
Zero-Knowledge Proofs

- Zero-knowledge property:
 - Verifier's view could have been "simulated"
- For every adversarial strategy in the protocol, there is a strategy in the "ideal world", which can be used to obtain the same view



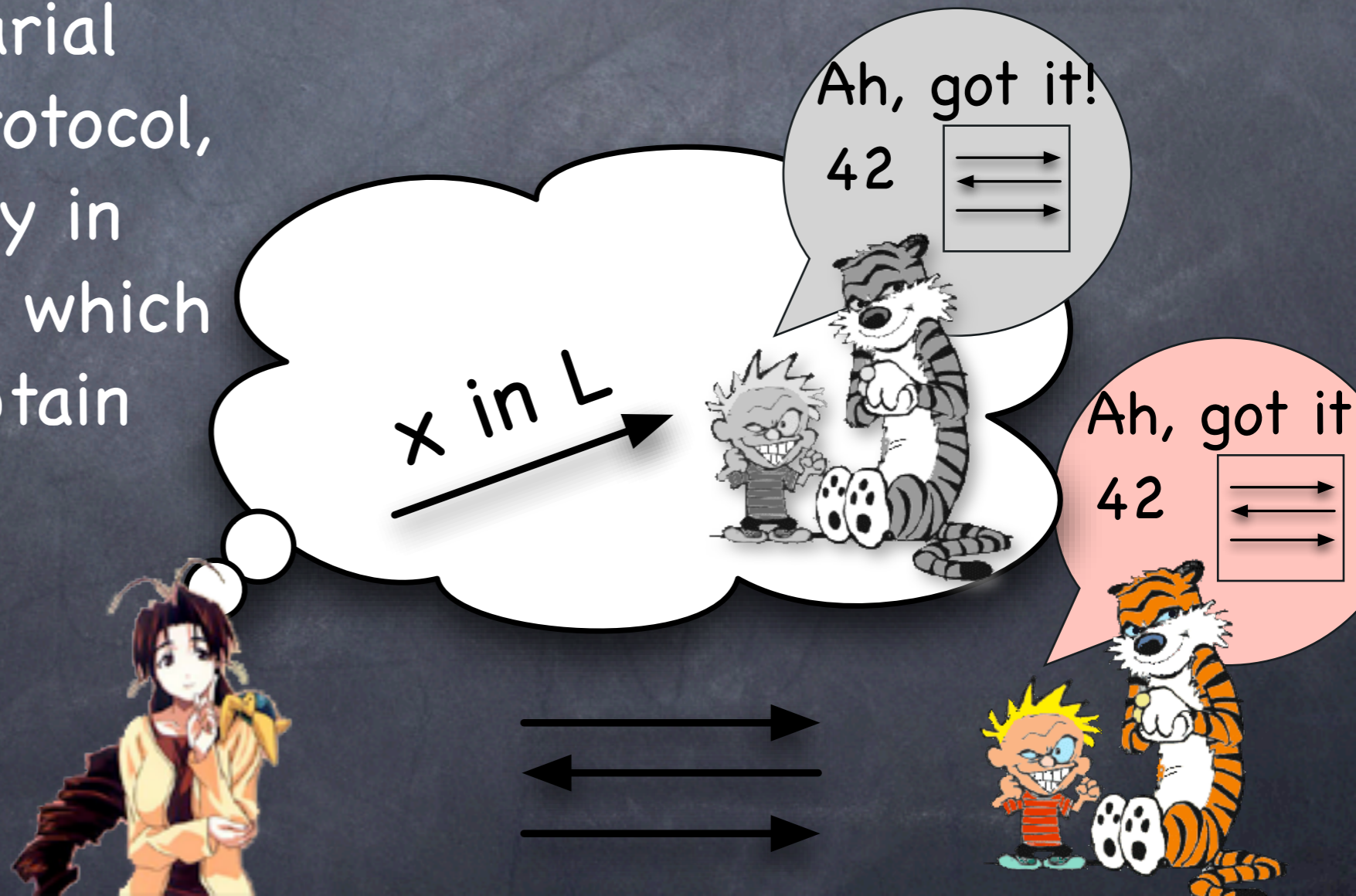
Zero-Knowledge Proofs

- Zero-knowledge property:
 - Verifier's view could have been "simulated"
- For every adversarial strategy in the protocol, there is a strategy in the "ideal world", which can be used to obtain the same view



Zero-Knowledge Proofs

- Zero-knowledge property:
 - Verifier's view could have been "simulated"
- For every adversarial strategy in the protocol, there is a strategy in the "ideal world", which can be used to obtain the same view



Secure Multi-party Computation

Secure Multi-party Computation

- **Goal:** To emulate the presence of a trusted third party using a protocol

Secure Multi-party Computation

- **Goal:** To emulate the presence of a trusted third party using a protocol
- **e.g. ZK proofs:** Prover and verifier will emulate the presence of a mediator. Prover will send a (non-zero-knowledge) proof to the mediator, who verifies it and tells the verifier yes or no.

Secure Multi-party Computation

- **Goal:** To emulate the presence of a trusted third party using a protocol
- **e.g. ZK proofs:** Prover and verifier will emulate the presence of a mediator. Prover will send a (non-zero-knowledge) proof to the mediator, who verifies it and tells the verifier yes or no.
- **Several other possibilities:** privacy-preserving data-mining, online games, auctions, voting etc. without a trusted central authority, ...

Secure Multi-party Computation

- **Goal:** To emulate the presence of a trusted third party using a protocol
- **e.g. ZK proofs:** Prover and verifier will emulate the presence of a mediator. Prover will send a (non-zero-knowledge) proof to the mediator, who verifies it and tells the verifier yes or no.
- Several other possibilities: privacy-preserving data-mining, online games, auctions, voting etc. without a trusted central authority, ...
 - Well developed theory, slowly moving into practice

Secure Multi-party Computation

- **Goal:** To emulate the presence of a trusted third party using a protocol
- **e.g. ZK proofs:** Prover and verifier will emulate the presence of a mediator. Prover will send a (non-zero-knowledge) proof to the mediator, who verifies it and tells the verifier yes or no.
- Several other possibilities: privacy-preserving data-mining, online games, auctions, voting etc. without a trusted central authority, ...
 - Well developed theory, slowly moving into practice
- **Security depends on gap between ease of verifying and of solving, and related phenomena**

Secure Multi-party Computation

- **Goal:** To emulate the presence of a trusted third party using a protocol
- **e.g. ZK proofs:** Prover and verifier will emulate the presence of a mediator. Prover will send a (non-zero-knowledge) proof to the mediator, who verifies it and tells the verifier yes or no.
- Several other possibilities: privacy-preserving data-mining, online games, auctions, voting etc. without a trusted central authority, ...
 - Well developed theory, slowly moving into practice
- **Security depends on gap between ease of verifying and of solving, and related phenomena**
 - Studied in computational complexity theory

That's all folks!

That's all folks!

- In this course:

That's all folks!

- In this course:

- Abstracting computational problems (languages) and computational models (DFA, PDA, TM)

That's all folks!

- In this course:
 - Abstracting **computational problems** (languages) and **computational models** (DFA, PDA, TM)
 - Reasoning about **power and limitations** of computational models

That's all folks!

- In this course:

- Abstracting **computational problems** (languages) and **computational models** (DFA, PDA, TM)
- Reasoning about **power and limitations** of computational models
 - Irregularity, Non-context-freeness, undecidability, unrecognizability

That's all folks!

- In this course:

- Abstracting computational problems (languages) and computational models (DFA, PDA, TM)
- Reasoning about power and limitations of computational models
 - Irregularity, Non-context-freeness, undecidability, unrecognizability

- Beyond:

That's all folks!

- In this course:

- Abstracting computational problems (languages) and computational models (DFA, PDA, TM)
- Reasoning about power and limitations of computational models
 - Irregularity, Non-context-freeness, undecidability, unrecognizability

- Beyond:

- Formal methods for reasoning, Natural Language Processing

That's all folks!

- In this course:

- Abstracting computational problems (languages) and computational models (DFA, PDA, TM)
- Reasoning about power and limitations of computational models
 - Irregularity, Non-context-freeness, undecidability, unrecognizability

- Beyond:

- Formal methods for reasoning, Natural Language Processing
- Computational complexity (with TMs)

That's all folks!

- In this course:

- Abstracting computational problems (languages) and computational models (DFA, PDA, TM)
- Reasoning about power and limitations of computational models
 - Irregularity, Non-context-freeness, undecidability, unrecognizability

- Beyond:

- Formal methods for reasoning, Natural Language Processing
- Computational complexity (with TMs)
 - Studying further limitations: e.g. hardness of solving problems which have proofs that are easy to verify

That's all folks!

- In this course:

- Abstracting **computational problems** (languages) and **computational models** (DFA, PDA, TM)
- Reasoning about **power and limitations** of computational models
 - Irregularity, Non-context-freeness, undecidability, unrecognizability

- Beyond:

- **Formal methods for reasoning, Natural Language Processing**
- **Computational complexity (with TMs)**
 - Studying further limitations: e.g. hardness of solving problems which have proofs that are easy to verify
 - Also other computational models: e.g. quantum computation

That's all folks!

- In this course:

- Abstracting computational problems (languages) and computational models (DFA, PDA, TM)
- Reasoning about power and limitations of computational models
 - Irregularity, Non-context-freeness, undecidability, unrecognizability

- Beyond:

- Formal methods for reasoning, Natural Language Processing
- Computational complexity (with TMs)
 - Studying further limitations: e.g. hardness of solving problems which have proofs that are easy to verify
 - Also other computational models: e.g. quantum computation
 - Applications: e.g. cryptography