Defining Encryption (ctd.)

Lecture 3

CPA/CCA security

Computational Indistinguishability

Pseudo-randomness, One-Way Functions

Security of Encryption

Security of Encryption

- Perfect secrecy (IND-Onetime security) is too strong (though too weak in some other respects...)

Security of Encryption

- Perfect secrecy (IND-Onetime security) is too strong (though too weak in some other respects...)
  - Necessitates keys as long as the messages

Security of Encryption

- Perfect secrecy (IND-Onetime security) is too strong (though too weak in some other respects...)
  - Necessitates keys as long as the messages
- Relax the requirement by restricting to computationally bounded adversaries (and environments)

Security of Encryption

- Perfect secrecy (IND-Onetime security) is too strong (though too weak in some other respects...)
  - Necessitates keys as long as the messages
- Relax the requirement by restricting to computationally bounded adversaries (and environments)
- Coming up: Formalizing notions of "computational" security (as opposed to perfect/statistical security)

Security of Encryption

- Perfect secrecy (IND-Onetime security) is too strong (though too weak in some other respects...)
  - Necessitates keys as long as the messages
- Relax the requirement by restricting to computationally bounded adversaries (and environments)
- Coming up: Formalizing notions of "computational" security (as opposed to perfect/statistical security)
  - Then, security definitions used for encryption of multiple messages

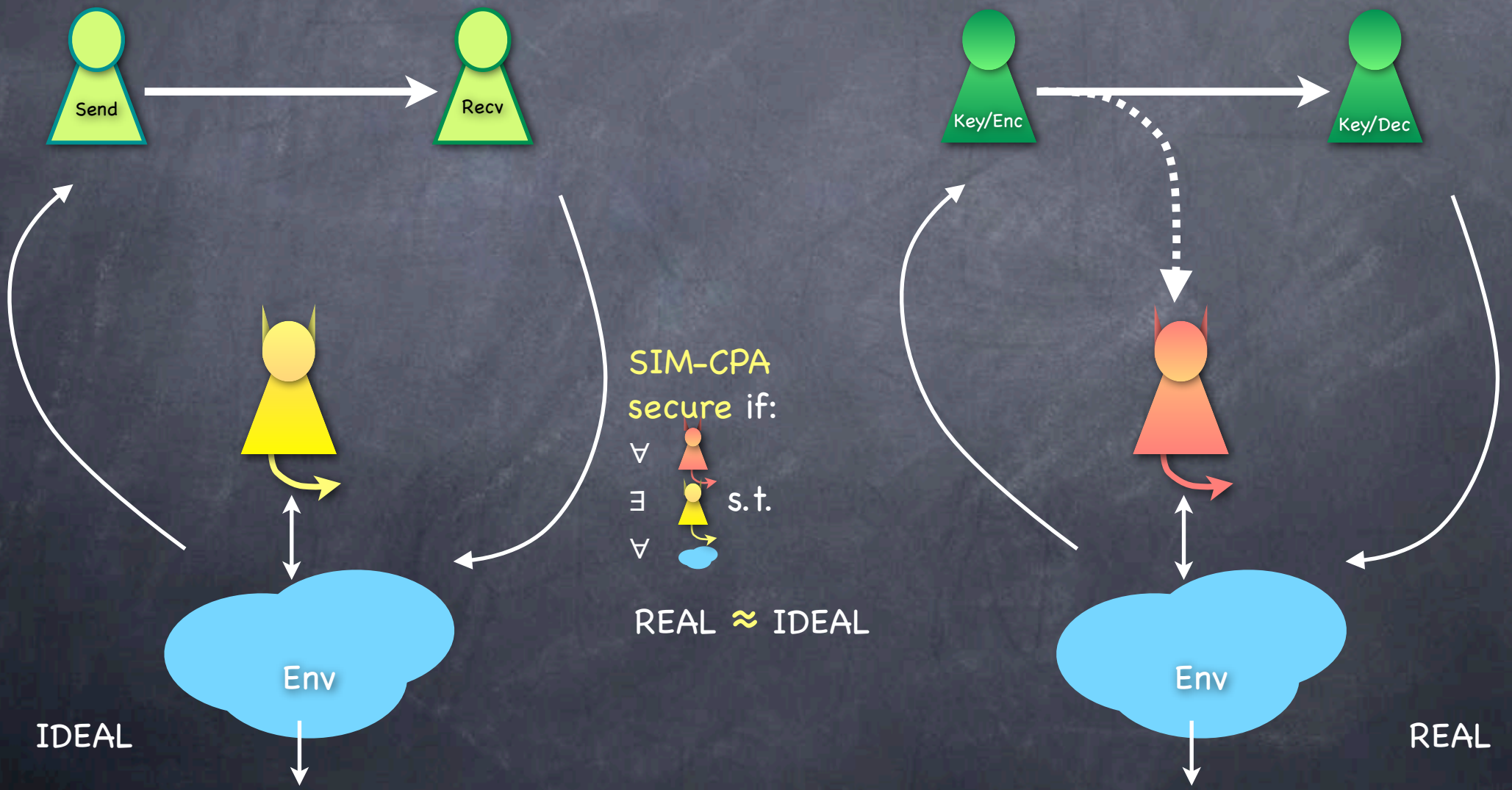
Symmetric-Key Encryption

The Syntax

- Shared-key (Private-key) Encryption
  - Key Generation: Randomized
    - $K \leftarrow \mathcal{K}$, uniformly randomly drawn from the key-space (or according to a key-distribution)
  - Encryption: Randomized
    - Enc: $\mathcal{M} \times \mathcal{K} \times \mathcal{R} \rightarrow \mathcal{C}$. During encryption a fresh random string will be chosen uniformly at random from $\mathcal{R}$
  - Decryption: Deterministic
    - Dec: $\mathcal{C} \times \mathcal{K} \rightarrow \mathcal{M}$

Symmetric-Key Encryption

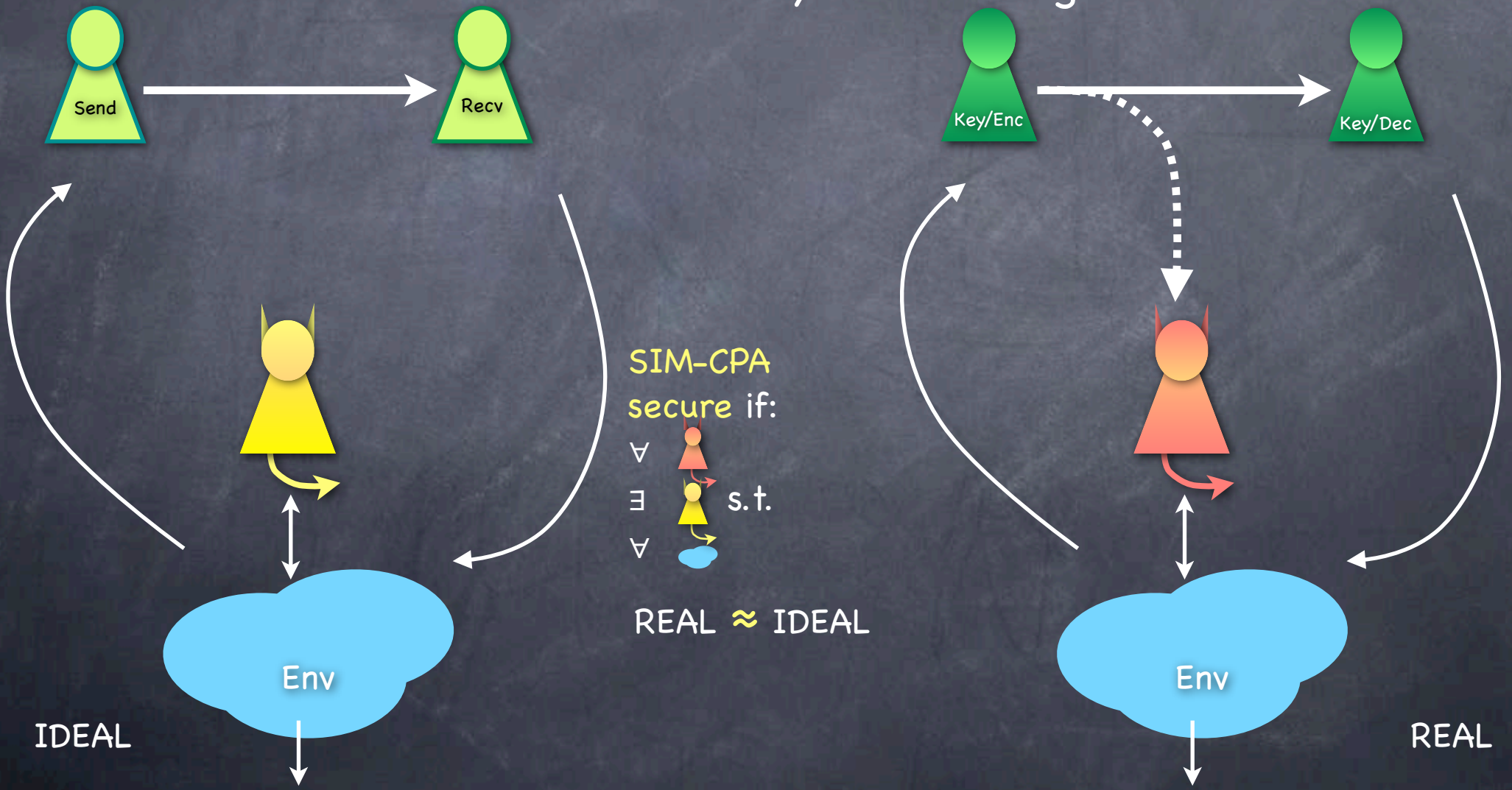
SIM-CPA Security



Symmetric-Key Encryption

SIM-CPA Security

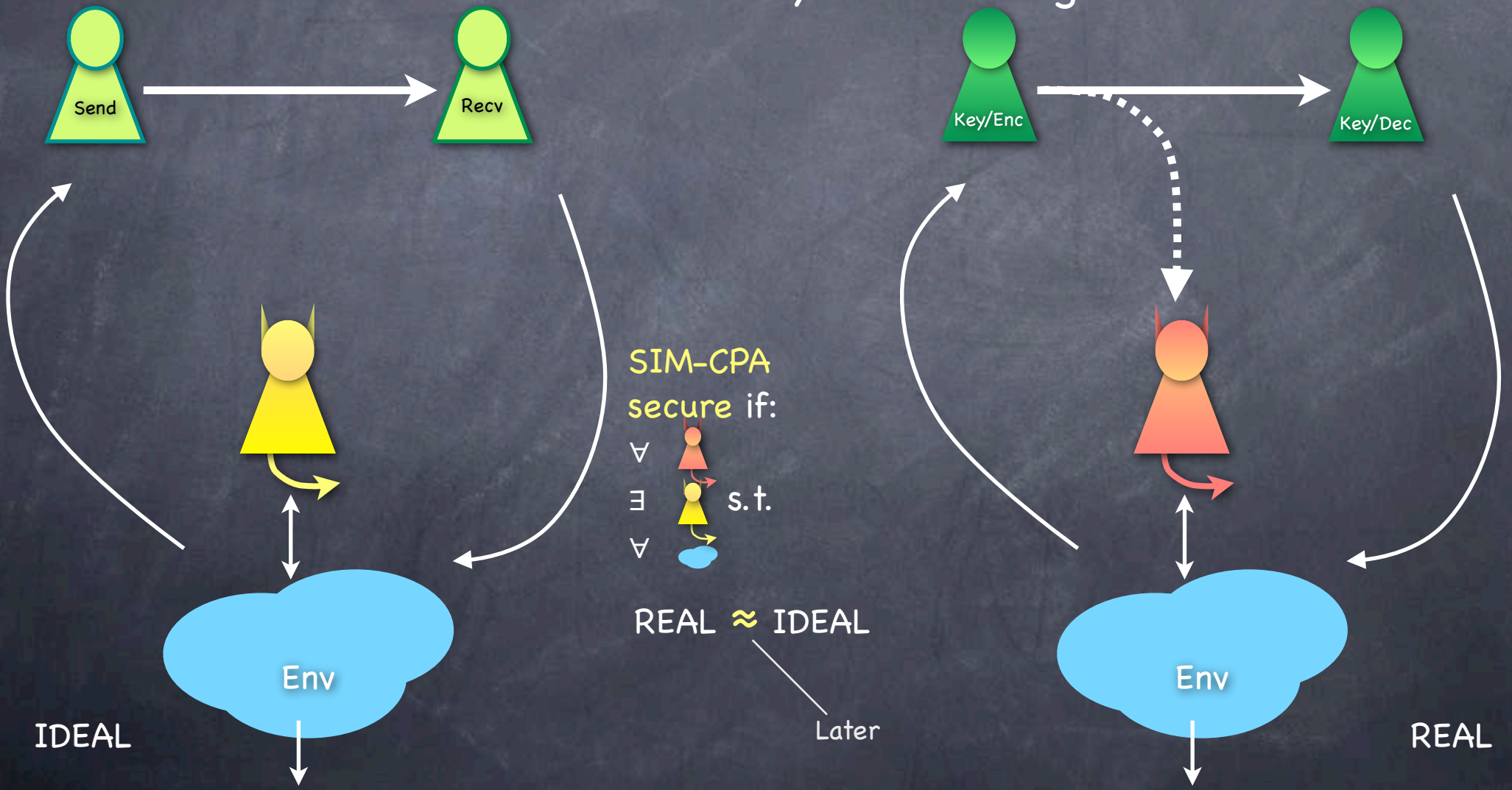
- Same as SIM-onetime security, but not restricted to environments which send only one message



Symmetric-Key Encryption

SIM-CPA Security

- Same as SIM-onetime security, but not restricted to environments which send only one message



Symmetric-Key Encryption

IND-CPA Security

Symmetric-Key Encryption

IND-CPA Security

- Experiment picks a random bit b. It also runs KeyGen to get a key K

b←{0,1}

Symmetric-Key Encryption

IND-CPA Security

- Experiment picks a random bit b. It also runs KeyGen to get a key K
  - For as long as Adversary wants

b←{0,1}

Symmetric-Key Encryption

IND-CPA Security

- Experiment picks a random bit b. It also runs KeyGen to get a key K

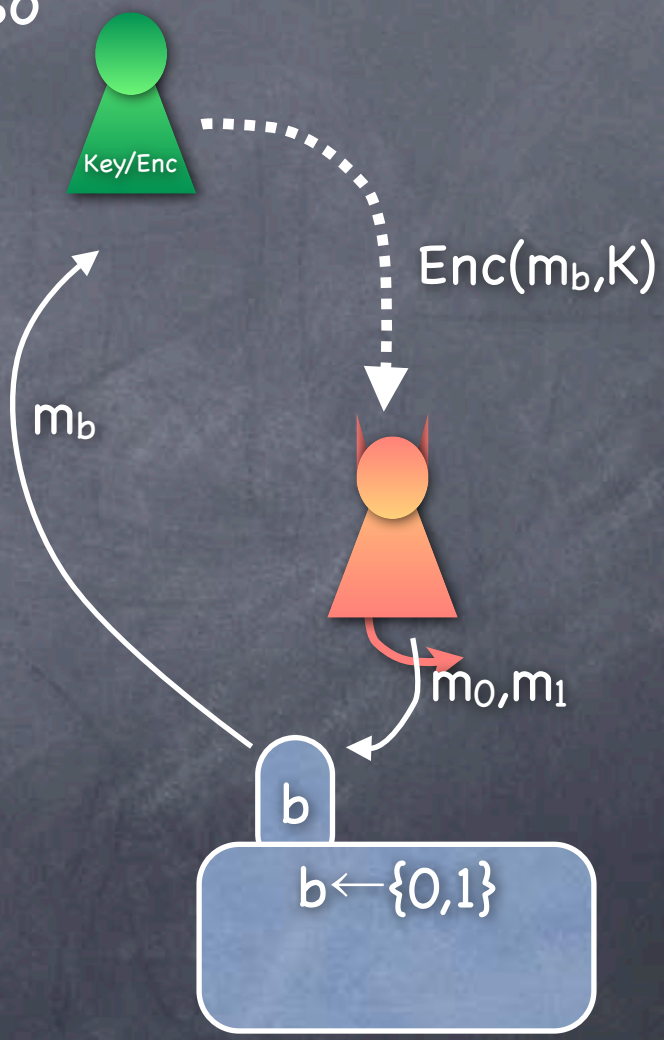
  - For as long as Adversary wants
    - Adv sends two messages m_0, m_1 to the experiment

$b \leftarrow \{0,1\}$

Symmetric-Key Encryption

IND-CPA Security

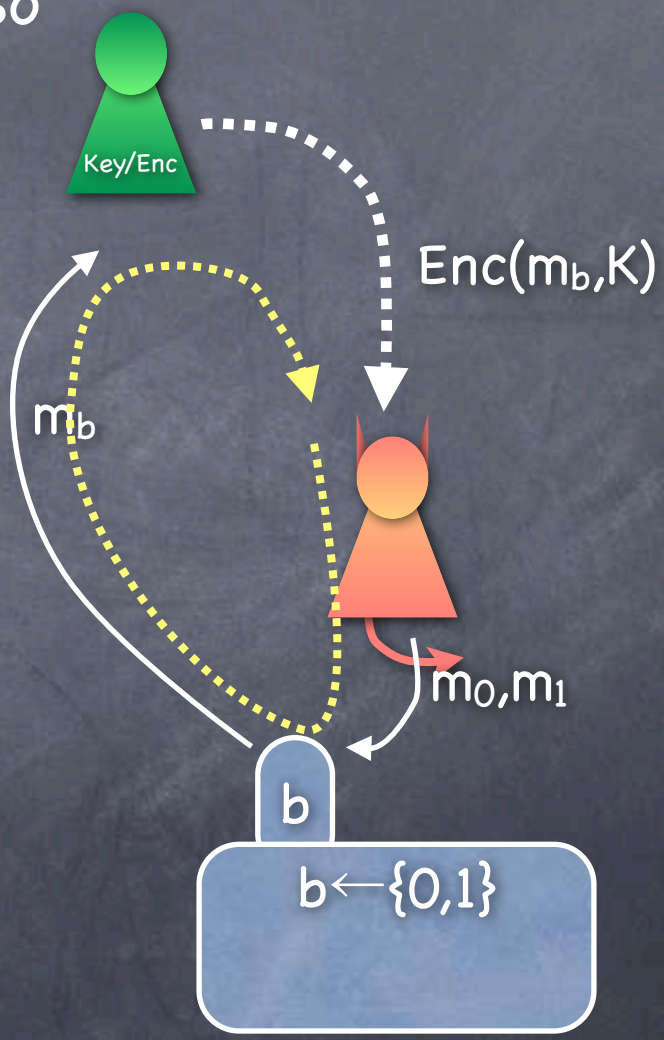
- Experiment picks a random bit b. It also runs KeyGen to get a key K
  - For as long as Adversary wants
    - Adv sends two messages m_0, m_1 to the experiment
    - Expt returns Enc(m_b,K) to the adversary



Symmetric-Key Encryption

IND-CPA Security

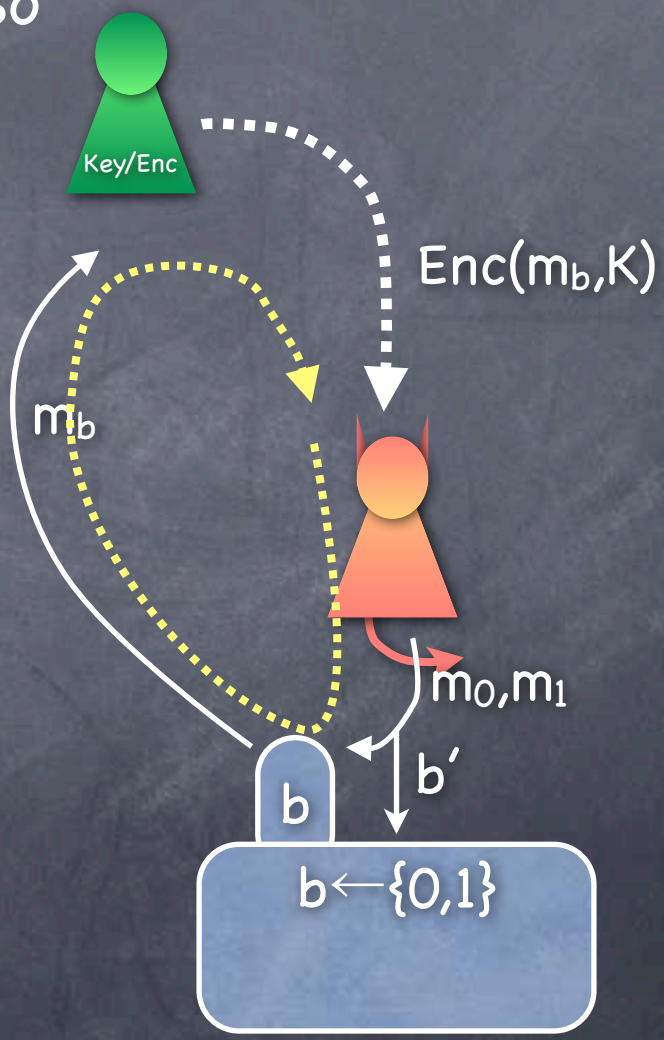
- Experiment picks a random bit b. It also runs KeyGen to get a key K
- For as long as Adversary wants
  - Adv sends two messages m_0, m_1 to the experiment
  - Expt returns Enc(m_b,K) to the adversary



Symmetric-Key Encryption

IND-CPA Security

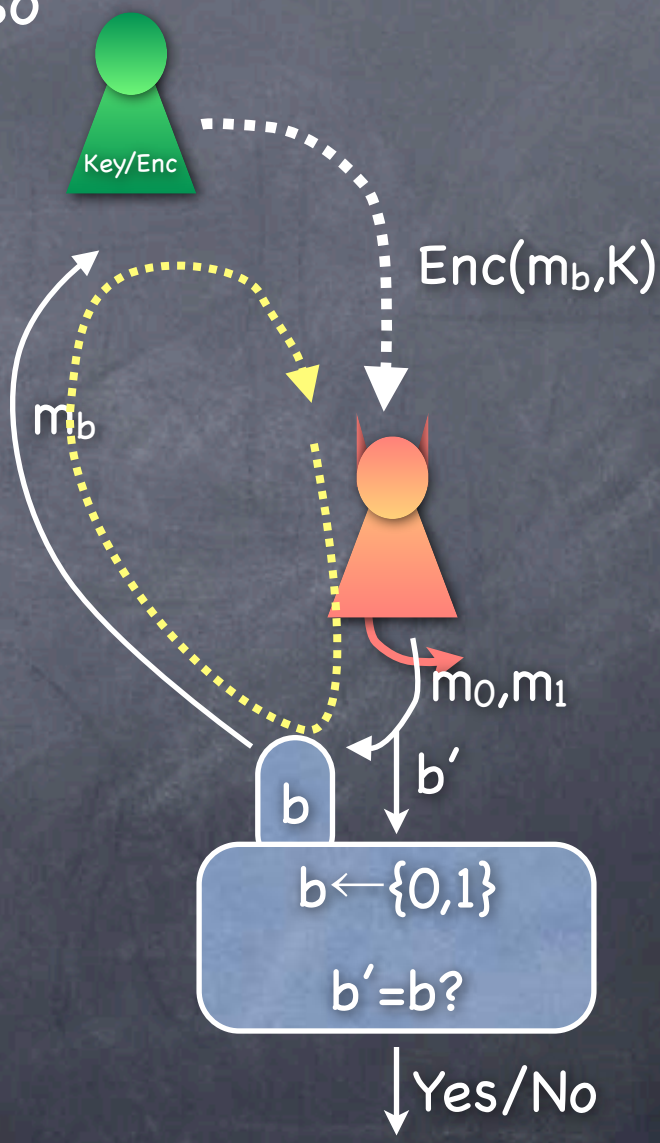
- Experiment picks a random bit b. It also runs KeyGen to get a key K
- For as long as Adversary wants
  - Adv sends two messages m_0, m_1 to the experiment
  - Expt returns Enc(m_b,K) to the adversary
- Adversary returns a guess b′



Symmetric-Key Encryption

IND-CPA Security

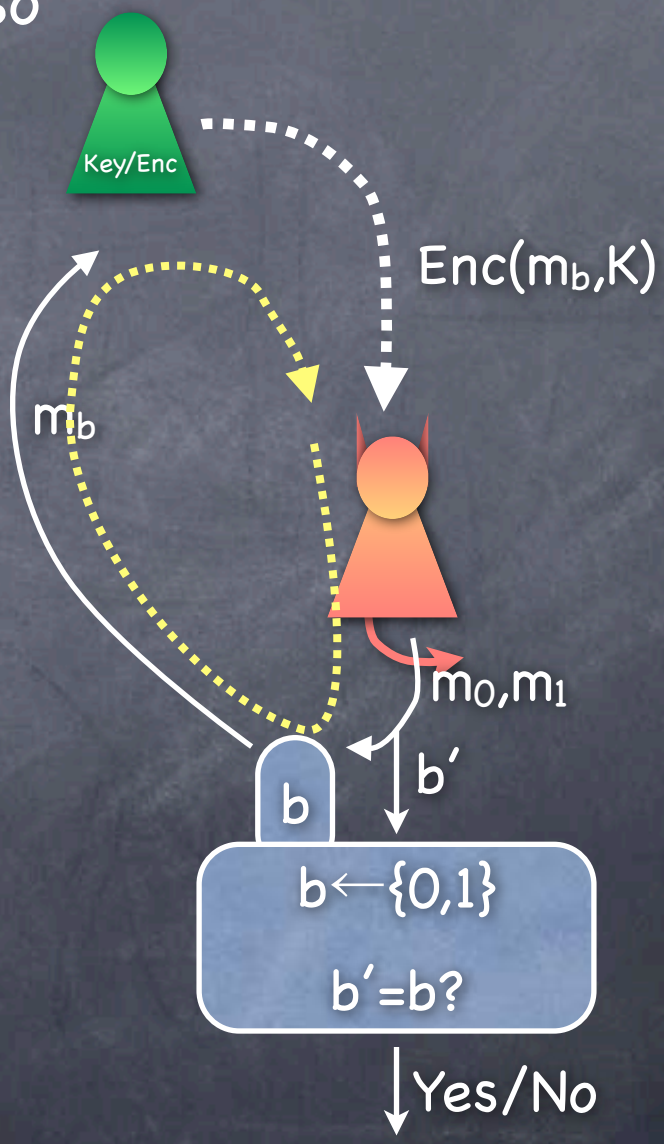
- Experiment picks a random bit b. It also runs KeyGen to get a key K
  - For as long as Adversary wants
    - Adv sends two messages m_0, m_1 to the experiment
    - Expt returns $Enc(m_b,K)$ to the adversary
- Adversary returns a guess b′
- Experiment outputs 1 iff b′=b



Symmetric-Key Encryption

IND-CPA Security

- Experiment picks a random bit b. It also runs KeyGen to get a key K
  - For as long as Adversary wants
    - Adv sends two messages m_0, m_1 to the experiment
    - Expt returns $Enc(m_b,K)$ to the adversary
  - Adversary returns a guess b′
  - Experiment outputs 1 iff b′=b
- IND-CPA secure if for all "feasible" adversaries $Pr[b'=b] \approx 1/2$

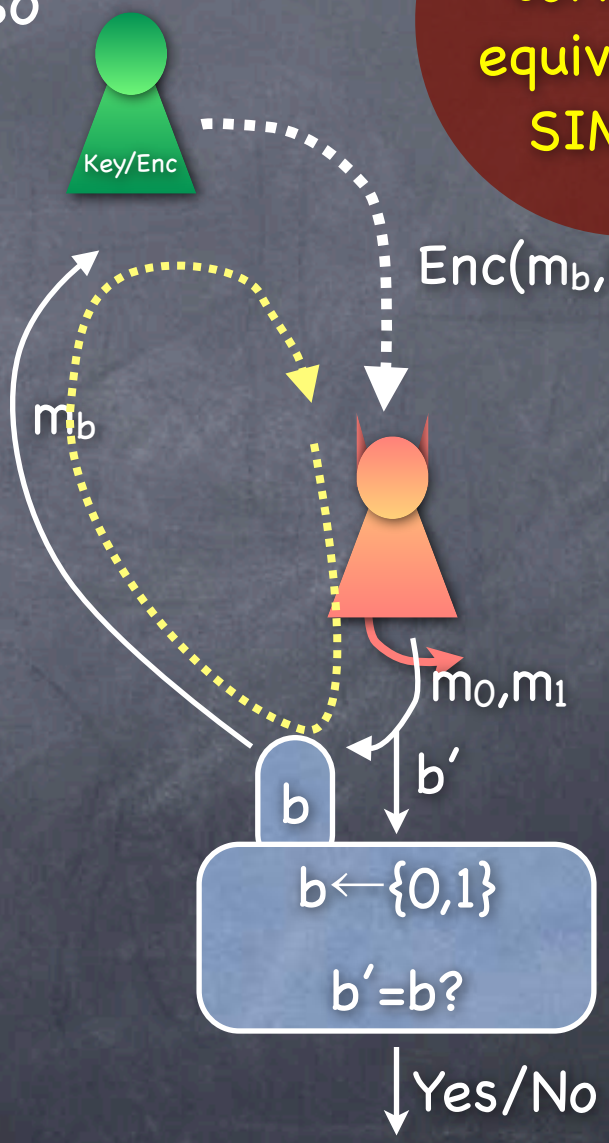


Symmetric-Key Encryption

IND-CPA Security

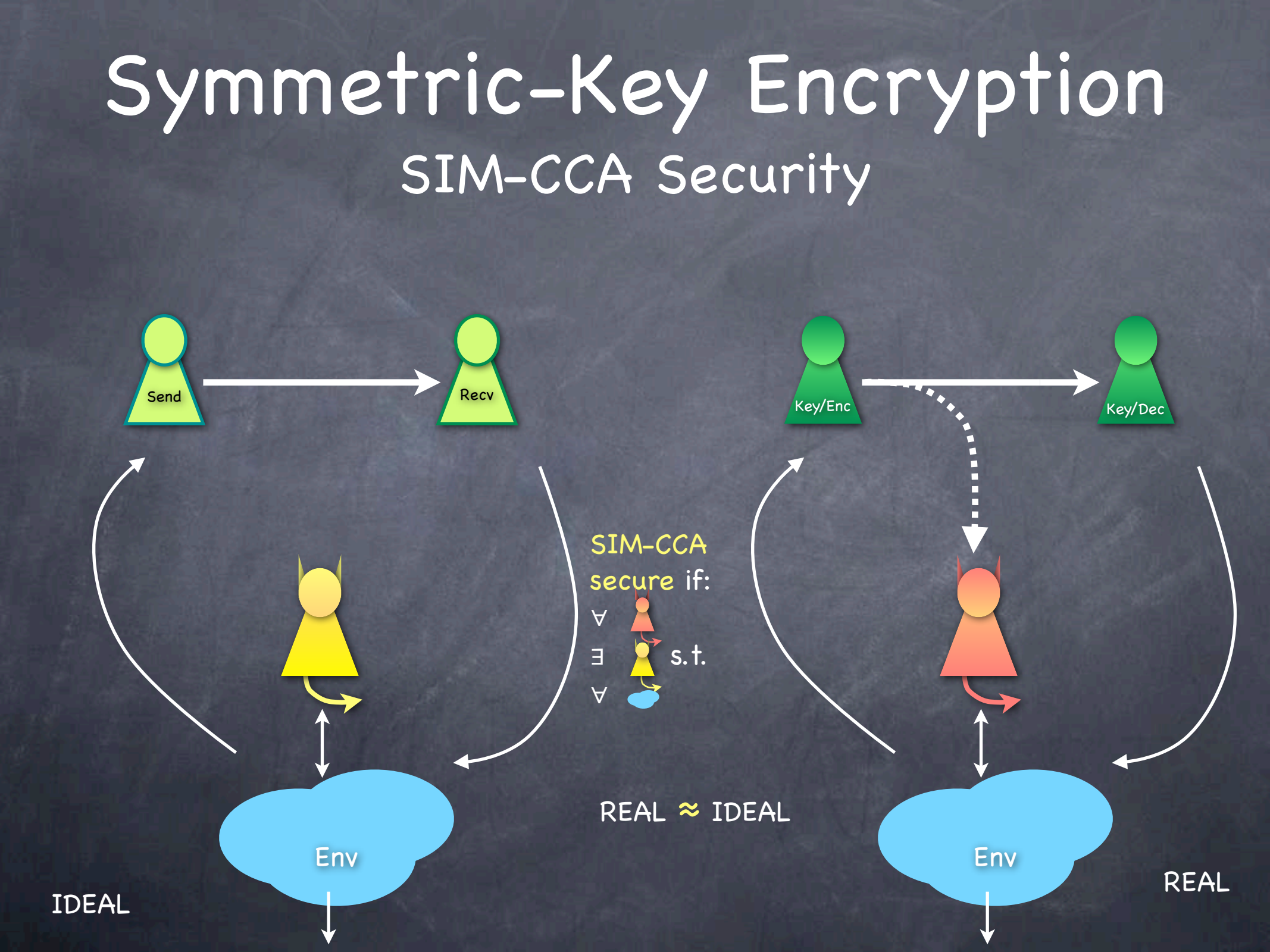
- Experiment picks a random bit b. It also runs KeyGen to get a key K
  - For as long as Adversary wants
    - Adv sends two messages m_0, m_1 to the experiment
    - Expt returns $Enc(m_b,K)$ to the adversary
  - Adversary returns a guess b′
  - Experiment outputs 1 iff b′=b
- IND-CPA secure if for all "feasible" adversaries $Pr[b'=b] \approx 1/2$

IND-CPA + ~correctness equivalent to SIM-CPA



Symmetric-Key Encryption

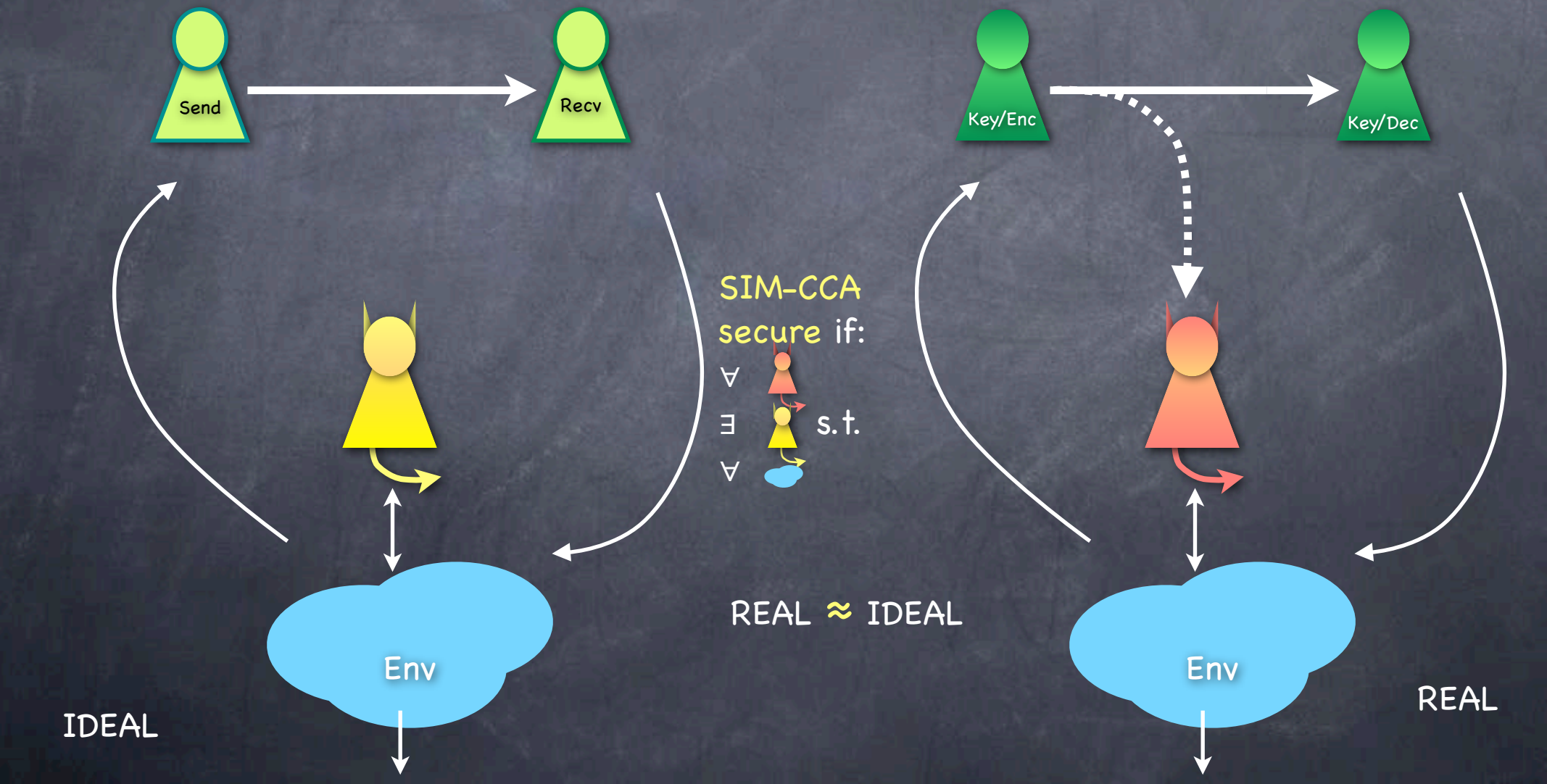
SIM-CCA Security



Symmetric-Key Encryption

SIM-CCA Security

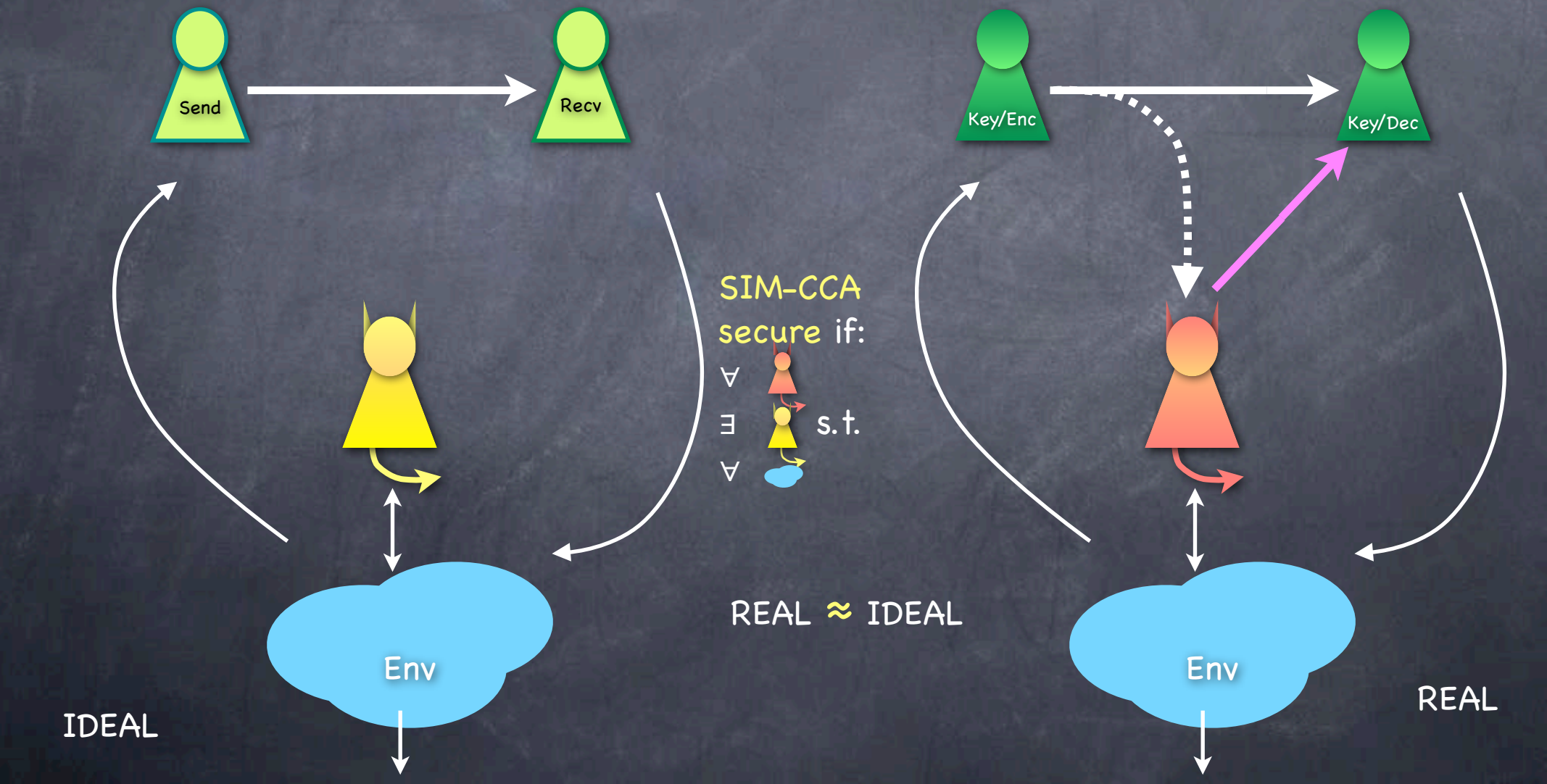
- An active adversary can inject its own ciphertexts into the channel and get them "decrypted"



Symmetric-Key Encryption

SIM-CCA Security

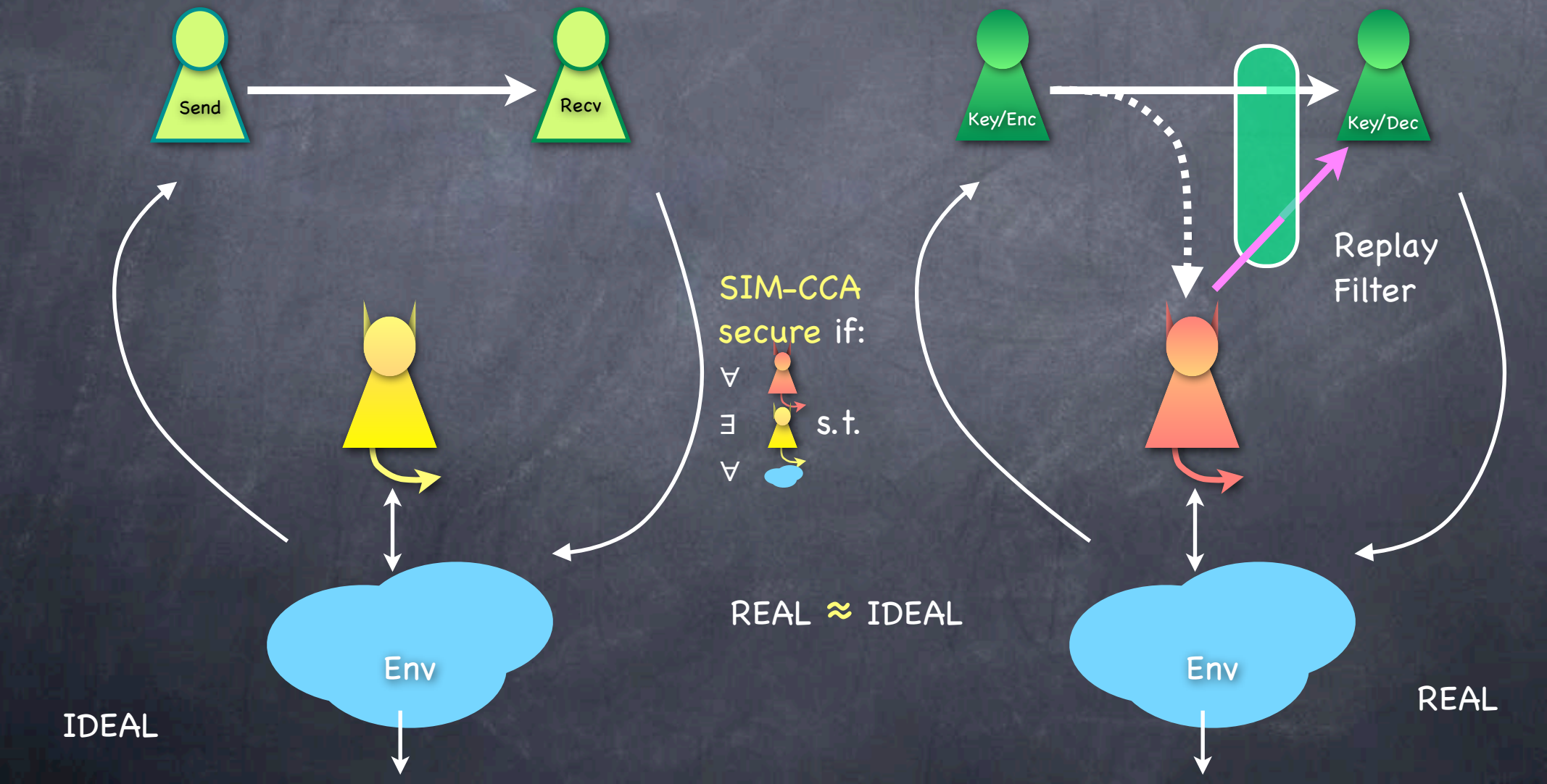
- An active adversary can inject its own ciphertexts into the channel and get them "decrypted"



Symmetric-Key Encryption

SIM-CCA Security

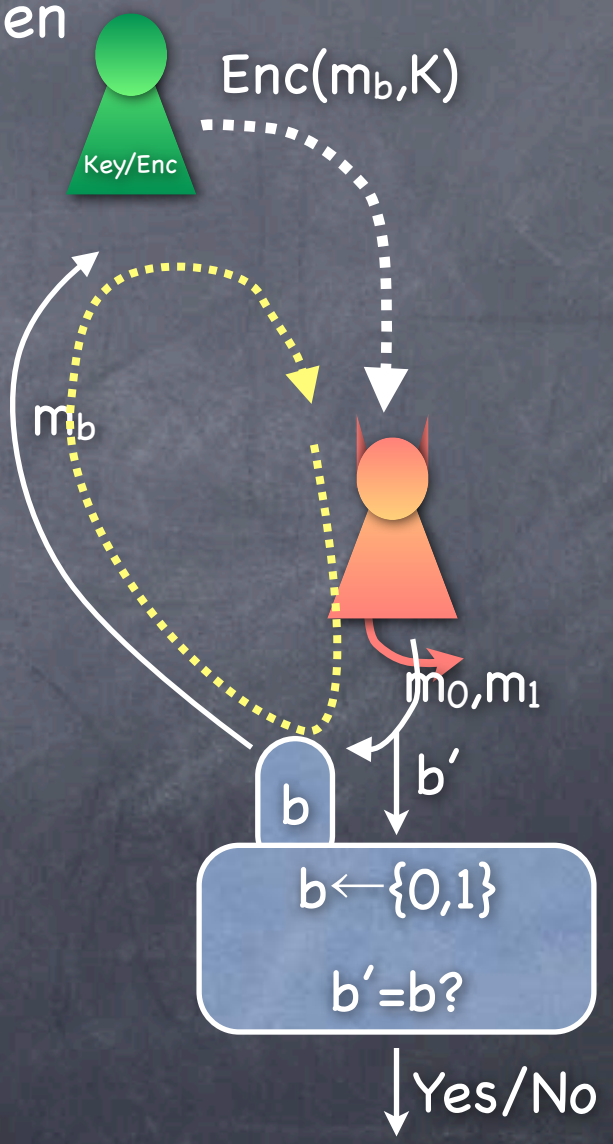
- An active adversary can inject its own ciphertexts into the channel and get them "decrypted"



Symmetric-Key Encryption

IND-CCA Security

- Experiment picks $b \leftarrow \{0,1\}$ and $K \leftarrow KeyGen$
  - For as long as Adversary wants
    - Adv sends two messages m_0, m_1 to the experiment
    - Expt returns $Enc(m_b,K)$ to the adversary
- Adversary returns a guess b'
- Experiments outputs 1 iff $b'=b$
- IND-CCA secure if for all feasible adversaries $Pr[b'=b] \approx 1/2$



Symmetric-Key Encryption

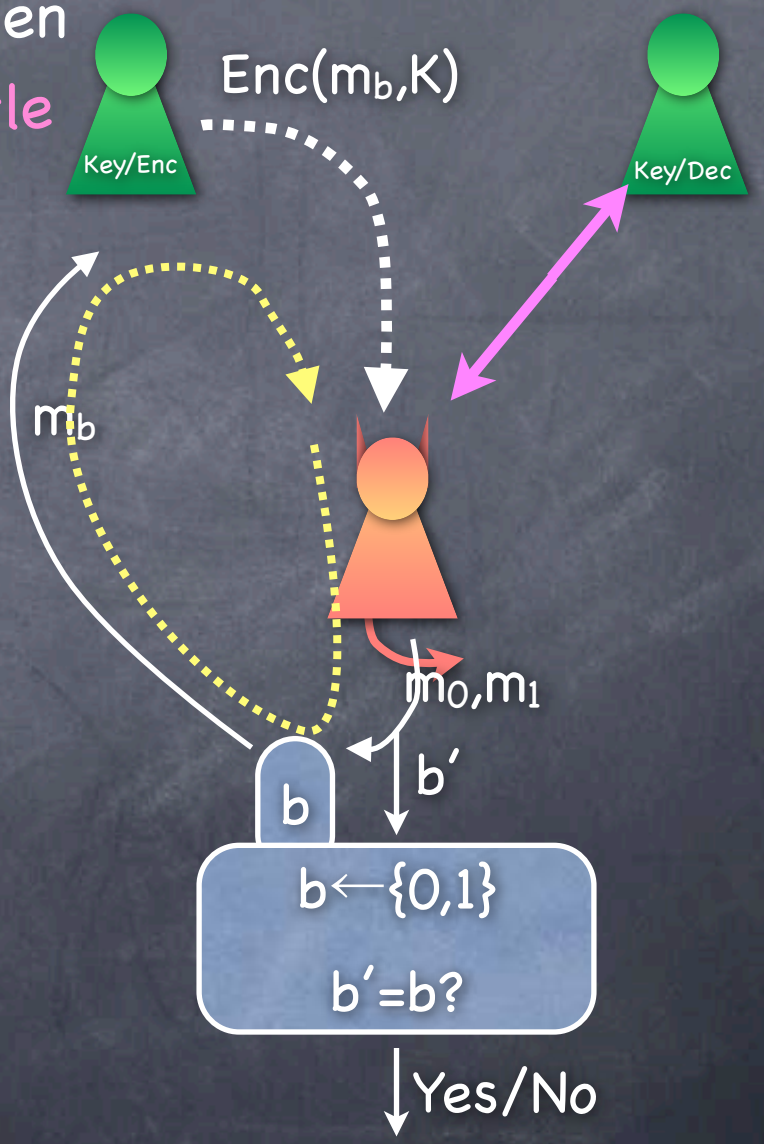
IND-CCA Security

- Experiment picks $b \leftarrow \{0,1\}$ and $K \leftarrow KeyGen$

  Adv gets (guarded) access to Dec_K oracle

  - For as long as Adversary wants
    - Adv sends two messages m_0, m_1 to the experiment
    - Expt returns $Enc(m_b,K)$ to the adversary

- Adversary returns a guess b'
- Experiments outputs 1 iff $b'=b$
- IND-CCA secure if for all feasible adversaries $Pr[b'=b] \approx 1/2$



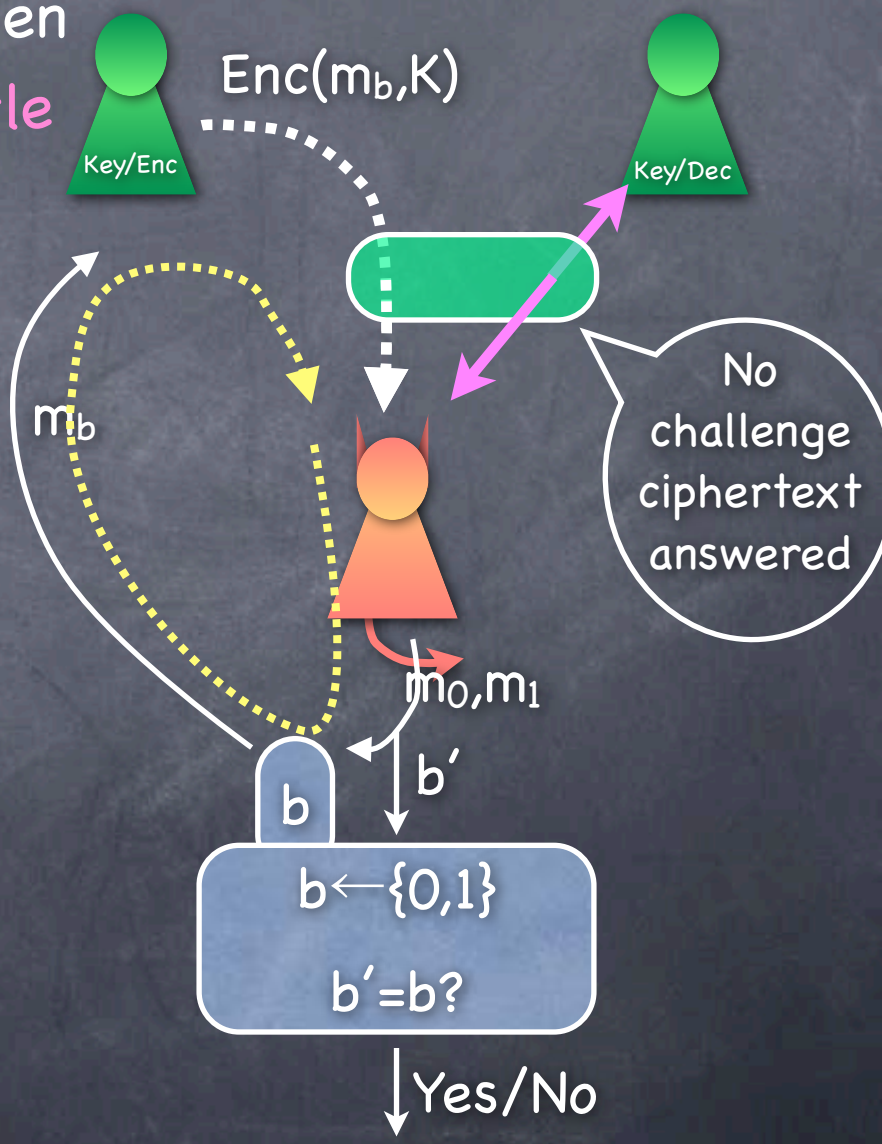
Symmetric-Key Encryption

IND-CCA Security

- Experiment picks $b \leftarrow \{0,1\}$ and $K \leftarrow KeyGen$

  Adv gets (guarded) access to Dec_K oracle

  - For as long as Adversary wants
    - Adv sends two messages m_0, m_1 to the experiment
    - Expt returns $Enc(m_b,K)$ to the adversary
- Adversary returns a guess b'
- Experiments outputs 1 iff $b'=b$
- IND-CCA secure if for all feasible adversaries $Pr[b'=b] \approx 1/2$



Symmetric-Key Encryption

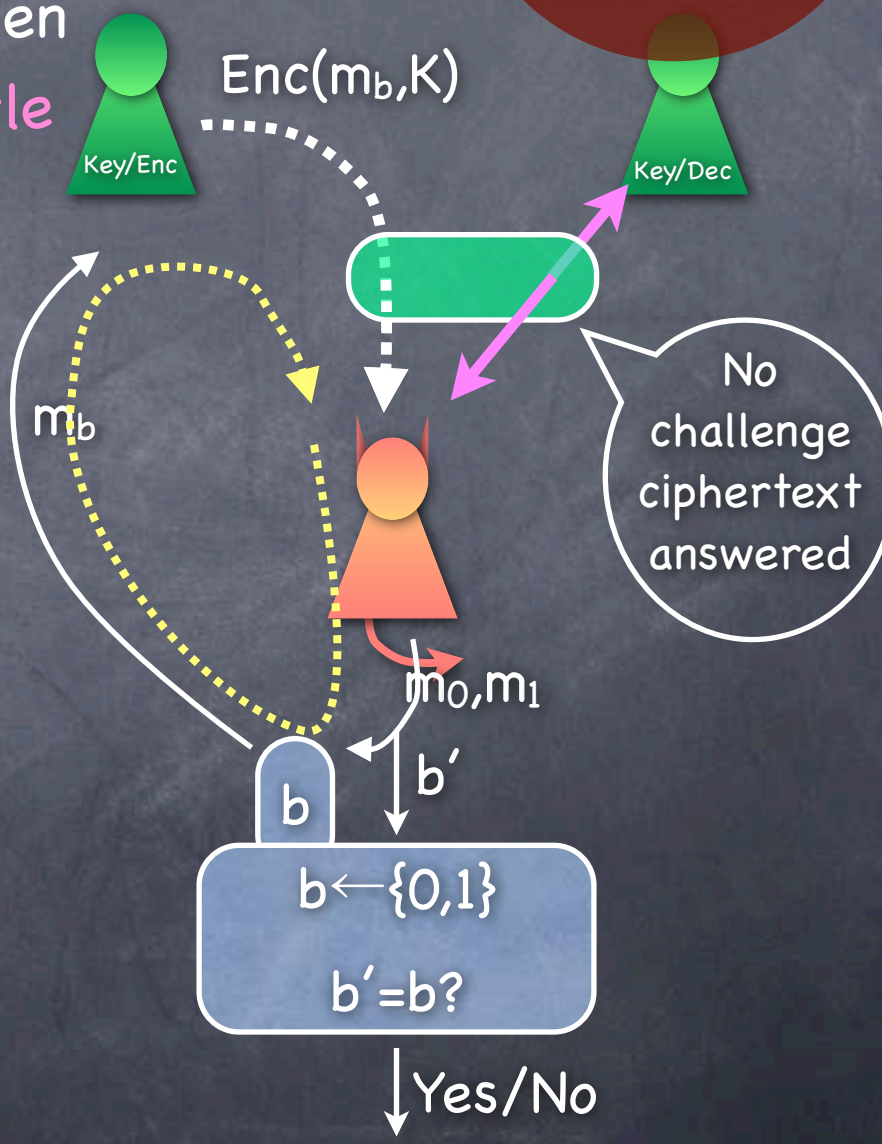
IND-CCA Security

IND-CCA + ~correctness equivalent to SIM-CCA

- Experiment picks $b \leftarrow \{0,1\}$ and $K \leftarrow KeyGen$

  Adv gets (guarded) access to Dec_K oracle

  - For as long as Adversary wants
    - Adv sends two messages m_0, m_1 to the experiment
    - Expt returns $Enc(m_b,K)$ to the adversary
- Adversary returns a guess b'
- Experiments outputs 1 iff $b'=b$
- IND-CCA secure if for all feasible adversaries $Pr[b'=b] \approx 1/2$



Definitions Summary

Definitions Summary

- Security definitions:

Definitions Summary

- Security definitions:
  - SIM-Onetime = IND-Onetime/Perfect Secrecy + correctness

Definitions Summary

- Security definitions:
  - SIM-Onetime = IND-Onetime/Perfect Secrecy + correctness
  - SIM-CPA = IND-CPA + ~correctness: allows using the same key for multiple messages

Definitions Summary

- Security definitions:
  - SIM-Onetime = IND-Onetime/Perfect Secrecy + correctness
  - SIM-CPA = IND-CPA + ~correctness: allows using the same key for multiple messages
  - SIM-CCA = IND-CCA + ~correctness: allows active attacks

Definitions Summary

- Security definitions:
  - SIM-Onetime = IND-Onetime/Perfect Secrecy + correctness
  - SIM-CPA = IND-CPA + ~correctness: allows using the same key for multiple messages
  - SIM-CCA = IND-CCA + ~correctness: allows active attacks
- Next

Definitions Summary

- Security definitions:
  - SIM-Onetime = IND-Onetime/Perfect Secrecy + correctness
  - SIM-CPA = IND-CPA + ~correctness: allows using the same key for multiple messages
  - SIM-CCA = IND-CCA + ~correctness: allows active attacks
- Next
  - For multi-message schemes we relaxed the "perfect" simulation requirement

Definitions Summary

- Security definitions:
  - SIM-Onetime = IND-Onetime/Perfect Secrecy + correctness
  - SIM-CPA = IND-CPA + ~correctness: allows using the same key for multiple messages
  - SIM-CCA = IND-CCA + ~correctness: allows active attacks
- Next
  - For multi-message schemes we relaxed the "perfect" simulation requirement
  - But what is $\approx$?

Feasible Computation

Feasible Computation

- In analyzing complexity of algorithms: Rate at which computational complexity grows with input size
  - e.g. Can do sorting in O(n log n)

Feasible Computation

- In analyzing complexity of algorithms: Rate at which computational complexity grows with input size
  - e.g. Can do sorting in O(n log n)
- Only the rough rate considered
  - Exact time depends on the technology

Feasible Computation

- In analyzing complexity of algorithms: Rate at which computational complexity grows with input size
  - e.g. Can do sorting in O(n log n)
- Only the rough rate considered
  - Exact time depends on the technology
  - How much more computation will be needed as the instances of the problem get larger. (Do we scale well?)

Feasible Computation

- In analyzing complexity of algorithms: Rate at which computational complexity grows with input size
  - e.g. Can do sorting in O(n log n)
- Only the rough rate considered
  - Exact time depends on the technology
  - How much more computation will be needed as the instances of the problem get larger. (Do we scale well?)

Feasible Computation

- In analyzing complexity of algorithms: Rate at which computational complexity grows with input size
  - e.g. Can do sorting in O(n log n)
- Only the rough rate considered
  - Exact time depends on the technology
  - How much more computation will be needed as the instances of the problem get larger. (Do we scale well?)
  - "Polynomial time" ($O(n)$, $O(n^2)$, $O(n^3)$, ...) considered feasible

Infeasible Computation

Infeasible Computation

- "Super-Polynomial time" considered infeasible

Infeasible Computation

- "Super-Polynomial time" considered infeasible
  - e.g. 2^n, $2^{\sqrt{n}}$, $n^{\log(n)}$

Infeasible Computation

- "Super-Polynomial time" considered infeasible
  - e.g. 2^n, $2^{\sqrt{n}}$, $n^{\log(n)}$
  - i.e., as n grows, quickly becomes "infeasibly large"

Infeasible Computation

- "Super-Polynomial time" considered infeasible
  - e.g. 2^n, $2^{\sqrt{n}}$, $n^{\log(n)}$
  - i.e., as n grows, quickly becomes "infeasibly large"
- Can we make breaking security infeasible for Eve?

Infeasible Computation

- "Super-Polynomial time" considered infeasible
  - e.g. 2^n, $2^{\sqrt{n}}$, $n^{\log(n)}$
  - i.e., as n grows, quickly becomes "infeasibly large"
- Can we make breaking security infeasible for Eve?
  - What is n (that can grow)?

Infeasible Computation

- "Super-Polynomial time" considered infeasible
  - e.g. 2^n, $2^{\sqrt{n}}$, $n^{\log(n)}$
  - i.e., as n grows, quickly becomes "infeasibly large"
- Can we make breaking security infeasible for Eve?
  - What is n (that can grow)?
  - Message size?

Infeasible Computation

- "Super-Polynomial time" considered infeasible
  - e.g. 2^n, $2^{\sqrt{n}}$, $n^{\log(n)}$
  - i.e., as n grows, quickly becomes "infeasibly large"
- Can we make breaking security infeasible for Eve?
  - What is n (that can grow)?
  - Message size?
    - We need security even if sending only one bit!

Security Parameter

Security Parameter

- A parameter that is part of the encryption scheme

Security Parameter

- A parameter that is part of the encryption scheme
  - Not related to message size

Security Parameter

- A parameter that is part of the encryption scheme
  - Not related to message size
  - A knob that can be used to set the security level

Security Parameter

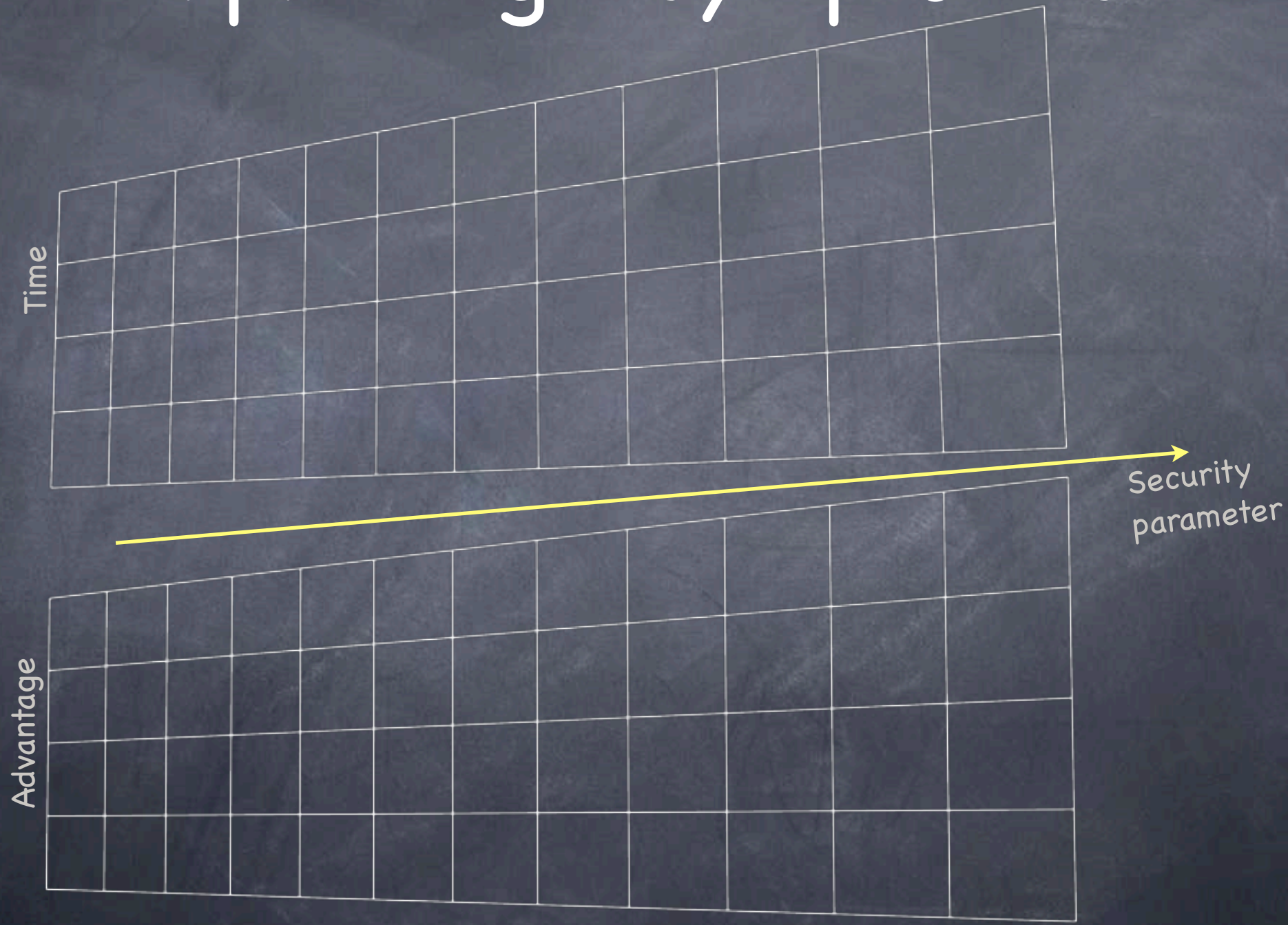
- A parameter that is part of the encryption scheme
  - Not related to message size
  - A knob that can be used to set the security level
  - Will denote by k

Security Parameter

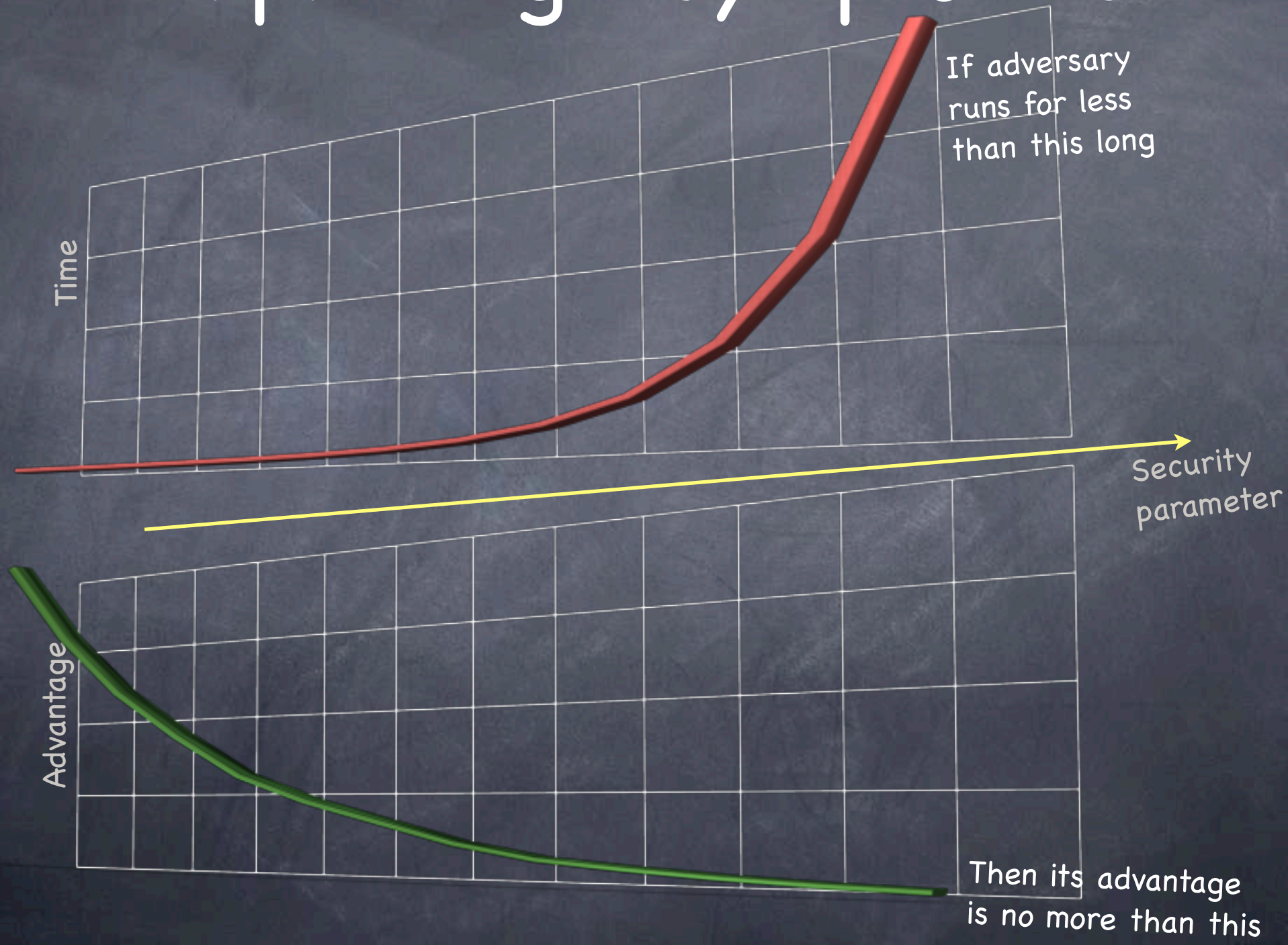
- A parameter that is part of the encryption scheme
  - Not related to message size
  - A knob that can be used to set the security level
  - Will denote by k
- Security guarantees are given asymptotically as a function of the security parameter

Interpreting Asymptotics

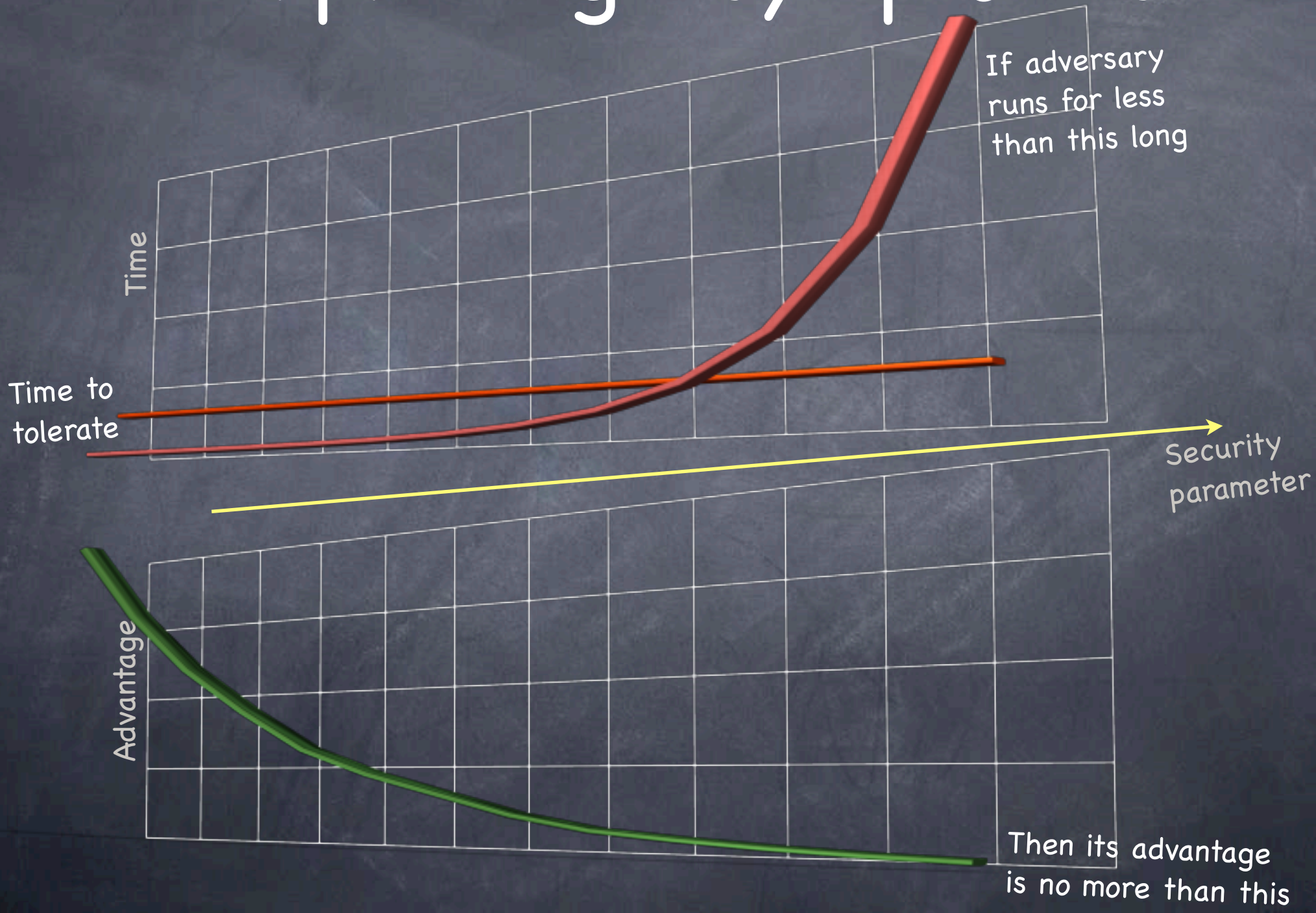
Interpreting Asymptotics



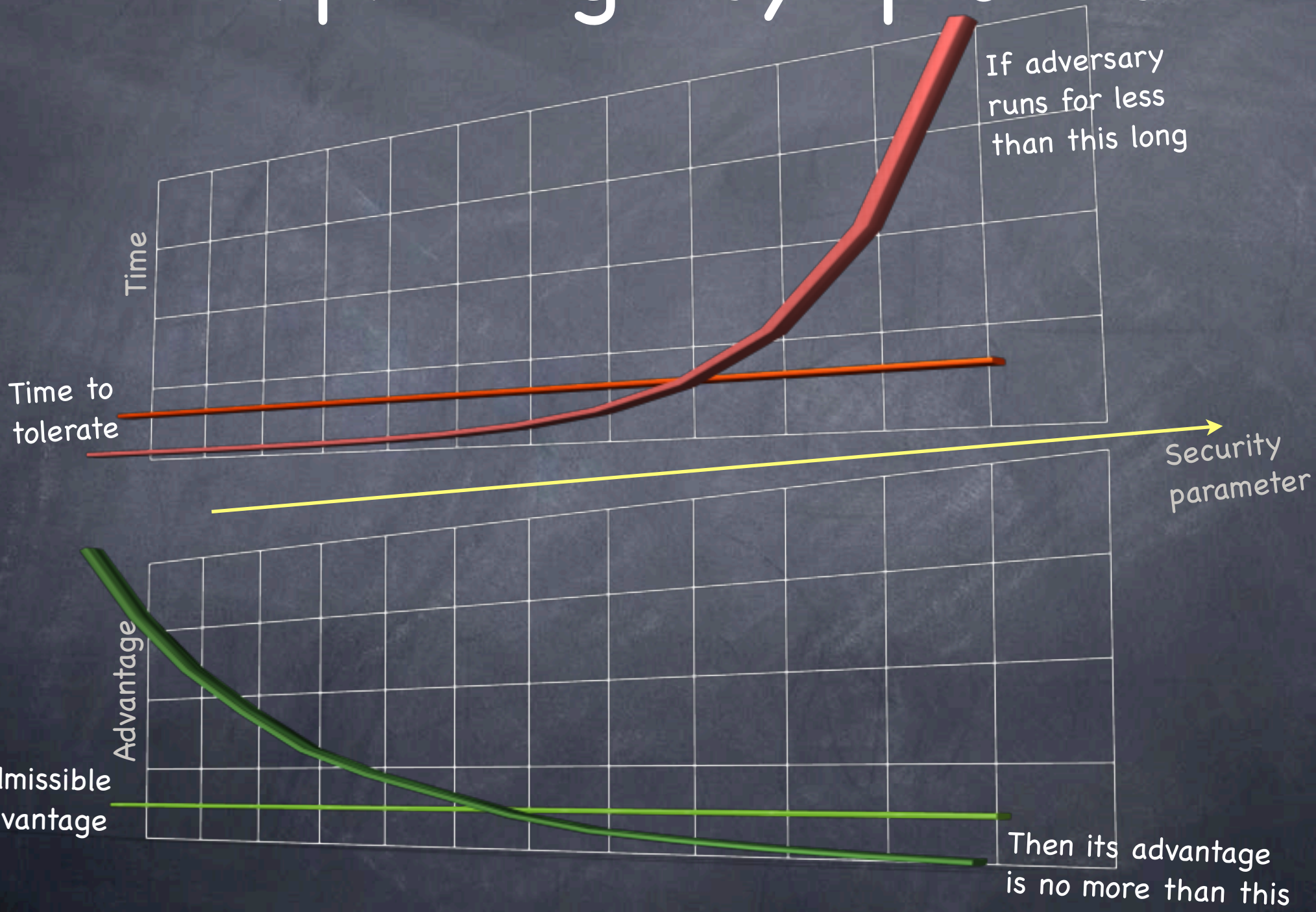
Interpreting Asymptotics



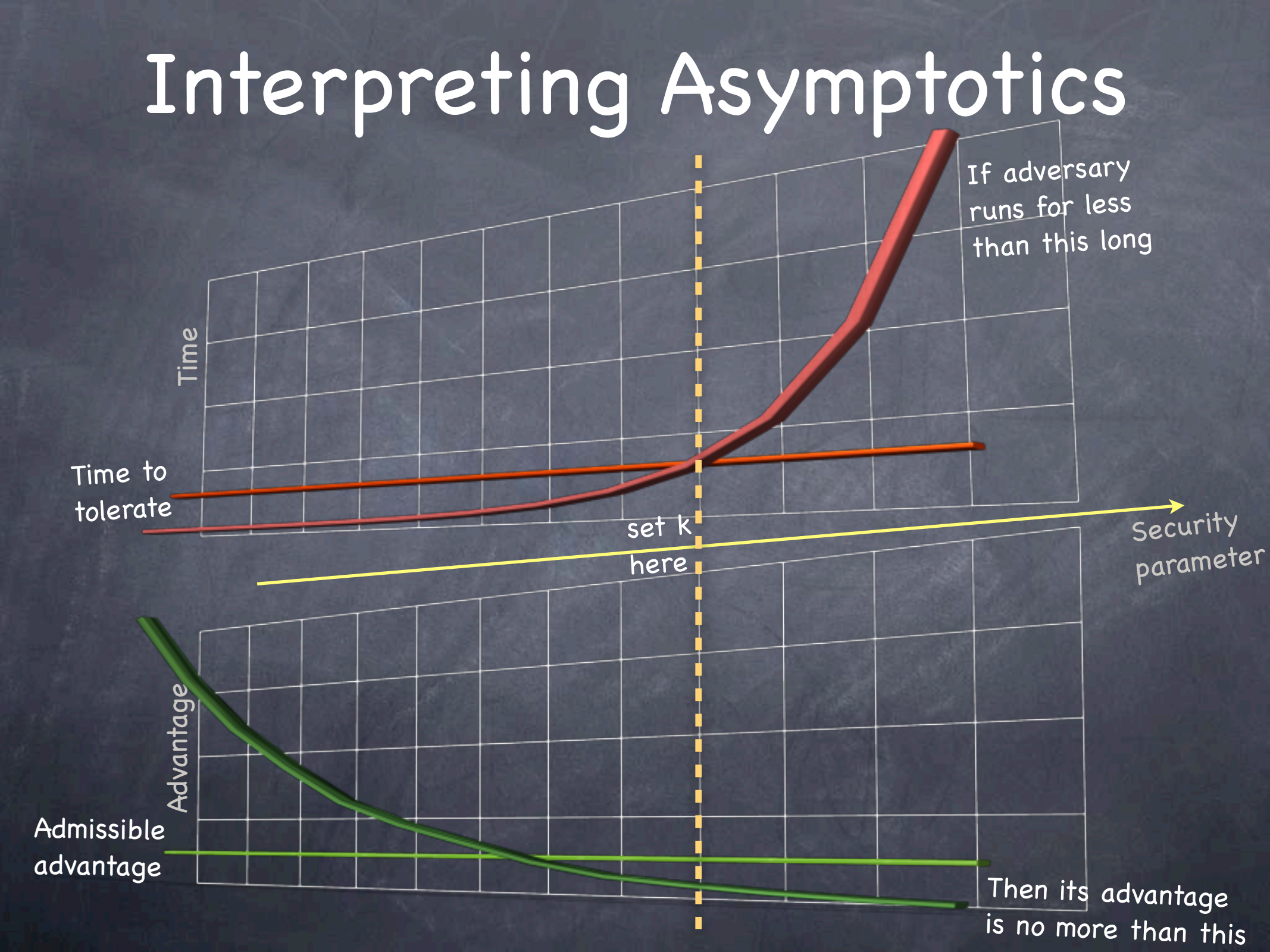
Interpreting Asymptotics



Interpreting Asymptotics



Interpreting Asymptotics



Feasible and Negligible

Feasible and Negligible

- We want to tolerate Eves who have a running time bounded by some polynomial in k

Feasible and Negligible

- We want to tolerate Eves who have a running time bounded by some polynomial in k
  - Eve could toss coins: Probabilistic Polynomial-Time (PPT)

Feasible and Negligible

- We want to tolerate Eves who have a running time bounded by some polynomial in k
  - Eve could toss coins: Probabilistic Polynomial-Time (PPT)
  - It is better that we allow Eve high polynomial times too (we'll typically allow Eve some super-polynomial time)

Feasible and Negligible

- We want to tolerate Eves who have a running time bounded by some polynomial in k
  - Eve could toss coins: Probabilistic Polynomial-Time (PPT)
  - It is better that we allow Eve high polynomial times too (we'll typically allow Eve some super-polynomial time)
    - But algorithms for Alice/Bob better be very efficient

Feasible and Negligible

- We want to tolerate Eves who have a running time bounded by some polynomial in k
  - Eve could toss coins: Probabilistic Polynomial-Time (PPT)
  - It is better that we allow Eve high polynomial times too (we'll typically allow Eve some super-polynomial time)
    - But algorithms for Alice/Bob better be very efficient
  - Eve could be non-uniform: a different strategy for each k

Feasible and Negligible

- We want to tolerate Eves who have a running time bounded by some polynomial in k
  - Eve could toss coins: Probabilistic Polynomial-Time (PPT)
  - It is better that we allow Eve high polynomial times too (we'll typically allow Eve some super-polynomial time)
    - But algorithms for Alice/Bob better be very efficient
  - Eve could be non-uniform: a different strategy for each k
- Such an Eve should have only a "negligible" advantage (or, should cause at most a "negligible" difference in the behavior of the environment in the SIM definition)

Feasible and Negligible

- We want to tolerate Eves who have a running time bounded by some polynomial in k
  - Eve could toss coins: Probabilistic Polynomial-Time (PPT)
  - It is better that we allow Eve high polynomial times too (we'll typically allow Eve some super-polynomial time)
    - But algorithms for Alice/Bob better be very efficient
  - Eve could be non-uniform: a different strategy for each k
- Such an Eve should have only a "negligible" advantage (or, should cause at most a "negligible" difference in the behavior of the environment in the SIM definition)
  - What is negligible?

Negligibly Small

Negligibly Small

- A negligible quantity: As we turn the knob the quantity should "decrease extremely fast"

Negligibly Small

- A negligible quantity: As we turn the knob the quantity should "decrease extremely fast"
  - Negligible: decreases as 1/superpoly(k)

Negligibly Small

- A negligible quantity: As we turn the knob the quantity should "decrease extremely fast"
  - Negligible: decreases as 1/superpoly(k)
    - i.e., faster than 1/poly(k) for every polynomial

Negligibly Small

- A negligible quantity: As we turn the knob the quantity should "decrease extremely fast"
  - Negligible: decreases as 1/superpoly(k)
    - i.e., faster than 1/poly(k) for every polynomial
    - e.g.: 2^{-k}, $2^{-\sqrt{k}}$, $k^{-(\log k)}$.

Negligibly Small

- A negligible quantity: As we turn the knob the quantity should "decrease extremely fast"
  - Negligible: decreases as 1/superpoly(k)
    - i.e., faster than 1/poly(k) for every polynomial
    - e.g.: 2^{-k}, $2^{-\sqrt{k}}$, $k^{-(\log k)}$.
    - Formally: T negligible if $\forall c>0\ \exists k_0\ \forall k>k_0\ \ T(k) < 1/k^c$

Negligibly Small

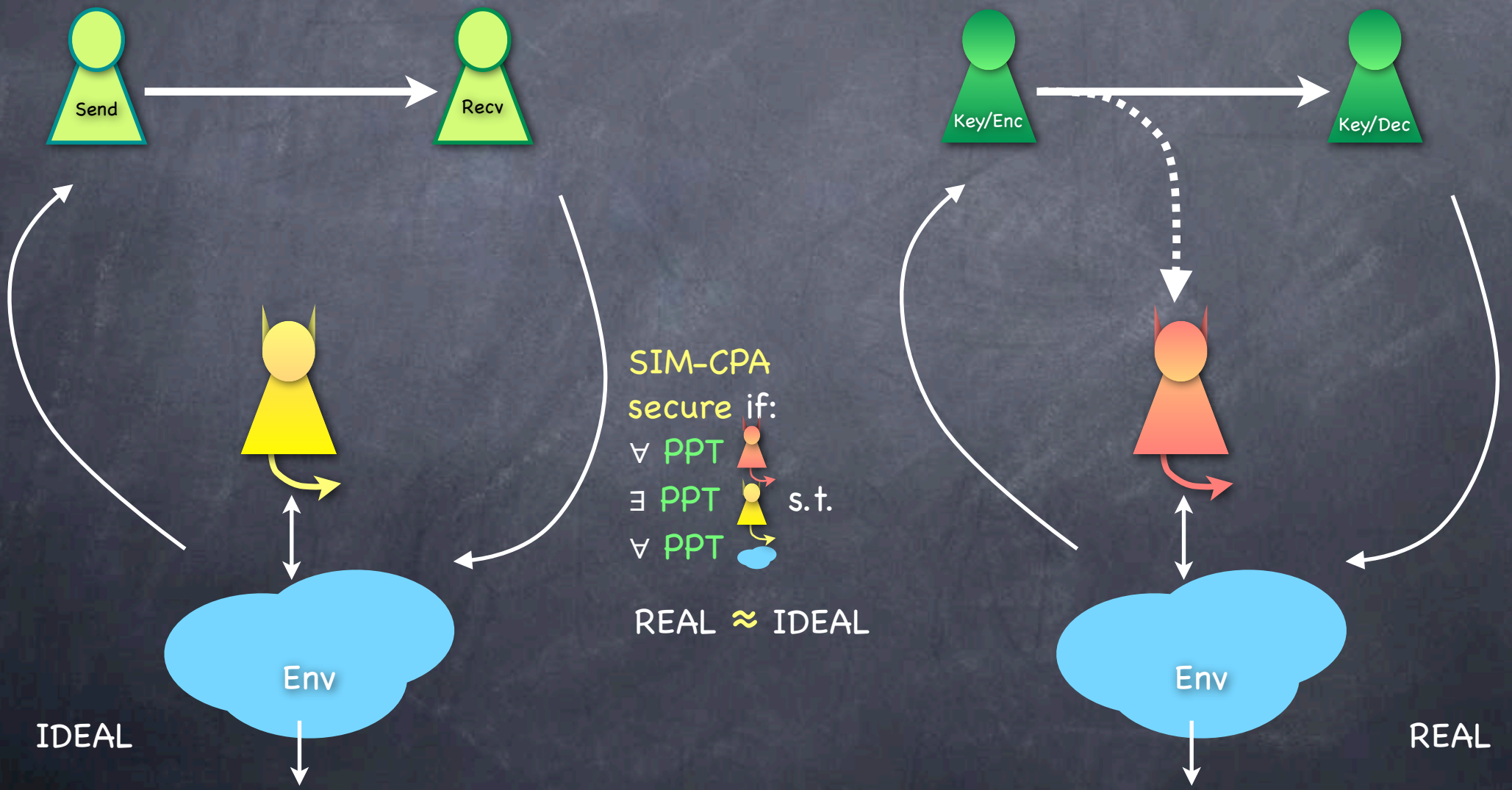
- A negligible quantity: As we turn the knob the quantity should "decrease extremely fast"
  - Negligible: decreases as 1/superpoly(k)
    - i.e., faster than 1/poly(k) for every polynomial
    - e.g.: 2^{-k}, $2^{-\sqrt{k}}$, $k^{-(\log k)}$.
    - Formally: T negligible if $\forall c>0\ \exists k_0\ \forall k>k_0\ \ T(k) < 1/k^c$
  - So that negl(k) $\times$ poly(k) = negl'(k)

Negligibly Small

- A negligible quantity: As we turn the knob the quantity should "decrease extremely fast"
  - Negligible: decreases as 1/superpoly(k)
    - i.e., faster than 1/poly(k) for every polynomial
    - e.g.: 2^{-k}, $2^{-\sqrt{k}}$, $k^{-(\log k)}$.
    - Formally: T negligible if $\forall c>0\ \exists k_0\ \forall k>k_0\ \ T(k) < 1/k^c$
  - So that negl(k) $\times$ poly(k) = negl'(k)
    - Needed, because Eve can often increase advantage polynomially by spending that much more time/by seeing that many more messages

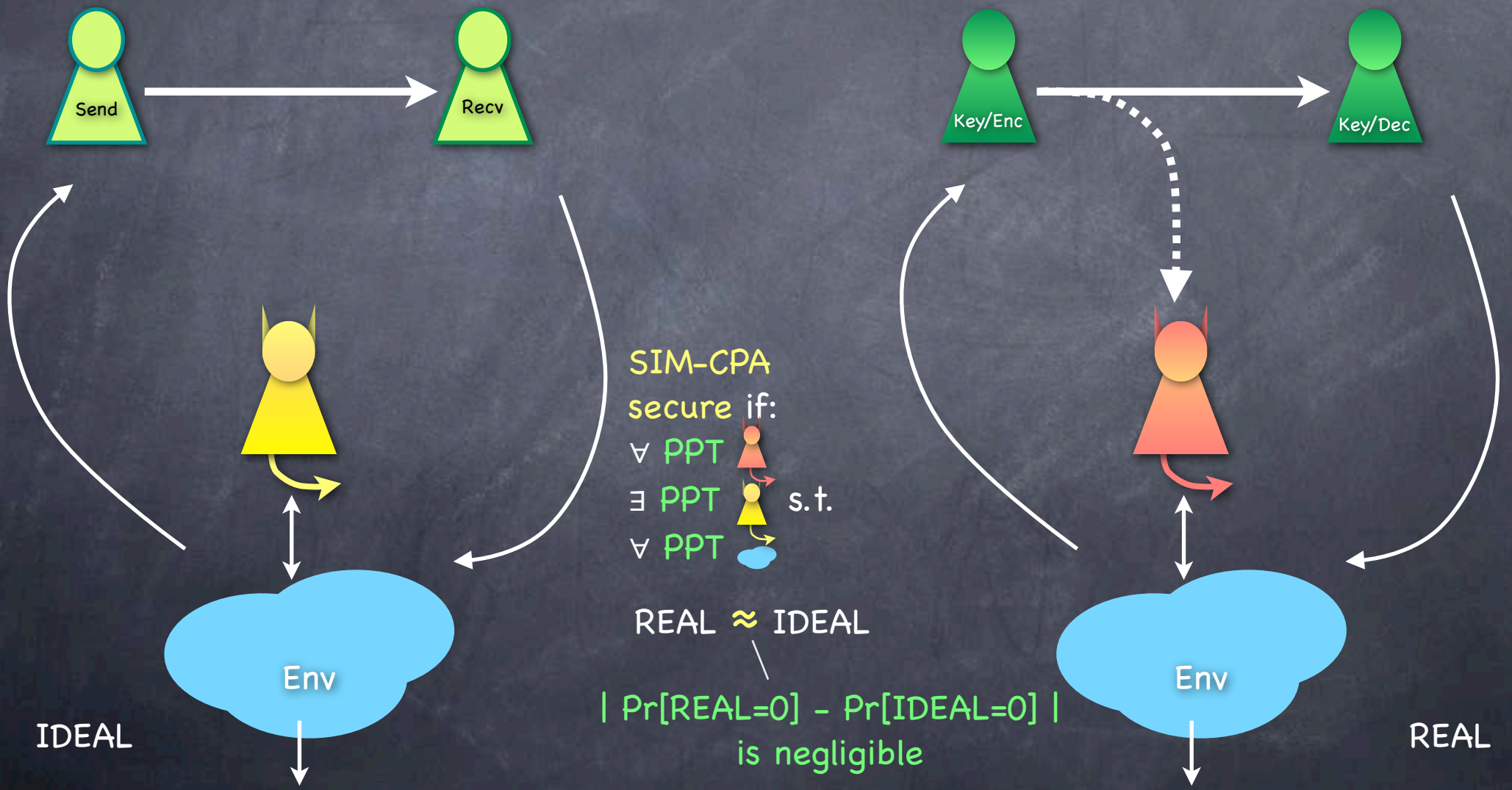
Symmetric-Key Encryption

SIM-CPA Security



Symmetric-Key Encryption

SIM-CPA Security



Constructing SKE schemes

Constructing SKE schemes

- Basic idea: extensible pseudo-random one-time pads (kept compressed in the key)

Constructing SKE schemes

- Basic idea: extensible pseudo-random one-time pads (kept compressed in the key)
- (Will also need a mechanism to ensure that the same piece of the one-time pad is not used more than once)

Constructing SKE schemes

- Basic idea: extensible pseudo-random one-time pads (kept compressed in the key)
- (Will also need a mechanism to ensure that the same piece of the one-time pad is not used more than once)
- Approach used in practice today: complex functions which are conjectured to have the requisite pseudo-randomness properties (stream-ciphers, block-ciphers)

Constructing SKE schemes

- Basic idea: extensible pseudo-random one-time pads (kept compressed in the key)
- (Will also need a mechanism to ensure that the same piece of the one-time pad is not used more than once)
- Approach used in practice today: complex functions which are conjectured to have the requisite pseudo-randomness properties (stream-ciphers, block-ciphers)
- Theoretical Constructions: Security relies on certain computational hardness assumptions related to simple functions

Constructing SKE schemes

- Basic idea: extensible pseudo-random one-time pads (kept compressed in the key)
- (Will also need a mechanism to ensure that the same piece of the one-time pad is not used more than once)
- Approach used in practice today: complex functions which are conjectured to have the requisite pseudo-randomness properties (stream-ciphers, block-ciphers)
- Theoretical Constructions: Security relies on certain computational hardness assumptions related to simple functions
  - Coming up: One-Way Functions, Hardcore predicates, PRG, ...

Pseudorandomness Generator (PRG)

Pseudorandomness Generator (PRG)

- Expand a short random seed to a "random-looking" string

Pseudorandomness Generator (PRG)

- Expand a short random seed to a "random-looking" string
  - So that we can build "stream ciphers" (to encrypt a stream of data, using just one short shared key)

Pseudorandomness Generator (PRG)

- Expand a short random seed to a "random-looking" string
  - So that we can build "stream ciphers" (to encrypt a stream of data, using just one short shared key)
- PRG with fixed stretch: G_k: $\{0,1\}^k \rightarrow \{0,1\}^{n(k)}$, $n(k) > k$

Pseudorandomness Generator (PRG)

- Expand a short random seed to a "random-looking" string
  - So that we can build "stream ciphers" (to encrypt a stream of data, using just one short shared key)
- PRG with fixed stretch: G_k: $\{0,1\}^k \rightarrow \{0,1\}^{n(k)}$, $n(k) > k$
- Random-looking:

Pseudorandomness Generator (PRG)

- Expand a short random seed to a "random-looking" string
  - So that we can build "stream ciphers" (to encrypt a stream of data, using just one short shared key)
- PRG with fixed stretch: $G_k: \{0,1\}^k \rightarrow \{0,1\}^{n(k)}$, $n(k) > k$
- Random-looking:
  - Next-Bit Unpredictability: PPT adversary can't predict i^{th} bit of a sample from its first $(i-1)$ bits (for every $i \in \{0,1,\ldots,n-1\}$)

Pseudorandomness Generator (PRG)

- Expand a short random seed to a "random-looking" string
  - So that we can build "stream ciphers" (to encrypt a stream of data, using just one short shared key)
- PRG with fixed stretch: $G_k: \{0,1\}^k \rightarrow \{0,1\}^{n(k)}$, $n(k) > k$
- Random-looking:
  - Next-Bit Unpredictability: PPT adversary can't predict i^{th} bit of a sample from its first (i-1) bits (for every $i \in \{0,1,\ldots,n-1\}$)
  - A "more correct" definition:

Pseudorandomness Generator (PRG)

- Expand a short random seed to a "random-looking" string
  - So that we can build "stream ciphers" (to encrypt a stream of data, using just one short shared key)
- PRG with fixed stretch: $G_k: \{0,1\}^k \rightarrow \{0,1\}^{n(k)}$, $n(k) > k$
- Random-looking:
  - Next-Bit Unpredictability: PPT adversary can't predict i^{th} bit of a sample from its first (i-1) bits (for every $i \in \{0,1,\ldots,n-1\}$)
  - A "more correct" definition:
    - PPT adversary can't distinguish between a sample from $\{G_k(x)\}_{x \leftarrow \{0,1\}^k}$ and one from $\{0,1\}^{n(k)}$

Pseudorandomness Generator (PRG)

- Expand a short random seed to a "random-looking" string
  - So that we can build "stream ciphers" (to encrypt a stream of data, using just one short shared key)
- PRG with fixed stretch: $G_k: \{0,1\}^k \rightarrow \{0,1\}^{n(k)}$, $n(k) > k$
- Random-looking:
  - Next-Bit Unpredictability: PPT adversary can't predict i^{th} bit of a sample from its first (i-1) bits (for every $i \in \{0,1,\ldots,n-1\}$)
  - A "more correct" definition:
    - PPT adversary can't distinguish between a sample from $\{G_k(x)\}_{x \leftarrow \{0,1\}^k}$ and one from $\{0,1\}^{n(k)}$

$|\ Pr_{y \leftarrow PRG}[A(y)=0] - Pr_{y \leftarrow rand}[A(y)=0]\ |$ is negligible for all PPT A

Pseudorandomness Generator (PRG)

- Expand a short random seed to a "random-looking" string
  - So that we can build "stream ciphers" (to encrypt a stream of data, using just one short shared key)
- PRG with fixed stretch: $G_k: \{0,1\}^k \rightarrow \{0,1\}^{n(k)}$, $n(k) > k$
- Random-looking:
  - Next-Bit Unpredictability: PPT adversary can't predict i^{th} bit of a sample from its first (i-1) bits (for every $i \in \{0,1,\ldots,n-1\}$)
  - A "more correct" definition:
    - PPT adversary can't distinguish between a sample from $\{G_k(x)\}_{x \leftarrow \{0,1\}^k}$ and one from $\{0,1\}^{n(k)}$
  - Turns out they are equivalent!

$|\ Pr_{y \leftarrow PRG}[A(y)=0] - Pr_{y \leftarrow rand}[A(y)=0]\ |$ is negligible for all PPT A

One-Way Function, Hardcore Predicate

One-Way Function, Hardcore Predicate

- f_k: $\{0,1\}^k \rightarrow \{0,1\}^{n(k)}$ is a one-way function (OWF) if

One-Way Function, Hardcore Predicate

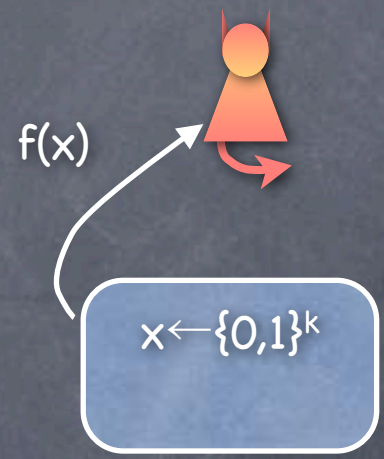
- f_k: $\{0,1\}^k \rightarrow \{0,1\}^{n(k)}$ is a one-way function (OWF) if
  - f is polynomial time computable

One-Way Function, Hardcore Predicate

- f_k: $\{0,1\}^k \rightarrow \{0,1\}^{n(k)}$ is a one-way function (OWF) if
  - f is polynomial time computable
  - For all (non-uniform) PPT adversary, probability of success in the "OWF experiment" is negligible

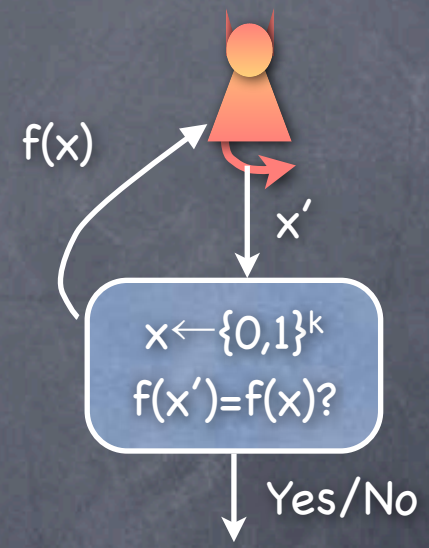
One-Way Function, Hardcore Predicate

- $f_k: \{0,1\}^k \rightarrow \{0,1\}^{n(k)}$ is a one-way function (OWF) if
  - f is polynomial time computable
  - For all (non-uniform) PPT adversary, probability of success in the "OWF experiment" is negligible



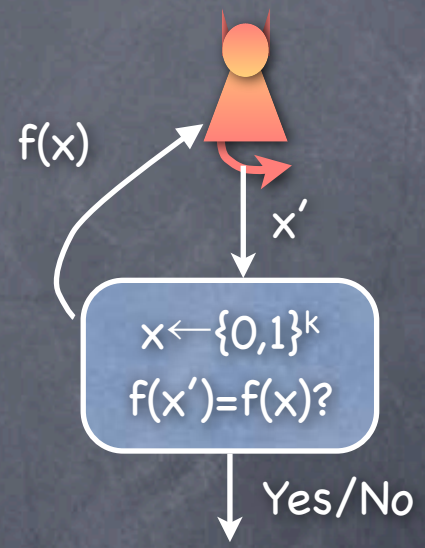
One-Way Function, Hardcore Predicate

- $f_k: \{0,1\}^k \rightarrow \{0,1\}^{n(k)}$ is a one-way function (OWF) if
  - f is polynomial time computable
  - For all (non-uniform) PPT adversary, probability of success in the "OWF experiment" is negligible



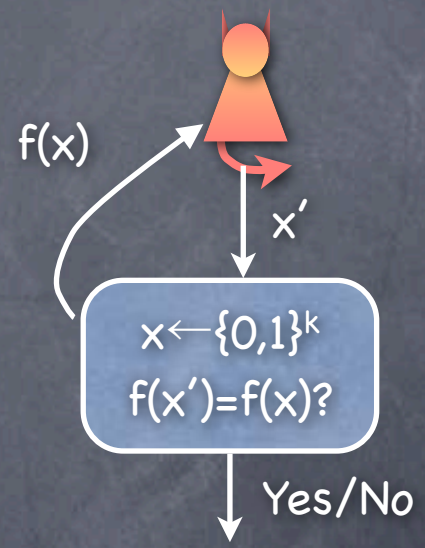
One-Way Function, Hardcore Predicate

- $f_k: \{0,1\}^k \rightarrow \{0,1\}^{n(k)}$ is a one-way function (OWF) if
  - f is polynomial time computable
  - For all (non-uniform) PPT adversary, probability of success in the "OWF experiment" is negligible
  - But x may not be completely hidden by f(x)



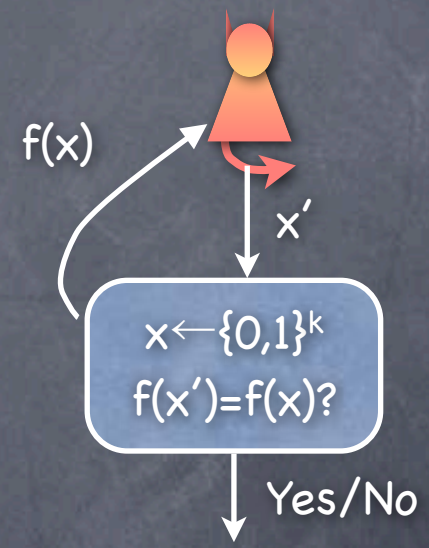
One-Way Function, Hardcore Predicate

- $f_k: \{0,1\}^k \rightarrow \{0,1\}^{n(k)}$ is a one-way function (OWF) if
  - f is polynomial time computable
  - For all (non-uniform) PPT adversary, probability of success in the "OWF experiment" is negligible
  - But x may not be completely hidden by f(x)
- B is a hardcore predicate of a OWF f if



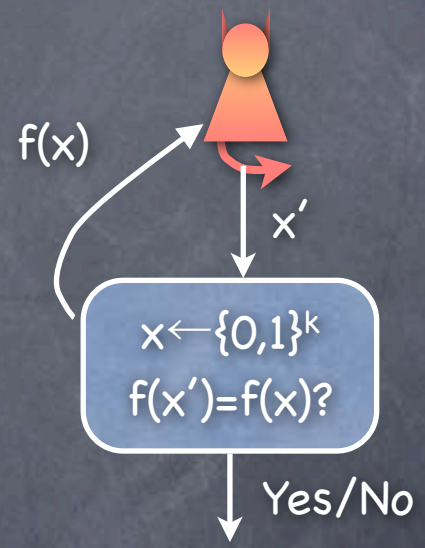
One-Way Function, Hardcore Predicate

- $f_k: \{0,1\}^k \rightarrow \{0,1\}^{n(k)}$ is a one-way function (OWF) if
  - f is polynomial time computable
  - For all (non-uniform) PPT adversary, probability of success in the "OWF experiment" is negligible
  - But x may not be completely hidden by f(x)
- B is a hardcore predicate of a OWF f if
  - B is polynomial time computable



One-Way Function, Hardcore Predicate

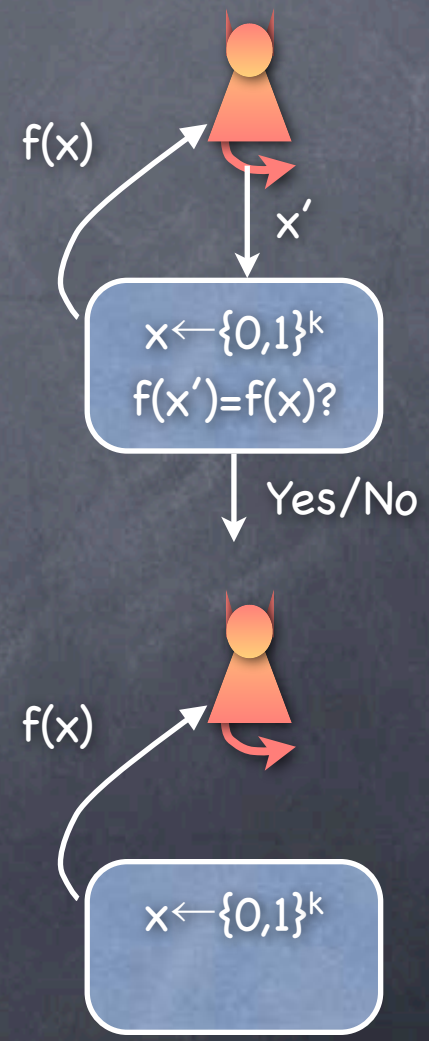
- f_k: $\{0,1\}^k \rightarrow \{0,1\}^{n(k)}$ is a one-way function (OWF) if
  - f is polynomial time computable
  - For all (non-uniform) PPT adversary, probability of success in the "OWF experiment" is negligible
  - But x may not be completely hidden by f(x)



- B is a hardcore predicate of a OWF f if
  - B is polynomial time computable
  - For all (non-uniform) PPT adversary, advantage in the Hardcore-predicate experiment is negligible

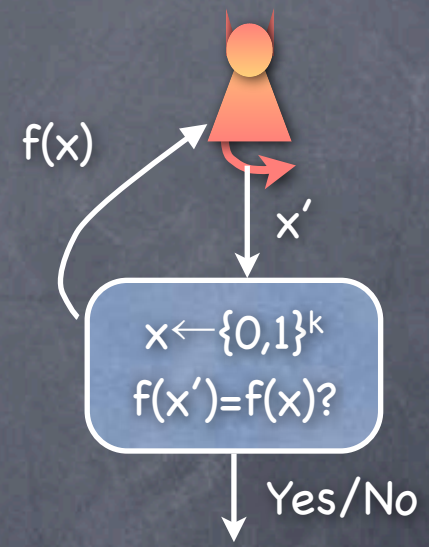
One-Way Function, Hardcore Predicate

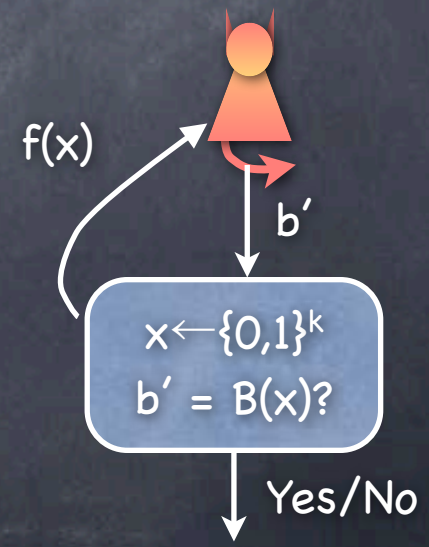
- $f_k: \{0,1\}^k \rightarrow \{0,1\}^{n(k)}$ is a one-way function (OWF) if
  - f is polynomial time computable
  - For all (non-uniform) PPT adversary, probability of success in the "OWF experiment" is negligible
  - But x may not be completely hidden by f(x)
- B is a hardcore predicate of a OWF f if
  - B is polynomial time computable
  - For all (non-uniform) PPT adversary, advantage in the Hardcore-predicate experiment is negligible



One-Way Function, Hardcore Predicate

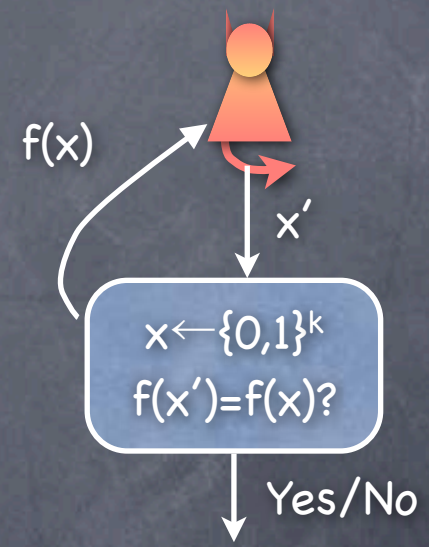
- f_k: $\{0,1\}^k \rightarrow \{0,1\}^{n(k)}$ is a one-way function (OWF) if
  - f is polynomial time computable
  - For all (non-uniform) PPT adversary, probability of success in the "OWF experiment" is negligible
  - But x may not be completely hidden by f(x)
- B is a hardcore predicate of a OWF f if
  - B is polynomial time computable
  - For all (non-uniform) PPT adversary, advantage in the Hardcore-predicate experiment is negligible

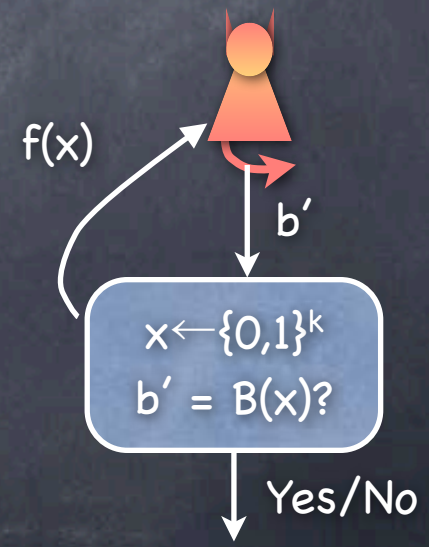




One-Way Function, Hardcore Predicate

- f_k: $\{0,1\}^k \rightarrow \{0,1\}^{n(k)}$ is a one-way function (OWF) if
  - f is polynomial time computable
  - For all (non-uniform) PPT adversary, probability of success in the "OWF experiment" is negligible
  - But x may not be completely hidden by f(x)
- B is a hardcore predicate of a OWF f if
  - B is polynomial time computable
  - For all (non-uniform) PPT adversary, advantage in the Hardcore-predicate experiment is negligible
  - B(x) remains "completely" hidden, given f(x)





Next

Next

- Candidate OWFs

Next

- Candidate OWFs
- Using OWF/Hardcore-predicates to build PRG and (CPA-secure) SKE