Some Basic Primitives

Lecture 3

One-Way Functions, PRG

Today

Today

- An important notion for symmetric-key encryption: pseudorandomness generators (PRG)

Today

- An important notion for symmetric-key encryption: pseudorandomness generators (PRG)
- Can be (in principle) built from a more basic primitive, namely One-Way Functions (OWF)

Today

- An important notion for symmetric-key encryption: pseudorandomness generators (PRG)
- Can be (in principle) built from a more basic primitive, namely One-Way Functions (OWF)
- Existence of these notions depends on computational complexity assumptions

Today

- An important notion for symmetric-key encryption: pseudorandomness generators (PRG)
- Can be (in principle) built from a more basic primitive, namely One-Way Functions (OWF)
- Existence of these notions depends on computational complexity assumptions
- First, some complexity-speak...

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

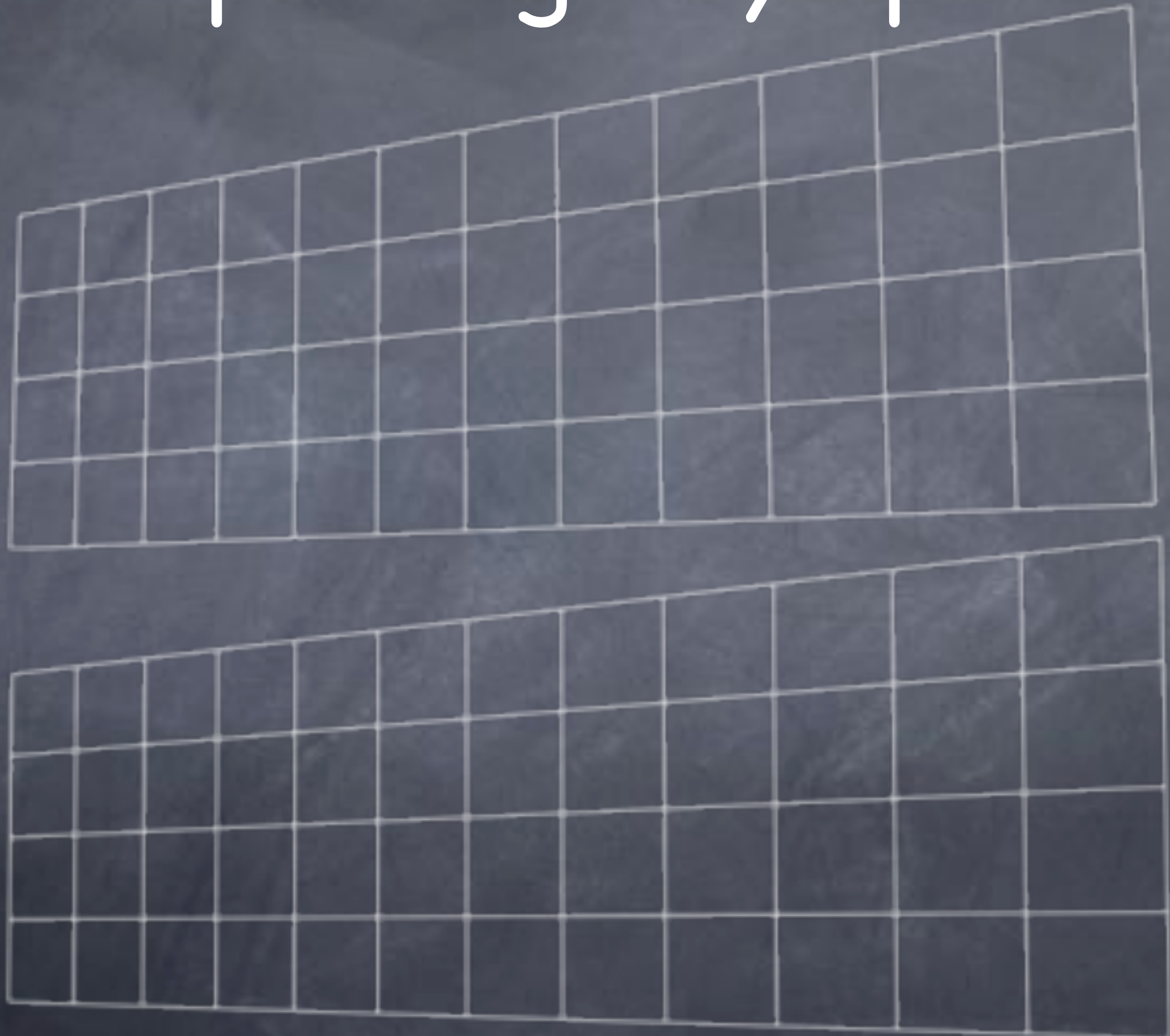
- A parameter that is part of the encryption scheme
  - Not related to message size
  - A knob that can be used to set the security level
  - Will denote by k

Security Parameter

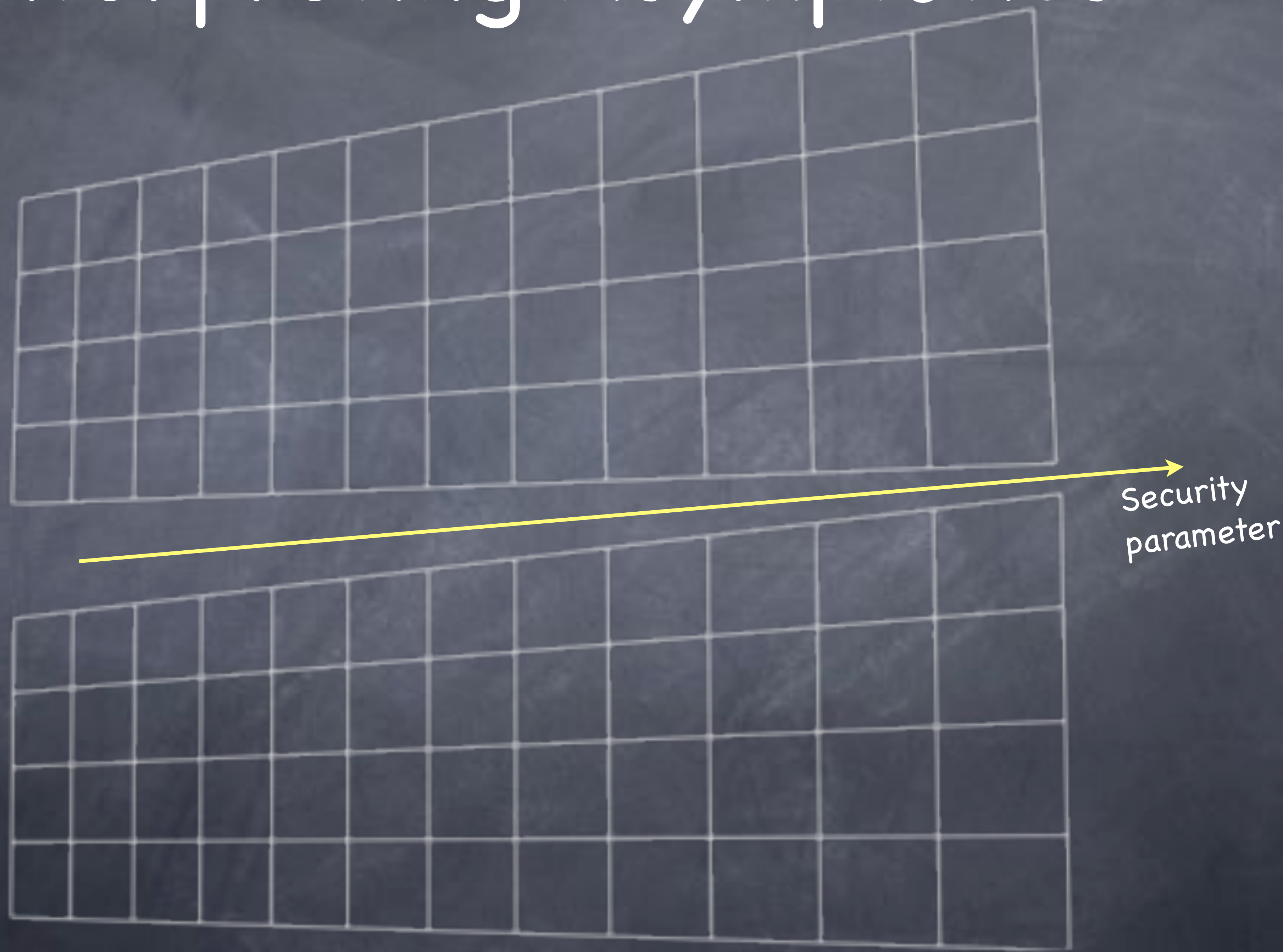
- A parameter that is part of the encryption scheme
  - Not related to message size
  - A knob that can be used to set the security level
  - Will denote by k
- Security guarantees are given asymptotically as a function of the security parameter

Interpreting Asymptotics

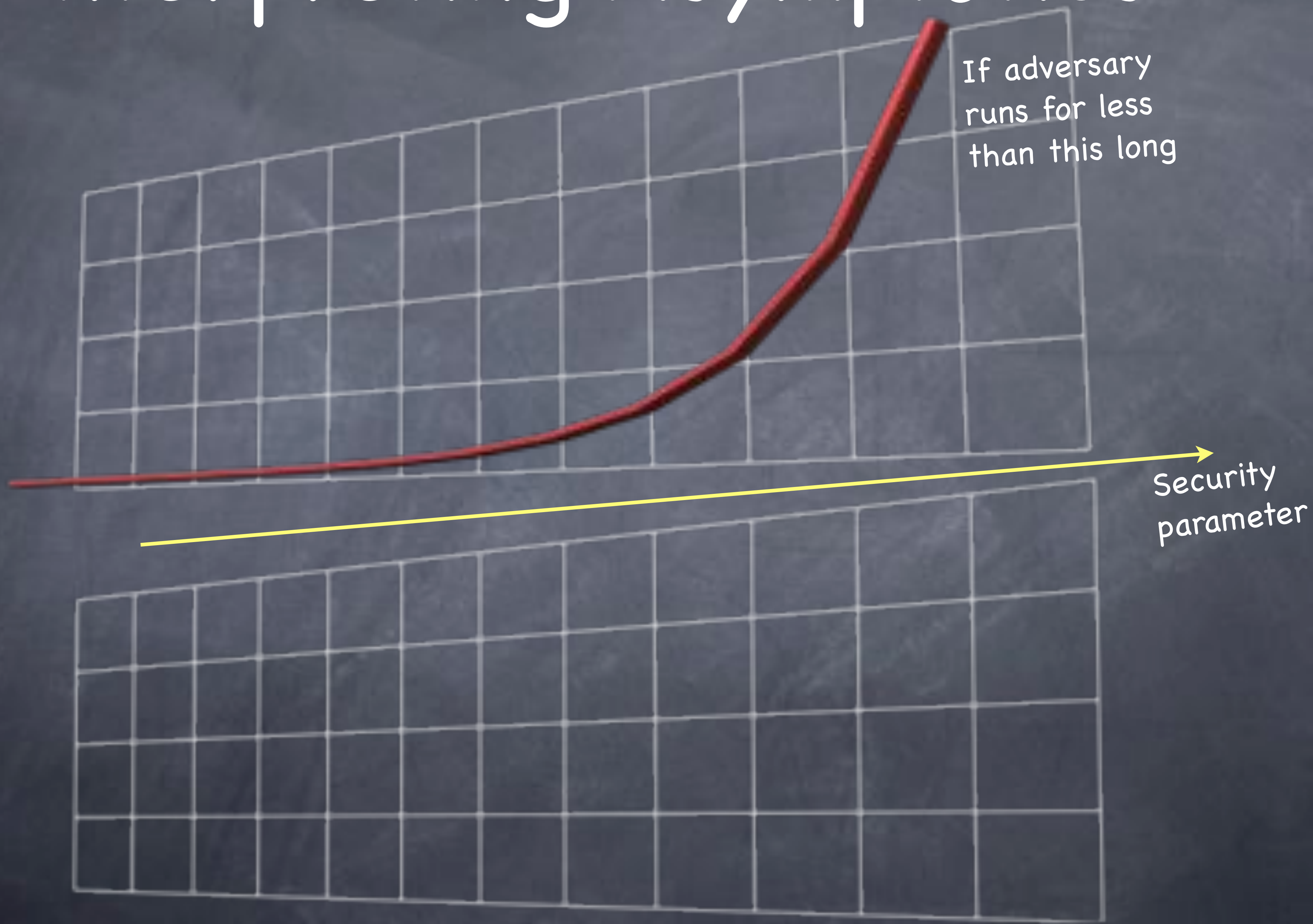
Interpreting Asymptotics



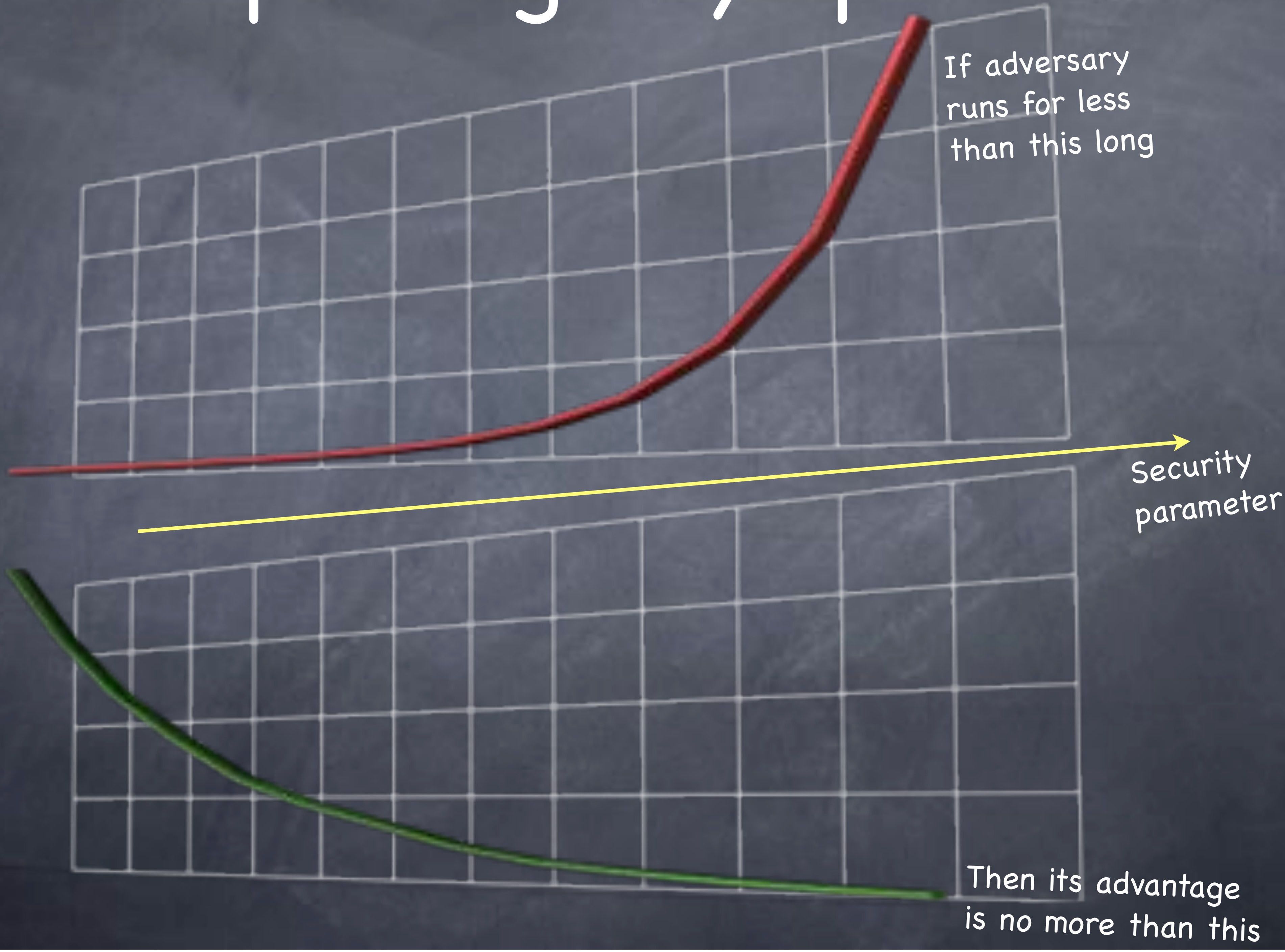
Interpreting Asymptotics



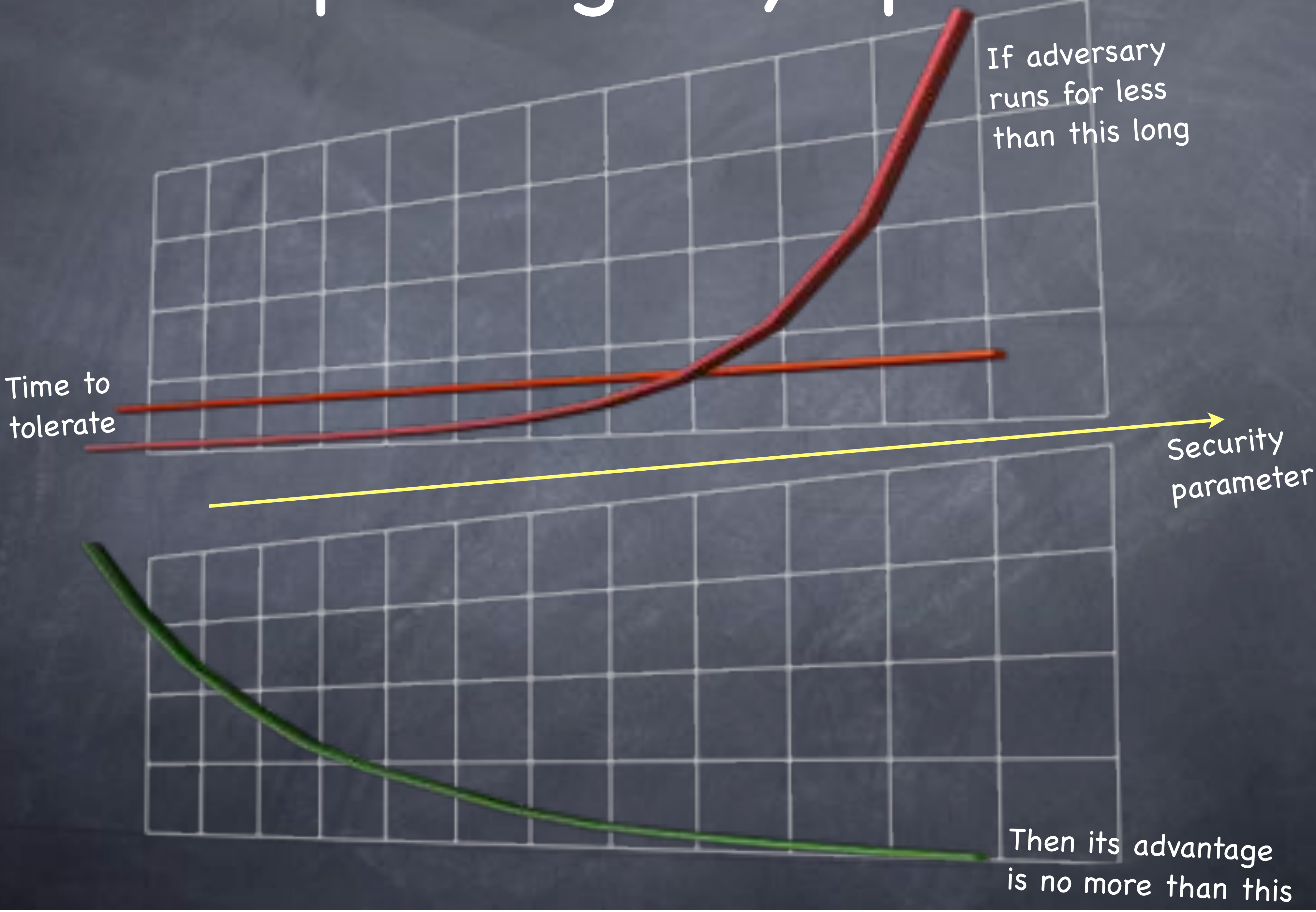
Interpreting Asymptotics



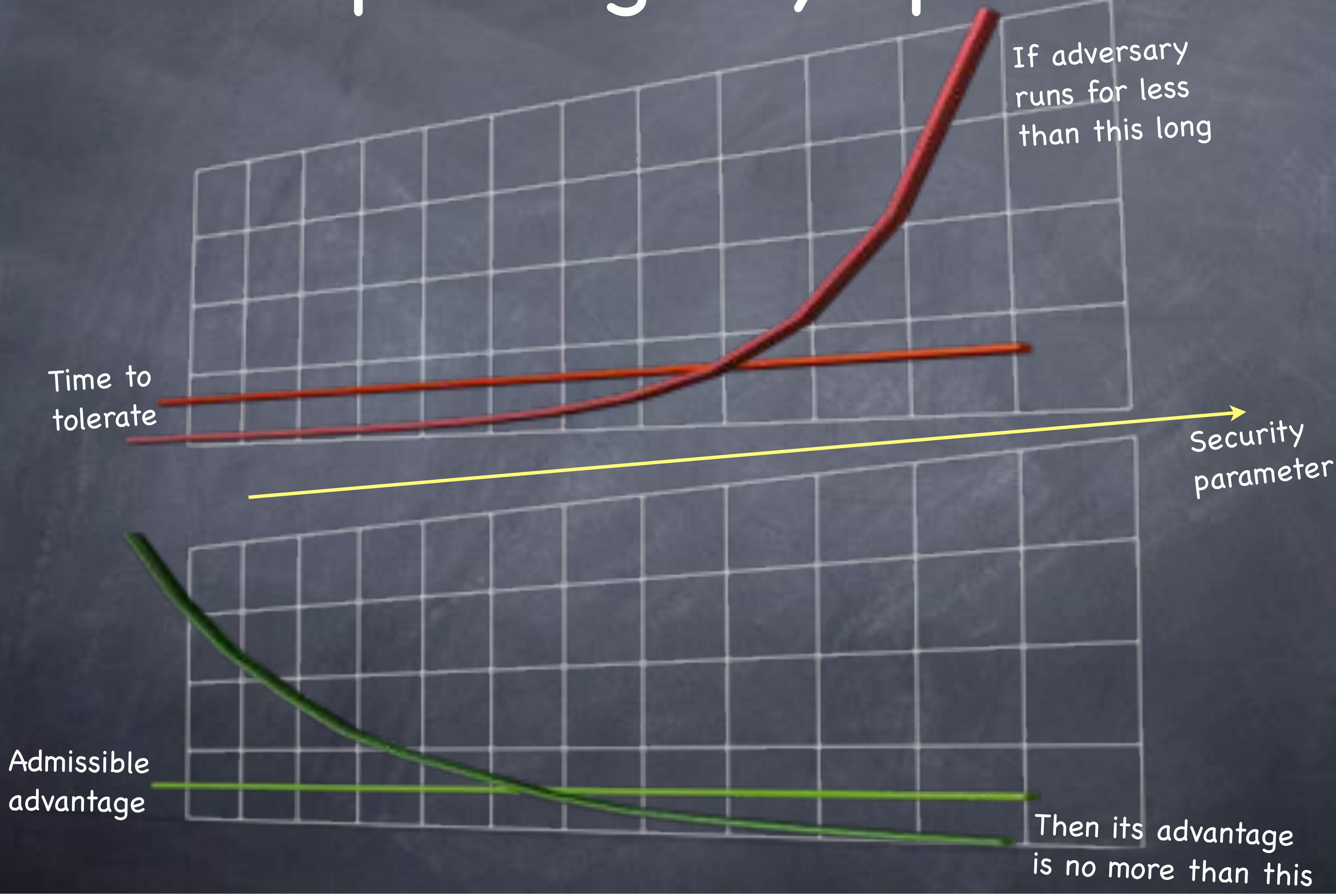
Interpreting Asymptotics



Interpreting Asymptotics



Interpreting Asymptotics



Feasible and Negligible

Feasible and Negligible

- We shall restrict ourselves to Eves who have a running time bounded by some polynomial in k

Feasible and Negligible

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

Feasible and Negligible

- We shall restrict ourselves to 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

Feasible and Negligible

- We shall restrict ourselves to 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
    - But algorithms for Alice/Bob better be very efficient

Feasible and Negligible

- We shall restrict ourselves to 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
    - But algorithms for Alice/Bob better be very efficient
  - Eve could be non-uniform: a different strategy for each k

Feasible and Negligible

- We shall restrict ourselves to 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
    - 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 shall restrict ourselves to 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
    - 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

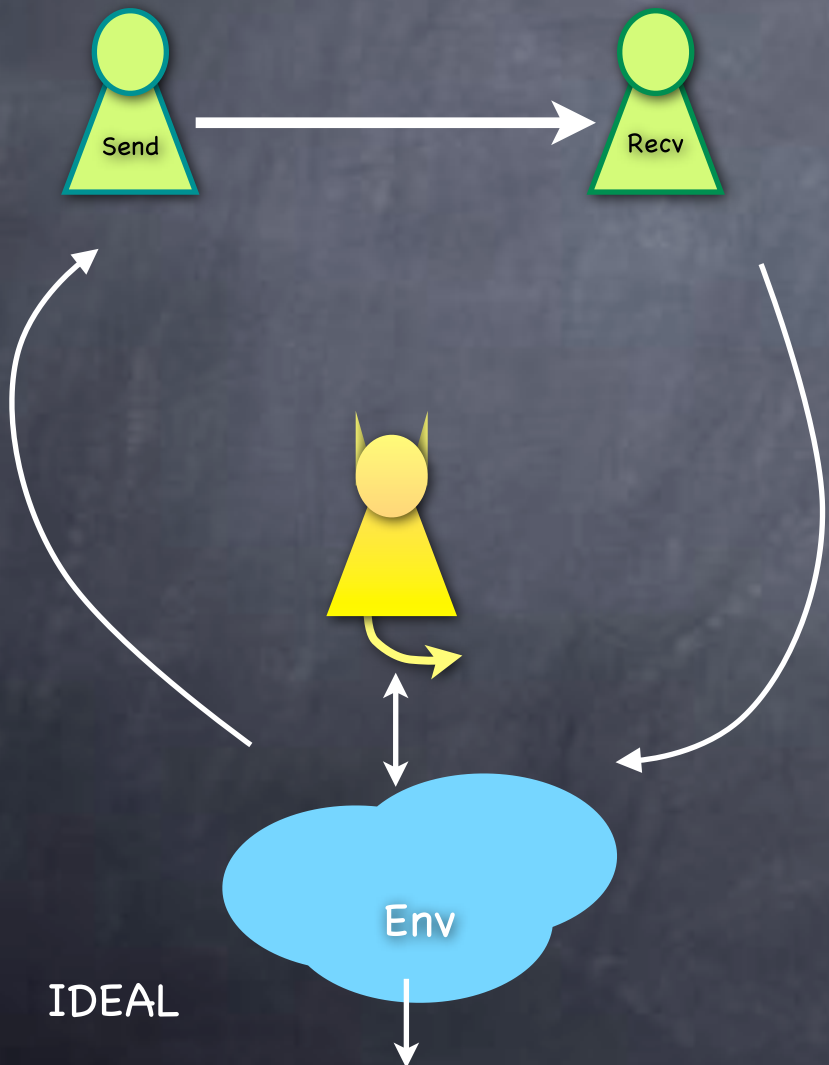
- A negligible quantity: As we turn the knob the quantity should "decrease extremely fast"
  - Negligible: decreases as 1/superpoly(k)
    - i.e., faster than 1/poly(k) for every polynomial
    - e.g.: 2^{-k}, $2^{-\sqrt{k}}$, $k^{-(\log k)}$.
  - So that negl(k) × poly(k) = negl′(k)

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}, $2^{-\sqrt{k}}$, $k^{-(\log k)}$.
  - So that negl(k) × 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



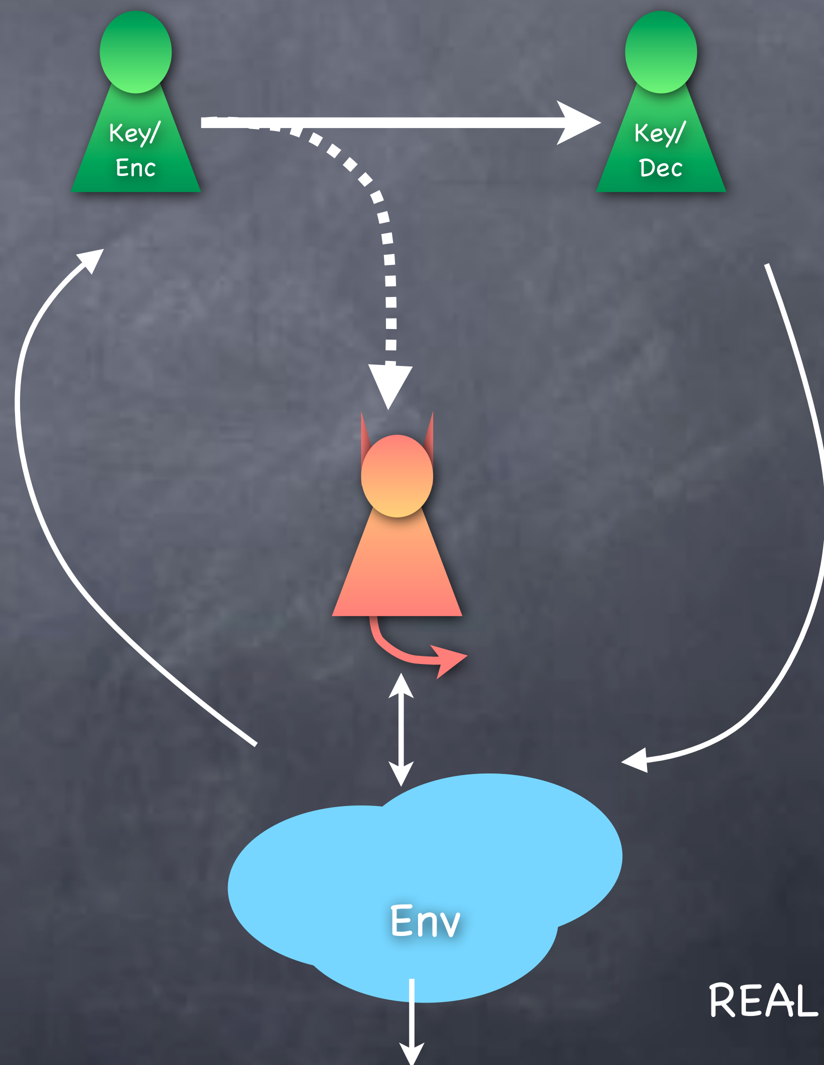
SIM-CPA secure if:

∀ PPT

∃ PPT s.t.

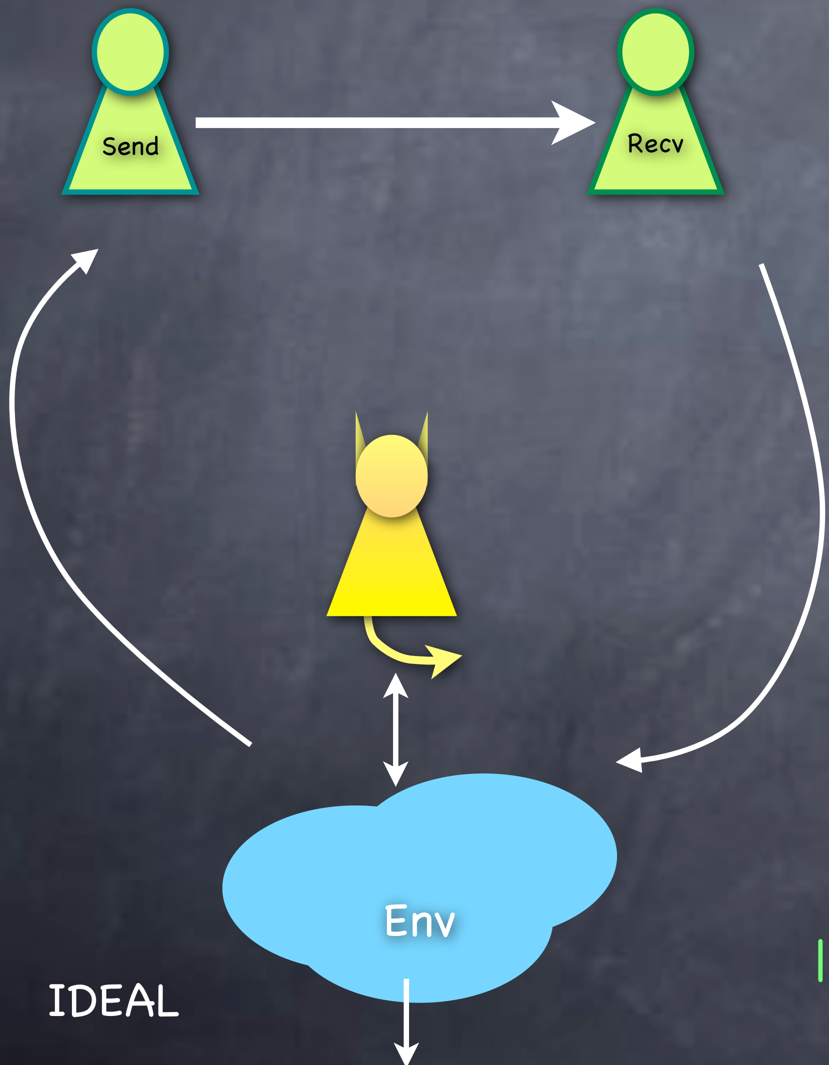
∀ PPT

REAL ≈ IDEAL



Symmetric-Key Encryption

SIM-CPA Security



SIM-CPA secure if:

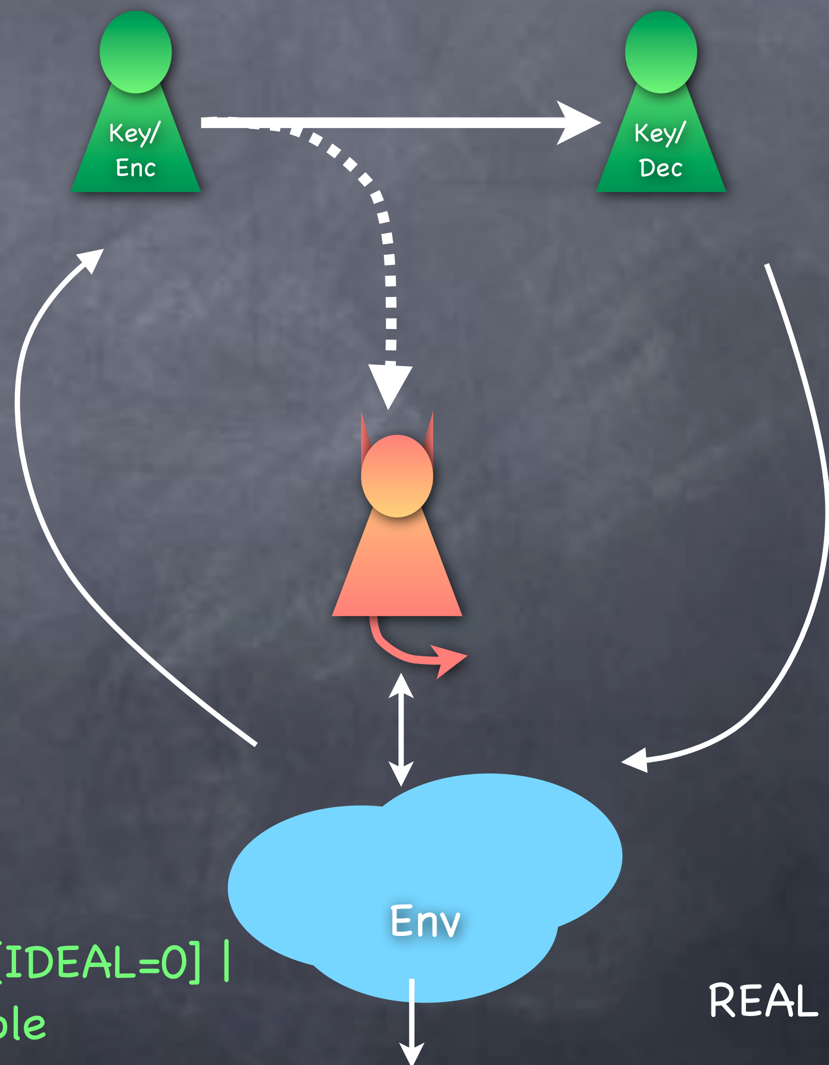
∀ PPT

∃ PPT s.t.

∀ PPT

REAL ≈ IDEAL

| Pr[REAL=0] - Pr[IDEAL=0] | is negligible



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, ...

One-Way Function, Hardcore Predicate

One-Way Function, Hardcore Predicate

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

One-Way Function, Hardcore Predicate

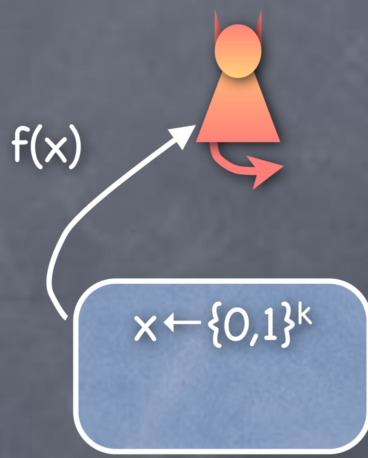
- f: $\{0,1\}^k \rightarrow \{0,1\}^n$ is a one-way function (OWF) if
  - f polynomial time computable

One-Way Function, Hardcore Predicate

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

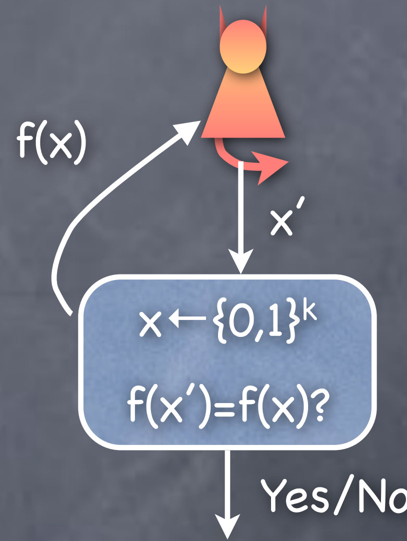
One-Way Function, Hardcore Predicate

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



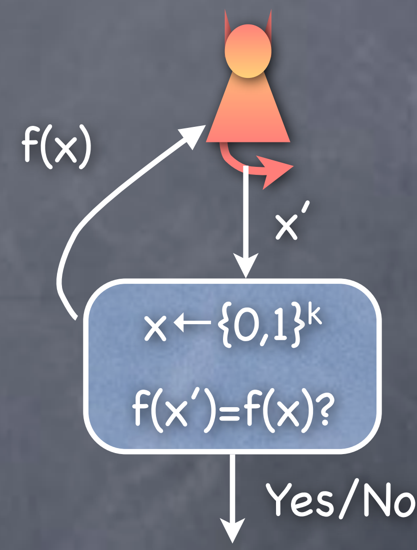
One-Way Function, Hardcore Predicate

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



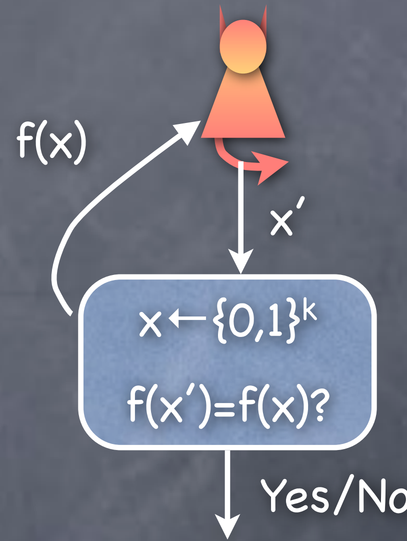
One-Way Function, Hardcore Predicate

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

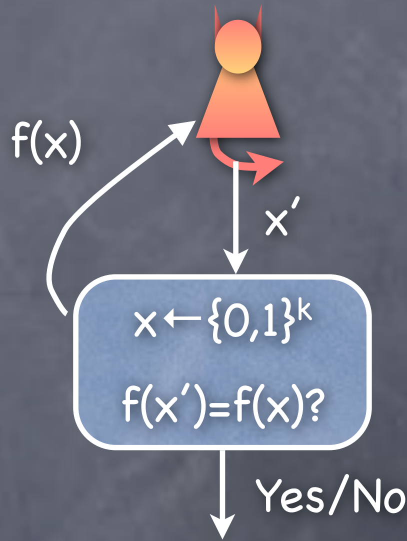
- f: $\{0,1\}^k \rightarrow \{0,1\}^n$ is a one-way function (OWF) if
  - f polynomial time computable
  - For all (non-uniform) PPT adversary, probability of success in the "OWF experiment" is negligible
  - But x may not be completely hidden by f(x)



- B is a hardcore predicate of a OWF f if

One-Way Function, Hardcore Predicate

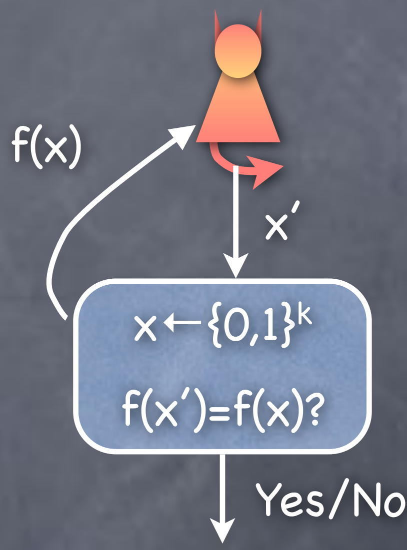
- f: $\{0,1\}^k \rightarrow \{0,1\}^n$ is a one-way function (OWF) if
  - f polynomial time computable
  - For all (non-uniform) PPT adversary, probability of success in the "OWF experiment" is negligible
  - But x may not be completely hidden by f(x)



- B is a hardcore predicate of a OWF f if
  - B is polynomial time computable

One-Way Function, Hardcore Predicate

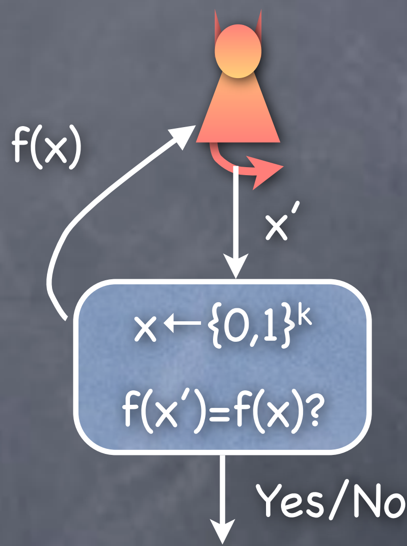
- f: $\{0,1\}^k \rightarrow \{0,1\}^n$ is a one-way function (OWF) if
  - f polynomial time computable
  - For all (non-uniform) PPT adversary, probability of success in the "OWF experiment" is negligible
  - But x may not be completely hidden by f(x)



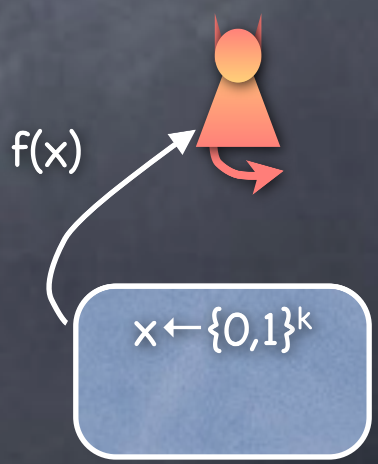
- B is a hardcore predicate of a OWF f if
  - B is polynomial time computable
  - For all (non-uniform) PPT adversary, <u>advantage</u> in the Hardcore-predicate experiment is negligible

One-Way Function, Hardcore Predicate

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

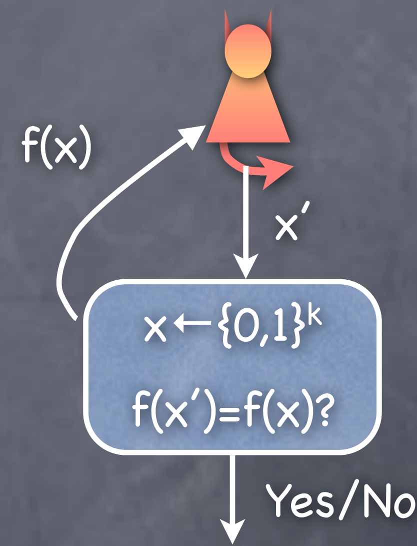


- B is a hardcore predicate of a OWF f if
  - B is polynomial time computable
  - For all (non-uniform) PPT adversary, advantage in the Hardcore-predicate experiment is negligible

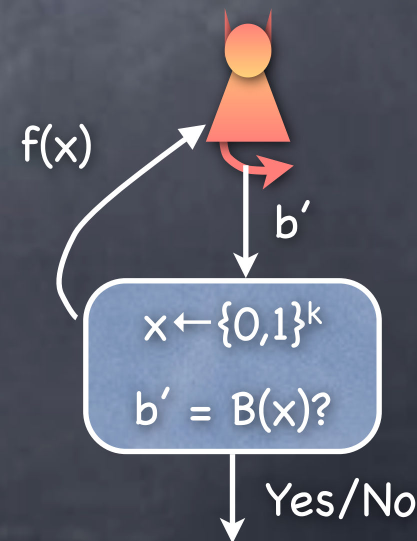


One-Way Function, Hardcore Predicate

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

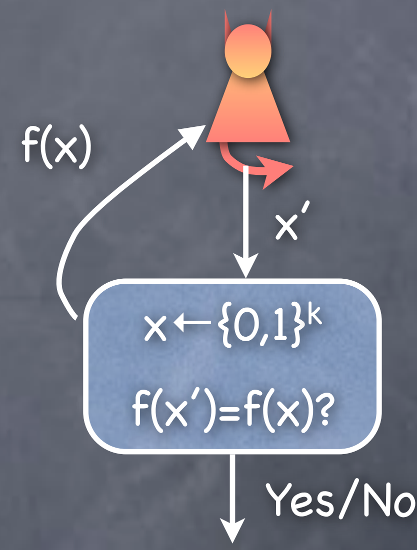


- B is a hardcore predicate of a OWF f if
  - B is polynomial time computable
  - For all (non-uniform) PPT adversary, advantage in the Hardcore-predicate experiment is negligible

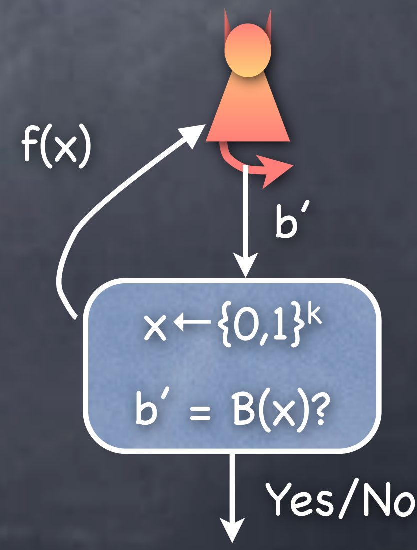


One-Way Function, Hardcore Predicate

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



One-Way Function Candidates

One-Way Function Candidates

- Integer factorization:

One-Way Function Candidates

- Integer factorization:
  - $f_{mult}(x,y) = x.y$

One-Way Function Candidates

- Integer factorization:
  - $f_{mult}(x,y) = x.y$
  - Input distribution: (x,y) random k-bit primes

One-Way Function Candidates

- Integer factorization:
  - $f_{mult}(x,y) = x.y$
  - Input distribution: (x,y) random k-bit primes
  - Fact: input distribution (x,y) random k-bit integers will also work (if k-bit primes distribution works)

One-Way Function Candidates

- Integer factorization:
  - $f_{mult}(x,y) = x.y$
  - Input distribution: (x,y) random k-bit primes
  - Fact: input distribution (x,y) random k-bit integers will also work (if k-bit primes distribution works)
    - Important that we require $|x|=|y|=k$, not $|x.y|=k$ (otherwise, 2 is a factor of x.y with 3/4 probability)

One-Way Function Candidates

One-Way Function Candidates

- Solving Subset Sum:

One-Way Function Candidates

- Solving Subset Sum:
  - $f_{subsum}(x_1 \ldots x_k, S) = (x_1 \ldots x_k, \Sigma_{i \in S}\ x_i)$

One-Way Function Candidates

- Solving Subset Sum:
  - $f_{subsum}(x_1...x_k, S) = (x_1...x_k, \Sigma_{i\in S}\ x_i\ )$
  - Input distribution: x_i k-bit integers, $S\subseteq\{1...k\}$. Uniform

One-Way Function Candidates

- Solving Subset Sum:
  - $f_{subsum}(x_1...x_k, S) = (x_1...x_k, \Sigma_{i \in S} x_i)$
  - Input distribution: x_i k-bit integers, $S \subseteq \{1...k\}$. Uniform
  - Inverting f_{subsum} known to be NP-complete, but assuming that it is a OWF is "stronger" than assuming P≠NP

One-Way Function Candidates

One-Way Function Candidates

- Rabin OWF: $f_{Rabin}(x; n) = (x^2 \bmod n, n)$, where $n = pq$, and p, q are random k-bit primes, and x is uniform from $\{0...n\}$

One-Way Function Candidates

- Rabin OWF: $f_{Rabin}(x; n) = (x^2 \bmod n, n)$, where $n = pq$, and p, q are random k-bit primes, and x is uniform from {0...n}
  - Note that n is part of the output. This OWF can be used as a "OWF collection" indexed by n (many n's for the same k)

One-Way Function Candidates

- Rabin OWF: $f_{Rabin}(x; n) = (x^2 \bmod n, n)$, where $n = pq$, and p, q are random k-bit primes, and x is uniform from {0...n}
  - Note that n is part of the output. This OWF can be used as a "OWF collection" indexed by n (many n's for the same k)
- More e.g.: RSA function (uses as index: n=pq, and an exponent e), Discrete Logarithm (uses as index: a group and a generator)

One-Way Function Candidates

- Rabin OWF: $f_{Rabin}(x; n) = (x^2 \bmod n, n)$, where $n = pq$, and p, q are random k-bit primes, and x is uniform from $\{0...n\}$
  - Note that n is part of the output. This OWF can be used as a "OWF collection" indexed by n (many n's for the same k)
- More e.g.: RSA function (uses as index: n=pq, and an exponent e), Discrete Logarithm (uses as index: a group and a generator)
  - Later

Hardcore Predicates

Hardcore Predicates

- For candidate OWFs, often hardcore predicates known

Hardcore Predicates

- For candidate OWFs, often hardcore predicates known
  - e.g. if $f_{Rabin}(x;n)$ (with certain restrictions on sampling x and n) is a OWF, then LSB(x) is a hardcore predicate for it

Hardcore Predicates

- For candidate OWFs, often hardcore predicates known
  - e.g. if $f_{Rabin}(x;n)$ (with certain restrictions on sampling x and n) is a OWF, then LSB(x) is a hardcore predicate for it
    - Reduction: Given an algorithm for finding LSB(x) from $f_{Rabin}(x;n)$ for random x, show how to invert f_{Rabin}

Goldreich-Levin Predicate

Goldreich-Levin Predicate

- Given any OWF f, can slightly modify it to get a OWF g_f such that

Goldreich-Levin Predicate

- Given any OWF f, can slightly modify it to get a OWF g_f such that
  - g_f has a simple hardcore predicate

Goldreich-Levin Predicate

- Given any OWF f, can slightly modify it to get a OWF g_f such that
  - g_f has a simple hardcore predicate
  - g_f is almost as efficient as f; is a permutation if f is one

Goldreich-Levin Predicate

- Given any OWF f, can slightly modify it to get a OWF g_f such that
  - g_f has a simple hardcore predicate
  - g_f is almost as efficient as f; is a permutation if f is one
- $g_f(x,r) = (f(x), r)$, where $|r|=|x|$

Goldreich-Levin Predicate

- Given any OWF f, can slightly modify it to get a OWF g_f such that
  - g_f has a simple hardcore predicate
  - g_f is almost as efficient as f; is a permutation if f is one
- $g_f(x,r) = (f(x), r)$, where $|r|=|x|$
  - Input distribution: x as for f, and r independently random

Goldreich-Levin Predicate

- Given any OWF f, can slightly modify it to get a OWF g_f such that
  - g_f has a simple hardcore predicate
  - g_f is almost as efficient as f; is a permutation if f is one
- $g_f(x,r) = (f(x), r)$, where $|r|=|x|$
  - Input distribution: x as for f, and r independently random
- GL-predicate: $B(x,r) = \langle x,r \rangle$ (dot product of bit vectors)

Goldreich-Levin Predicate

- Given any OWF f, can slightly modify it to get a OWF g_f such that
  - g_f has a simple hardcore predicate
  - g_f is almost as efficient as f; is a permutation if f is one
- $g_f(x,r) = (f(x), r)$, where $|r|=|x|$
  - Input distribution: x as for f, and r independently random
- GL-predicate: $B(x,r) = \langle x,r \rangle$ (dot product of bit vectors)
  - Can show that a predictor of B(x,r) with non-negligible advantage can be turned into an inversion algorithm for f

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)
- First, PRG with fixed stretch: $G_k: \{0,1\}^k \to \{0,1\}^{n(k)}$, $n(k) > k$

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)
- First, 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)
- First, 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)
- First, 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 $i \in \{0,1,...,n-1\}$)
  - 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)
- First, 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)
- First, 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)
- First, 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

PRG from One-Way Permutations

PRG from One-Way Permutations

- One-bit stretch PRG, $G_k: \{0,1\}^k \rightarrow \{0,1\}^{k+1}$

PRG from One-Way Permutations

- One-bit stretch PRG, G_k: $\{0,1\}^k \rightarrow \{0,1\}^{k+1}$

PRG from One-Way Permutations

- One-bit stretch PRG, G_k: $\{0,1\}^k \rightarrow \{0,1\}^{k+1}$
  - $G(x) = f(x) \circ B(x)$

PRG from One-Way Permutations

- One-bit stretch PRG, G_k: $\{0,1\}^k \to \{0,1\}^{k+1}$
  - $G(x) = f(x) \circ B(x)$
  - Where f: $\{0,1\}^k \to \{0,1\}^k$ is a one-way permutation, and B a hardcore predicate for f

PRG from One-Way Permutations

- One-bit stretch PRG, G_k: $\{0,1\}^k \rightarrow \{0,1\}^{k+1}$

  - $G(x) = f(x) \circ B(x)$
  - Where f: $\{0,1\}^k \rightarrow \{0,1\}^k$ is a one-way permutation, and B a hardcore predicate for f
  - For a random x, f(x) is also random, and hence all of f(x) is next-bit unpredictable. B is a hardcore predicate, so B(x) remains unpredictable after seeing f(x)

PRG from One-Way Permutations

- One-bit stretch PRG, G_k: $\{0,1\}^k \rightarrow \{0,1\}^{k+1}$

  - $G(x) = f(x) \circ B(x)$
  - Where f: $\{0,1\}^k \rightarrow \{0,1\}^k$ is a one-way permutation, and B a hardcore predicate for f
  - For a random x, f(x) is also random, and hence all of f(x) is next-bit unpredictable. B is a hardcore predicate, so B(x) remains unpredictable after seeing f(x)
  - Important: holds only when the seed x is kept hidden, and is random

PRG from One-Way Permutations

- One-bit stretch PRG, G_k: $\{0,1\}^k \rightarrow \{0,1\}^{k+1}$

  - $G(x) = f(x) \circ B(x)$
  - Where f: $\{0,1\}^k \rightarrow \{0,1\}^k$ is a one-way permutation, and B a hardcore predicate for f
  - For a random x, f(x) is also random, and hence all of f(x) is next-bit unpredictable. B is a hardcore predicate, so B(x) remains unpredictable after seeing f(x)
  - Important: holds only when the seed x is kept hidden, and is random
    - ... or pseudorandom

PRG from One-Way Permutations

PRG from One-Way Permutations

- One-bit stretch PRG, G_k: $\{0,1\}^k \rightarrow \{0,1\}^{k+1}$

PRG from One-Way Permutations

- One-bit stretch PRG, $G_k: \{0,1\}^k \rightarrow \{0,1\}^{k+1}$

- Increasing the stretch

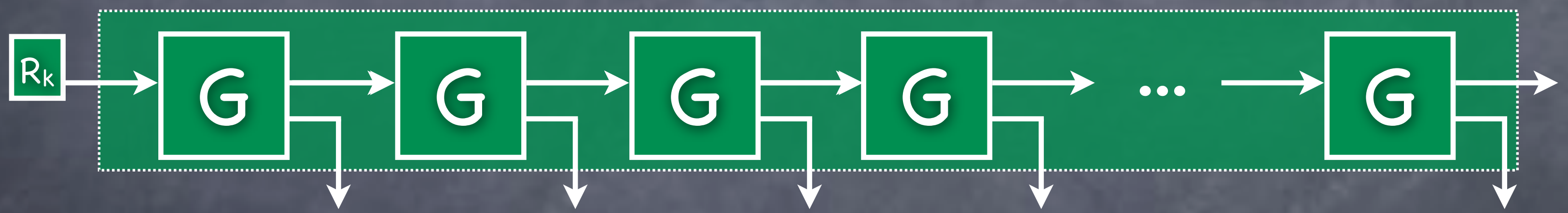
PRG from One-Way Permutations

- One-bit stretch PRG, G_k: $\{0,1\}^k \rightarrow \{0,1\}^{k+1}$
- Increasing the stretch
  - Can use part of the PRG output as a new seed

PRG from One-Way Permutations

- One-bit stretch PRG, $G_k: \{0,1\}^k \rightarrow \{0,1\}^{k+1}$

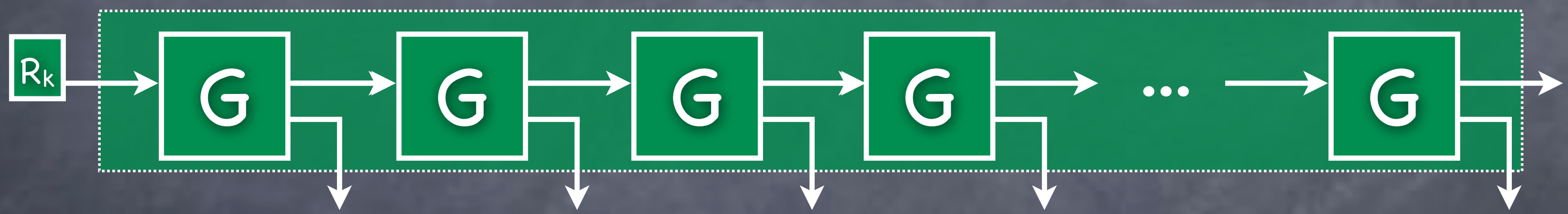
- Increasing the stretch
  - Can use part of the PRG output as a new seed



PRG from One-Way Permutations

- One-bit stretch PRG, $G_k: \{0,1\}^k \rightarrow \{0,1\}^{k+1}$

- Increasing the stretch
  - Can use part of the PRG output as a new seed

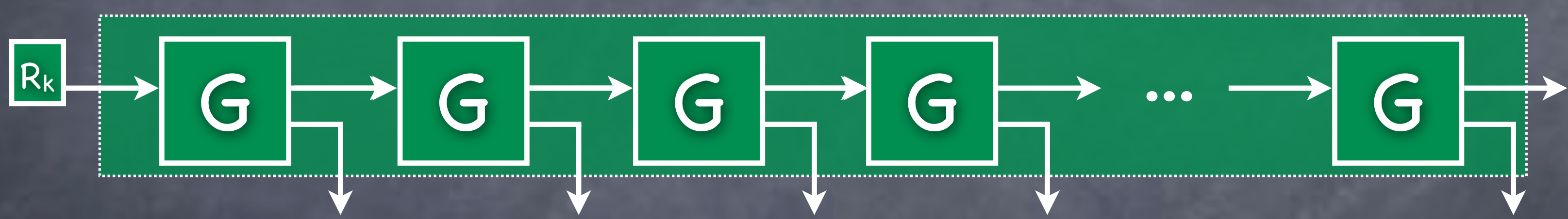


  - If the intermediate seeds are never output, can keep stretching on demand (for any "polynomial length")

PRG from One-Way Permutations

- One-bit stretch PRG, $G_k: \{0,1\}^k \rightarrow \{0,1\}^{k+1}$

- Increasing the stretch
  - Can use part of the PRG output as a new seed



  - If the intermediate seeds are never output, can keep stretching on demand (for any "polynomial length")
  - A stream cipher

One-time CPA-secure SKE with a Stream-Cipher

One-time CPA-secure SKE with a Stream-Cipher

- One-time Encryption with a stream-cipher:

One-time CPA-secure SKE with a Stream-Cipher

- One-time Encryption with a stream-cipher:
  - Generate a one-time pad from a short seed

One-time CPA-secure SKE with a Stream-Cipher

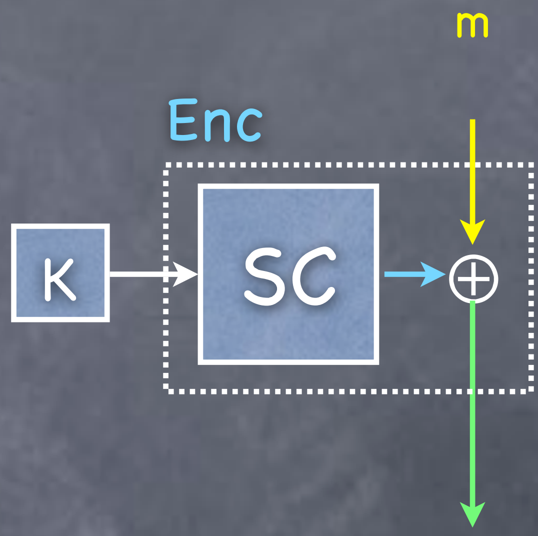
- One-time Encryption with a stream-cipher:
  - Generate a one-time pad from a short seed
  - Can share just the seed as the key

One-time CPA-secure SKE with a Stream-Cipher

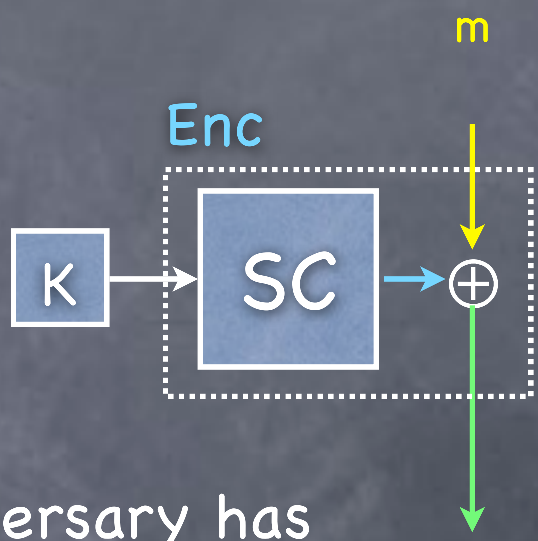
- One-time Encryption with a stream-cipher:
  - Generate a one-time pad from a short seed
  - Can share just the seed as the key
  - Mask message with the pseudorandom pad

One-time CPA-secure SKE with a Stream-Cipher

- One-time Encryption with a stream-cipher:
  - Generate a one-time pad from a short seed
  - Can share just the seed as the key
  - Mask message with the pseudorandom pad



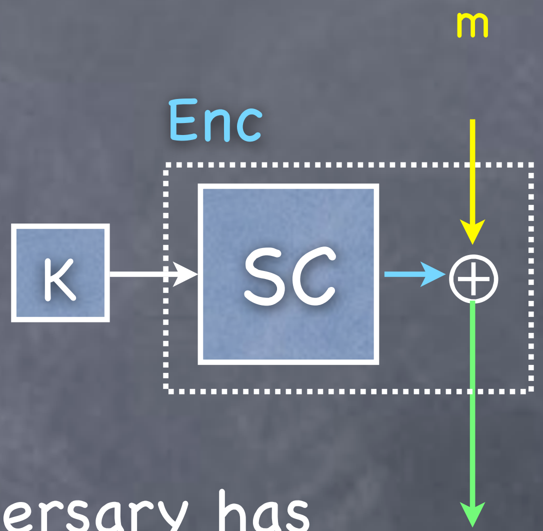
One-time CPA-secure SKE with a Stream-Cipher



- One-time Encryption with a stream-cipher:
  - Generate a one-time pad from a short seed
  - Can share just the seed as the key
  - Mask message with the pseudorandom pad
- Security: if using a real (long) random pad, adversary has no advantage in the IND-CPA experiment. If the adversary has a non-negligible advantage when the output of SC is used, then that output is not pseudorandom: this adversary can be used to distinguish it from random

One-time CPA-secure SKE with a Stream-Cipher

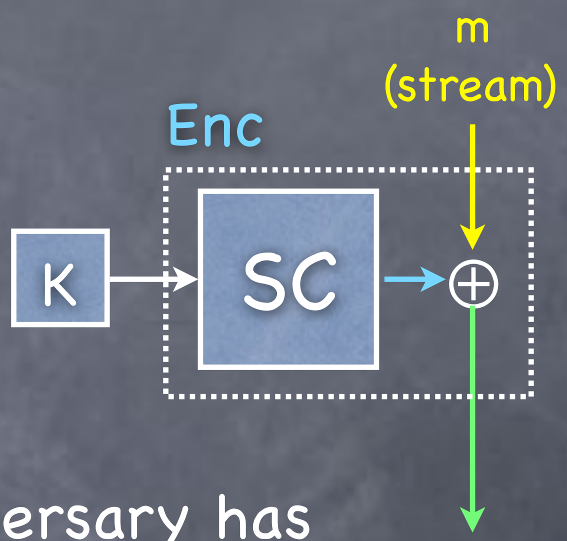
- One-time Encryption with a stream-cipher:
  - Generate a one-time pad from a short seed
  - Can share just the seed as the key
  - Mask message with the pseudorandom pad



- Security: if using a real (long) random pad, adversary has no advantage in the IND-CPA experiment. If the adversary has a non-negligible advantage when the output of SC is used, then that output is not pseudorandom: this adversary can be used to distinguish it from random
- SC can spit out bits on demand, so the message can arrive bit by bit, and the length of the message doesn't have to be a priori fixed

One-time CPA-secure SKE with a Stream-Cipher

- One-time Encryption with a stream-cipher:
  - Generate a one-time pad from a short seed
  - Can share just the seed as the key
  - Mask message with the pseudorandom pad



- Security: if using a real (long) random pad, adversary has no advantage in the IND-CPA experiment. If the adversary has a non-negligible advantage when the output of SC is used, then that output is not pseudorandom: this adversary can be used to distinguish it from random
- SC can spit out bits on demand, so the message can arrive bit by bit, and the length of the message doesn't have to be a priori fixed

Story So Far

Story So Far

- OWF, OWP, Hardcore predicates

Story So Far

- OWF, OWP, Hardcore predicates
- Output of a PRG on a random (hidden) seed is computationally indistinguishable from random

Story So Far

- OWF, OWP, Hardcore predicates
- Output of a PRG on a random (hidden) seed is computationally indistinguishable from random
    - A PRG can be constructed from a OWP and a hardcore predicate. Possible from OWF too, but more complicated. (Any way, many candidate OWFs are in fact permutations.)

Story So Far

- OWF, OWP, Hardcore predicates
- Output of a PRG on a random (hidden) seed is computationally indistinguishable from random
  - A PRG can be constructed from a OWP and a hardcore predicate. Possible from OWF too, but more complicated. (Any way, many candidate OWFs are in fact permutations.)
  - Useful in SKE: Can use PRG to stretch a short key to a long (one-time) pad. Or use as a Stream Cipher.

Story So Far

- OWF, OWP, Hardcore predicates
- Output of a PRG on a random (hidden) seed is computationally indistinguishable from random
  - A PRG can be constructed from a OWP and a hardcore predicate. Possible from OWF too, but more complicated. (Any way, many candidate OWFs are in fact permutations.)
  - Useful in SKE: Can use PRG to stretch a short key to a long (one-time) pad. Or use as a Stream Cipher.
  - Next: Constructing a proper (multi-message) SKE scheme