Symmetric-Key Encryption: constructions

Lecture 4

OWF, PRG, Stream Cipher

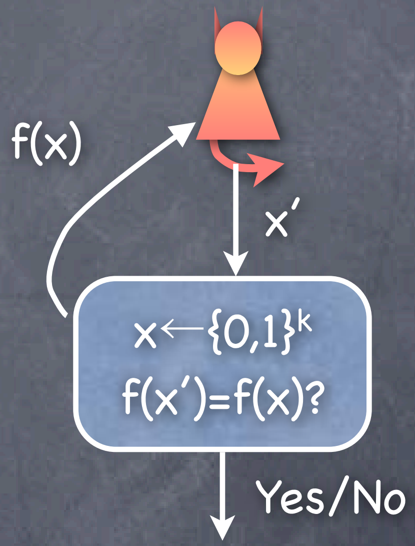
RECALL

One-Way Function, Hardcore Predicate

RECALL

One-Way Function, Hardcore Predicate

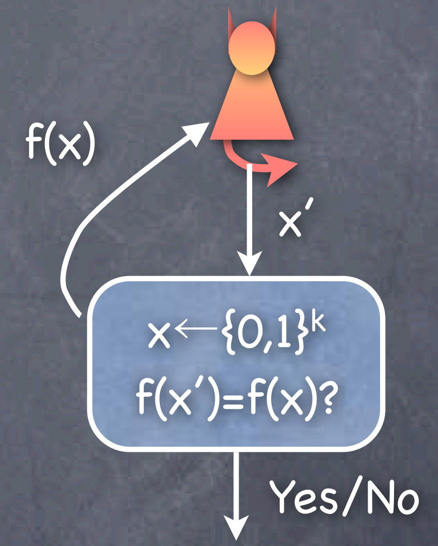
- f_k: $\{0,1\}^k \rightarrow \{0,1\}^{n(k)}$ is a one-way function (OWF) if
  - f is polynomial time computable
  - For all (non-uniform) PPT adversary, probability of success in the "OWF experiment" is negligible
  - But x may not be completely hidden by f(x)



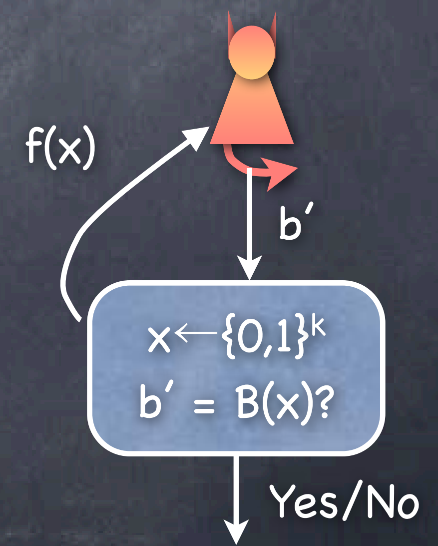
RECALL

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, <u>advantage</u> 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:
  - $f_{mult}(x,y) = x \cdot y$

One-Way Function Candidates

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

One-Way Function Candidates

- Integer factorization:
  - $f_{mult}(x,y) = x \cdot y$
  - Input distribution: (x,y) random k-bit primes
  - Fact: taking input domain to be the set of all k-bit integers, with input distribution being uniform over it, will also work (if k-bit primes distribution works)

One-Way Function Candidates

- Integer factorization:
  - $f_{mult}(x,y) = x \cdot y$
  - Input distribution: (x,y) random k-bit primes
  - Fact: taking input domain to be the set of all k-bit integers, with input distribution being uniform over it, will also work (if k-bit primes distribution works)
    - Important that we require $|x|=|y|=k$, not $|x \cdot y|=k$ (otherwise, 2 is a valid 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:
  - $f_{subsum}(x_1 \ldots x_k, S) = (x_1 \ldots x_k, \sum_{i \in S} x_i)$

One-Way Function Candidates

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

One-Way Function Candidates

- Solving Subset Sum:
  - $f_{subsum}(x_1 \ldots x_k,\ S) = (x_1 \ldots x_k,\ \sum_{i \in S} x_i\ )$
  - Input distribution: x_i k-bit integers, $S \subseteq \{1 \ldots k\}$. 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: n is part of the input and the output (i.e., n is "public"). This OWF can be used as a "OWF collection" indexed by n (many functions for the same k, using different 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: n is part of the input and the output (i.e., n is "public"). This OWF can be used as a "OWF collection" indexed by n (many functions for the same k, using different n)
- More: e.g, Discrete Logarithm (uses as index: a group & generator), RSA function (uses as index: n=pq & an exponent e).

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: n is part of the input and the output (i.e., n is "public"). This OWF can be used as a "OWF collection" indexed by n (many functions for the same k, using different n)
- More: e.g, Discrete Logarithm (uses as index: a group & generator), RSA function (uses as index: n=pq & an exponent e).
  - 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 <u>any</u> OWF f, can slightly modify it to get a OWF g_f such that

Goldreich-Levin Predicate

- Given <u>any</u> 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 <u>any</u> 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 <u>any</u> 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 <u>any</u> 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 <u>any</u> 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

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
    - Predictor for B(x,r) is a "noisy channel" through which x, encoded as $(\langle x,0\rangle,\langle x,1\rangle \ldots \langle x,2^{|x|}-1\rangle)$ (Walsh-Hadamard code), is transmitted. Can recover x by error-correction (local list decoding)

RECALL

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

Computational Indistinguishability

Computational Indistinguishability

- Distribution ensemble: A sequence of distributions (typically on a growing sample-space) indexed by k. Denoted $\{X_k\}$

Computational Indistinguishability

- Distribution ensemble: A sequence of distributions (typically on a growing sample-space) indexed by k. Denoted $\{X_k\}$
  - E.g., ciphertext distributions, indexed by security parameter

Computational Indistinguishability

- Distribution ensemble: A sequence of distributions (typically on a growing sample-space) indexed by k. Denoted $\{X_k\}$
  - E.g., ciphertext distributions, indexed by security parameter
- Two distribution ensembles $\{X_k\}$ and $\{X'_k\}$ are said to be computationally indistinguishable if

Computational Indistinguishability

- Distribution ensemble: A sequence of distributions (typically on a growing sample-space) indexed by k. Denoted $\{X_k\}$
  - E.g., ciphertext distributions, indexed by security parameter
- Two distribution ensembles $\{X_k\}$ and $\{X'_k\}$ are said to be computationally indistinguishable if
  - $\exists$ negligible $\nu(k)$ such that $\forall$ (non-uniform) PPT distinguisher D

Computational Indistinguishability

- Distribution ensemble: A sequence of distributions (typically on a growing sample-space) indexed by k. Denoted $\{X_k\}$
  - E.g., ciphertext distributions, indexed by security parameter
- Two distribution ensembles $\{X_k\}$ and $\{X'_k\}$ are said to be computationally indistinguishable if
  - $\exists$ negligible $\nu(k)$ such that $\forall$ (non-uniform) PPT distinguisher D
    - $| \Pr_{x \leftarrow X_k}[D(x)=1] - \Pr_{x \leftarrow X'_k}[D(x)=1] | \leq \nu(k)$

Computational Indistinguishability

- Distribution ensemble: A sequence of distributions (typically on a growing sample-space) indexed by k. Denoted $\{X_k\}$
  - E.g., ciphertext distributions, indexed by security parameter
- Two distribution ensembles $\{X_k\}$ and $\{X'_k\}$ are said to be computationally indistinguishable if
  - $\exists$ negligible $\nu(k)$ such that $\forall$ (non-uniform) PPT distinguisher D
    - $| \Pr_{x \leftarrow X_k}[D(x)=1] - \Pr_{x \leftarrow X'_k}[D(x)=1] | \leq \nu(k)$
- $\Delta_{PPT}(X_k, X'_k) :=$ "max"$_{PPT\ D}$ $| \Pr_{x \leftarrow X_k}[D(x)=1] - \Pr_{x \leftarrow X'_k}[D(x)=1] |$

Computational Indistinguishability

- Distribution ensemble: A sequence of distributions (typically on a growing sample-space) indexed by k. Denoted $\{X_k\}$
  - E.g., ciphertext distributions, indexed by security parameter
- Two distribution ensembles $\{X_k\}$ and $\{X'_k\}$ are said to be computationally indistinguishable if

$\Delta_{PPT}(X_k,X'_k) \leq \nu(k)$

  - $\exists$ negligible $\nu(k)$ such that $\forall$ (non-uniform) PPT distinguisher D
    - $| \Pr_{x\leftarrow X_k}[D(x)=1] - \Pr_{x\leftarrow X'_k}[D(x)=1] | \leq \nu(k)$
- $\Delta_{PPT}(X_k,X'_k) :=$ "max"$_{PPT\ D}$ $| \Pr_{x\leftarrow X_k}[D(x)=1] - \Pr_{x\leftarrow X'_k}[D(x)=1] |$

Computational Indistinguishability

Computational Indistinguishability

- Two distribution ensembles $\{X_k\}$ and $\{X'_k\}$ are said to be computationally indistinguishable if

Computational Indistinguishability

- Two distribution ensembles $\{X_k\}$ and $\{X'_k\}$ are said to be computationally indistinguishable if
  - $\exists$ negligible $\nu(k)$ such that $\Delta_{PPT}(X_k, X'_k) \leq \nu(k)$

Computational Indistinguishability

- Two distribution ensembles $\{X_k\}$ and $\{X'_k\}$ are said to be computationally indistinguishable if
  - $\exists$ negligible $\nu(k)$ such that $\Delta_{PPT}(X_k, X'_k) \leq \nu(k)$
  - $\Delta_{PPT}(X_k, X'_k) := \sup_{PPT\ D} \mid \Pr_{x \leftarrow X_k}[D(x)=1] - \Pr_{x \leftarrow X'_k}[D(x)=1] \mid$

Computational Indistinguishability

- Two distribution ensembles $\{X_k\}$ and $\{X'_k\}$ are said to be computationally indistinguishable if

  $X_k \approx X'_k$

  - $\exists$ negligible $\nu(k)$ such that $\Delta_{PPT}(X_k,X'_k) \leq \nu(k)$
  - $\Delta_{PPT}(X_k,X'_k) := \sup_{PPT\ D} \mid Pr_{x\leftarrow X_k}[D(x)=1] - Pr_{x\leftarrow X'_k}[D(x)=1] \mid$

Computational Indistinguishability

- Two distribution ensembles $\{X_k\}$ and $\{X'_k\}$ are said to be computationally indistinguishable if

  $X_k \approx X'_k$

  - $\exists$ negligible $\nu(k)$ such that $\Delta_{PPT}(X_k,X'_k) \leq \nu(k)$
  - $\Delta_{PPT}(X_k,X'_k) := \sup_{PPT\ D} \mid Pr_{x\leftarrow X_k}[D(x)=1] - Pr_{x\leftarrow X'_k}[D(x)=1] \mid$

- cf.: Two distribution ensembles $\{X_k\}$ and $\{X'_k\}$ are said to be statistically indistinguishable if $\Delta(X_k,X'_k) \leq \nu(k)$

Computational Indistinguishability

- Two distribution ensembles $\{X_k\}$ and $\{X'_k\}$ are said to be computationally indistinguishable if

  $X_k \approx X'_k$

  - $\exists$ negligible $\nu(k)$ such that $\Delta_{PPT}(X_k,X'_k) \leq \nu(k)$
  - $\Delta_{PPT}(X_k,X'_k) := \sup_{PPT\ D} | Pr_{x\leftarrow X_k}[D(x)=1] - Pr_{x\leftarrow X'_k}[D(x)=1] |$

- cf.: Two distribution ensembles $\{X_k\}$ and $\{X'_k\}$ are said to be statistically indistinguishable if $\Delta(X_k,X'_k) \leq \nu(k)$
  - $\Delta(X_k,X'_k) := \max_T | Pr_{x\leftarrow X_k}[T(x)=1] - Pr_{x\leftarrow X'_k}[T(x)=1] |$

Computational Indistinguishability

- Two distribution ensembles $\{X_k\}$ and $\{X'_k\}$ are said to be computationally indistinguishable if

$X_k \approx X'_k$

  - $\exists$ negligible $\nu(k)$ such that $\Delta_{PPT}(X_k,X'_k) \leq \nu(k)$
  - $\Delta_{PPT}(X_k,X'_k) := \sup_{PPT\ D} \mid Pr_{x \leftarrow X_k}[D(x)=1] - Pr_{x \leftarrow X'_k}[D(x)=1] \mid$
- cf.: Two distribution ensembles $\{X_k\}$ and $\{X'_k\}$ are said to be statistically indistinguishable if $\Delta(X_k,X'_k) \leq \nu(k)$
  - $\Delta(X_k,X'_k) := \max_T \mid Pr_{x \leftarrow X_k}[T(x)=1] - Pr_{x \leftarrow X'_k}[T(x)=1] \mid$
- If X_k, X'_k are short (say a single bit), $X_k \approx X'_k$ iff X_k, X'_k are statistically indistinguishable (Exercise)

Pseudorandomness Generator (PRG)

Pseudorandomness Generator (PRG)

- Takes a short seed and (deterministically) outputs a long string

Pseudorandomness Generator (PRG)

- Takes a short seed and (deterministically) outputs a long string
  - G_k: $\{0,1\}^k \rightarrow \{0,1\}^{n(k)}$ where $n(k) > k$

Pseudorandomness Generator (PRG)

- Takes a short seed and (deterministically) outputs a long string
  - $G_k: \{0,1\}^k \to \{0,1\}^{n(k)}$ where $n(k) > k$
- Security definition: Output distribution induced by random input seed should be "pseudorandom"

Pseudorandomness Generator (PRG)

- Takes a short seed and (deterministically) outputs a long string
  - $G_k: \{0,1\}^k \rightarrow \{0,1\}^{n(k)}$ where $n(k) > k$
- Security definition: Output distribution induced by random input seed should be "pseudorandom"
  - i.e., Computationally indistinguishable from uniformly random

Pseudorandomness Generator (PRG)

- Takes a short seed and (deterministically) outputs a long string
  - $G_k: \{0,1\}^k \rightarrow \{0,1\}^{n(k)}$ where $n(k) > k$
- Security definition: Output distribution induced by random input seed should be "pseudorandom"
  - i.e., Computationally indistinguishable from uniformly random
  - $\{G_k(x)\}_{x \leftarrow \{0,1\}^k} \approx U_{n(k)}$

Pseudorandomness Generator (PRG)

- Takes a short seed and (deterministically) outputs a long string
    - $G_k: \{0,1\}^k \rightarrow \{0,1\}^{n(k)}$ where $n(k) > k$
- Security definition: Output distribution induced by random input seed should be "pseudorandom"
    - i.e., Computationally indistinguishable from uniformly random
    - $\{G_k(x)\}_{x \leftarrow \{0,1\}^k} \approx U_{n(k)}$
    - Note: $\{G_k(x)\}_{x \leftarrow \{0,1\}^k}$ cannot be statistically indistinguishable from $U_{n(k)}$ unless $n(k) \leq k$ (Exercise)

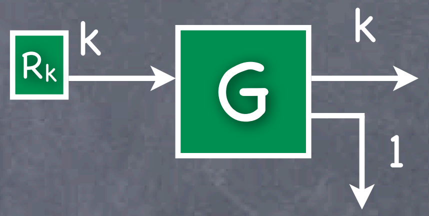
PRG from One-Way Permutations

PRG from One-Way Permutations

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

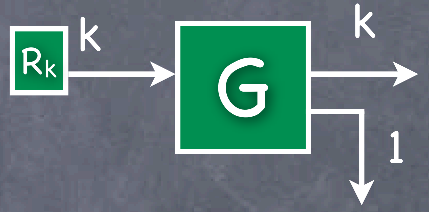
PRG from One-Way Permutations

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



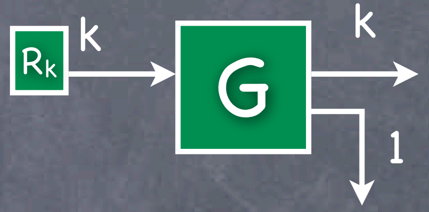
PRG from One-Way Permutations

- 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

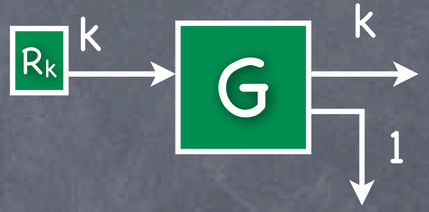
- One-bit stretch PRG, G_k: $\{0,1\}^k \rightarrow \{0,1\}^{k+1}$



  - $G(x) = f(x) \circ B(x)$
  - Where f: $\{0,1\}^k \rightarrow \{0,1\}^k$ is a one-way <u>permutation</u>, and B a hardcore predicate for f

PRG from One-Way Permutations

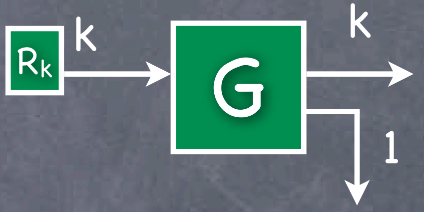
- One-bit stretch PRG, G_k: $\{0,1\}^k \to \{0,1\}^{k+1}$



  - $G(x) = f(x) \circ B(x)$
  - Where f: $\{0,1\}^k \to \{0,1\}^k$ is a one-way <u>permutation</u>, 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

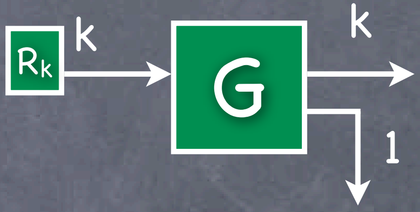
- One-bit stretch PRG, G_k: $\{0,1\}^k \rightarrow \{0,1\}^{k+1}$



  - $G(x) = f(x) \circ B(x)$
  - 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

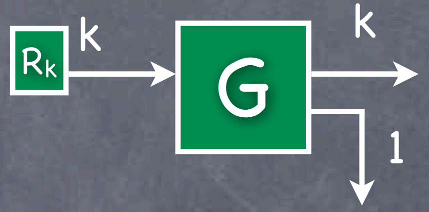
- One-bit stretch PRG, $G_k: \{0,1\}^k \to \{0,1\}^{k+1}$



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

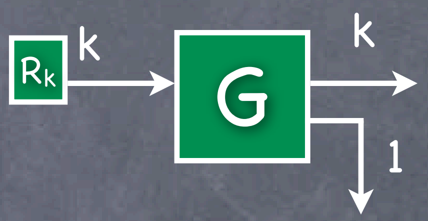
PRG from One-Way Permutations

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



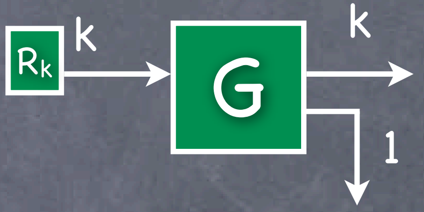
PRG from One-Way Permutations

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



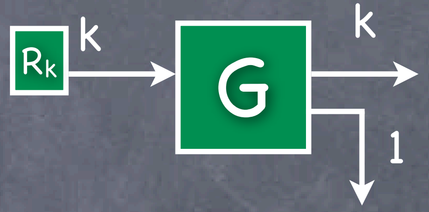
PRG from One-Way Permutations

- 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

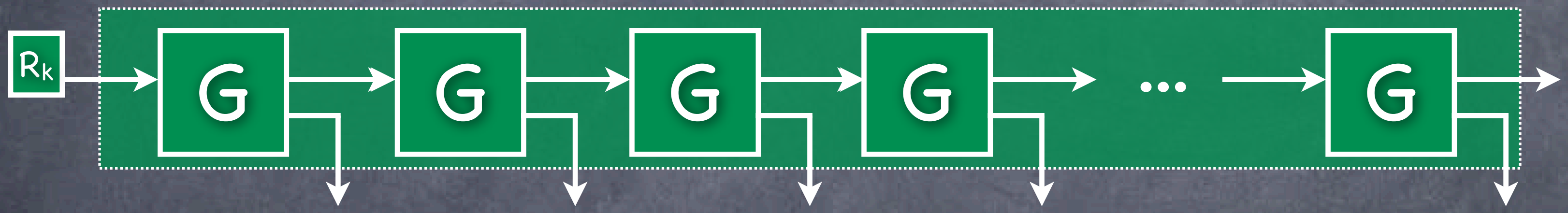


PRG from One-Way Permutations

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

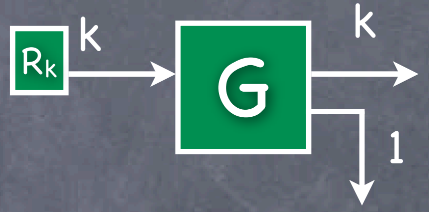


- Increasing the stretch
  - Can use part of the PRG output as a new seed

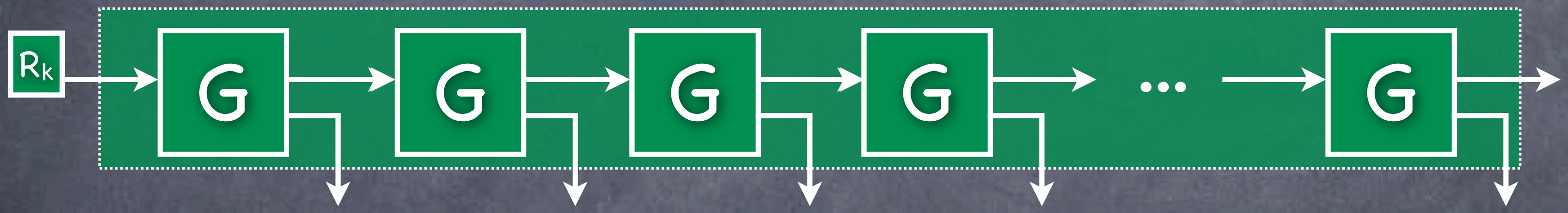


PRG from One-Way Permutations

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



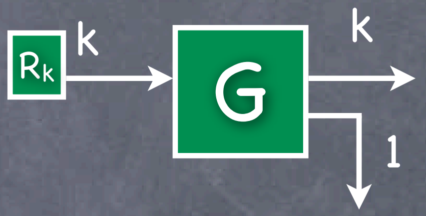
- Increasing the stretch
  - Can use part of the PRG output as a new seed



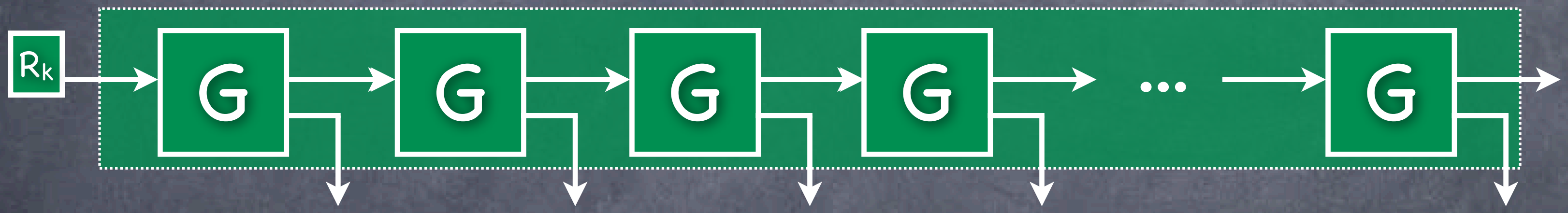
  - If the intermediate seeds are never output, can keep stretching on demand (for any "polynomial length")

PRG from One-Way Permutations

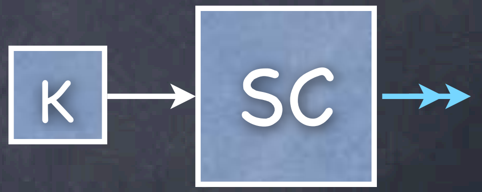
- One-bit stretch PRG, G_k: $\{0,1\}^k \rightarrow \{0,1\}^{k+1}$



- 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

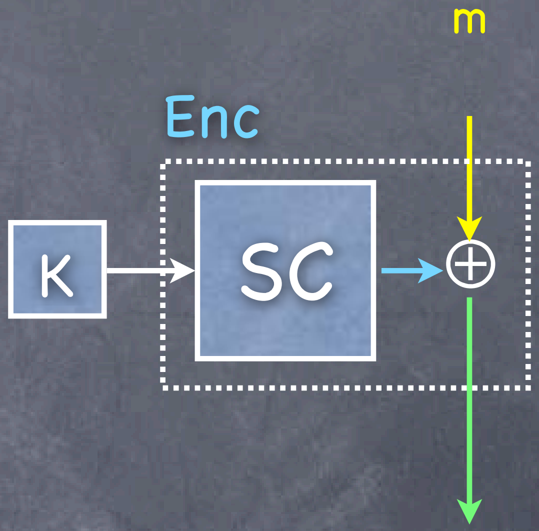
One-time CPA-secure SKE with a Stream-Cipher

- 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

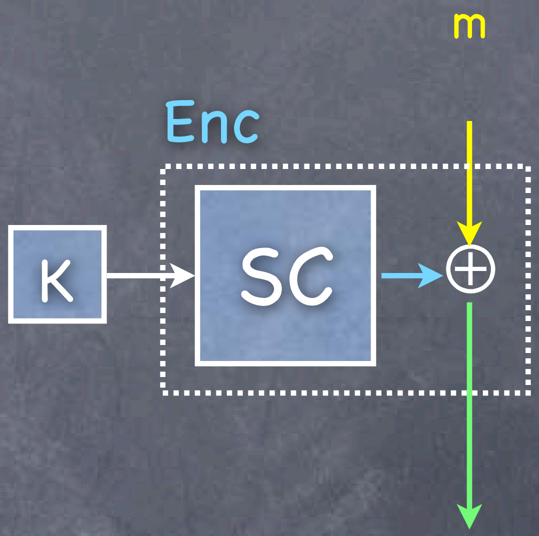
One-time CPA-secure SKE with a Stream-Cipher



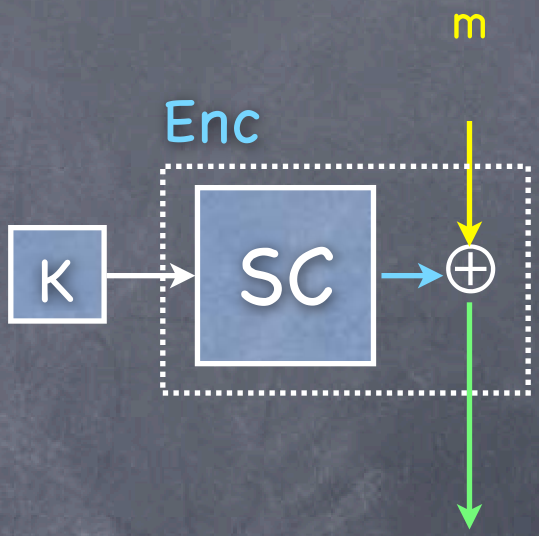
- One-time Encryption with a stream-cipher:
  - Generate a one-time pad from a short seed
  - Can share just the seed as the key
  - Mask message with the pseudorandom pad

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
- Decryption is symmetric: plaintext & ciphertext interchanged

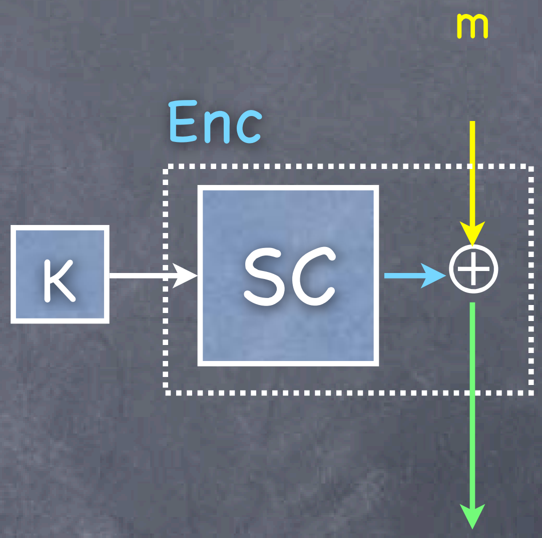


One-time CPA-secure SKE with a Stream-Cipher



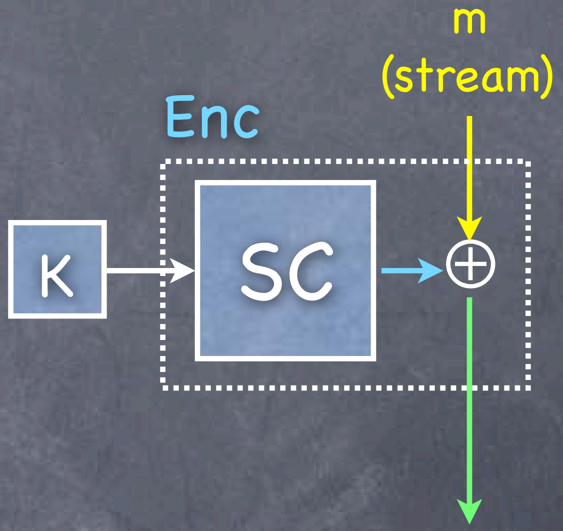
- One-time Encryption with a stream-cipher:
  - Generate a one-time pad from a short seed
  - Can share just the seed as the key
  - Mask message with the pseudorandom pad
- Decryption is symmetric: plaintext & ciphertext interchanged
- SC can spit out bits on demand, so the message can arrive bit by bit, and the length of the message doesn't have to be a priori fixed

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



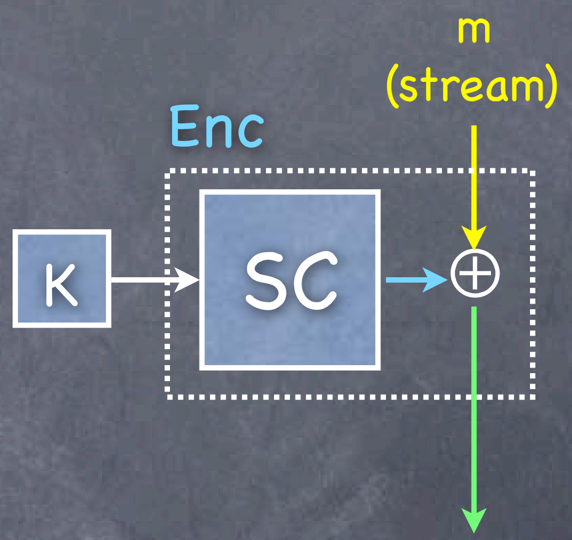
- One-time Encryption with a stream-cipher:
  - Generate a one-time pad from a short seed
  - Can share just the seed as the key
  - Mask message with the pseudorandom pad
- Decryption is symmetric: plaintext & ciphertext interchanged
- SC can spit out bits on demand, so the message can arrive bit by bit, and the length of the message doesn't have to be a priori fixed
- Security: indistinguishability from using a truly random pad

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



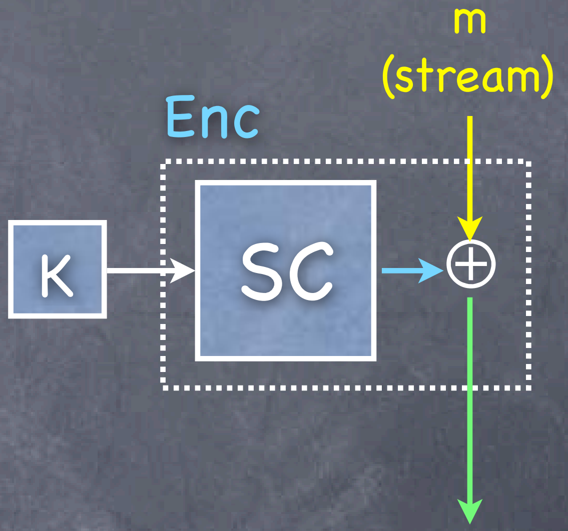
- One-time Encryption with a stream-cipher:
  - Generate a one-time pad from a short seed
  - Can share just the seed as the key
  - Mask message with the pseudorandom pad
- Decryption is symmetric: plaintext & ciphertext interchanged
- SC can spit out bits on demand, so the message can arrive bit by bit, and the length of the message doesn't have to be a priori fixed
- Security: indistinguishability from using a truly random pad

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



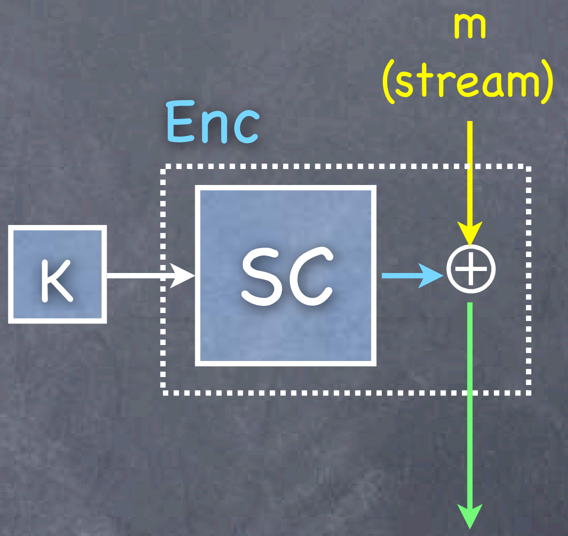
One-time CPA-secure SKE with a Stream-Cipher

- In IDEAL experiment, consider simulator that uses a truly random string as the ciphertext



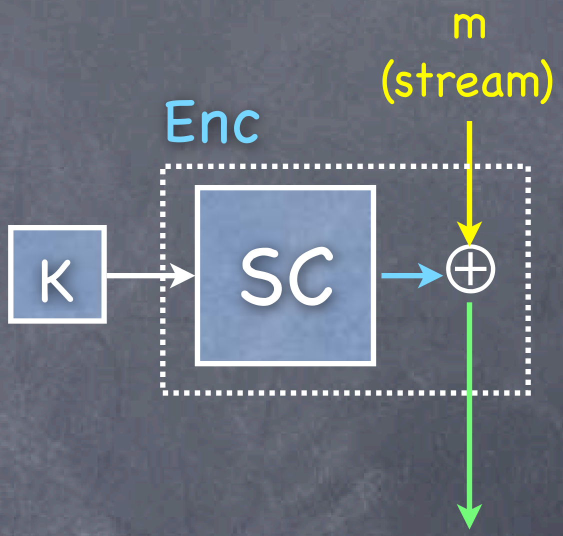
One-time CPA-secure SKE with a Stream-Cipher

- In IDEAL experiment, consider simulator that uses a truly random string as the ciphertext
- To show REAL ≈ IDEAL

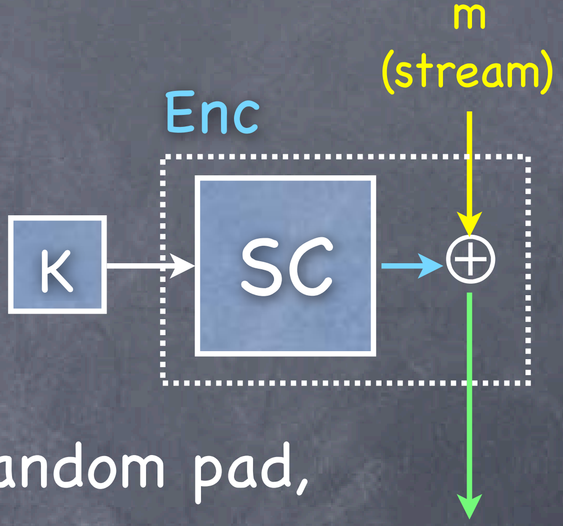


One-time CPA-secure SKE with a Stream-Cipher

- In IDEAL experiment, consider simulator that uses a truly random string as the ciphertext
- To show REAL ≈ IDEAL
- Consider an intermediate world, HYBRID:

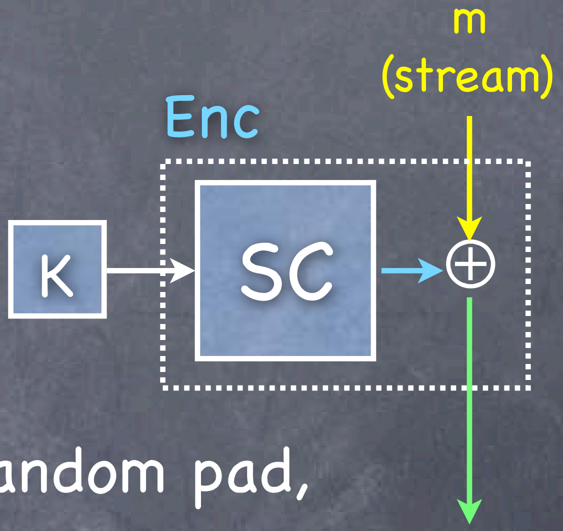


One-time CPA-secure SKE with a Stream-Cipher



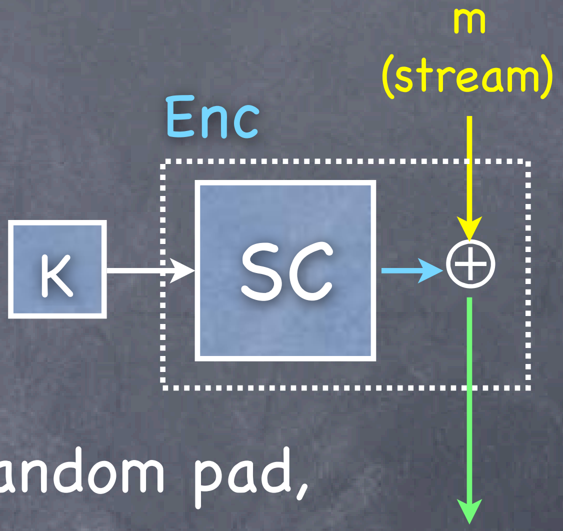
- In IDEAL experiment, consider simulator that uses a truly random string as the ciphertext
- To show REAL ≈ IDEAL
- Consider an intermediate world, HYBRID:
  - Like REAL, but Enc/Dec use a (long) truly random pad, instead of the output from the stream-cipher

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



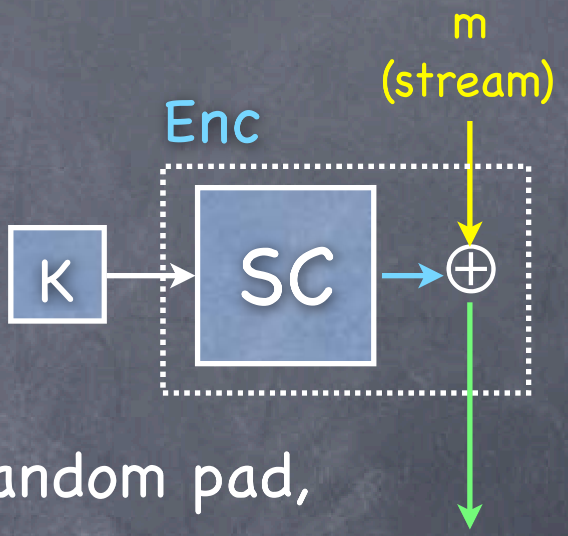
- In IDEAL experiment, consider simulator that uses a truly random string as the ciphertext
- To show REAL ≈ IDEAL
- Consider an intermediate world, HYBRID:
    - Like REAL, but Enc/Dec use a (long) truly random pad, instead of the output from the stream-cipher
    - HYBRID = IDEAL (recall perfect security of one-time pad)

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



- In IDEAL experiment, consider simulator that uses a truly random string as the ciphertext
- To show REAL ≈ IDEAL
- Consider an intermediate world, HYBRID:
  - Like REAL, but Enc/Dec use a (long) truly random pad, instead of the output from the stream-cipher
  - HYBRID = IDEAL (recall perfect security of one-time pad)
  - Claim: REAL ≈ HYBRID

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



- In IDEAL experiment, consider simulator that uses a truly random string as the ciphertext
- To show REAL ≈ IDEAL
- Consider an intermediate world, HYBRID:
  - Like REAL, but Enc/Dec use a (long) truly random pad, instead of the output from the stream-cipher
  - HYBRID = IDEAL (recall perfect security of one-time pad)
  - Claim: REAL ≈ HYBRID
    - Consider the experiments as a system that accepts the pad from outside (R' = SC(K) for a random K, or truly random R) and outputs the environment's output. This system is PPT, and so can't distinguish pseudorandom from random.

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.

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. (And, 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. (And, 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. (And, 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