Symmetric-Key Encryption: constructions

Lecture 4

PRG, Stream Cipher

Story So Far

Story So Far

- We defined (passive) security of Symmetric Key Encryption (SKE)

Story So Far

- We defined (passive) security of Symmetric Key Encryption (SKE)
  - **SIM-CPA = IND-CPA + approximate correctness**

Story So Far

- We defined (passive) security of Symmetric Key Encryption (SKE)
  - **SIM-CPA = IND-CPA + approximate correctness**
    - Exploits the restriction to PPT entities

Story So Far

- We defined (passive) security of Symmetric Key Encryption (SKE)
  - **SIM-CPA = IND-CPA + approximate correctness**
    - Exploits the restriction to PPT entities
    - Allows negligible advantage to the adversary

Story So Far

- We defined (passive) security of Symmetric Key Encryption (SKE)
  - **SIM-CPA** = **IND-CPA** + **approximate correctness**
    - Exploits the restriction to PPT entities
    - Allows negligible advantage to the adversary
- Today: Constructing SKE from Pseudorandomness

Story So Far

- We defined (passive) security of Symmetric Key Encryption (SKE)
  - **SIM-CPA = IND-CPA + approximate correctness**
    - Exploits the restriction to PPT entities
    - Allows negligible advantage to the adversary
- Today: Constructing SKE from Pseudorandomness
- Next time: Pseudorandomness ← One-Way Permutations

Constructing SKE schemes

Constructing SKE schemes

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

Constructing SKE schemes

- Basic idea: "stretchable" 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: "stretchable" 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: "stretchable" 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

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
- First, 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
- First, PRG with fixed stretch: $G_k: \{0,1\}^k \rightarrow \{0,1\}^{n(k)}$, $n(k) > k$
- How does one define random-looking?

Pseudorandomness Generator (PRG)

- Expand a short random **seed** to a "random-looking" string
- First, PRG with fixed stretch: $G_k: \{0,1\}^k \rightarrow \{0,1\}^{n(k)}$, $n(k) > k$
- How does one define 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
- First, PRG with fixed stretch: $G_k: \{0,1\}^k \rightarrow \{0,1\}^{n(k)}$, $n(k) > k$
- How does one define 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
- First, PRG with fixed stretch: $G_k: \{0,1\}^k \rightarrow \{0,1\}^{n(k)}$, $n(k) > k$
- How does one define 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
- First, PRG with fixed stretch: $G_k: \{0,1\}^k \rightarrow \{0,1\}^{n(k)}$, $n(k) > k$
- How does one define 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:
    - 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!**

Pseudorandomness Generator (PRG)

- Expand a short random **seed** to a "random-looking" string
- First, PRG with fixed stretch: $G_k: \{0,1\}^k \rightarrow \{0,1\}^{n(k)}$, $n(k) > k$
- How does one define 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
  - $\forall$ (non-uniform) PPT distinguisher D, $\exists$ negligible $\nu(k)$ such that

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

Computational Indistinguishability

- Two distribution ensembles $\{X_k\}$ and $\{X'_k\}$ are said to be **computationally indistinguishable** if
  - $\forall$ (non-uniform) PPT distinguisher D, $\exists$ negligible $\nu(k)$ such that $|\ \Pr_{x \leftarrow X_k}[D(x)=1] - \Pr_{x \leftarrow X'_k}[D(x)=1]\ | \leq \nu(k)$

Computational Indistinguishability

- Two distribution ensembles $\{X_k\}$ and $\{X'_k\}$ are said to be **computationally indistinguishable** if
  - $\forall$ (non-uniform) PPT distinguisher D, $\exists$ negligible $\nu(k)$ such that $| \Pr_{x \leftarrow X_k}[D(x)=1] - \Pr_{x \leftarrow X'_k}[D(x)=1] | \leq \nu(k)$

$X_k \approx X'_k$

Computational Indistinguishability

- Two distribution ensembles $\{X_k\}$ and $\{X'_k\}$ are said to be **computationally indistinguishable** if
  - $\forall$ (non-uniform) PPT distinguisher D, $\exists$ negligible $\nu(k)$ such that $| \Pr_{x \leftarrow X_k}[D(x)=1] - \Pr_{x \leftarrow X'_k}[D(x)=1] | \leq \nu(k)$

$X_k \approx X'_k$

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

Computational Indistinguishability

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

  $X_k \approx X'_k$

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

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

  - Can rewrite as, $\exists$ negligible $\nu(k)$ s.t. $\Delta(X_k,X'_k) \leq \nu(k)$ where $\Delta(X_k,X'_k) := \max_T | \Pr_{x \leftarrow X_k}[T(x)=1] - \Pr_{x \leftarrow X'_k}[T(x)=1] |$

Computational Indistinguishability

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

  $X_k \approx X'_k$

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

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

  - Can rewrite as, $\exists$ negligible $\nu(k)$ s.t. $\Delta(X_k,X'_k) \leq \nu(k)$ where $\Delta(X_k,X'_k) := \max_T | \Pr_{x \leftarrow X_k}[T(x)=1] - \Pr_{x \leftarrow X'_k}[T(x)=1] |$

- 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
  - $G_k: \{0,1\}^k \rightarrow \{0,1\}^{n(k)}$ 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)

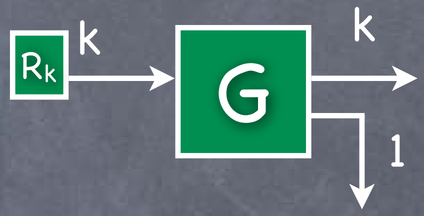
Pseudorandomness Generator (PRG)

- 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)
    - i.e., no PRG against unbounded adversaries

PRG from One-Way Permutations

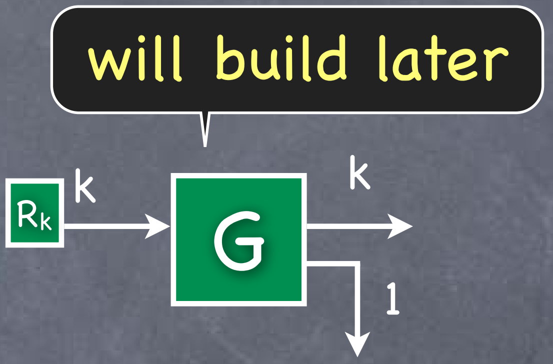
PRG from One-Way Permutations

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



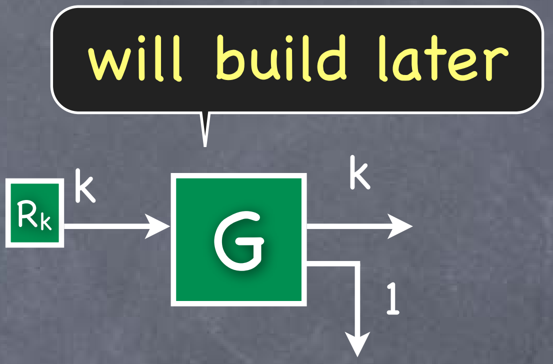
PRG from One-Way Permutations

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



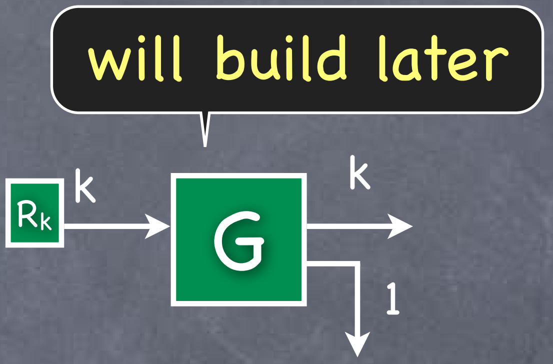
PRG from One-Way Permutations

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

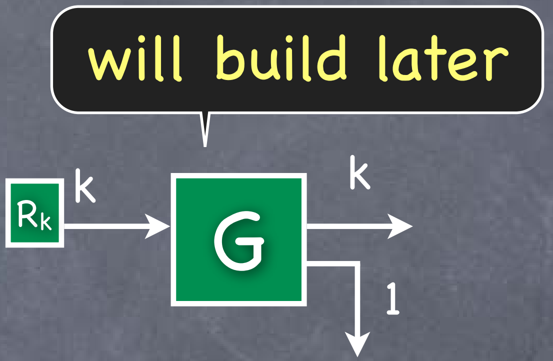


PRG from One-Way Permutations

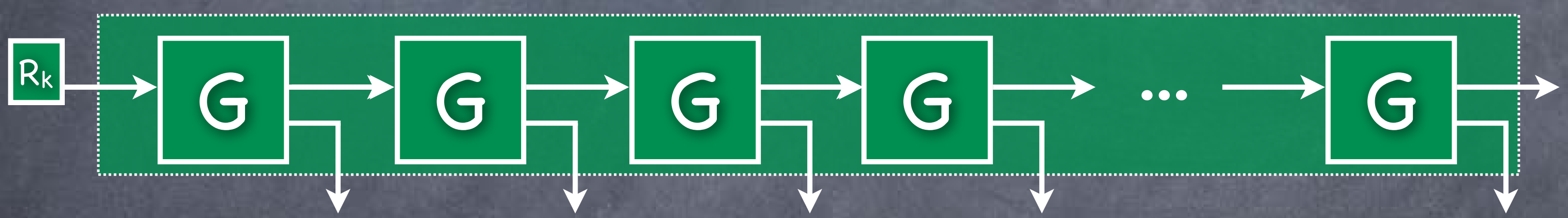
- One-bit stretch PRG, $G_k: \{0,1\}^k \rightarrow \{0,1\}^{k+1}$
- Increasing the stretch
  - Can use part of the PRG output as a new seed



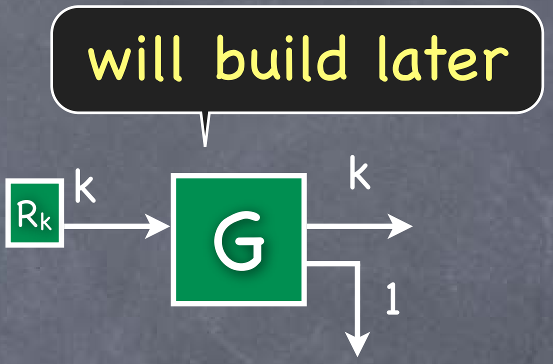
PRG from One-Way Permutations



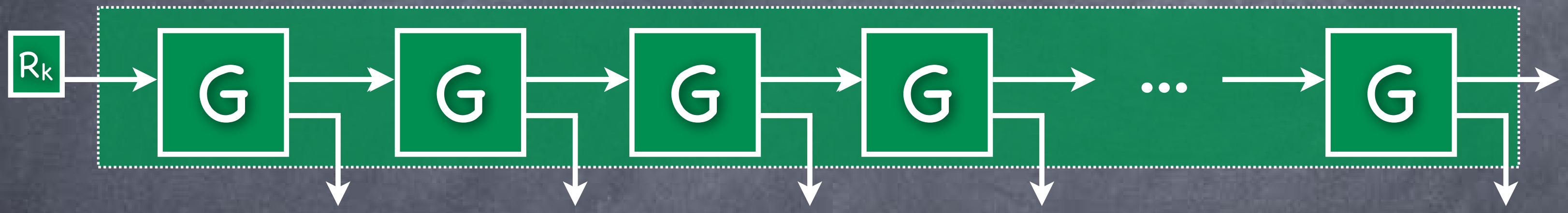
- One-bit stretch PRG, G_k: $\{0,1\}^k \rightarrow \{0,1\}^{k+1}$
- Increasing the stretch
  - Can use part of the PRG output as a new seed



PRG from One-Way Permutations

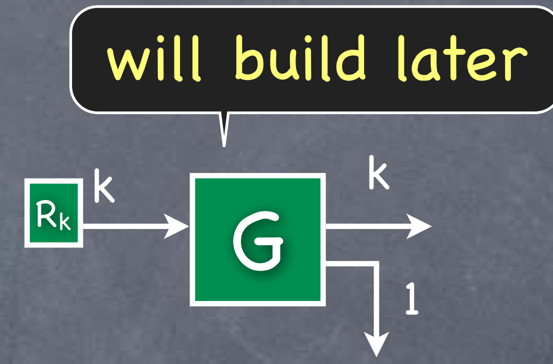


- One-bit stretch PRG, G_k: $\{0,1\}^k \rightarrow \{0,1\}^{k+1}$
- Increasing the stretch
  - Can use part of the PRG output as a new seed

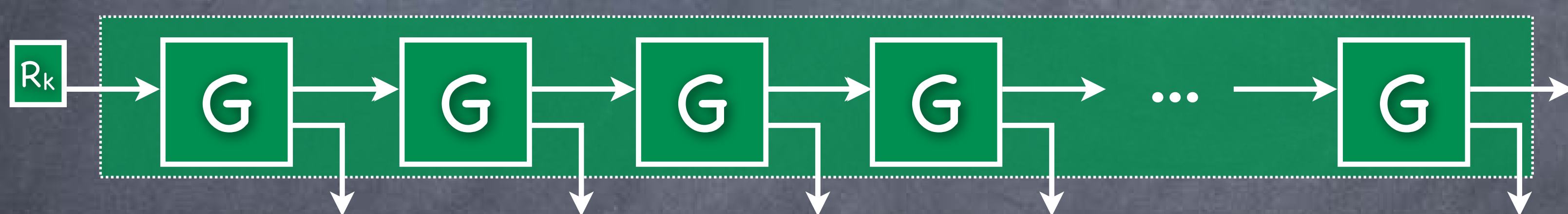


  - If the intermediate seeds are never output, can keep stretching on demand (for any "polynomial length")

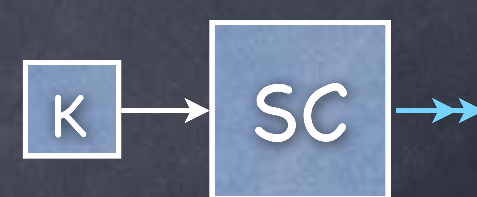
PRG from One-Way Permutations



- 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

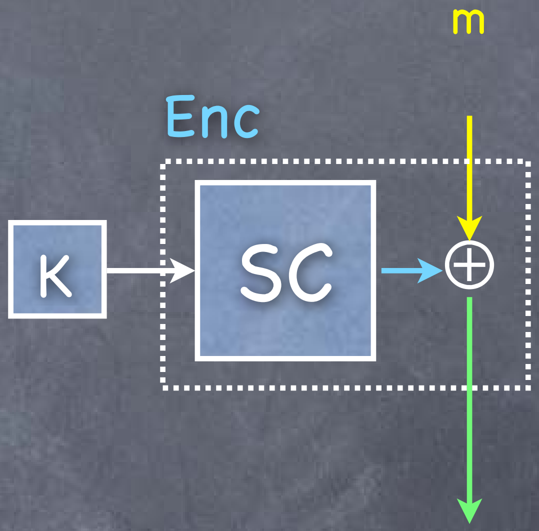
- One-time Encryption with a stream-cipher:
  - Generate a one-time pad from a short seed
  - Can share just the seed as the key

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

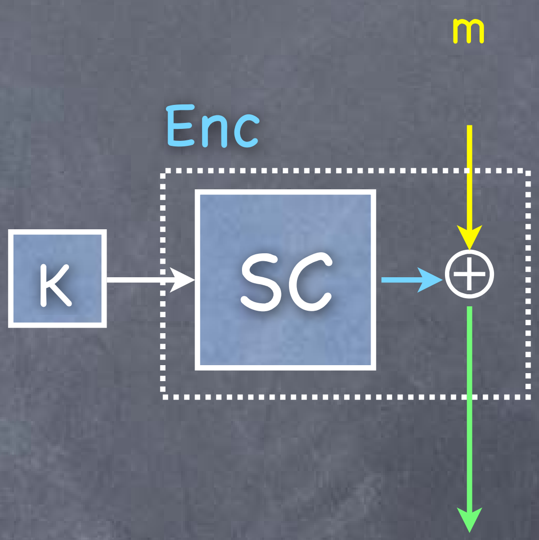
- One-time Encryption with a stream-cipher:
  - Generate a one-time pad from a short seed
  - Can share just the seed as the key
  - Mask message with the pseudorandom pad

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

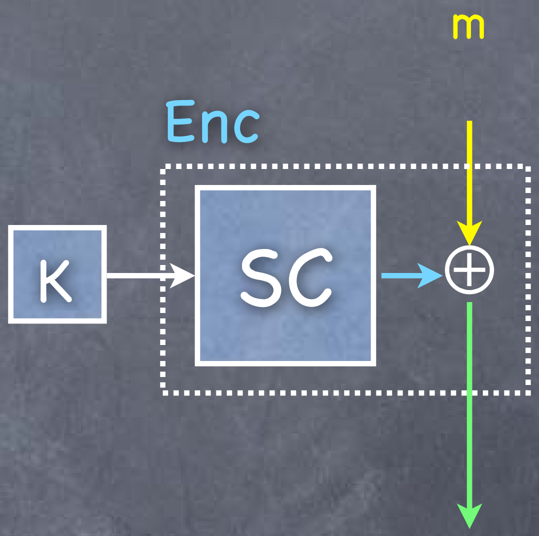


One-time CPA-secure SKE with a Stream-Cipher



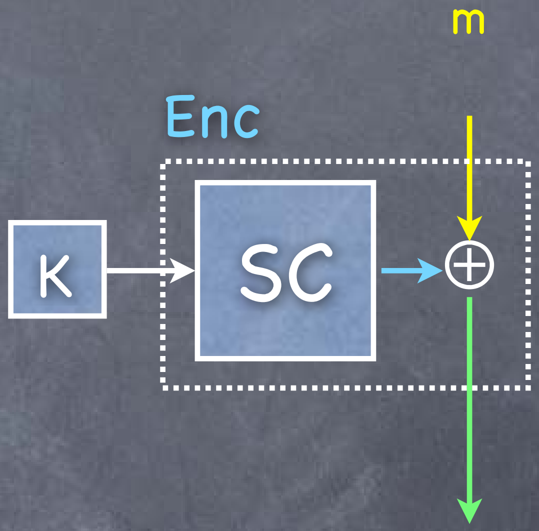
- One-time Encryption with a stream-cipher:
  - Generate a one-time pad from a short seed
  - Can share just the seed as the key
  - Mask message with the pseudorandom pad
- Decryption is symmetric: plaintext & ciphertext interchanged

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



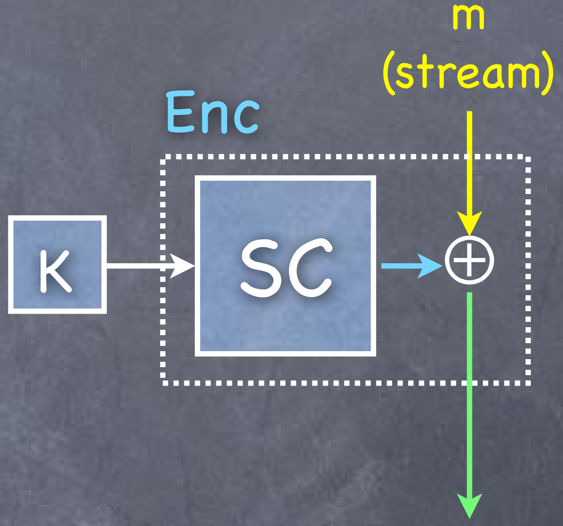
- One-time Encryption with a stream-cipher:
  - Generate a one-time pad from a short seed
  - Can share just the seed as the key
  - Mask message with the pseudorandom pad
- Decryption is symmetric: plaintext & ciphertext interchanged
- SC can spit out bits on demand, so the message can arrive bit by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

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



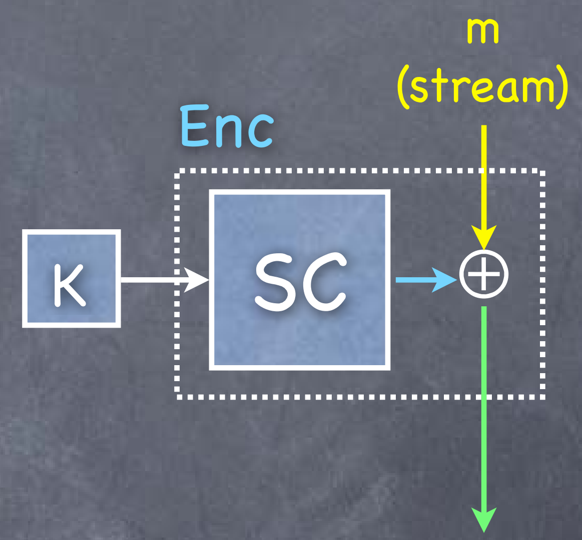
- One-time Encryption with a stream-cipher:
  - Generate a one-time pad from a short seed
  - Can share just the seed as the key
  - Mask message with the pseudorandom pad
- Decryption is symmetric: plaintext & ciphertext interchanged
- SC can spit out bits on demand, so the message can arrive bit by 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



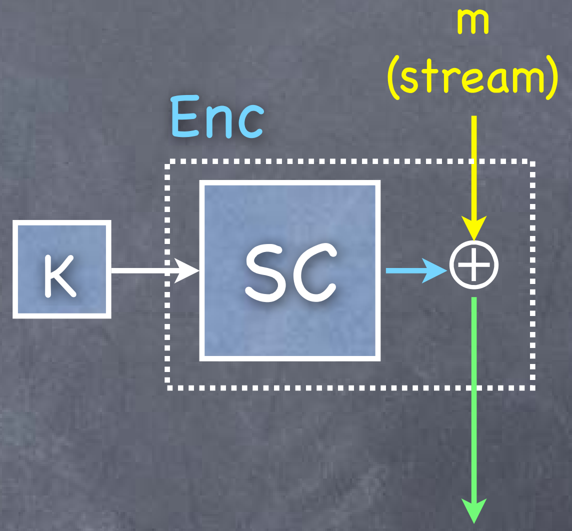
- One-time Encryption with a stream-cipher:
  - Generate a one-time pad from a short seed
  - Can share just the seed as the key
  - Mask message with the pseudorandom pad
- Decryption is symmetric: plaintext & ciphertext interchanged
- SC can spit out bits on demand, so the message can arrive bit by 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



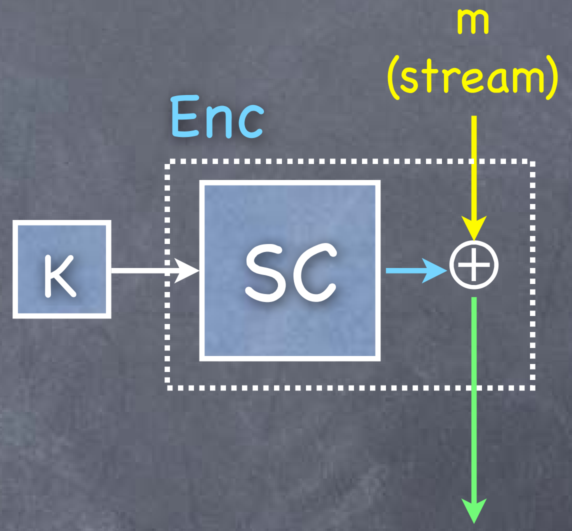
One-time CPA-secure SKE with a Stream-Cipher

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



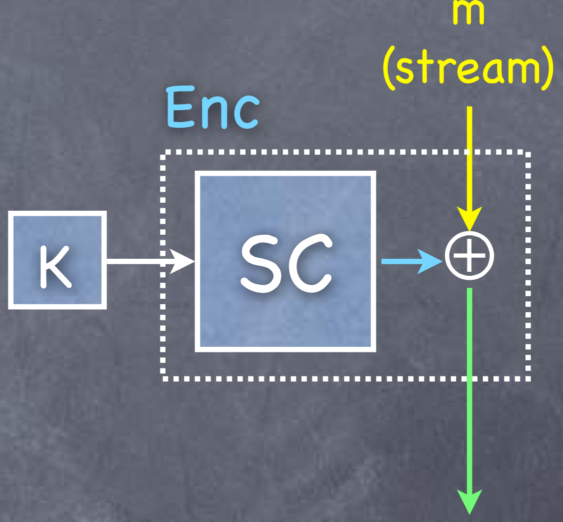
One-time CPA-secure SKE with a Stream-Cipher

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



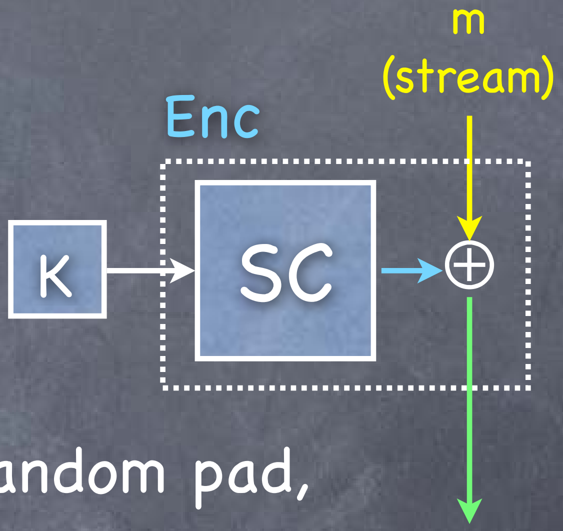
One-time CPA-secure SKE with a Stream-Cipher

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

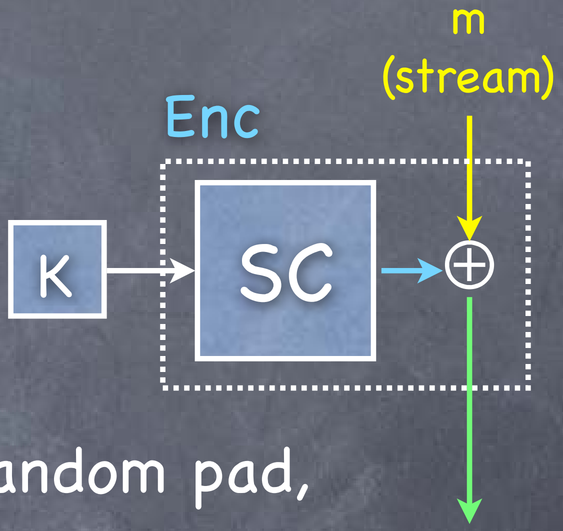


One-time CPA-secure SKE with a Stream-Cipher

- 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

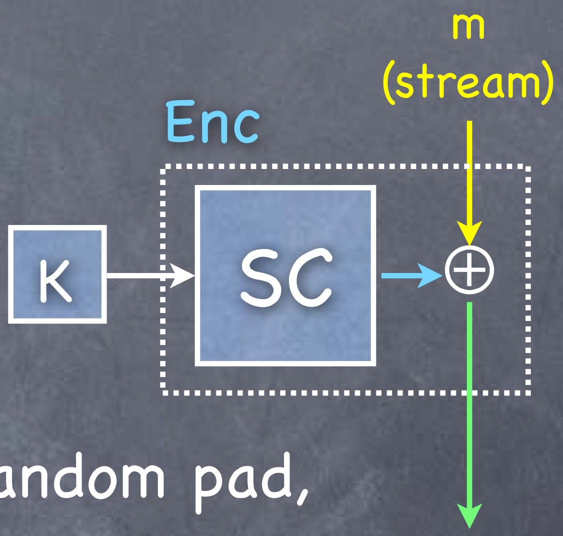


One-time CPA-secure SKE with a Stream-Cipher



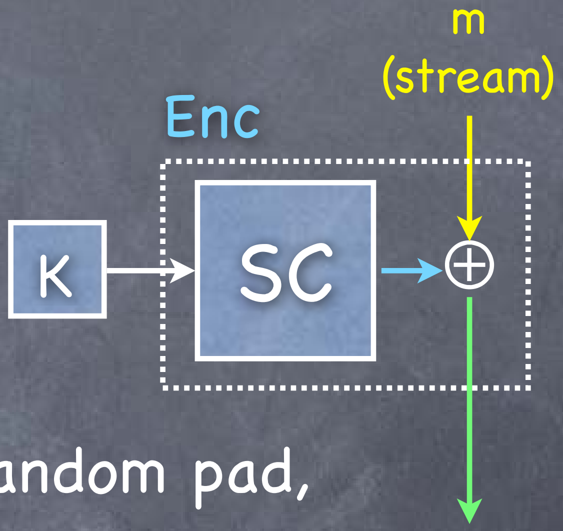
- In IDEAL experiment, consider simulator that uses a truly random string as the ciphertext
- To show REAL ≈ IDEAL
- Consider an intermediate world, HYBRID:
  - Like REAL, but Enc/Dec use a (long) truly random pad, instead of the output from the stream-cipher
  - HYBRID = IDEAL (recall perfect security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