Symmetric-Key Encryption: constructions

Lecture 5
PRG from One-Way Permutations
PRF, Block Cipher
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-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-Way Function, Hardcore Predicate
One-Way Function, Hardcore Predicate

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

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

$f$ is polynomial time computable
One-Way Function, Hardcore Predicate

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

- $f$ is polynomial time computable
- For all (non-uniform) PPT adversary, probability of success in the “OWF experiment” is negligible
One-Way Function, Hardcore Predicate

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

- \( f \) is polynomial time computable
- For all (non-uniform) PPT adversary, probability of success in the “OWF experiment” is negligible.

\( x \leftarrow \{0,1\}^k \)

\( f(x) \)
One-Way Function, Hardcore Predicate

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

- \( f \) is polynomial time computable
- For all (non-uniform) PPT adversary, probability of success in the "OWF experiment" is negligible
One-Way Function, Hardcore Predicate

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

- \( f \) is polynomial time computable
- For all (non-uniform) PPT adversary, probability of success in the "OWF experiment" is negligible
- But \( x \) may not be completely hidden by \( f(x) \)
One-Way Function, Hardcore Predicate

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

B is a hardcore predicate of a OWF \( f \) if
- \( x \leftarrow \{0,1\}^k \)
- \( f(x') = f(x) \)?
One-Way Function, Hardcore Predicate

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

  - \(f\) is polynomial time computable
  - For all (non-uniform) PPT adversary, probability of success in the "OWF experiment" is negligible
  - But \(x\) may not be completely hidden by \(f(x)\)

- \(B\) is a **hardcore predicate** of a OWF \(f\) if

  - \(B\) is polynomial time computable
One-Way Function, Hardcore Predicate

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

- \( f \) is polynomial time computable
- For all (non-uniform) PPT adversary, probability of success in the “OWF experiment” is negligible
- But \( x \) may not be completely hidden by \( f(x) \)

\( B \) is a **hardcore predicate** of a OWF \( f \) if

- \( B \) is polynomial time computable
- For all (non-uniform) PPT adversary, **advantage** over random prediction in the Hardcore-predicate experiment is negligible
One-Way Function, Hardcore Predicate

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

- \( f \) is polynomial time computable
- For all (non-uniform) PPT adversary, probability of success in the “OWF experiment” is negligible
- But \( x \) may not be completely hidden by \( f(x) \)

\( B \) is a hardcore predicate of a OWF \( f \) if

- \( B \) is polynomial time computable
- For all (non-uniform) PPT adversary, advantage over random prediction in the Hardcore-predicate experiment is negligible
One-Way Function, Hardcore Predicate

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

- $B$ is a hardcore predicate of a OWF $f$ if
  - $B$ is polynomial time computable
  - For all (non-uniform) PPT adversary, advantage over random prediction in the Hardcore-predicate experiment is negligible
One-Way Function, Hardcore Predicate

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

- \( f \) is polynomial time computable
- For all (non-uniform) PPT adversary, probability of success in the "OWF experiment" is negligible
- But \( x \) may not be completely hidden by \( f(x) \)

\( B \) is a **hardcore predicate** of a OWF \( f \) if

- \( B \) is polynomial time computable
- For all (non-uniform) PPT adversary, **advantage** over random prediction 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:

\[ f_{\text{mult}}(x, y) = x \cdot y \]
One-Way Function Candidates

- Integer factorization:
  - $f_{\text{mult}}(x,y) = x \cdot y$

- Input distribution: $(x,y)$ random k-bit primes
One-Way Function Candidates

- Integer factorization:
  \[ f_{\text{mult}}(x,y) = x \cdot y \]

- 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:
  \[ f_{\text{mult}}(x,y) = x \cdot y \]

- 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 just \(|x \cdot y|=2k\)
  (otherwise, 2 is a valid factor of \(x \cdot y\) with \(3/4\) probability)
One-Way Function Candidates
One-Way Function Candidates

Solving Subset Sum:
One-Way Function Candidates

Solving Subset Sum:

\[ f_{\text{subsum}}(x_1 \ldots x_k, S) = (x_1 \ldots x_k, \sum_{i \in S} x_i) \]
One-Way Function Candidates

Solving Subset Sum:

\[ f_{\text{subsum}}(x_1...x_k, S) = (x_1...x_k, \sum_{i \in S} x_i) \]

Input distribution: \( x_i \) k-bit integers, \( S \subseteq \{1...k\} \). Uniform
One-Way Function Candidates

Solving Subset Sum:

\[ f_{\text{subsum}}(x_1...x_k, S) = (x_1...x_k, \sum_{i \in S} x_i) \]

Input distribution: \( x_i \) k-bit integers, \( S \subseteq \{1...k\} \). Uniform

Inverting \( f_{\text{subsum}} \) known to be NP-complete, but assuming that it is a OWF is “stronger” than assuming \( P \neq NP \)
One-Way Function Candidates

Solving Subset Sum:

\[ f_{\text{subsum}}(x_1...x_k, S) = (x_1...x_k, \sum_{i \in S} x_i) \]

Input distribution: \( x_i \) k-bit integers, \( S \subseteq \{1...k\} \). Uniform

Inverting \( f_{\text{subsum}} \) known to be NP-complete, but assuming that it is a OWF is “stronger” than assuming \( P \neq NP \)

Note: \((x_1,...,x_k)\) is “public” (given as part of the output to be inverted)
One-Way Function Candidates

Solving Subset Sum:

\[ f_{\text{subsum}}(x_1...x_k, S) = (x_1...x_k, \sum_{i \in S} x_i) \]

Input distribution: \( x_i \) k-bit integers, \( S \subseteq \{1...k\} \). Uniform

Inverting \( f_{\text{subsum}} \) known to be NP-complete, but assuming that it is a OWF is “stronger” than assuming \( P \neq NP \)

Note: \((x_1,...,x_k)\) is “public” (given as part of the output to be inverted)

**OWF Collection:** A collection of subset sum problems, all with the same \((x_1,...,x_k)\) (and independent \(S\))
One-Way Function
Candidates
One-Way Function Candidates

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

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

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

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

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

For candidate OWFs, often hardcore predicates known
Hardcore Predicates

For candidate OWFs, often hardcore predicates known

e.g. if \( f_{\text{Rabin}}(x;n) \) is a OWF, then \( \text{LSB}(x) \) is a hardcore predicate for it
Hardcore Predicates

For candidate OWFs, often hardcore predicates known

e.g. if $f_{Rabin}(x;n)$ is a OWF, then $\text{LSB}(x)$ is a hardcore predicate for it

**Reduction:** Given an algorithm for finding $\text{LSB}(x)$ from $f_{Rabin}(x;n)$ for random $x$, one can use it 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 $|r| = |x|$
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)
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 $f$
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

\[ g_f(x, r) = (f(x), r), \text{ where } |r| = |x| \]

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)
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, $G_K: \{0,1\}^k \rightarrow \{0,1\}^{k+1}$
- $G(x) = f(x) \circ B(x)$
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 $f$
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 $f$
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 $f$

Claim: $G$ is a PRG
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 $f$

Claim: $G$ is a PRG

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, $G_k: \{0,1\}^k \to \{0,1\}^{k+1}$

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

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

Claim: $G$ is a PRG

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 $f$

Claim: $G$ is a PRG

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
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
PRG Summary

- OWF, OWP, Hardcore predicates
PRG Summary

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

- 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.
PRG Summary

- 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.)
PRG Summary

- 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.
PRG Summary

- 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
Beyond One-Time
Beyond One-Time

- Need to make sure same part of the one-time pad is never reused
Beyond One-Time

Need to make sure same part of the one-time pad is never reused.

Sender and receiver will need to maintain state and stay in sync (indicating how much of the pad has already been used).
Beyond One-Time

- Need to make sure same part of the one-time pad is never reused

- Sender and receiver will need to maintain state and stay in sync (indicating how much of the pad has already been used)

- Or only sender maintains the index, but sends it to the receiver. Then receiver will need to run the stream-cipher to get to that index.
Beyond One-Time

Need to make sure same part of the one-time pad is never reused

Sender and receiver will need to maintain state and stay in sync (indicating how much of the pad has already been used)

Or only sender maintains the index, but sends it to the receiver. Then receiver will need to run the stream-cipher to get to that index.

A PRG with direct access to any part of the output stream?
Beyond One-Time

Need to make sure same part of the one-time pad is never reused

Sender and receiver will need to maintain state and stay in sync (indicating how much of the pad has already been used)

Or only sender maintains the index, but sends it to the receiver. Then receiver will need to run the stream-cipher to get to that index.

A PRG with direct access to any part of the output stream?

Pseudo Random Function (PRF)
Pseudorandom Function (PRF)
Pseudorandom Function (PRF)

A compact representation of an exponentially long (pseudorandom) string
Pseudorandom Function (PRF)

- A compact representation of an exponentially long (pseudorandom) string
- Allows “random-access” (instead of just sequential access)
Pseudorandom Function (PRF)

- A compact representation of an exponentially long (pseudorandom) string
- Allows “random-access” (instead of just sequential access)
- A function $F(s; i)$ outputs the $i^{th}$ block of the pseudorandom string corresponding to seed $s$
Pseudorandom Function (PRF)

- A compact representation of an exponentially long (pseudorandom) string
- Allows “random-access” (instead of just sequential access)
- A function $F(s; i)$ outputs the $i^{th}$ block of the pseudorandom string corresponding to seed $s$
- Exponentially many blocks (i.e., large domain for $i$)
Pseudorandom Function (PRF)

- A compact representation of an exponentially long (pseudorandom) string
- Allows “random-access” (instead of just sequential access)
- A function $F(s; i)$ outputs the $i^{th}$ block of the pseudorandom string corresponding to seed $s$
- Exponentially many blocks (i.e., large domain for $i$)

Pseudorandom Function
Pseudorandom Function (PRF)

- A compact representation of an exponentially long (pseudorandom) string
  - Allows “random-access” (instead of just sequential access)
  - A function $F(s;i)$ outputs the $i^{th}$ block of the pseudorandom string corresponding to seed $s$
  - Exponentially many blocks (i.e., large domain for $i$)

Pseudorandom Function

- Need to define pseudorandomness for a function (not a string)
Pseudorandom Function (PRF)
Pseudorandom Function (PRF)

\[ F: \{0,1\}^k \times \{0,1\}^{m(k)} \rightarrow \{0,1\}^{n(k)} \text{ is a PRF if all PPT adversaries have negligible advantage in the PRF experiment} \]
Pseudorandom Function (PRF)

\( F: \{0,1\}^k \times \{0,1\}^{m(k)} \rightarrow \{0,1\}^{n(k)} \) is a PRF if all PPT adversaries have negligible advantage in the PRF experiment.

Adversary given oracle access to either \( F \) with a random seed, or a random function \( R: \{0,1\}^{m(k)} \rightarrow \{0,1\}^{n(k)} \). Needs to guess which.
Pseudorandom Function (PRF)

\[ F : \{0,1\}^k \times \{0,1\}^{m(k)} \rightarrow \{0,1\}^{n(k)} \] is a PRF if all PPT adversaries have negligible advantage in the PRF experiment.

Adversary given oracle access to either \( F \) with a random seed, or a random function \( R : \{0,1\}^{m(k)} \rightarrow \{0,1\}^{n(k)} \). Needs to guess which.
F: \(\{0,1\}^k \times \{0,1\}^{m(k)} \rightarrow \{0,1\}^{n(k)}\) is a PRF if all PPT adversaries have negligible advantage in the PRF experiment.

Adversary given oracle access to either F with a random seed, or a random function \(R: \{0,1\}^{m(k)} \rightarrow \{0,1\}^{n(k)}\). Needs to guess which.
Pseudorandom Function (PRF)

F: \{0,1\}^k \times \{0,1\}^m(k) \rightarrow \{0,1\}^n(k) is a PRF if all PPT adversaries have negligible advantage in the PRF experiment.

Adversary given oracle access to either F with a random seed, or a random function R: \{0,1\}^m(k) \rightarrow \{0,1\}^n(k). Needs to guess which.

Note: Only $2^k$ seeds for F
Pseudorandom Function (PRF)

$F: \{0,1\}^k \times \{0,1\}^{m(k)} \rightarrow \{0,1\}^{n(k)}$ is a PRF if all PPT adversaries have negligible advantage in the PRF experiment.

Adversary given oracle access to either $F$ with a random seed, or a random function $R: \{0,1\}^{m(k)} \rightarrow \{0,1\}^{n(k)}$. Needs to guess which.

Note: Only $2^k$ seeds for $F$.

But $2^{(n2^m)}$ functions $R$.
Pseudorandom Function (PRF)

F: \{0,1\}^k \times \{0,1\}^{m(k)} \rightarrow \{0,1\}^{n(k)} is a PRF if all PPT adversaries have negligible advantage in the PRF experiment.

Adversary given oracle access to either F with a random seed, or a random function R: \{0,1\}^{m(k)} \rightarrow \{0,1\}^{n(k)}. Needs to guess which.

Note: Only $2^k$ seeds for F

But $2^{(n2^m)}$ functions R

PRF stretches k bits to $n2^m$ bits
Pseudorandom Function (PRF)
Pseudorandom Function (PRF)

A PRF can be constructed from any PRG
A PRF can be constructed from any PRG
Pseudorandom Function (PRF)

A PRF can be constructed from any PRG

G is a length-doubling PRG
Pseudorandom Function (PRF)

A PRF can be constructed from any PRG

$G$ is a length-doubling PRG
A PRF can be constructed from any PRG

G is a length-doubling PRG
A PRF can be constructed from any PRG

$G$ is a length-doubling PRG
A PRF can be constructed from any PRG

G is a length-doubling PRG
A PRF can be constructed from any PRG.
Pseudorandom Function (PRF)

A PRF can be constructed from any PRG

G is a length-doubling PRG
Pseudorandom Function (PRF)

A PRF can be constructed from any PRG

\[ G \text{ is a length-doubling PRG} \]
A PRF can be constructed from any PRG
Pseudorandom Function (PRF)

A PRF can be constructed from any PRG
Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG
- Not blazing fast
Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG
- Not blazing fast
- Faster constructions based on specific number-theoretic computational complexity assumptions
A PRF can be constructed from any PRG
  - Not blazing fast

Faster constructions based on specific number-theoretic computational complexity assumptions

Fast heuristic constructions
A PRF can be constructed from any PRG

Not blazing fast

Faster constructions based on specific number-theoretic computational complexity assumptions

Fast heuristic constructions

PRF in practice: Block Cipher
Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG
  - Not blazing fast
- Faster constructions based on specific number-theoretic computational complexity assumptions
- Fast heuristic constructions
- PRF in practice: Block Cipher
Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG
- Not blazing fast
- Faster constructions based on specific number-theoretic computational complexity assumptions
- Fast heuristic constructions

PRF in practice: Block Cipher
- Extra features/requirements:
Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG
  - Not blazing fast

- Faster constructions based on specific number-theoretic computational complexity assumptions

- Fast heuristic constructions

PRF in practice: Block Cipher

- Extra features/requirements:
  - Permutation: input block (r) to output block
A PRF can be constructed from any PRG
- Not blazing fast

Faster constructions based on specific number-theoretic computational complexity assumptions

Fast heuristic constructions

PRF in practice: Block Cipher

Extra features/requirements:
- Permutation: input block \( (r) \) to output block
- Key can be used as an inversion trapdoor
Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG
  - Not blazing fast

- Faster constructions based on specific number-theoretic computational complexity assumptions

- Fast heuristic constructions

PRF in practice: Block Cipher

- Extra features/requirements:
  - Permutation: input block (r) to output block
  - Key can be used as an inversion trapdoor
  - Pseudorandomness even with access to inversion
CPA-secure SKE with a Block Cipher
CPA-secure SKE with a Block Cipher

Suppose Alice and Bob have shared a key (seed) for a block-cipher (PRF) BC
Suppose Alice and Bob have shared a key (seed) for a block-cipher (PRF) \( BC \)

For each encryption, Alice will pick a fresh pseudorandom pad, by picking a fresh value \( r \) and setting \( \text{pad} = BC_k(r) \)
Suppose Alice and Bob have shared a key (seed) for a block-cipher (PRF) $BC$

For each encryption, Alice will pick a fresh pseudorandom pad, by picking a fresh value $r$ and setting $pad = BC_K(r)$
Suppose Alice and Bob have shared a key (seed) for a block-cipher (PRF) \( BC \).

For each encryption, Alice will pick a fresh pseudorandom pad, by picking a fresh value \( r \) and setting \( pad = BC_K(r) \).

Bob needs to be able to generate the same pad, so Alice sends \( r \) (in the clear, as part of the ciphertext) to Bob.
Suppose Alice and Bob have shared a key (seed) for a block-cipher (PRF) $BC$

For each encryption, Alice will pick a fresh pseudorandom pad, by picking a fresh value $r$ and setting $pad=BC_K(r)$

Bob needs to be able to generate the same pad, so Alice sends $r$ (in the clear, as part of the ciphertext) to Bob
Suppose Alice and Bob have shared a key (seed) for a block-cipher (PRF) \( BC \)

For each encryption, Alice will pick a fresh pseudorandom pad, by picking a fresh value \( r \) and setting \( pad = BC_K(r) \)

Bob needs to be able to generate the same pad, so Alice sends \( r \) (in the clear, as part of the ciphertext) to Bob.
Suppose Alice and Bob have shared a key (seed) for a block-cipher (PRF) $BC$

For each encryption, Alice will pick a fresh pseudorandom pad, by picking a fresh value $r$ and setting $pad = BC_K(r)$

Bob needs to be able to generate the same pad, so Alice sends $r$ (in the clear, as part of the ciphertext) to Bob

Even if Eve sees $r$, PRF security guarantees that $BC_K(r)$ is pseudorandom. (In fact, Eve could have picked $r$, as long as we ensure no $r$ is reused.)
Suppose Alice and Bob have shared a key (seed) for a block-cipher (PRF) BC

For each encryption, Alice will pick a fresh pseudorandom pad, by picking a fresh value \( r \) and setting \( \text{pad} = \text{BC}_K(r) \)

Bob needs to be able to generate the same pad, so Alice sends \( r \) (in the clear, as part of the ciphertext) to Bob

Even if Eve sees \( r \), PRF security guarantees that \( \text{BC}_K(r) \) is pseudorandom. (In fact, Eve could have picked \( r \), as long as we ensure no \( r \) is reused.)

How to pick a fresh \( r \)?
Suppose Alice and Bob have shared a key (seed) for a block-cipher (PRF) $BC$

For each encryption, Alice will pick a fresh pseudorandom pad, by picking a fresh value $r$ and setting $pad = BC_K(r)$

Bob needs to be able to generate the same pad, so Alice sends $r$ (in the clear, as part of the ciphertext) to Bob

Even if Eve sees $r$, PRF security guarantees that $BC_K(r)$ is pseudorandom. (In fact, Eve could have picked $r$, as long as we ensure no $r$ is reused.)

How to pick a fresh $r$?

Pick at random!
CPA-secure SKE with a Block Cipher
CPA-secure SKE with a Block Cipher

How to encrypt a long message (multiple blocks)?
How to encrypt a long message (multiple blocks)?

Can chop the message into blocks and independently encrypt each block as before. Works, but ciphertext size is double that of the plaintext (if $|r|$ is one-block long)
CPA-secure SKE with a Block Cipher

How to encrypt a long message (multiple blocks)?

Can chop the message into blocks and independently encrypt each block as before. Works, but ciphertext size is double that of the plaintext (if |r| is one-block long).

Extend output length of PRF (w/o increasing input length)
CPA-secure SKE with a Block Cipher

How to encrypt a long message (multiple blocks)?

Can chop the message into blocks and independently encrypt each block as before. Works, but ciphertext size is double that of the plaintext (if $|r|$ is one-block long)

Extend output length of PRF (w/o increasing input length)
CPA-secure SKE with a Block Cipher

How to encrypt a long message (multiple blocks)?

Can chop the message into blocks and independently encrypt each block as before. Works, but ciphertext size is double that of the plaintext (if $|r|$ is one-block long)

Extend output length of PRF (w/o increasing input length)
CPA-secure SKE with a Block Cipher

How to encrypt a long message (multiple blocks)?

- Can chop the message into blocks and independently encrypt each block as before. Works, but ciphertext size is double that of the plaintext (if $|r|$ is one-block long).

- Extend output length of PRF (w/o increasing input length)

\[ F_K, F_K, \ldots, F_K \]
\[ r \]
\[ F_K, F_K, \ldots, F_K \]
\[ r, 1 \]
\[ r, 2 \]
\[ r, t \]
How to encrypt a long message (multiple blocks)?

Can chop the message into blocks and independently encrypt each block as before. Works, but ciphertext size is double that of the plaintext (if $|r|$ is one-block long)

Extend output length of PRF (w/o increasing input length)
How to encrypt a long message (multiple blocks)?

Can chop the message into blocks and independently encrypt each block as before. Works, but ciphertext size is double that of the plaintext (if $|r|$ is one-block long)

Extend output length of PRF (w/o increasing input length)

Output is indistinguishable from $t$ random blocks (even if input to $F_k$ known/chosen)
CPA-secure SKE with a Block Cipher
CPA-secure SKE with a Block Cipher

Various “modes” of operation of a Block-cipher (i.e., encryption schemes using a block-cipher). All with one block overhead.
CPA-secure SKE with a Block Cipher

Various “modes” of operation of a Block-cipher (i.e., encryption schemes using a block-cipher). All with one block overhead.

Output Feedback (OFB) mode: Extend the pseudorandom output using the first construction in the previous slide
CPA-secure SKE with a Block Cipher

Various “modes” of operation of a Block-cipher (i.e., encryption schemes using a block-cipher). All with one block overhead.

**Output Feedback (OFB) mode:** Extend the pseudorandom output using the first construction in the previous slide.
CPA-secure SKE with a Block Cipher

Various “modes” of operation of a Block-cipher (i.e., encryption schemes using a block-cipher). All with one block overhead.

Output Feedback (OFB) mode: Extend the pseudorandom output using the first construction in the previous slide.
Various “modes” of operation of a Block-cipher (i.e., encryption schemes using a block-cipher). All with one block overhead.

Output Feedback (OFB) mode: Extend the pseudorandom output using the first construction in the previous slide

Counter (CTR) Mode: Similar idea as in the second construction. No a priori limit on number of blocks in a message. Security from low likelihood of \((r+1,\ldots,r+t)\) running into \((r'+1,\ldots,r'+t')\)
CPA-secure SKE with a Block Cipher

Various “modes” of operation of a Block-cipher (i.e., encryption schemes using a block-cipher). All with one block overhead.

Output Feedback (OFB) mode: Extend the pseudorandom output using the first construction in the previous slide.

Counter (CTR) Mode: Similar idea as in the second construction. No a priori limit on number of blocks in a message. Security from low likelihood of \((r+1,\ldots,r+t)\) running into \((r'+1,\ldots,r'+t')\)

Not a PRF (Why?)
CPA-secure SKE with a Block Cipher

Various “modes” of operation of a Block-cipher (i.e., encryption schemes using a block-cipher). All with one block overhead.

Output Feedback (OFB) mode: Extend the pseudorandom output using the first construction in the previous slide.

Counter (CTR) Mode: Similar idea as in the second construction. No a priori limit on number of blocks in a message. Security from low likelihood of \((r+1,\ldots,r+t)\) running into \((r'+1,\ldots,r'+t')\).

Cipher Block Chaining (CBC) mode:
Sequential encryption. Decryption uses \(F_K^{-1}\). Ciphertext an integral number of blocks.

Not a PRF (Why?)