Pseudorandom Functions

How do we build a CPA-secure encryption scheme?

- For EAV-security we had to rely on PRGs

Pseudorandom Functions

How do we build a CPA-secure encryption scheme?

- For EAV-security we had to rely on PRGs
- For CPA-security we need a new cryptographic primitive: **pseudorandom functions** (PRFs)

(Pseudo-)Random Functions

What does it mean for a function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ to be random?

(Pseudo-)Random Functions

What does it mean for a function $f : \{0,1\}^* \to \{0,1\}^*$ to be random?

The question is ill-posed!

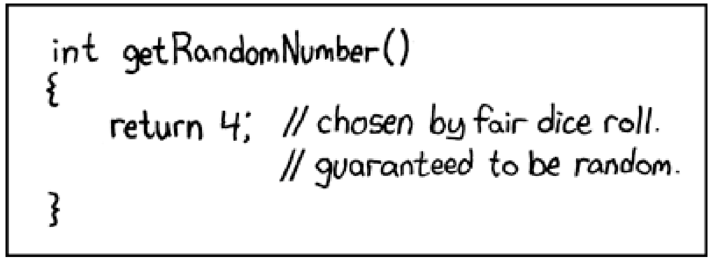
- It does not make sense to say that a *fixed* function is random

(Pseudo-)Random Functions

What does it mean for a function $f : \{0,1\}^* \to \{0,1\}^*$ to be random?

The question is ill-posed!

- It does not make sense to say that a *fixed* function is random
- Just like it does not make sense to say that 0010110 is random, or that the number 4 is random



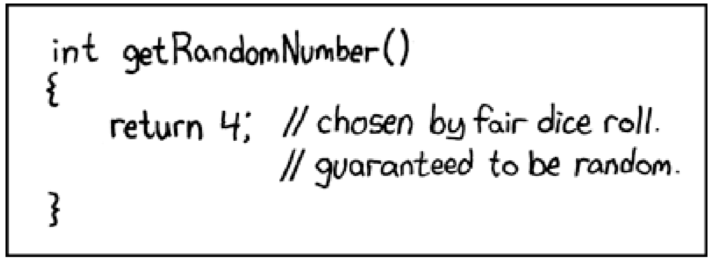
xkcd.com

(Pseudo-)Random Functions

What does it mean for a function $f : \{0,1\}^* \to \{0,1\}^*$ to be random?

The question is ill-posed!

- It does not make sense to say that a *fixed* function is random
- Just like it does not make sense to say that 0010110 is random, or that the number 4 is random



xkcd.com

We need to talk about probability distributions over functions instead

This is formalized using the notion of **keyed functions**

Keyed Functions

A **keyed function** is a function $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$

This function has two inputs. The first input is called the key

Keyed Functions

A **keyed function** is a function $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$

This function has two inputs. The first input is called the key

A keyed function is said to be **efficient** if there is a polynomial-time algorithm that takes as input $k \in \{0,1\}^*$ and $x \in \{0,1\}^*$, and computes $F(k, x)$

Keyed Functions

A **keyed function** is a function $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$

This function has two inputs. The first input is called the key

A keyed function is said to be **efficient** if there is a polynomial-time algorithm that takes as input $k \in \{0,1\}^*$ and $x \in \{0,1\}^*$, and computes $F(k,x)$

We are usually interested in keyed function in which:

- The key has some fixed length ℓ_{key}
- The second input has some fixed length ℓ_{in}
- The output has some fixed length ℓ_{out}

Keyed Functions

A **keyed function** is a function $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$

This function has two inputs. The first input is called the key

A keyed function is said to be **efficient** if there is a polynomial-time algorithm that takes as input $k \in \{0,1\}^*$ and $x \in \{0,1\}^*$, and computes $F(k, x)$

We are usually interested in keyed function in which:

- The key has some fixed length $\ell_{key}(n)$
- The second input has some fixed length $\ell_{in}(n)$
- The output has some fixed length $\ell_{out}(n)$

These quantities are actually functions of the security parameter!

Keyed Functions

A **keyed function** is a function $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$

This function has two inputs. The first input is called the key

A keyed function is said to be **efficient** if there is a polynomial-time algorithm that takes as input $k \in \{0,1\}^*$ and $x \in \{0,1\}^*$, and computes $F(k,x)$

We are usually interested in keyed function in which:

- The key has some fixed length $\ell_{key}(n)$
- The second input has some fixed length $\ell_{in}(n)$
- The output has some fixed length $\ell_{out}(n)$

These quantities are actually functions of the security parameter!

Simplifying assumption (can be removed): F is **length-preserving**

$$\ell_{key}(n) = \ell_{in}(n) = \ell_{out}(n) = n$$

Number of functions

Let Func$_n$ be the set of all functions $f : \{0,1\}^n \to \{0,1\}^n$

How big is Func$_n$?

Number of functions

Let Func_n be the set of all functions $f : \{0,1\}^n \to \{0,1\}^n$

How big is Func_n?

Think of the function as a huge table:

$$
2^n \text{ rows} \quad \left\{ \begin{array}{c|c}
x & F(x) \\
\hline
\texttt{00...000} & \texttt{10...011} \\
\texttt{00...001} & \texttt{01...010} \\
\texttt{00...010} & \texttt{00...110} \\
\vdots & \vdots \\
\texttt{11...111} & \texttt{10...001}
\end{array} \right.
$$

Number of functions

Let Func_n be the set of all functions $f : \{0,1\}^n \to \{0,1\}^n$

How big is Func_n?

Think of the function as a huge table:

2^n rows

x	$F(x)$
00...000	10...011
00...001	01...010
00...010	00...110
$\vdots$	$\vdots$
11...111	10...001

How many distinct tables?

Number of functions

Let Func_n be the set of all functions $f : \{0,1\}^n \to \{0,1\}^n$

How big is Func_n?

Think of the function as a huge table:

2^n rows

x	$F(x)$
`00...000`	`10...011`
`00...001`	`01...010`
`00...010`	`00...110`
$\vdots$	$\vdots$
`11...111`	`10...001`

We have 2^n choices
per row

How many distinct tables?

Number of functions

Let Func_n be the set of all functions $f : \{0,1\}^n \to \{0,1\}^n$

How big is Func_n?

Think of the function as a huge table:

$$
2^n \text{ rows} \quad \left\{ \begin{array}{c|c}
x & F(x) \\
\hline
\texttt{00...000} & \texttt{10...011} \\
\texttt{00...001} & \texttt{01...010} \\
\texttt{00...010} & \texttt{00...110} \\
\vdots & \vdots \\
\texttt{11...111} & \texttt{10...001}
\end{array} \right.
$$

We have 2^n choices
per row

How many distinct tables?

$$(\text{choices per row})^{\#\text{rows}} = (2^n)^{2^n} = 2^{n \cdot 2^n}$$

Number of functions

Let Func_n be the set of all functions $f : \{0,1\}^n \to \{0,1\}^n$

How big is Func_n?

Think of the function as a huge table:

$$
2^n \text{ rows} \quad \left\{ \begin{array}{c|c}
x & F(x) \\
\hline
\texttt{00...000} & \texttt{10...011} \\
\texttt{00...001} & \texttt{01...010} \\
\texttt{00...010} & \texttt{00...110} \\
\vdots & \vdots \\
\texttt{11...111} & \texttt{10...001}
\end{array} \right.
$$

We have 2^n choices **per row**

How many distinct tables?

$$(\text{choices per row})^{\#\text{rows}} = (2^n)^{2^n} = 2^{n \cdot 2^n}$$

For $n = 4$ there 2^{64} functions

Random functions

When we talk about a random function f (for some security parameter n), we actually mean that f is sampled **uniformly at random** from the set Func_n

Random functions

When we talk about a random function f (for some security parameter n), we actually mean that f is sampled **uniformly at random** from the set Func_n

By the *principle of deferred decisions*, we can **equivalently** think of f:

- As a function whose outputs are completely determined at sampling time
  (i.e., for each x, choose a random string $f(x)$ in $\{0,1\}^n$)

Random functions

When we talk about a random function f (for some security parameter n), we actually mean that f is sampled **uniformly at random** from the set Func_n

By the *principle of deferred decisions*, we can **equivalently** think of f:

- As a function whose outputs are completely determined at sampling time (i.e., for each x, choose a random string $f(x)$ in $\{0,1\}^n$)

- As a function whose outputs are decided **lazily**: whenever we need to evaluate $f(x)$:
  - If $f(x)$ was never evaluated before with input x:
    - Return a binary string chosen u.a.r. from $\{0,1\}^n$
  - Otherwise, return the previously chosen string for input x

Back to keyed functions

We will typically use efficient keyed functions as follows:

- Chose some key $k \in \{0,1\}^n$
- Evaluate the function $F(k, x)$ for different choices of x, while k stays the same

Back to keyed functions

We will typically use efficient keyed functions as follows:

- Chose some key $k \in \{0,1\}^n$
- Evaluate the function $F(k,x)$ for different choices of x, while k stays the same
- Is is then convenient to define the **single-input** function $F_k(x) = F(k,x)$

Back to keyed functions

We will typically use efficient keyed functions as follows:

- Chose some key $k \in \{0,1\}^n$
- Evaluate the function $F(k,x)$ for different choices of x, while k stays the same
- Is is then convenient to define the **single-input** function $F_k(x) = F(k,x)$

Choosing k is equivalent to choosing a function $F_k \in \mathsf{Func}_n$!

Pick a uniform k. We now have a **distribution** over the functions in Func_n

Back to keyed functions

We will typically use efficient keyed functions as follows:

- Chose some key $k \in \{0,1\}^n$
- Evaluate the function $F(k,x)$ for different choices of x, while k stays the same
- Is is then convenient to define the **single-input** function $F_k(x) = F(k,x)$

Choosing k is equivalent to choosing a function $F_k \in \mathsf{Func}_n$!

Pick a uniform k. We now have a **distribution** over the functions in Func_n

How big is the support of this distribution?

Back to keyed functions

We will typically use efficient keyed functions as follows:

- Chose some key $k \in \{0,1\}^n$
- Evaluate the function $F(k,x)$ for different choices of x, while k stays the same
- Is is then convenient to define the **single-input** function $F_k(x) = F(k,x)$

Choosing k is equivalent to choosing a function $F_k \in \mathsf{Func}_n$!

Pick a uniform k. We now have a **distribution** over the functions in Func_n

How big is the support of this distribution?

There can be at most as many functions F_k as keys $k \in \{0,1\}^n \implies$ at most 2^n functions!

(out of $2^{n \cdot 2^n}$)

Back to keyed functions

We will typically use efficient keyed functions as follows:

- Chose some key $k \in \{0,1\}^n$
- Evaluate the function $F(k, x)$ for different choices of x, while k stays the same
- Is is then convenient to define the **single-input** function $F_k(x) = F(k, x)$

Choosing k is equivalent to choosing a function $F_k \in \mathsf{Func}_n$!

Pick a uniform k. We now have a **distribution** over the functions in Func_n

How big is the support of this distribution?

There can be at most as many functions F_k as keys $k \in \{0,1\}^n \implies$ at most 2^n functions!

(out of $2^{n \cdot 2^n}$)

For $n = 4$ there are $2^4 = 16$ possible choices... out of 2^{64} possible functions!

Back to keyed functions

We will typically use efficient keyed functions as follows:

- Chose some key $k \in \{0,1\}^n$
- Evaluate the function $F(k,x)$ for different choices of x, while k stays the same
- Is is then convenient to define the **single-input** function $F_k(x) = F(k,x)$

Choosing k is equivalent to choosing a function $F_k \in \mathsf{Func}_n$!

Pick a uniform k. We now have a **distribution** over the functions in Func_n

How big is the support of this distribution?

There can be at most as many functions F_k as keys $k \in \{0,1\}^n \implies$ at most 2^n functions!

(out of $2^{n \cdot 2^n}$)

For $n = 4$ there are $2^4 = 16$ possible choices... out of 2^{64} possible functions!

We can only sample a **tiny** fractions of the functions in Func_n!

Defining pseudorandom functions

Intuition: $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is pseudorandom if no polynomial-time algorithm D can distinguish the function F_k (where k is chosen u.a.r.) from a random function $f \in \mathsf{Func}_n$, except for a negligible probability.

Defining pseudorandom functions

Intuition: $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is pseudorandom if no polynomial-time algorithm D can distinguish the function F_k (where k is chosen u.a.r.) from a random function $f \in \mathsf{Func}_n$, except for a negligible probability.

Caution! What's the input to D?

Defining pseudorandom functions

Intuition: $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is pseudorandom if no polynomial-time algorithm D can distinguish the function F_k (where k is chosen u.a.r.) from a random function $f \in \mathsf{Func}_n$, except for a negligible probability.

Caution! What's the input to D?

- We cannot use an encoding of F_k and f as the input to D
- Such an encoding would be (super)exponential in n !

Defining pseudorandom functions

Intuition: $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is pseudorandom if no polynomial-time algorithm D can distinguish the function F_k (where k is chosen u.a.r.) from a random function $f \in \mathsf{Func}_n$, except for a negligible probability.

Caution! What's the input to D?

- We cannot use an encoding of F_k and f as the input to D
- Such an encoding would be (super)exponential in n !
- D needs to run in a time that is polynomially bounded **by the size of its input**

Defining pseudorandom functions

Intuition: $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is pseudorandom if no polynomial-time algorithm D can distinguish the function F_k (where k is chosen u.a.r.) from a random function $f \in \mathsf{Func}_n$, except for a negligible probability.

Caution! What's the input to D?

- We cannot use an encoding of F_k and f as the input to D
- Such an encoding would be (super)exponential in n !
- D needs to run in a time that is polynomially bounded **by the size of its input**

Workaround: we give D **oracle access** to F_k and f and input 1^n:

Defining pseudorandom functions

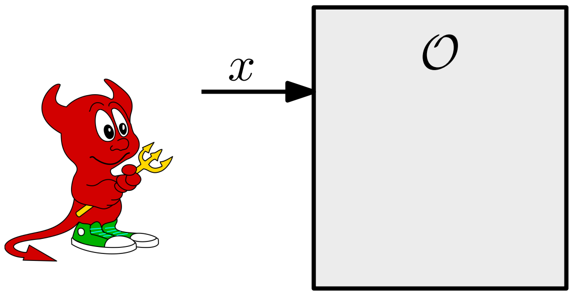
Intuition: $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is pseudorandom if no polynomial-time algorithm D can distinguish the function F_k (where k is chosen u.a.r.) from a random function $f \in \mathsf{Func}_n$, except for a negligible probability.

Caution! What's the input to D?

- We cannot use an encoding of F_k and f as the input to D
- Such an encoding would be (super)exponential in n !
- D needs to run in a time that is polynomially bounded **by the size of its input**

Workaround: we give D **oracle access** to F_k and f and input 1^n:

- There is an oracle $\mathcal{O}$ that can be queried with a string $x \in \{0,1\}^n$



Defining pseudorandom functions

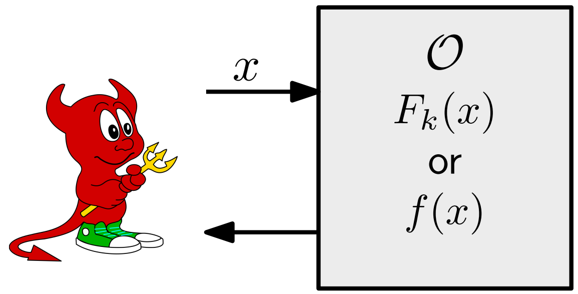
Intuition: $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is pseudorandom if no polynomial-time algorithm D can distinguish the function F_k (where k is chosen u.a.r.) from a random function $f \in \mathsf{Func}_n$, except for a negligible probability.

Caution! What's the input to D?

- We cannot use an encoding of F_k and f as the input to D
- Such an encoding would be (super)exponential in n !
- D needs to run in a time that is polynomially bounded **by the size of its input**

Workaround: we give D **oracle access** to F_k and f and input 1^n:

- There is an oracle $\mathcal{O}$ that can be queried with a string $x \in \{0,1\}^n$
- $\mathcal{O}$ either always answers with $F_k(x)$, or it always answers with $f(x)$



Defining pseudorandom functions

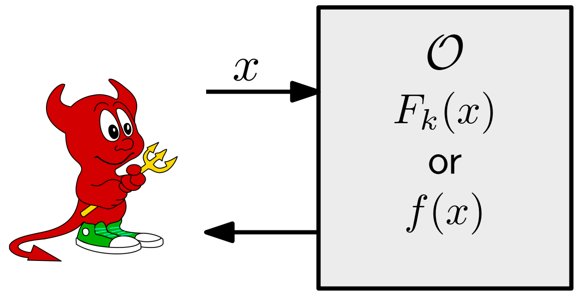
Intuition: $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is pseudorandom if no polynomial-time algorithm D can distinguish the function F_k (where k is chosen u.a.r.) from a random function $f \in \mathsf{Func}_n$, except for a negligible probability.

Caution! What's the input to D?

- We cannot use an encoding of F_k and f as the input to D
- Such an encoding would be (super)exponential in n !
- D needs to run in a time that is polynomially bounded **by the size of its input**

Workaround: we give D **oracle access** to F_k and f and input 1^n:

- There is an oracle $\mathcal{O}$ that can be queried with a string $x \in \{0,1\}^n$
- $\mathcal{O}$ either always answers with $F_k(x)$, or it always answers with $f(x)$
- D can query $\mathcal{O}$ many times



Defining pseudorandom functions

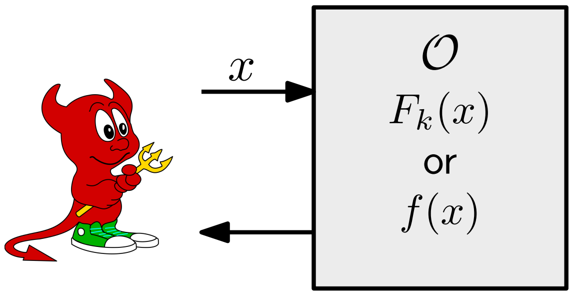
Intuition: $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is pseudorandom if no polynomial-time algorithm D can distinguish the function F_k (where k is chosen u.a.r.) from a random function $f \in \mathsf{Func}_n$, except for a negligible probability.

Caution! What's the input to D?

- We cannot use an encoding of F_k and f as the input to D
- Such an encoding would be (super)exponential in n !
- D needs to run in a time that is polynomially bounded **by the size of its input**

Workaround: we give D **oracle access** to F_k and f and input 1^n:

- There is an oracle $\mathcal{O}$ that can be queried with a string $x \in \{0,1\}^n$
- $\mathcal{O}$ either always answers with $F_k(x)$, or it always answers with $f(x)$
- D can query $\mathcal{O}$ many times
- D needs to guess whether $\mathcal{O}$ is evaluating F_k or f



Defining pseudorandom functions

"World 1":

k is chosen u.a.r. in $\{0,1\}^n$

$\mathcal{O}$

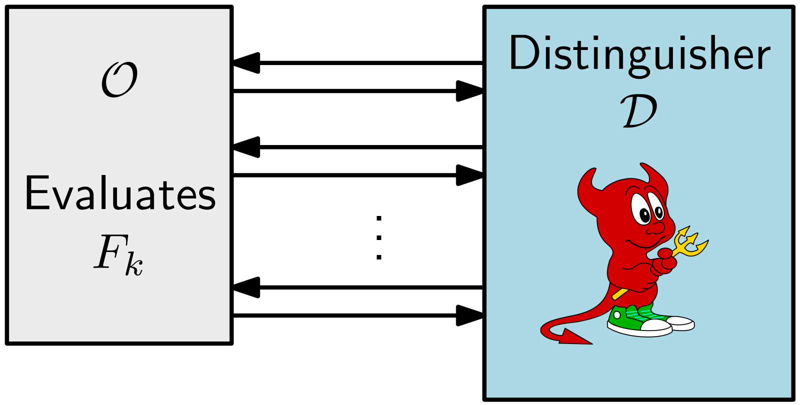
Evaluates F_k

Distinguisher $\mathcal{D}$

Defining pseudorandom functions

"World 1":

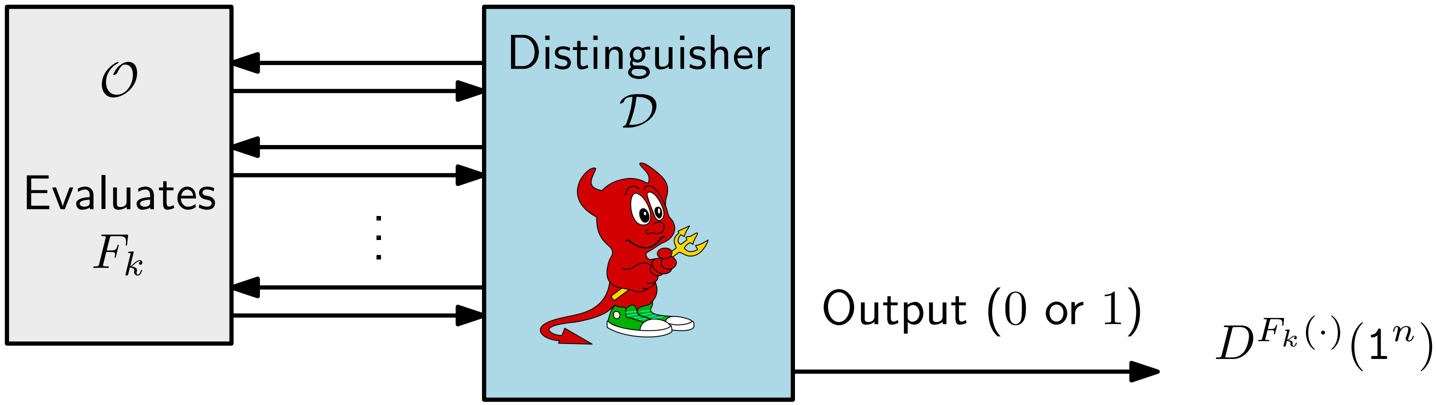
k is chosen u.a.r. in $\{0,1\}^n$



Defining pseudorandom functions

"World 1":

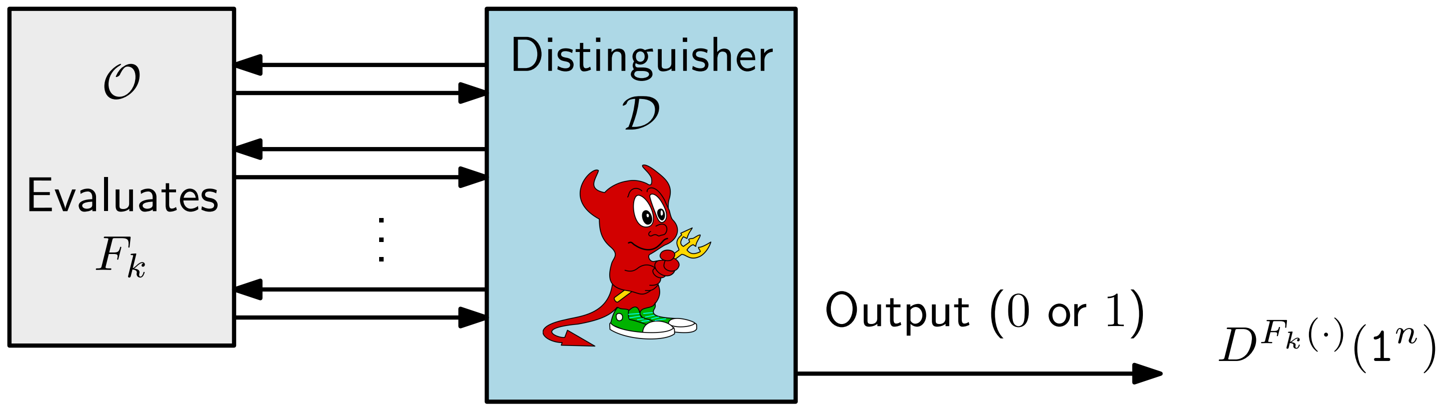
k is chosen u.a.r. in $\{0,1\}^n$



Defining pseudorandom functions

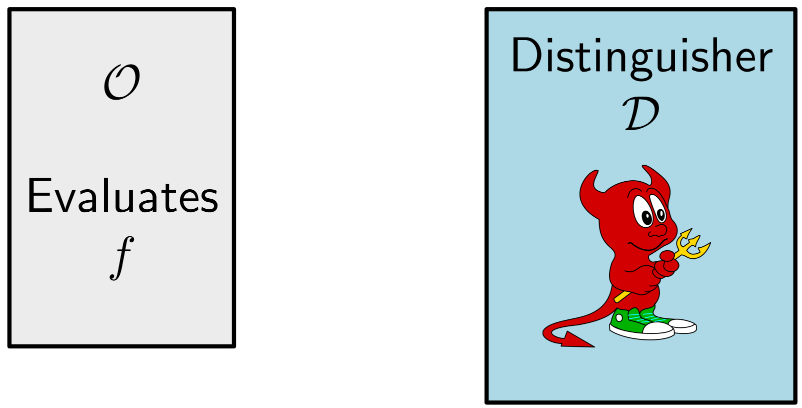
"World 1":

k is chosen u.a.r. in $\{0,1\}^n$



"World 0":

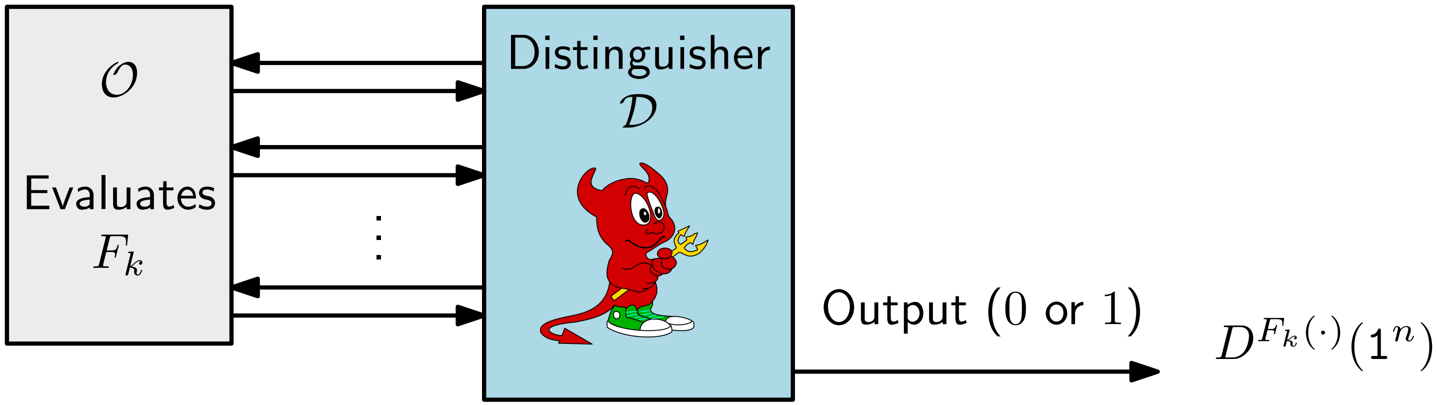
f is chosen u.a.r. in Func_n



Defining pseudorandom functions

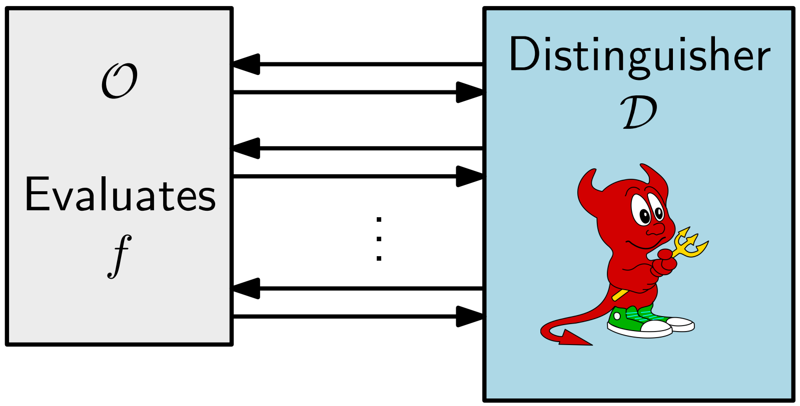
"World 1":

k is chosen u.a.r. in $\{0,1\}^n$



"World 0":

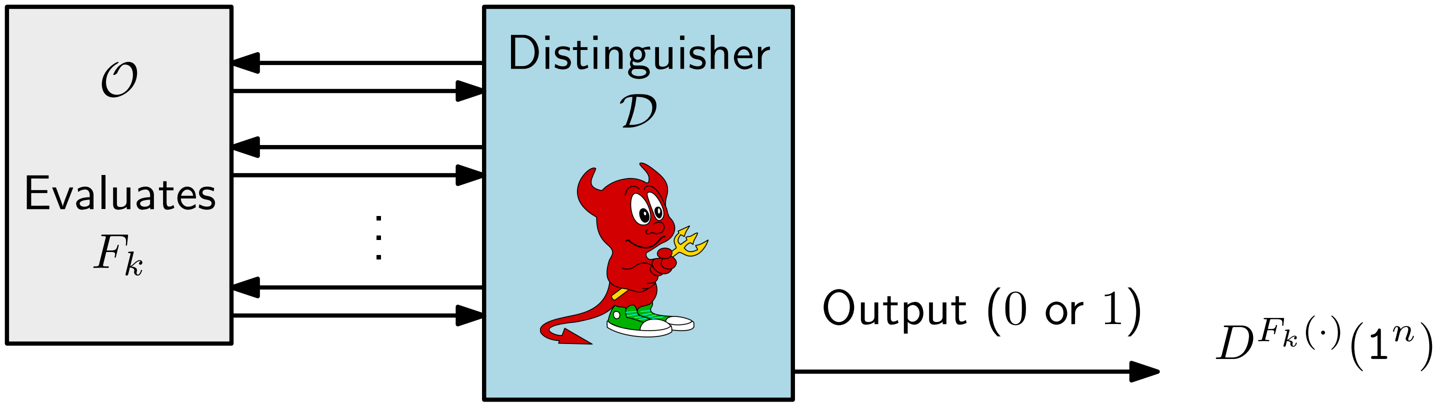
f is chosen u.a.r. in Func_n



Defining pseudorandom functions

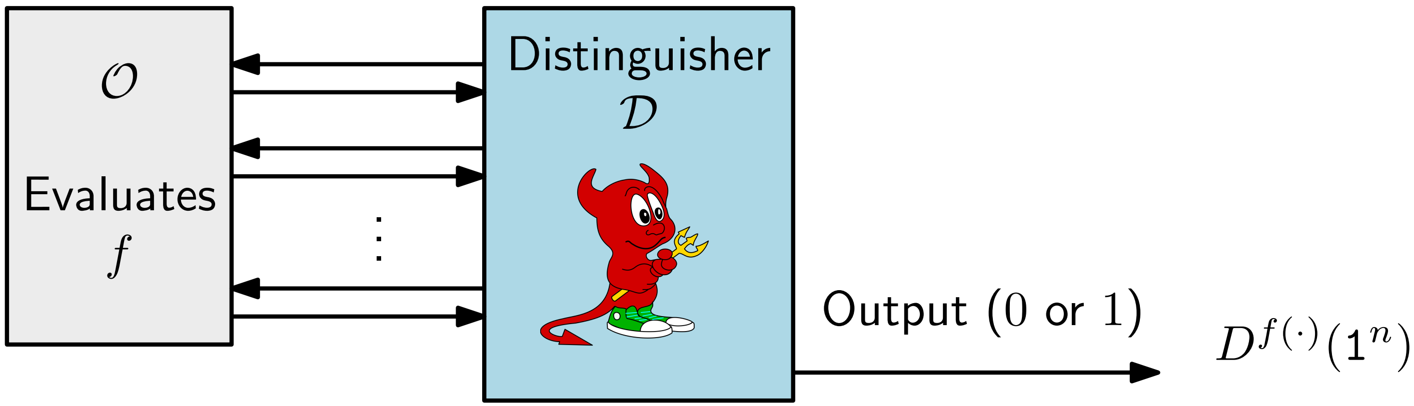
"World 1":

k is chosen u.a.r. in $\{0,1\}^n$

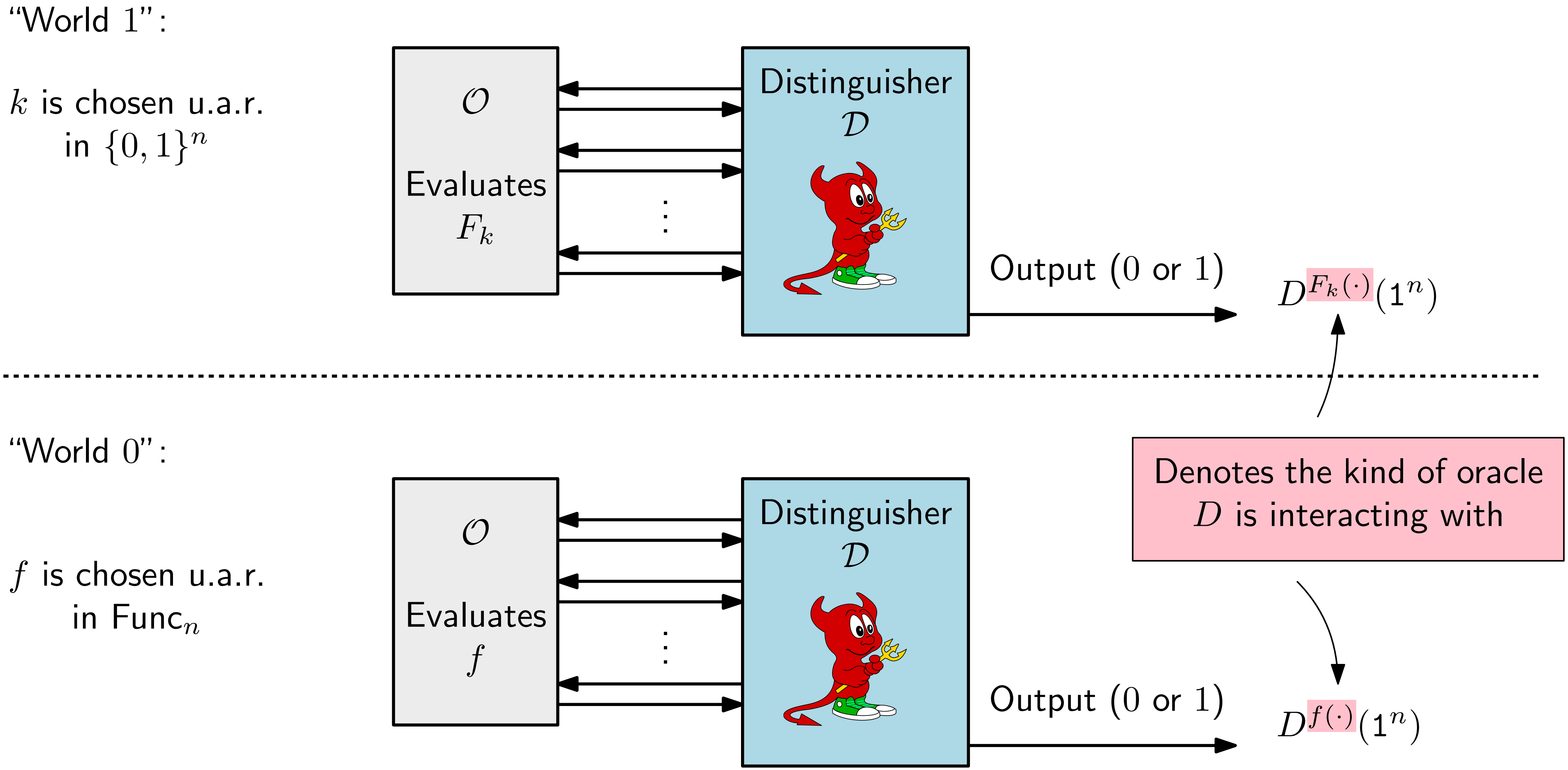


"World 0":

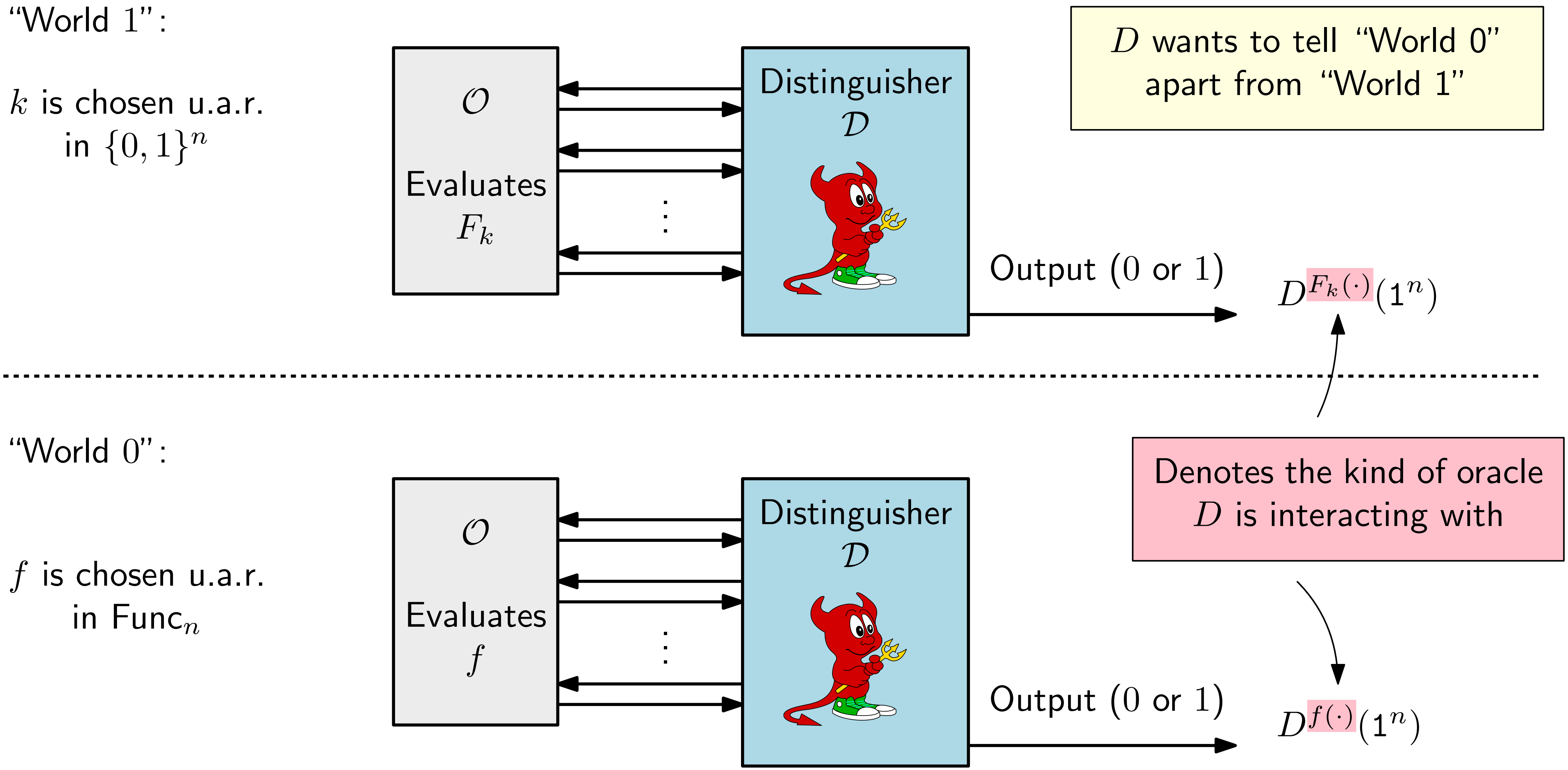
f is chosen u.a.r. in Func_n



Defining pseudorandom functions



Defining pseudorandom functions



Defining pseudorandom functions (formal)

Definition: An efficient, length preserving, keyed function $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is a **pseudorandom function** if for all probabilistic polynomial-time distinguishers D, there is a negligible function ε such that:

$$\left| \Pr[D^{F_k(\cdot)}(1^n) = 1] - \Pr[D^{f(\cdot)}(1^n) = 1] \right| \leq \varepsilon(n)$$

Defining pseudorandom functions (formal)

Definition: An efficient, length preserving, keyed function $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is a **pseudorandom function** if for all probabilistic polynomial-time distinguishers D, there is a negligible function ε such that:

$$\left|\ \Pr[D^{F_k(\cdot)}(\mathtt{1}^n) = 1] - \Pr[D^{f(\cdot)}(\mathtt{1}^n) = 1]\ \right| \le \varepsilon(n)$$

Probability over the randomness of the distinguisher and the choice of k

Defining pseudorandom functions (formal)

Definition: An efficient, length preserving, keyed function $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is a **pseudorandom function** if for all probabilistic polynomial-time distinguishers D, there is a negligible function ε such that:

$$\left|\ \Pr[D^{F_k(\cdot)}(1^n) = 1] - \Pr[D^{f(\cdot)}(1^n) = 1]\ \right| \leq \varepsilon(n)$$

Probability over the randomness of the distinguisher and the choice of k

Probability over the randomness of the distinguisher and the uniform choice of $f \in \mathsf{Func}_n$

Examples

What are some possible distinguishers from the following (failed attempts at) pseudorandom functions?

- $F(k, x) = 1^n$
- $F(k, x) = k$
- $F(k, x) = k \vee x$
- $F(k, x) = k \wedge x$
- $F(k, x) = k \oplus x$

PRFs and PRGs

If we have a PRF $F(k, x)$ we can use it to build a PRG G.

$G(s)$:

- Return $F_s(\mathtt{0...000}) \parallel F_s(\mathtt{0...001})$

expansion factor $\ell(n) = 2n$

PRFs and PRGs

If we have a PRF $F(k, x)$ we can use it to build a PRG G.

$G(k)$:

- Return $F_k(\langle 0 \rangle) \parallel F_k(\langle 1 \rangle) \parallel \ldots \parallel F_k(\langle L \rangle)$

$\langle x \rangle$ = binary encoding of x with n bits

expansion factor $\ell(n) = n \cdot L$ (for $L = O(\text{poly}(n))$)

PRFs and PRGs

If we have a PRF $F(k, x)$ we can use it to build a PRG G.

$G(k)$:

- Return $F_k(\langle 0 \rangle) \parallel F_k(\langle 1 \rangle) \parallel \ldots \parallel F_k(\langle L \rangle)$

$\langle x \rangle$ = binary encoding of x with n bits

expansion factor $\ell(n) = n \cdot L$ (for $L = O(\text{poly}(n))$)

Proof that G is a PRG? Security reduction ("breaking G implies breaking F")

- Suppose that G is not a PRG, then there is some distinguisher D for G (with non negligible gap)
- Use D to build a distinguisher $\mathcal{A}$ for F (with non negligible gap)
- This contradicts the fact that F is a PRF (i.e., no such D can exist)

PRFs and PRGs

$G(k)$:

- Return $F_k(\langle 0 \rangle) \,\|\, F_k(\langle 1 \rangle) \,\|\, \ldots \,\|F_k(\langle L \rangle)$

- Suppose that G is not a PRG, then there is some D such that:

$$|\Pr[D(G(k)) = 1] - \Pr[D(r) = 1]| = \varepsilon(n) \text{ where } \varepsilon(n) \text{ is not negligible}$$

PRFs and PRGs

$G(k)$:

- Return $F_k(\langle 0 \rangle) \parallel F_k(\langle 1 \rangle) \parallel \ldots \parallel F_k(\langle L \rangle)$

- Suppose that G is not a PRG, then there is some D such that:

$$|\Pr[D(G(k)) = 1] - \Pr[D(r) = 1]| = \varepsilon(n) \text{ where } \varepsilon(n) \text{ is not negligible}$$

- We design a distinguisher $\mathcal{A}$ for F. $\mathcal{A}^{\Phi}$ has access to an oracle Φ and returns:

$$D(\ \Phi(\langle 0 \rangle) \parallel \Phi(\langle 1 \rangle) \parallel \ldots \parallel \Phi(\langle L \rangle)\ )$$

PRFs and PRGs

$G(k)$:

- Return $F_k(\langle 0 \rangle) \,\|\, F_k(\langle 1 \rangle) \,\|\, \dots \,\| F_k(\langle L \rangle)$

- Suppose that G is not a PRG, then there is some D such that:

$$|\Pr[D(G(k)) = 1] - \Pr[D(r) = 1]| = \varepsilon(n) \text{ where } \varepsilon(n) \text{ is not negligible}$$

- We design a distinguisher $\mathcal{A}$ for F. $\mathcal{A}^{\Phi}$ has access to an oracle Φ and returns:

$$D(\ \Phi(\langle 0 \rangle) \,\|\, \Phi(\langle 1 \rangle) \,\|\, \dots \,\|\, \Phi(\langle L \rangle)\ )$$

$\Pr[\mathcal{A}^{F_k(\cdot)}(1^n) = 1] = \Pr[D(G(k)) = 1]$

PRFs and PRGs

$G(k)$:

- Return $F_k(\langle 0\rangle) \,\|\, F_k(\langle 1\rangle) \,\|\, \ldots \,\|F_k(\langle L\rangle)$

- Suppose that G is not a PRG, then there is some D such that:

$$|\Pr[D(G(k)) = 1] - \Pr[D(r) = 1]| = \varepsilon(n) \text{ where } \varepsilon(n) \text{ is not negligible}$$

- We design a distinguisher $\mathcal{A}$ for F. $\mathcal{A}^{\Phi}$ has access to an oracle Φ and returns:

$$D(\ \Phi(\langle 0\rangle) \,\|\, \Phi(\langle 1\rangle) \,\|\, \ldots \,\|\, \Phi(\langle L\rangle)\ )$$

$\Pr[\mathcal{A}^{F_k(\cdot)}(1^n) = 1] = \Pr[D(G(k)) = 1]$

$\Pr[\mathcal{A}^{f(\cdot)}(1^n) = 1] = \Pr[D(r) = 1]$

PRFs and PRGs

$G(k)$:

- Return $F_k(\langle 0 \rangle) \,\|\, F_k(\langle 1 \rangle) \,\|\, \ldots \,\|F_k(\langle L \rangle)$

- Suppose that G is not a PRG, then there is some D such that:

$$|\Pr[D(G(k)) = 1] - \Pr[D(r) = 1]| = \varepsilon(n) \text{ where } \varepsilon(n) \text{ is not negligible}$$

- We design a distinguisher $\mathcal{A}$ for F. $\mathcal{A}^{\Phi}$ has access to an oracle Φ and returns:

$$D(\ \Phi(\langle 0 \rangle) \,\|\, \Phi(\langle 1 \rangle) \,\|\, \ldots \,\|\, \Phi(\langle L \rangle)\ )$$

Random string in $\{0,1\}^{L \cdot n}$

$$\Pr[\mathcal{A}^{F_k(\cdot)}(1^n) = 1] = \Pr[D(G(k)) = 1] \qquad \Pr[\mathcal{A}^{f(\cdot)}(1^n) = 1] = \Pr[D(r) = 1]$$

PRFs and PRGs

$G(k)$:

- Return $F_k(\langle 0\rangle) \,\|\, F_k(\langle 1\rangle) \,\|\, \ldots \,\|F_k(\langle L\rangle)$

- Suppose that G is not a PRG, then there is some D such that:

$$|\Pr[D(G(k)) = 1] - \Pr[D(r) = 1]| = \varepsilon(n) \text{ where } \varepsilon(n) \text{ is not negligible}$$

- We design a distinguisher $\mathcal{A}$ for F. $\mathcal{A}^{\Phi}$ has access to an oracle Φ and returns:

$$D(\ \Phi(\langle 0\rangle) \,\|\, \Phi(\langle 1\rangle) \,\|\, \ldots \,\|\, \Phi(\langle L\rangle)\ )$$

Random string in $\{0,1\}^{L\cdot n}$

$$\Pr[\mathcal{A}^{F_k(\cdot)}(1^n) = 1] = \Pr[D(G(k)) = 1] \qquad \Pr[\mathcal{A}^{f(\cdot)}(1^n) = 1] = \Pr[D(r) = 1]$$

$$\left|\ \Pr[\mathcal{A}^{F_k(\cdot)}(1^n) = 1] - \Pr[\mathcal{A}^{f(\cdot)}(1^n) = 1]\ \right| = \left|\ \Pr[D(G(k))] - \Pr[D(r)]\ \right| = \varepsilon(n)$$

PRFs and PRGs

$G(k)$:

- Return $F_k(\langle 0\rangle) \,\|\, F_k(\langle 1\rangle) \,\|\, \ldots \,\| F_k(\langle L\rangle)$

- Suppose that G is not a PRG, then there is some D such that:

$$|\Pr[D(G(k)) = 1] - \Pr[D(r) = 1]| = \varepsilon(n) \text{ where } \varepsilon(n) \text{ is not negligible}$$

- We design a distinguisher $\mathcal{A}$ for F. $\mathcal{A}^{\Phi}$ has access to an oracle Φ and returns:

$$D(\ \Phi(\langle 0\rangle) \,\|\, \Phi(\langle 1\rangle) \,\|\, \ldots \,\|\, \Phi(\langle L\rangle)\ )$$

Random string in $\{0,1\}^{L\cdot n}$

$$\Pr[\mathcal{A}^{F_k(\cdot)}(1^n) = 1] = \Pr[D(G(k)) = 1] \qquad \Pr[\mathcal{A}^{f(\cdot)}(1^n) = 1] = \Pr[D(r) = 1]$$

$$\left|\ \Pr[\mathcal{A}^{F_k(\cdot)}(1^n) = 1] - \Pr[\mathcal{A}^{f(\cdot)}(1^n) = 1]\ \right| = \left|\ \Pr[D(G(k))] - \Pr[D(r)]\ \right| = \varepsilon(n)$$

- Therefore F is not a PRF. $\lightning$ $\square$

PRFs and PRGs

If we have a PRF $F(k, x)$ we can use it to build a PRG G.

Are PRFs a stronger cryptographic primitive than PRGs?

PRFs and PRGs

If we have a PRF $F(k, x)$ we can use it to build a PRG G.

Are PRFs a stronger cryptographic primitive than PRGs? No. PRFs exist $\iff$ PRGs exist

PRFs and PRGs

If we have a PRF $F(k, x)$ we can use it to build a PRG G.

Are PRFs a stronger cryptographic primitive than PRGs? No. PRFs exist $\iff$ PRGs exist

If we have a PRG G we can use it to build a PRF $F(k, x)$.

PRFs and PRGs

If we have a PRF $F(k, x)$ we can use it to build a PRG G.

Are PRFs a stronger cryptographic primitive than PRGs? No. PRFs exist $\iff$ PRGs exist

If we have a PRG G we can use it to build a PRF $F(k, x)$.

A simple case: consider a PRG $G(k)$ with expansion factor $\ell(n) = n \cdot 2^{t(n)}$

$$G(k) = 11011010010010110000101001011110$$

PRFs and PRGs

If we have a PRF $F(k,x)$ we can use it to build a PRG G.

Are PRFs a stronger cryptographic primitive than PRGs? No. PRFs exist $\iff$ PRGs exist

If we have a PRG G we can use it to build a PRF $F(k,x)$.

A simple case: consider a PRG $G(k)$ with expansion factor $\ell(n) = n \cdot 2^{t(n)}$

Divide the output of $G(k)$ into $2^{t(n)}$ "chunks" of n bits each

$G(k) =$	1101	1010	0100	1011	0000	1010	0101	1110

PRFs and PRGs

If we have a PRF $F(k, x)$ we can use it to build a PRG G.

Are PRFs a stronger cryptographic primitive than PRGs?

No. PRFs exist $\iff$ PRGs exist

If we have a PRG G we can use it to build a PRF $F(k, x)$.

A simple case: consider a PRG $G(k)$ with expansion factor $\ell(n) = n \cdot 2^{t(n)}$

Divide the output of $G(k)$ into $2^{t(n)}$ "chunks" of n bits each

$$G(k) = \boxed{1101}\boxed{1010}\boxed{0100}\boxed{1011}\boxed{0000}\boxed{1010}\boxed{0101}\boxed{1110}$$

x	$F_k(x)$
000	1101
001	1010
010	0100
011	1011
100	0000
101	1010
110	0101
111	1110

PRFs and PRGs

If we have a PRF $F(k, x)$ we can use it to build a PRG G.

Are PRFs a stronger cryptographic primitive than PRGs? No. PRFs exist $\iff$ PRGs exist

If we have a PRG G we can use it to build a PRF $F(k, x)$.

A simple case: consider a PRG $G(k)$ with expansion factor $\ell(n) = n \cdot 2^{t(n)}$

Divide the output of $G(k)$ into $2^{t(n)}$ "chunks" of n bits each

$$G(k) = \boxed{1101}\boxed{1010}\boxed{0100}\boxed{1011}\boxed{0000}\boxed{1010}\boxed{0101}\boxed{1110}$$

$F_k(\langle i \rangle)$ returns the i-th group of bits (counting from 0) of $G(k)$

$\ell_{in}(n) = t(n)$, $\ell_{out}(n) = n$

x	$F_k(x)$
000	1101
001	1010
010	0100
011	1011
100	0000
101	1010
110	0101
111	1110

PRFs and PRGs

If we have a PRF $F(k,x)$ we can use it to build a PRG G.

Are PRFs a stronger cryptographic primitive than PRGs? No. PRFs exist $\iff$ PRGs exist

If we have a PRG G we can use it to build a PRF $F(k,x)$.

A simple case: consider a PRG $G(k)$ with expansion factor $\ell(n) = n \cdot 2^{t(n)}$

Divide the output of $G(k)$ into $2^{t(n)}$ "chunks" of n bits each

$$G(k) = \boxed{1101}\boxed{1010}\boxed{0100}\boxed{1011}\boxed{0000}\boxed{1010}\boxed{0101}\boxed{1110}$$

x	$F_k(x)$
000	1101
001	1010
010	0100
011	1011
100	0000
101	1010
110	0101
111	1110

$F_k(\langle i \rangle)$ returns the i-th group of bits (counting from 0) of $G(k)$

$\ell_{in}(n) = t(n)$, $\ell_{out}(n) = n$

Caveat: To construct the table in polynomial time we need $t(n) = O(\log n)$

PRFs and PRGs

If we have a PRF $F(k,x)$ we can use it to build a PRG G.

Are PRFs a stronger cryptographic primitive than PRGs? No. PRFs exist $\iff$ PRGs exist

If we have a PRG G we can use it to build a PRF $F(k,x)$.

x	$F_k(x)$
000	1101
001	1010
010	0100
011	1011
100	0000
101	1010
110	0101
111	1110

A simple case: consider a PRG $G(k)$ with expansion factor $\ell(n) = n \cdot 2^{t(n)}$

Divide the output of $G(k)$ into $2^{t(n)}$ "chunks" of n bits each

$$G(k) = \boxed{1101}\boxed{1010}\boxed{0100}\boxed{1011}\boxed{0000}\boxed{1010}\boxed{0101}\boxed{1110}$$

$F_k(\langle i \rangle)$ returns the i-th group of bits (counting from 0) of $G(k)$

$\ell_{in}(n) = t(n)$, $\ell_{out}(n) = n$

Caveat: To construct the table in polynomial time we need $t(n) = O(\log n)$ $\implies$ F has short inputs

PRFs and PRGs

Proof of security: $G(k) =$ 11011010010010110000101001011110

- Suppose that F is not a PRF, then there is $\mathcal{A}$ such that

$$\left| \Pr[\mathcal{A}^{F_k(\cdot)}(1^n) = 1] - \Pr[\mathcal{A}^{f(\cdot)}(1^n) = 1] \right| = \varepsilon(n) \text{ for non-negligible } \varepsilon(n)$$

PRFs and PRGs

Proof of security:

$G(k) = $ | 1101 | 1010 | 0100 | 1011 | 0000 | 1010 | 0101 | 1110 |

- Suppose that F is not a PRF, then there is $\mathcal{A}$ such that

  $\left| \Pr[\mathcal{A}^{F_k(\cdot)}(1^n) = 1] - \Pr[\mathcal{A}^{f(\cdot)}(1^n) = 1] \right| = \varepsilon(n)$ for non-negligible $\varepsilon(n)$

- Consider the following distinguisher $D(w)$ for G:
  - D splits w into blocks, and builds a table as before

x	$F_k(x)$
000	1101
001	1010
010	0100
011	1011
100	0000
101	1010
110	0101
111	1110

PRFs and PRGs

Proof of security:

$G(k) =$ | 1101 | 1010 | 0100 | 1011 | 0000 | 1010 | 0101 | 1110 |

x	$F_k(x)$
000	1101
001	1010
010	0100
011	1011
100	0000
101	1010
110	0101
111	1110

- Suppose that F is not a PRF, then there is $\mathcal{A}$ such that

  $\left| \Pr[\mathcal{A}^{F_k(\cdot)}(1^n) = 1] - \Pr[\mathcal{A}^{f(\cdot)}(1^n) = 1] \right| = \varepsilon(n)$ for non-negligible $\varepsilon(n)$

- Consider the following distinguisher $D(w)$ for G:
  - D splits w into blocks, and builds a table as before
  - D simulates the oracle Φ and calls $\mathcal{A}^{\Phi}$. Whenever $\mathcal{A}$ queries $\Phi(x)$, D answers with the output of the row labeled x in the table

PRFs and PRGs

Proof of security:

$G(k) =$ 1101 1010 0100 1011 0000 1010 0101 1110

- Suppose that F is not a PRF, then there is $\mathcal{A}$ such that

$$\left|\ \Pr[\mathcal{A}^{F_k(\cdot)}(1^n) = 1] - \Pr[\mathcal{A}^{f(\cdot)}(1^n) = 1]\ \right| = \varepsilon(n) \text{ for non-negligible } \varepsilon(n)$$

- Consider the following distinguisher $D(w)$ for G:
  - D splits w into blocks, and builds a table as before
  - D simulates the oracle Φ and calls $\mathcal{A}^{\Phi}$. Whenever $\mathcal{A}$ queries $\Phi(x)$, D answers with the output of the row labeled x in the table
  - D returns the same output as $\mathcal{A}$

x	$F_k(x)$
000	1101
001	1010
010	0100
011	1011
100	0000
101	1010
110	0101
111	1110

PRFs and PRGs

Proof of security:

$G(k) = $ 1101 1010 0100 1011 0000 1010 0101 1110

- Suppose that F is not a PRF, then there is $\mathcal{A}$ such that

$$\left| \Pr[\mathcal{A}^{F_k(\cdot)}(1^n) = 1] - \Pr[\mathcal{A}^{f(\cdot)}(1^n) = 1] \right| = \varepsilon(n) \text{ for non-negligible } \varepsilon(n)$$

- Consider the following distinguisher $D(w)$ for G:
  - D splits w into blocks, and builds a table as before
  - D simulates the oracle Φ and calls $\mathcal{A}^{\Phi}$. Whenever $\mathcal{A}$ queries $\Phi(x)$, D answers with the output of the row labeled x in the table
  - D returns the same output as $\mathcal{A}$

- $\Pr[D(G(k)) = 1] = \Pr[A^{F_k(\cdot)}(1^n) = 1]$

x	$F_k(x)$
000	1101
001	1010
010	0100
011	1011
100	0000
101	1010
110	0101
111	1110

PRFs and PRGs

Proof of security: $G(k) = \boxed{1101}\boxed{1010}\boxed{0100}\boxed{1011}\boxed{0000}\boxed{1010}\boxed{0101}\boxed{1110}$

x	$F_k(x)$
000	1101
001	1010
010	0100
011	1011
100	0000
101	1010
110	0101
111	1110

- Suppose that F is not a PRF, then there is $\mathcal{A}$ such that

  $\left| \Pr[\mathcal{A}^{F_k(\cdot)}(1^n) = 1] - \Pr[\mathcal{A}^{f(\cdot)}(1^n) = 1] \right| = \varepsilon(n)$ for non-negligible $\varepsilon(n)$

- Consider the following distinguisher $D(w)$ for G:
  - D splits w into blocks, and builds a table as before
  - D simulates the oracle Φ and calls $\mathcal{A}^{\Phi}$. Whenever $\mathcal{A}$ queries $\Phi(x)$, D answers with the output of the row labeled x in the table
  - D returns the same output as $\mathcal{A}$

- $\Pr[D(G(k)) = 1] = \Pr[A^{F_k(\cdot)}(1^n) = 1]$

- $\Pr[D(r) = 1] = \Pr[A^{f(\cdot)}(1^n) = 1]$

PRFs and PRGs

Proof of security:

$G(k) = \boxed{1101}\boxed{1010}\boxed{0100}\boxed{1011}\boxed{0000}\boxed{1010}\boxed{0101}\boxed{1110}$

- Suppose that F is not a PRF, then there is $\mathcal{A}$ such that

$$\left| \Pr[\mathcal{A}^{F_k(\cdot)}(1^n) = 1] - \Pr[\mathcal{A}^{f(\cdot)}(1^n) = 1] \right| = \varepsilon(n) \text{ for non-negligible } \varepsilon(n)$$

x	$F_k(x)$
000	1101
001	1010
010	0100
011	1011
100	0000
101	1010
110	0101
111	1110

- Consider the following distinguisher $D(w)$ for G:
  - D splits w into blocks, and builds a table as before
  - D simulates the oracle Φ and calls $\mathcal{A}^{\Phi}$. Whenever $\mathcal{A}$ queries $\Phi(x)$, D answers with the output of the row labeled x in the table
  - D returns the same output as $\mathcal{A}$

- $\Pr[D(G(k)) = 1] = \Pr[A^{F_k(\cdot)}(1^n) = 1]$
- $\Pr[D(r) = 1] = \Pr[A^{f(\cdot)}(1^n) = 1]$

$\Longrightarrow |\Pr[D(G(k))] - \Pr[D(r)]| = \varepsilon(n)$ non negligible

PRFs and PRGs

Proof of security:

$G(k) = $ 1101 1010 0100 1011 0000 1010 0101 1110

- Suppose that F is not a PRF, then there is $\mathcal{A}$ such that

$$\left| \Pr[\mathcal{A}^{F_k(\cdot)}(1^n) = 1] - \Pr[\mathcal{A}^{f(\cdot)}(1^n) = 1] \right| = \varepsilon(n) \text{ for non-negligible } \varepsilon(n)$$

x	$F_k(x)$
000	1101
001	1010
010	0100
011	1011
100	0000
101	1010
110	0101
111	1110

- Consider the following distinguisher $D(w)$ for G:
  - D splits w into blocks, and builds a table as before
  - D simulates the oracle Φ and calls $\mathcal{A}^{\Phi}$. Whenever $\mathcal{A}$ queries $\Phi(x)$, D answers with the output of the row labeled x in the table
  - D returns the same output as $\mathcal{A}$

- $\Pr[D(G(k)) = 1] = \Pr[A^{F_k(\cdot)}(1^n) = 1]$
- $\Pr[D(r) = 1] = \Pr[A^{f(\cdot)}(1^n) = 1]$

$\implies |\Pr[D(G(k))] - \Pr[D(r)]| = \varepsilon(n)$ non negligible

$\implies$ G is not a PRG $\square$

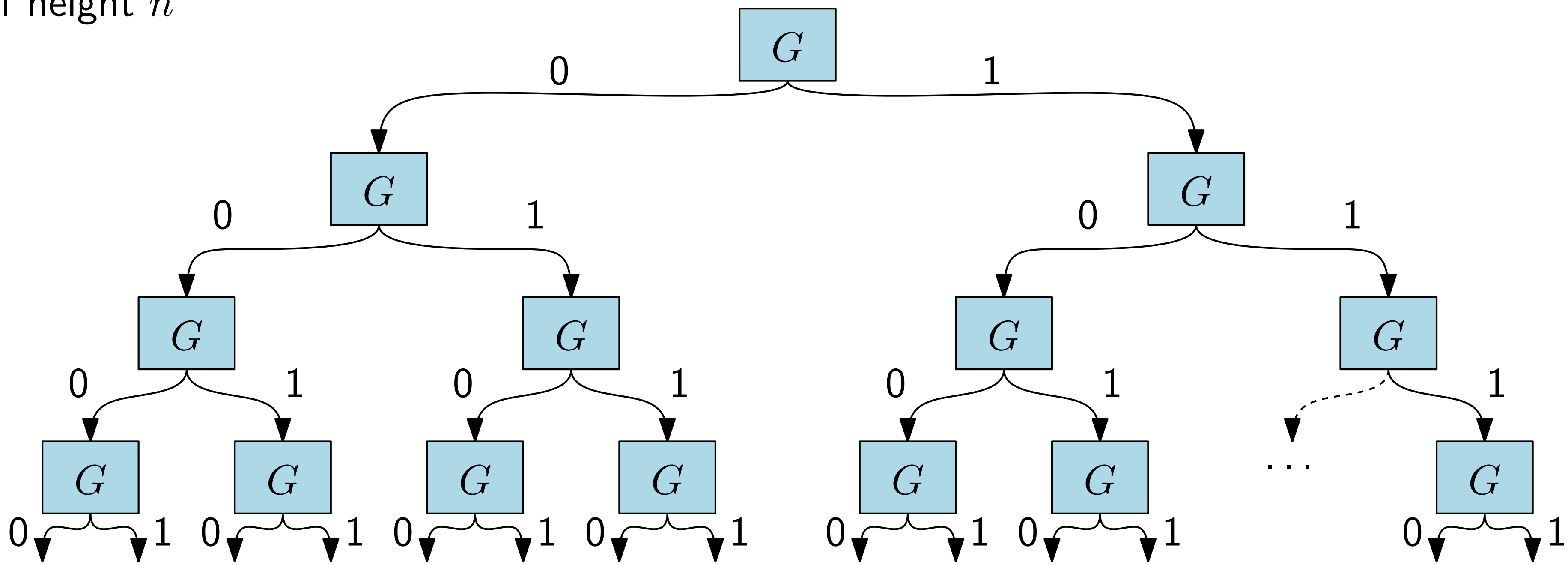
The Goldreich-Goldwasser-Micali construction

Let G be a *length-doubling* PRG, i.e., $\ell(n) = 2n$. $G(s) = G_0(s) \,\|\, G_1(s)$

The Goldreich-Goldwasser-Micali construction

Let G be a *length-doubling* PRG, i.e., $\ell(n) = 2n$. $\qquad G(s) = G_0(s) \,\|\, G_1(s)$

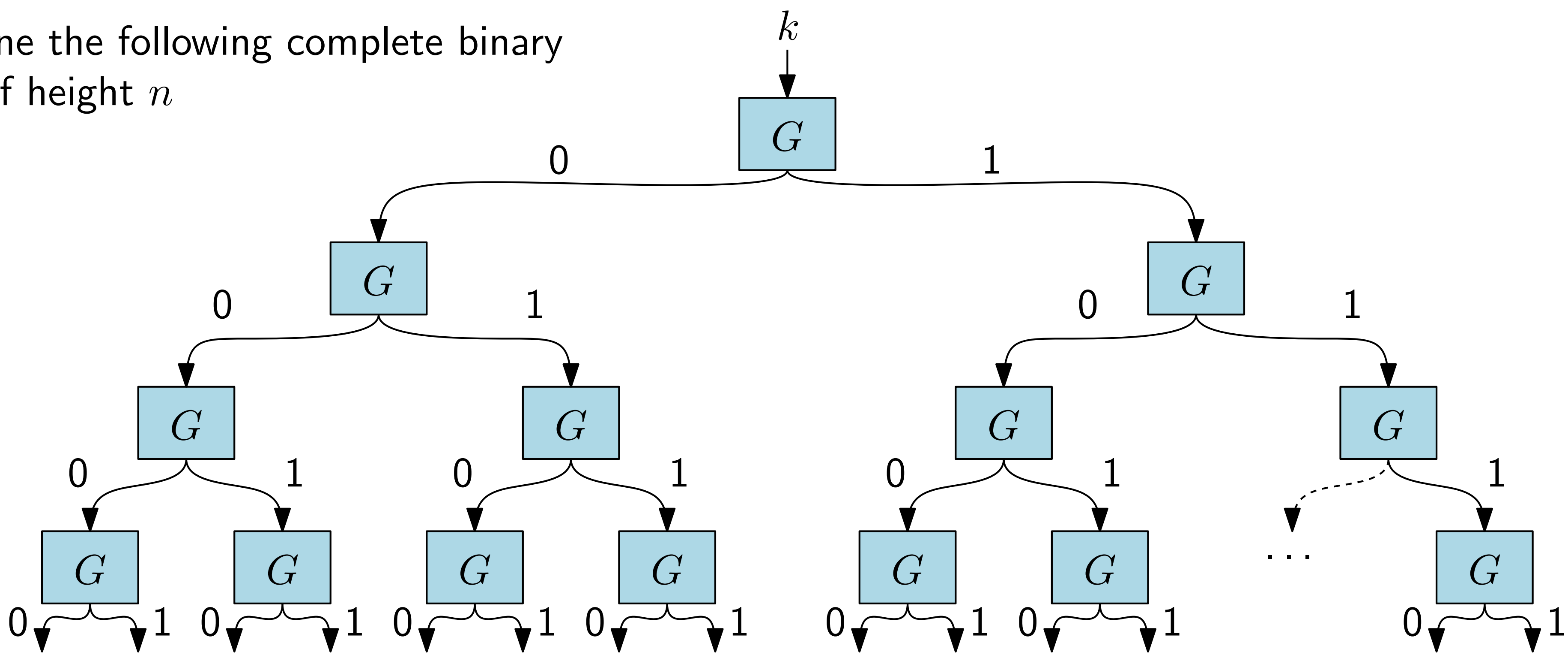
Imagine the following complete binary tree of height n



The Goldreich-Goldwasser-Micali construction

Let G be a *length-doubling* PRG, i.e., $\ell(n) = 2n$. $G(s) = G_0(s) \,\|\, G_1(s)$

Imagine the following complete binary tree of height n

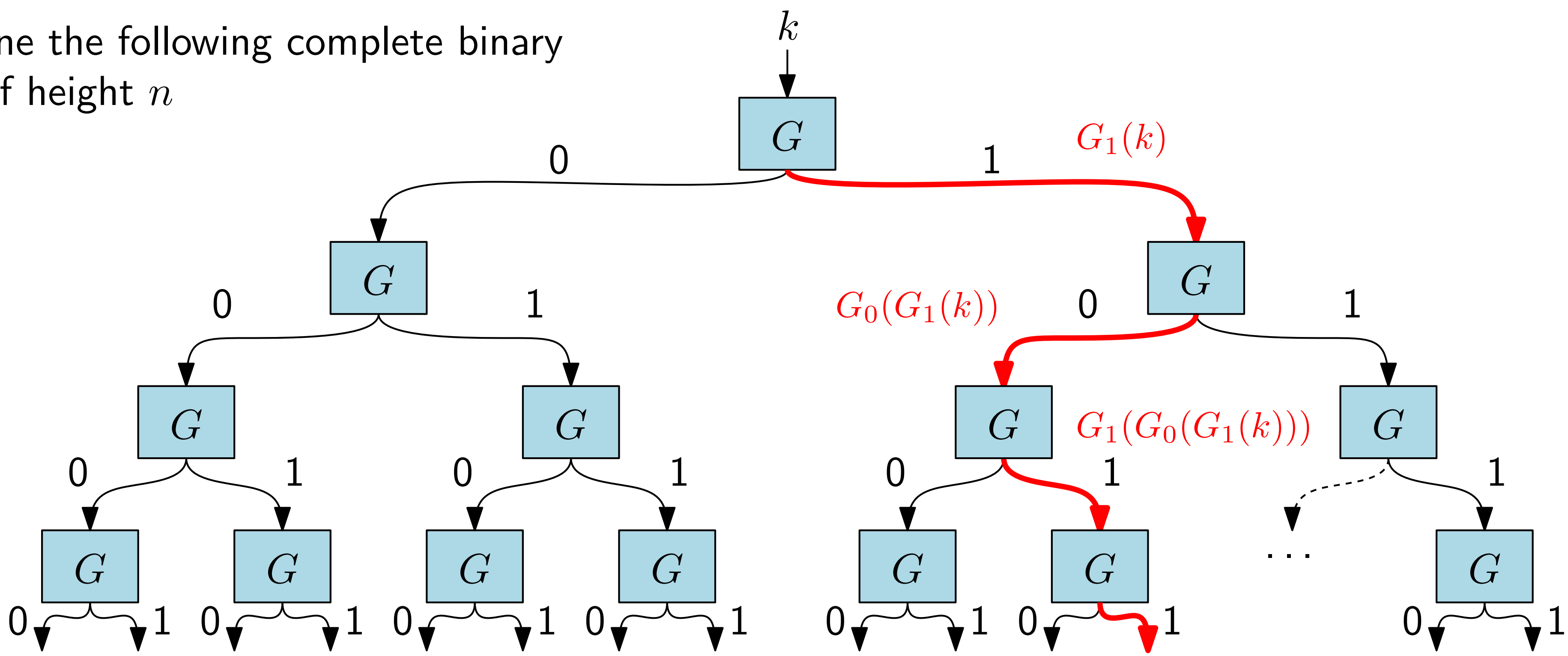


Interpret the key k of $F(k, x)$ as the seed of the root of the tree

The Goldreich-Goldwasser-Micali construction

Let G be a *length-doubling* PRG, i.e., $\ell(n) = 2n$. $G(s) = G_0(s) \,\|\, G_1(s)$

Imagine the following complete binary tree of height n



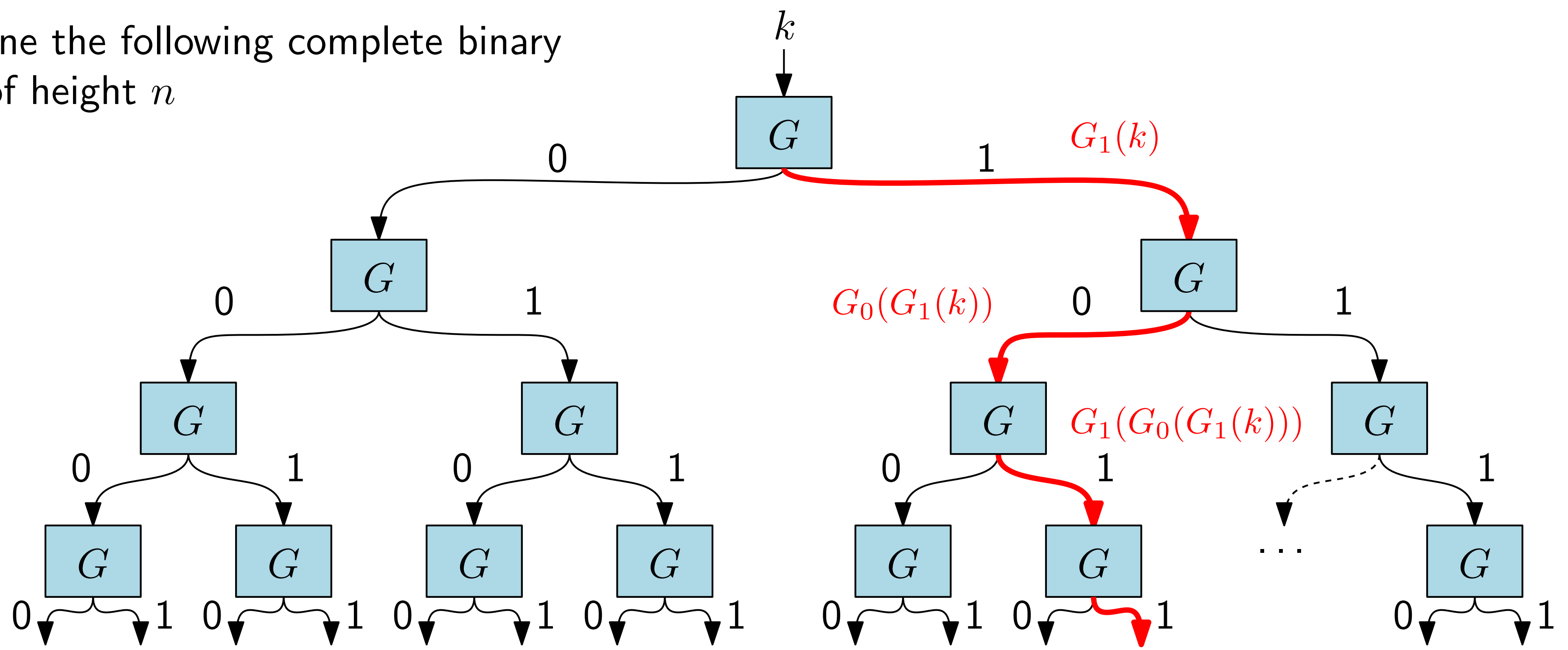
Interpret the key k of $F(k, x)$ as the seed of the root of the tree

Interpret the binary digits of x as a path in the tree

The Goldreich-Goldwasser-Micali construction

Let G be a *length-doubling* PRG, i.e., $\ell(n) = 2n$. $G(s) = G_0(s) \,\|\, G_1(s)$

Imagine the following complete binary tree of height n



Interpret the key k of $F(k, x)$ as the seed of the root of the tree

Interpret the binary digits of x as a path in the tree

Interpret the output of the leaf as the output of $F(k, x)$ $F(k, 1011) = G_1(G_1(G_0(G_1(k))))$

The Goldreich-Goldwasser-Micali construction

> If G is a secure length-doubling PRG, then the Goldreich-Goldwasser-Micali construction is a PRF

We won't see a proof of this fact (see Section 8.5 of the textbook if interested).

The Goldreich-Goldwasser-Micali construction

> If G is a secure length-doubling PRG, then the Goldreich-Goldwasser-Micali construction is a PRF

We won't see a proof of this fact (see Section 8.5 of the textbook if interested).

What if don't have a length-doubling PRG?

The Goldreich-Goldwasser-Micali construction

If G is a secure length-doubling PRG, then the Goldreich-Goldwasser-Micali construction is a PRF

We won't see a proof of this fact (see Section 8.5 of the textbook if interested).

What if don't have a length-doubling PRG?

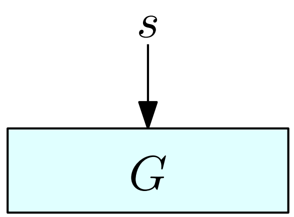
We can build one from any PRG, even if the expansion factor is just $\ell(n) = n + 1$

In fact, we can build a PRG with expansion factor $n + p(n)$ **for any polynomial** $p(n)$

Increasing the expansion factor

An easy case: increasing the expansion factor by 1

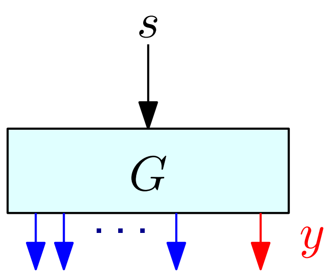
- Start from a PRG G with expansion factor $\ell(n) = n + 1$



Increasing the expansion factor

An easy case: increasing the expansion factor by 1

- Start from a PRG G with expansion factor $\ell(n) = n + 1$
- Call $G(s)$ and interpret the first n bits $x_1 x_2 \ldots x_n$ of the output as a new seed
- Let the last bit of $G(s)$ be y

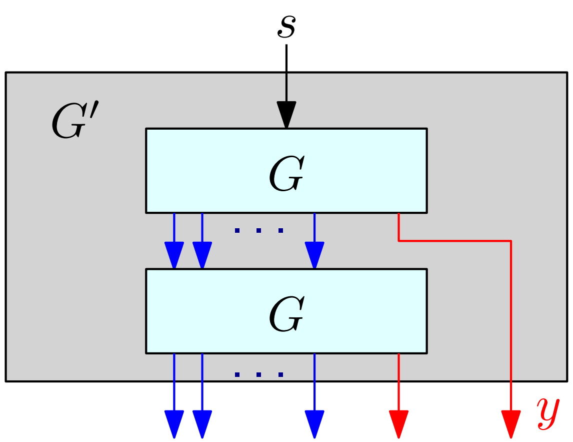


$$G(s) = x_1 x_2 x_3 \ldots x_n y$$

Increasing the expansion factor

An easy case: increasing the expansion factor by 1

- Start from a PRG G with expansion factor $\ell(n) = n + 1$
- Call $G(s)$ and interpret the first n bits $x_1 x_2 \dots x_n$ of the output as a new seed
- Let the last bit of $G(s)$ be y
- Return $G(x_1 x_2 \dots x_n) \,\|\, y$

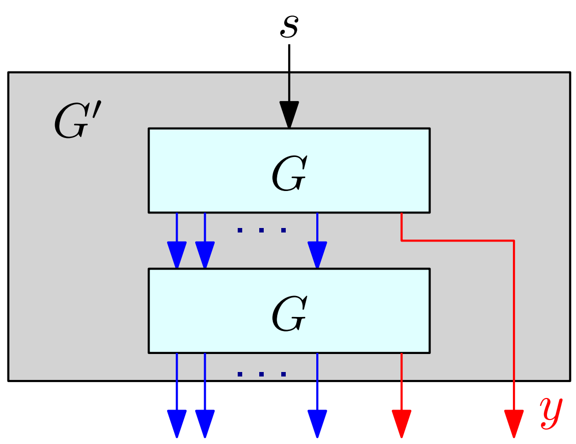


$$G(s) = x_1 x_2 x_3 \dots x_n y$$

Increasing the expansion factor

An easy case: increasing the expansion factor by 1

- Start from a PRG G with expansion factor $\ell(n) = n + 1$
- Call $G(s)$ and interpret the first n bits $x_1 x_2 \dots x_n$ of the output as a new seed
- Let the last bit of $G(s)$ be y
- Return $G(x_1 x_2 \dots x_n) \,\|\, y$



$G(s) = x_1 x_2 x_3 \dots x_n y$

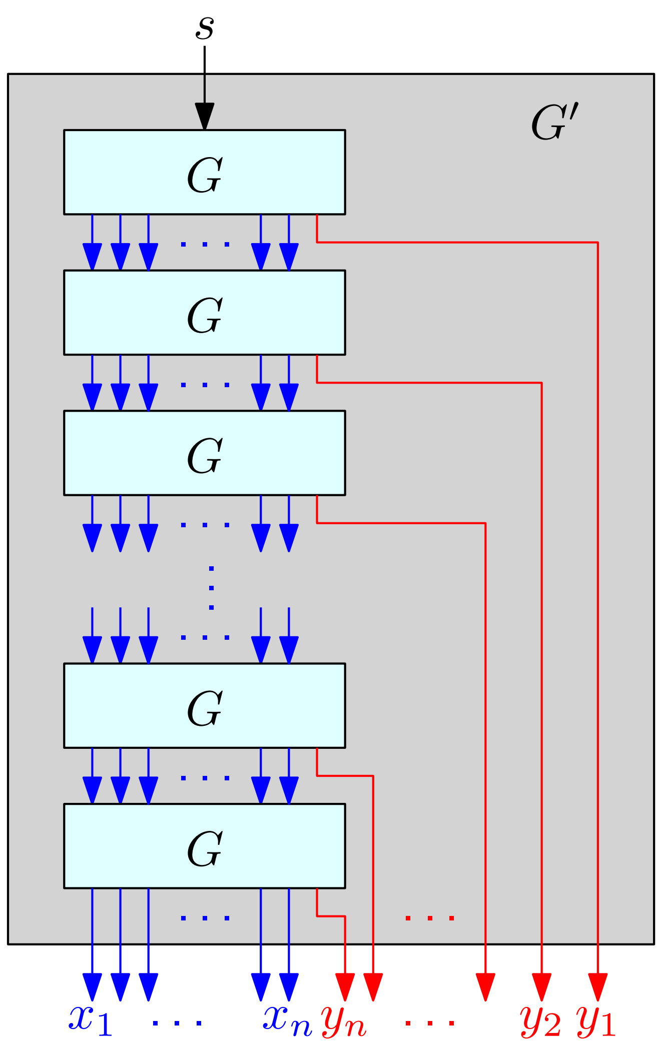
Overall expansion factor $\ell(n) = n + 2$

Increasing the expansion factor (length-doubling)

Increasing the expansion factor from $n+1$ to $2n$

- Start from a PRG G with expansion factor $\ell(n) = n+1$
- Repeat the previous idea for n levels
- The i-th intermediate level outputs $n+1$ bits
  - n bits are used as a seed for the next level
  - The $(n+1)$-th bit y_i will be part of the output of the whole construction
- The last level outputs $n+1$ bits $x_1 x_2 \dots x_n y_n$
- The final output is $x_1 x_2 \dots x_n y_n y_{n-1} \dots y_1$

Overall expansion factor: $\ell(n) = n + n = 2n$

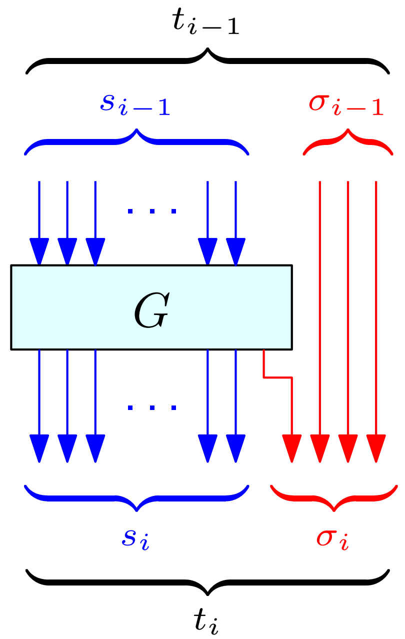


Increasing the expansion factor to $n + p(n)$

Repeat the previous idea $p(n)$ times

Algorithm $\widehat{G}(s)$: (here $s \in \{0,1\}^n$)

- $t_0 \leftarrow s$
- For $i = 1, 2, \ldots, p(n)$:
  - Interpret t_{i-1} as $s_{i-1} \| \sigma_{i-1}$ where $|s_{i-1}| = n$ and $|\sigma_{i-1}| = i - 1$
  - $t_i \leftarrow G(s_{i-1}) \| \sigma_{i-1}$
- Return $t_{p(n)}$

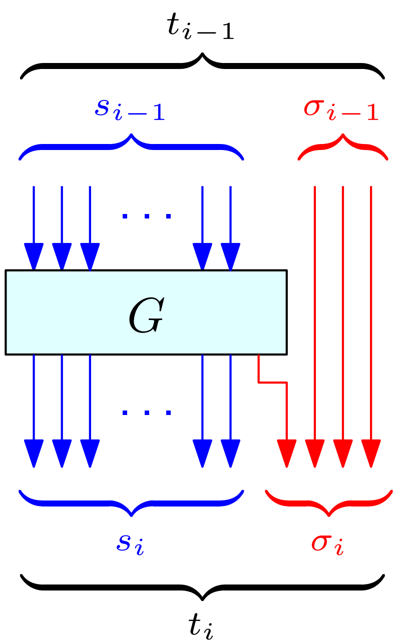


Increasing the expansion factor to $n + p(n)$

Repeat the previous idea $p(n)$ times

Algorithm $\widehat{G}(s)$: (here $s \in \{0,1\}^n$)

- $t_0 \leftarrow s$
- For $i = 1, 2, \ldots, p(n)$:
  - Interpret t_{i-1} as $s_{i-1} \| \sigma_{i-1}$ where $|s_{i-1}| = n$ and $|\sigma_{i-1}| = i - 1$
  - $t_i \leftarrow G(s_{i-1}) \| \sigma_{i-1}$
- Return $t_{p(n)}$



Theorem: If there exists a pseudorandom generator G with expansion factor $n + 1$ then, for any polynomial p, $\widehat{G}$ is a pseudorandom generator with expansion factor $n + p(n)$.

Increasing the expansion factor to $n + p(n)$

Theorem: If there exists a pseudorandom generator G with expansion factor $n+1$ then, for any polynomial p, $\widehat{G}$ is a pseudorandom generator with expansion factor $n + p(n)$.

Proof:

Increasing the expansion factor to $n + p(n)$

Theorem: If there exists a pseudorandom generator G with expansion factor $n+1$ then, for any polynomial p, $\widehat{G}$ is a pseudorandom generator with expansion factor $n + p(n)$.

Proof:

Define H_n^j to be the distribution on strings of length $n + p(n)$ output by the following process:

- Choose t_j u.a.r. from $\{0,1\}^{n+j}$
- Run $\widehat{G}$ starting from iteration $j+1$ of the for loop and returns its output

Increasing the expansion factor to $n + p(n)$

Theorem: If there exists a pseudorandom generator G with expansion factor $n+1$ then, for any polynomial p, $\widehat{G}$ is a pseudorandom generator with expansion factor $n + p(n)$.

Proof:

Define H_n^j to be the distribution on strings of length $n + p(n)$ output by the following process:

- Choose t_j u.a.r. from $\{0,1\}^{n+j}$
- Run $\widehat{G}$ starting from iteration $j+1$ of the for loop and returns its output

Note that: H_n^0 is the output distribution of $\widehat{G}(s)$ for a seed s choosen u.a.r. from $\{0,1\}^n$

Increasing the expansion factor to $n + p(n)$

Theorem: If there exists a pseudorandom generator G with expansion factor $n+1$ then, for any polynomial p, $\widehat{G}$ is a pseudorandom generator with expansion factor $n + p(n)$.

Proof:

Define H_n^j to be the distribution on strings of length $n + p(n)$ output by the following process:

- Choose t_j u.a.r. from $\{0,1\}^{n+j}$
- Run $\widehat{G}$ starting from iteration $j+1$ of the for loop and returns its output

Note that: H_n^0 is the output distribution of $\widehat{G}(s)$ for a seed s choosen u.a.r. from $\{0,1\}^n$

$H_n^{p(n)}$ is a string of length $p(n) + n$ chosen u.a.r. from $\{0,1\}^{n+p(n)}$

Increasing the expansion factor to $n + p(n)$

Theorem: If there exists a pseudorandom generator G with expansion factor $n + 1$ then, for any polynomial p, $\widehat{G}$ is a pseudorandom generator with expansion factor $n + p(n)$.

Proof:

Define H_n^j to be the distribution on strings of length $n + p(n)$ output by the following process:

- Choose t_j u.a.r. from $\{0,1\}^{n+j}$
- Run $\widehat{G}$ starting from iteration $j + 1$ of the for loop and returns its output

Note that: H_n^0 is the output distribution of $\widehat{G}(s)$ for a seed s choosen u.a.r. from $\{0,1\}^n$

$H_n^{p(n)}$ is a string of length $p(n) + n$ chosen u.a.r. from $\{0,1\}^{n+p(n)}$

We prove that if there exists a polynomial-time distinguisher $\widehat{D}$ (with non-negligible gap) for $\widehat{G}$, then there is a also a distinguisher D for G

Increasing the expansion factor to $n + p(n)$

Theorem: If there exists a pseudorandom generator G with expansion factor $n + 1$ then, for any polynomial p, $\widehat{G}$ is a pseudorandom generator with expansion factor $n + p(n)$.

Proof:

Define H_n^j to be the distribution on strings of length $n + p(n)$ output by the following process:

- Choose t_j u.a.r. from $\{0, 1\}^{n+j}$
- Run $\widehat{G}$ starting from iteration $j + 1$ of the for loop and returns its output

Note that: H_n^0 is the output distribution of $\widehat{G}(s)$ for a seed s choosen u.a.r. from $\{0, 1\}^n$

$H_n^{p(n)}$ is a string of length $p(n) + n$ chosen u.a.r. from $\{0, 1\}^{n+p(n)}$

We prove that if there exists a polynomial-time distinguisher $\widehat{D}$ (with non-negligible gap) for $\widehat{G}$, then there is a also a distinguisher D for G

Let D be a distinguisher such that:

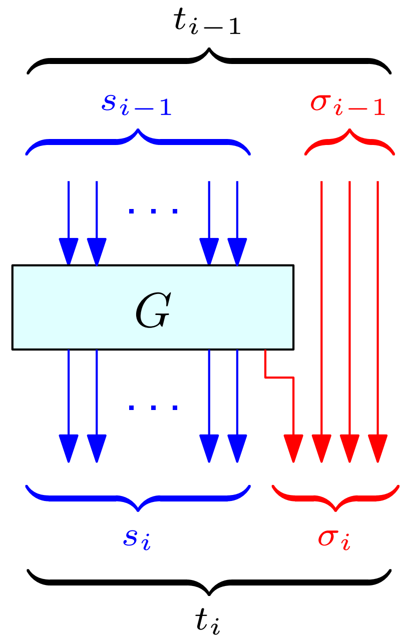
$$\left| \ \mathrm{Pr}_s[\widehat{D}(\widehat{G}(s))] - \mathrm{Pr}_r[\widehat{D}(r)] \ \right| = \varepsilon(n) \text{ for some non-negligible } \varepsilon(n)$$

Increasing the expansion factor to $n + p(n)$

Consider the following distinguisher D' for G:

Algorithm $D(w)$: (here $w \in \{0,1\}^{n+1}$)

- Choose j u.a.r. in $\{1, 2, \ldots, p(n)\}$
- Choose σ'_j u.a.r. in $\{0,1\}^{j-1}$
- Set $t_j = w \| \sigma'_j$ and run $\widehat{G}$ from iteration j to compute $t_{p(n)}$
- Run $\widehat{D}(t_{p(n)})$ and copy its output

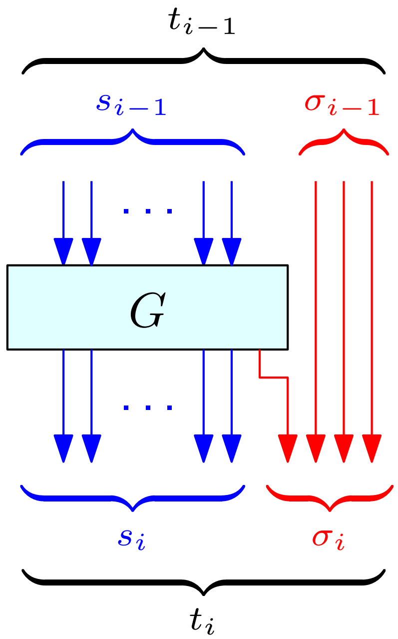


Increasing the expansion factor to $n + p(n)$

Consider the following distinguisher D' for G:

Algorithm $D(w)$: (here $w \in \{0,1\}^{n+1}$)

- Choose j u.a.r. in $\{1, 2, \ldots, p(n)\}$
- Choose σ'_j u.a.r. in $\{0,1\}^{j-1}$
- Set $t_j = w \| \sigma'_j$ and run $\widehat{G}$ from iteration j to compute $t_{p(n)}$
- Run $\widehat{D}(t_{p(n)})$ and copy its output



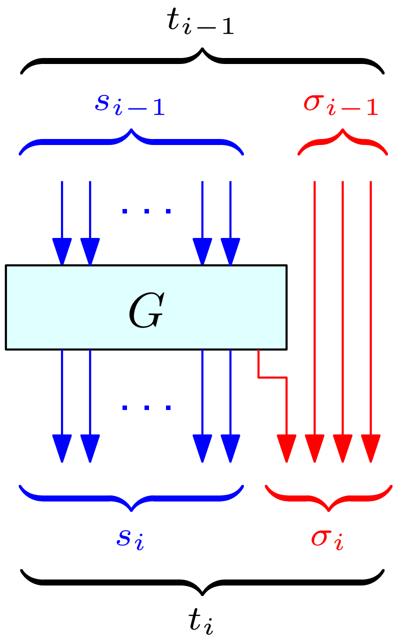
Fix $j^* \in \{1, 2, \ldots, p(n)\}$ and consider what happens when D chooses $j = j^*$

Increasing the expansion factor to $n + p(n)$

Consider the following distinguisher D' for G:

Algorithm $D(w)$: (here $w \in \{0,1\}^{n+1}$)

- Choose j u.a.r. in $\{1, 2, \ldots, p(n)\}$
- Choose σ'_j u.a.r. in $\{0,1\}^{j-1}$
- Set $t_j = w \| \sigma'_j$ and run $\widehat{G}$ from iteration j to compute $t_{p(n)}$
- Run $\widehat{D}(t_{p(n)})$ and copy its output



Fix $j^* \in \{1, 2, \ldots, p(n)\}$ and consider what happens when D chooses $j = j^*$

If w is a uniform string in $\{0,1\}^n$:

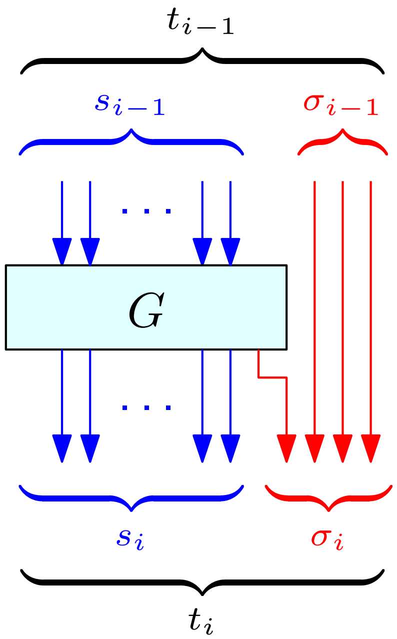
- Both w and σ'_j are chosen u.a.r., therefore t_{j^*} is a uniform string in $\{0,1\}^{n+j^*}$

Increasing the expansion factor to $n + p(n)$

Consider the following distinguisher D' for G:

Algorithm $D(w)$: (here $w \in \{0,1\}^{n+1}$)

- Choose j u.a.r. in $\{1, 2, \ldots, p(n)\}$
- Choose σ'_j u.a.r. in $\{0,1\}^{j-1}$
- Set $t_j = w \| \sigma'_j$ and run $\widehat{G}$ from iteration j to compute $t_{p(n)}$
- Run $\widehat{D}(t_{p(n)})$ and copy its output



Fix $j^* \in \{1, 2, \ldots, p(n)\}$ and consider what happens when D chooses $j = j^*$

If w is a uniform string in $\{0,1\}^n$:

- Both w and σ'_j are chosen u.a.r., therefore t_{j^*} is a uniform string in $\{0,1\}^{n+j^*}$
- The distribution of $t_{p(n)}$ is exactly $H_n^{j^*}$

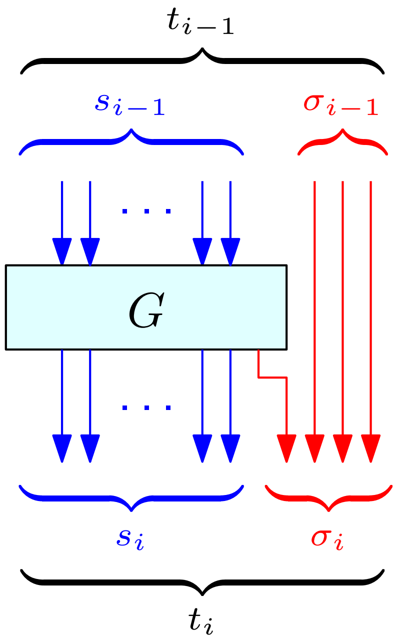
$$\Pr_r[D(r) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1]$$

Increasing the expansion factor to $n + p(n)$

Consider the following distinguisher D' for G:

Algorithm $D(w)$: (here $w \in \{0,1\}^{n+1}$)

- Choose j u.a.r. in $\{1, 2, \ldots, p(n)\}$
- Choose σ'_j u.a.r. in $\{0,1\}^{j-1}$
- Set $t_j = w \| \sigma'_j$ and run $\widehat{G}$ from iteration j to compute $t_{p(n)}$
- Run $\widehat{D}(t_{p(n)})$ and copy its output



Fix $j^* \in \{1, 2, \ldots, p(n)\}$ and consider what happens when D chooses $j = j^*$

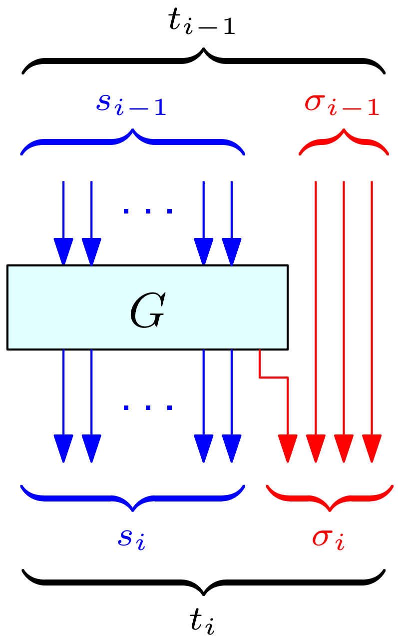
If w is the output of $G(s)$ on some seed s choosen u.a.r. from $\{0,1\}^n$:

Increasing the expansion factor to $n + p(n)$

Consider the following distinguisher D' for G:

Algorithm $D(w)$: (here $w \in \{0,1\}^{n+1}$)

- Choose j u.a.r. in $\{1, 2, \ldots, p(n)\}$
- Choose σ'_j u.a.r. in $\{0,1\}^{j-1}$
- Set $t_j = w \| \sigma'_j$ and run $\widehat{G}$ from iteration j to compute $t_{p(n)}$
- Run $\widehat{D}(t_{p(n)})$ and copy its output



Fix $j^* \in \{1, 2, \ldots, p(n)\}$ and consider what happens when D chooses $j = j^*$

If w is the output of $G(s)$ on some seed s choosen u.a.r. from $\{0,1\}^n$:

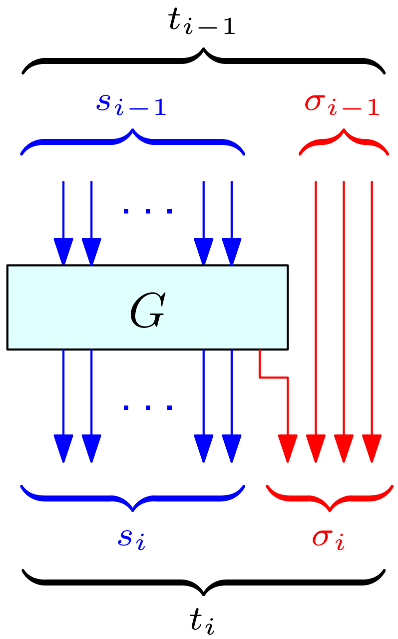
- Define $t_{j^*-1} = s \| \sigma'_j$ and notice that t_{j^*-1} is a uniform string in $\{0,1\}^{n+j^*-1}$

Increasing the expansion factor to $n + p(n)$

Consider the following distinguisher D' for G:

Algorithm $D(w)$: (here $w \in \{0,1\}^{n+1}$)

- Choose j u.a.r. in $\{1, 2, \ldots, p(n)\}$
- Choose σ'_j u.a.r. in $\{0,1\}^{j-1}$
- Set $t_j = w \| \sigma'_j$ and run $\widehat{G}$ from iteration j to compute $t_{p(n)}$
- Run $\widehat{D}(t_{p(n)})$ and copy its output



Fix $j^* \in \{1, 2, \ldots, p(n)\}$ and consider what happens when D chooses $j = j^*$

If w is the output of $G(s)$ on some seed s choosen u.a.r. from $\{0,1\}^n$:

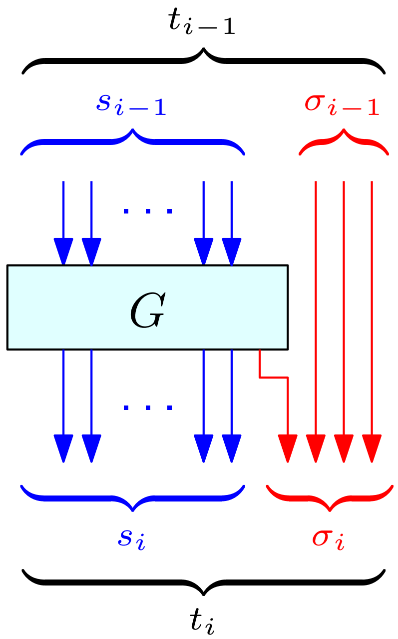
- Define $t_{j^*-1} = s \| \sigma'_j$ and notice that t_{j^*-1} is a uniform string in $\{0,1\}^{n+j^*-1}$
- Imagine running the j^*-th iteration of $\widehat{G}$. We would have $t_{j^*} = G(s) \| \sigma'_j = w \| \sigma'_j$

Increasing the expansion factor to $n + p(n)$

Consider the following distinguisher D' for G:

Algorithm $D(w)$: (here $w \in \{0,1\}^{n+1}$)

- Choose j u.a.r. in $\{1, 2, \dots, p(n)\}$
- Choose σ'_j u.a.r. in $\{0,1\}^{j-1}$
- Set $t_j = w \| \sigma'_j$ and run $\widehat{G}$ from iteration j to compute $t_{p(n)}$
- Run $\widehat{D}(t_{p(n)})$ and copy its output



Fix $j^* \in \{1, 2, \dots, p(n)\}$ and consider what happens when D chooses $j = j^*$

If w is the output of $G(s)$ on some seed s choosen u.a.r. from $\{0,1\}^n$:

- Define $t_{j^*-1} = s \| \sigma'_j$ and notice that t_{j^*-1} is a uniform string in $\{0,1\}^{n+j^*-1}$
- Imagine running the j^*-th iteration of $\widehat{G}$. We would have $t_{j^*} = G(s) \| \sigma'_j = w \| \sigma'_j$
- The distribution of $t_{p(n)}$ is exactly $H_n^{j^*-1}$

$$\Pr_s[D(G(s)) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*-1}}[\widehat{D}(t) = 1]$$

Increasing the expansion factor to $n + p(n)$

We have shown that:

$\Pr_r[D(r) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1]$

$\Pr_s[D(G(s)) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*-1}}[\widehat{D}(t) = 1]$

Increasing the expansion factor to $n + p(n)$

We have shown that:

$\Pr_r[D(r) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1]$ $\quad$ $\Pr_s[D(G(s)) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*-1}}[\widehat{D}(t) = 1]$

$$\Pr_r[D(r) = 1] = \sum_{j^*=1}^{p(n)} \Pr_r[D(r) = 1 \mid j = j^*] \cdot \Pr[j = j^*]$$

Increasing the expansion factor to $n + p(n)$

We have shown that:

$$\Pr_r[D(r) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1] \qquad \Pr_s[D(G(s)) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*-1}}[\widehat{D}(t) = 1]$$

$$\Pr_r[D(r) = 1] = \sum_{j^*=1}^{p(n)} \Pr_r[D(r) = 1 \mid j = j^*] \cdot \Pr[j = j^*] \quad = \frac{1}{p(n)} \sum_{j^*=1}^{p(n)} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1]$$

Increasing the expansion factor to $n + p(n)$

We have shown that:

$\Pr_r[D(r) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1]$ $\Pr_s[D(G(s)) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*-1}}[\widehat{D}(t) = 1]$

$$\Pr_r[D(r) = 1] = \sum_{j^*=1}^{p(n)} \Pr_r[D(r) = 1 \mid j = j^*] \cdot \Pr[j = j^*] \quad = \frac{1}{p(n)} \sum_{j^*=1}^{p(n)} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1]$$

$$\Pr_s[D(G(s)) = 1] = \sum_{j^*=1}^{p(n)} \Pr_s[D(G(s)) = 1 \mid j = j^*] \Pr[j = j^*]$$

Increasing the expansion factor to $n + p(n)$

We have shown that:

$$\Pr_r[D(r) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1] \qquad \Pr_s[D(G(s)) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*-1}}[\widehat{D}(t) = 1]$$

$$\Pr_r[D(r) = 1] = \sum_{j^*=1}^{p(n)} \Pr_r[D(r) = 1 \mid j = j^*] \cdot \Pr[j = j^*] \quad = \frac{1}{p(n)} \sum_{j^*=1}^{p(n)} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1]$$

$$\Pr_s[D(G(s)) = 1] = \sum_{j^*=1}^{p(n)} \Pr_s[D(G(s)) = 1 \mid j = j^*] \Pr[j = j^*] \quad = \frac{1}{p(n)} \sum_{j^*=1}^{p(n)} \Pr_{t \leftarrow H_n^{j^*-1}}[\widehat{D}(t) = 1]$$

Increasing the expansion factor to $n + p(n)$

We have shown that:

$\Pr_r[D(r) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1]$ $\qquad$ $\Pr_s[D(G(s)) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*-1}}[\widehat{D}(t) = 1]$

$$\Pr_r[D(r) = 1] = \sum_{j^*=1}^{p(n)} \Pr_r[D(r) = 1 \mid j = j^*] \cdot \Pr[j = j^*] \quad = \frac{1}{p(n)} \sum_{j^*=1}^{p(n)} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1]$$

$$\Pr_s[D(G(s)) = 1] = \sum_{j^*=1}^{p(n)} \Pr_s[D(G(s)) = 1 \mid j = j^*] \Pr[j = j^*] \quad = \frac{1}{p(n)} \sum_{j^*=0}^{p(n)-1} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1]$$

Increasing the expansion factor to $n + p(n)$

We have shown that:

$\Pr_r[D(r) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1]$ $\Pr_s[D(G(s)) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*-1}}[\widehat{D}(t) = 1]$

$$\Pr_r[D(r) = 1] = \sum_{j^*=1}^{p(n)} \Pr_r[D(r) = 1 \mid j = j^*] \cdot \Pr[j = j^*] \quad = \frac{1}{p(n)} \sum_{j^*=1}^{p(n)} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1]$$

$$\Pr_s[D(G(s)) = 1] = \sum_{j^*=1}^{p(n)} \Pr_s[D(G(s)) = 1 \mid j = j^*] \Pr[j = j^*] \quad = \frac{1}{p(n)} \sum_{j^*=0}^{p(n)-1} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1]$$

We can now bound:

$$\left| \Pr_s[D(G(s)) = 1] - \Pr_r[D(r) = 1] \right|$$

Increasing the expansion factor to $n + p(n)$

We have shown that:

$$\Pr_r[D(r) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1] \qquad \Pr_s[D(G(s)) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*-1}}[\widehat{D}(t) = 1]$$

$$\Pr_r[D(r) = 1] = \sum_{j^*=1}^{p(n)} \Pr_r[D(r) = 1 \mid j = j^*] \cdot \Pr[j = j^*] \quad = \frac{1}{p(n)} \sum_{j^*=1}^{p(n)} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1]$$

$$\Pr_s[D(G(s)) = 1] = \sum_{j^*=1}^{p(n)} \Pr_s[D(G(s)) = 1 \mid j = j^*] \Pr[j = j^*] \quad = \frac{1}{p(n)} \sum_{j^*=0}^{p(n)-1} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1]$$

We can now bound:

$$\Big| \Pr_s[D(G(s)) = 1] - \Pr_r[D(r) = 1] \Big| \quad = \left| \frac{1}{p(n)} \cdot \left(\sum_{j^*=1}^{p(n)} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1] - \sum_{j^*=0}^{p(n)-1} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1] \right) \right|$$

Increasing the expansion factor to $n + p(n)$

We have shown that:

$$\Pr_r[D(r) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1] \qquad \Pr_s[D(G(s)) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*-1}}[\widehat{D}(t) = 1]$$

$$\Pr_r[D(r) = 1] = \sum_{j^*=1}^{p(n)} \Pr_r[D(r) = 1 \mid j = j^*] \cdot \Pr[j = j^*] \quad = \frac{1}{p(n)} \sum_{j^*=1}^{p(n)} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1]$$

$$\Pr_s[D(G(s)) = 1] = \sum_{j^*=1}^{p(n)} \Pr_s[D(G(s)) = 1 \mid j = j^*] \Pr[j = j^*] \quad = \frac{1}{p(n)} \sum_{j^*=0}^{p(n)-1} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1]$$

We can now bound:

$$\left| \Pr_s[D(G(s)) = 1] - \Pr_r[D(r) = 1] \right| \quad = \left| \frac{1}{p(n)} \cdot \left(\sum_{j^*=1}^{p(n)} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1] - \sum_{j^*=0}^{p(n)-1} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1] \right) \right|$$

$$= \frac{1}{p(n)} \cdot \left| \Pr_{t \leftarrow H_n^{p(n)}}[\widehat{D}(t) = 1] - \Pr_{t \leftarrow H_n^{0}}[\widehat{D}(t) = 1] \right|$$

Increasing the expansion factor to $n + p(n)$

We have shown that:

$$\Pr_r[D(r) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1] \qquad \Pr_s[D(G(s)) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*-1}}[\widehat{D}(t) = 1]$$

$$\Pr_r[D(r) = 1] = \sum_{j^*=1}^{p(n)} \Pr_r[D(r) = 1 \mid j = j^*] \cdot \Pr[j = j^*] \quad = \frac{1}{p(n)} \sum_{j^*=1}^{p(n)} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1]$$

$$\Pr_s[D(G(s)) = 1] = \sum_{j^*=1}^{p(n)} \Pr_s[D(G(s)) = 1 \mid j = j^*] \Pr[j = j^*] \quad = \frac{1}{p(n)} \sum_{j^*=0}^{p(n)-1} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1]$$

We can now bound:

$$\left| \Pr_s[D(G(s)) = 1] - \Pr_r[D(r) = 1] \right| \quad = \left| \frac{1}{p(n)} \cdot \left(\sum_{j^*=1}^{p(n)} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1] - \sum_{j^*=0}^{p(n)-1} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1] \right) \right|$$

$$= \frac{1}{p(n)} \cdot \left| \Pr_{t \leftarrow H_n^{p(n)}}[\widehat{D}(t) = 1] - \Pr_{t \leftarrow H_n^{0}}[\widehat{D}(t) = 1] \right|$$

$$= \tfrac{1}{p(n)} \cdot \left| \Pr_r[\widehat{D}(r) = 1] - \Pr_s[\widehat{D}(\widehat{G}(s)) = 1] \right|$$

Increasing the expansion factor to $n + p(n)$

We have shown that:

$$\Pr_r[D(r) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1] \qquad \Pr_s[D(G(s)) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*-1}}[\widehat{D}(t) = 1]$$

$$\Pr_r[D(r) = 1] = \sum_{j^*=1}^{p(n)} \Pr_r[D(r) = 1 \mid j = j^*] \cdot \Pr[j = j^*] \quad = \frac{1}{p(n)} \sum_{j^*=1}^{p(n)} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1]$$

$$\Pr_s[D(G(s)) = 1] = \sum_{j^*=1}^{p(n)} \Pr_s[D(G(s)) = 1 \mid j = j^*] \Pr[j = j^*] \quad = \frac{1}{p(n)} \sum_{j^*=0}^{p(n)-1} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1]$$

We can now bound:

$$\Big| \Pr_s[D(G(s)) = 1] - \Pr_r[D(r) = 1] \Big| \quad = \left| \frac{1}{p(n)} \cdot \left(\sum_{j^*=1}^{p(n)} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1] - \sum_{j^*=0}^{p(n)-1} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1] \right) \right|$$

$$= \frac{1}{p(n)} \cdot \left| \Pr_{t \leftarrow H_n^{p(n)}}[\widehat{D}(t) = 1] - \Pr_{t \leftarrow H_n^{0}}[\widehat{D}(t) = 1] \right|$$

$$= \tfrac{1}{p(n)} \cdot \Big| \Pr_r[\widehat{D}(r) = 1] - \Pr_s[\widehat{D}(\widehat{G}(s)) = 1] \Big| \quad = \tfrac{\varepsilon(n)}{p(n)}$$

Increasing the expansion factor to $n + p(n)$

We have shown that:

$$\Pr_r[D(r) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1] \qquad \Pr_s[D(G(s)) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*-1}}[\widehat{D}(t) = 1]$$

$$\Pr_r[D(r) = 1] = \sum_{j^*=1}^{p(n)} \Pr_r[D(r) = 1 \mid j = j^*] \cdot \Pr[j = j^*] \quad = \frac{1}{p(n)} \sum_{j^*=1}^{p(n)} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1]$$

$$\Pr_s[D(G(s)) = 1] = \sum_{j^*=1}^{p(n)} \Pr_s[D(G(s)) = 1 \mid j = j^*] \Pr[j = j^*] \quad = \frac{1}{p(n)} \sum_{j^*=0}^{p(n)-1} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1]$$

We can now bound:

$$\left| \Pr_s[D(G(s)) = 1] - \Pr_r[D(r) = 1] \right| \quad = \left| \frac{1}{p(n)} \cdot \left(\sum_{j^*=1}^{p(n)} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1] - \sum_{j^*=0}^{p(n)-1} \Pr_{t \leftarrow H_n^{j^*}}[\widehat{D}(t) = 1] \right) \right|$$

$$= \frac{1}{p(n)} \cdot \left| \Pr_{t \leftarrow H_n^{p(n)}}[\widehat{D}(t) = 1] - \Pr_{t \leftarrow H_n^{0}}[\widehat{D}(t) = 1] \right|$$

Not negligible!

$$= \tfrac{1}{p(n)} \cdot \left| \Pr_r[\widehat{D}(r) = 1] - \Pr_s[\widehat{D}(\widehat{G}(s)) = 1] \right| \quad = \boxed{\frac{\varepsilon(n)}{p(n)}} \quad \lightning \quad \square$$

Pseudorandom permutations

To achieve CPA-security we need one more ingredient: **pseudorandom permutations** (PRPs)

Informal: A pseudorandom permutation is a pseudorandom function that is bijective

Pseudorandom permutations

To achieve CPA-security we need one more ingredient: **pseudorandom permutations** (PRPs)

Informal: A pseudorandom permutation is a pseudorandom function that is bijective

- Let Perm_n denote the set of all permutations in $\{0,1\}^n$, i.e., the set of all functions $F : \{0,1\}^n \to \{0,1\}^n$ that are bijective

- How big is Perm_n?

Pseudorandom permutations

To achieve CPA-security we need one more ingredient: **pseudorandom permutations** (PRPs)

Informal: A pseudorandom permutation is a pseudorandom function that is bijective

- Let Perm_n denote the set of all permutations in $\{0,1\}^n$, i.e., the set of all functions $F : \{0,1\}^n \to \{0,1\}^n$ that are bijective

- How big is Perm_n?

Think of a permutation F as a huge table in which all entries $F(x)$ are distinct:

	x	$F(x)$
	00...000	10...011
	00...001	01...010
2^n rows	00...010	00...110
	$\vdots$	$\vdots$
	11...111	10...001

Pseudorandom permutations

To achieve CPA-security we need one more ingredient: **pseudorandom permutations** (PRPs)

Informal: A pseudorandom permutation is a pseudorandom function that is bijective

- Let Perm_n denote the set of all permutations in $\{0,1\}^n$, i.e., the set of all functions $F : \{0,1\}^n \to \{0,1\}^n$ that are bijective
- How big is Perm_n?

Think of a permutation F as a huge table in which all entries $F(x)$ are distinct:

x	$F(x)$
`00...000`	`10...011`
`00...001`	`01...010`
`00...010`	`00...110`
$\vdots$	$\vdots$
`11...111`	`10...001`

2^n choices

2^n rows

Pseudorandom permutations

To achieve CPA-security we need one more ingredient: **pseudorandom permutations** (PRPs)

Informal: A pseudorandom permutation is a pseudorandom function that is bijective

- Let Perm_n denote the set of all permutations in $\{0,1\}^n$, i.e., the set of all functions $F : \{0,1\}^n \to \{0,1\}^n$ that are bijective
- How big is Perm_n?

Think of a permutation F as a huge table in which all entries $F(x)$ are distinct:

2^n rows

x	$F(x)$	
`00...000`	`10...011`	2^n choices
`00...001`	`01...010`	$2^n - 1$ choices
`00...010`	`00...110`	
$\vdots$	$\vdots$	
`11...111`	`10...001`	

Pseudorandom permutations

To achieve CPA-security we need one more ingredient: **pseudorandom permutations** (PRPs)

Informal: A pseudorandom permutation is a pseudorandom function that is bijective

- Let Perm_n denote the set of all permutations in $\{0,1\}^n$, i.e., the set of all functions $F : \{0,1\}^n \to \{0,1\}^n$ that are bijective

- How big is Perm_n?

Think of a permutation F as a huge table in which all entries $F(x)$ are distinct:

2^n rows

x	$F(x)$	
`00...000`	`10...011`	2^n choices
`00...001`	`01...010`	$2^n - 1$ choices
`00...010`	`00...110`	$\vdots$
$\vdots$	$\vdots$	
`11...111`	`10...001`	only 1 choice

Pseudorandom permutations

To achieve CPA-security we need one more ingredient: **pseudorandom permutations** (PRPs)

Informal: A pseudorandom permutation is a pseudorandom function that is bijective

- Let Perm_n denote the set of all permutations in $\{0,1\}^n$, i.e., the set of all functions $F : \{0,1\}^n \to \{0,1\}^n$ that are bijective
- How big is Perm_n?

Think of a permutation F as a huge table in which all entries $F(x)$ are distinct:

2^n rows

x	$F(x)$	
00...000	10...011	2^n choices
00...001	01...010	$2^n - 1$ choices
00...010	00...110	$\vdots$
$\vdots$	$\vdots$	
11...111	10...001	only 1 choice

$$\begin{aligned} |\mathsf{Perm}_n| &= 2^n \cdot (2^n - 1) \cdot \cdots \cdot 1 \\ &= (2^n)! \end{aligned}$$

Number of Permutations vs Number of Functions

Since a function $F \in \mathsf{Perm}_n$ is bijective, it must be **invertible**

$$F^{-1} \text{ exists and } F(x) = y \iff F^{-1}(y) = x$$

Number of Permutations vs Number of Functions

Since a function $F \in \mathsf{Perm}_n$ is bijective, it must be **invertible**

$$F^{-1} \text{ exists and } F(x) = y \iff F^{-1}(y) = x$$

What's the (asymptotic) proportion of functions in Func_n that are also permutations (i.e., invertible)?

Number of Permutations vs Number of Functions

Since a function $F \in \mathsf{Perm}_n$ is bijective, it must be **invertible**

$$F^{-1} \text{ exists and } F(x) = y \iff F^{-1}(y) = x$$

What's the (asymptotic) proportion of functions in Func_n that are also permutations (i.e., invertible)?

$$\lim_{n \to \infty} \frac{|\mathsf{Perm}_n|}{|\mathsf{Func}_n|}$$

Number of Permutations vs Number of Functions

Since a function $F \in \mathsf{Perm}_n$ is bijective, it must be **invertible**

$$F^{-1} \text{ exists and } F(x) = y \iff F^{-1}(y) = x$$

What's the (asymptotic) proportion of functions in Func_n that are also permutations (i.e., invertible)?

$$\lim_{n\to\infty} \frac{|\mathsf{Perm}_n|}{|\mathsf{Func}_n|} = \lim_{n\to\infty} \frac{(2^n)!}{2^{n2^n}}$$

Number of Permutations vs Number of Functions

Since a function $F \in \mathsf{Perm}_n$ is bijective, it must be **invertible**

$$F^{-1} \text{ exists and } F(x) = y \iff F^{-1}(y) = x$$

What's the (asymptotic) proportion of functions in Func_n that are also permutations (i.e., invertible)?

$$\lim_{n\to\infty} \frac{|\mathsf{Perm}_n|}{|\mathsf{Func}_n|} = \lim_{n\to\infty} \frac{(2^n)!}{2^{n2^n}} = \lim_{t\to\infty} \frac{t!}{t^t}$$

$t = 2^n$

Number of Permutations vs Number of Functions

Since a function $F \in \mathsf{Perm}_n$ is bijective, it must be **invertible**

$$F^{-1} \text{ exists and } F(x) = y \iff F^{-1}(y) = x$$

What's the (asymptotic) proportion of functions in Func_n that are also permutations (i.e., invertible)?

$$\lim_{n\to\infty} \frac{|\mathsf{Perm}_n|}{|\mathsf{Func}_n|} = \lim_{n\to\infty} \frac{(2^n)!}{2^{n2^n}} \underbrace{=}_{t = 2^n} \lim_{t\to\infty} \frac{t!}{t^t} = \lim_{t\to\infty} \frac{\sqrt{2\pi t} \cdot t^t/e^t}{t^t}$$

Stirling's approximation: $t! \sim \sqrt{2\pi t}\left(\frac{t}{e}\right)^t$

Number of Permutations vs Number of Functions

Since a function $F \in \mathsf{Perm}_n$ is bijective, it must be **invertible**

$$F^{-1} \text{ exists and } F(x) = y \iff F^{-1}(y) = x$$

What's the (asymptotic) proportion of functions in Func_n that are also permutations (i.e., invertible)?

$$\lim_{n\to\infty} \frac{|\mathsf{Perm}_n|}{|\mathsf{Func}_n|} = \lim_{n\to\infty} \frac{(2^n)!}{2^{n2^n}} \underset{t=2^n}{=} \lim_{t\to\infty} \frac{t!}{t^t} = \lim_{t\to\infty} \frac{\sqrt{2\pi t}\cdot t^t/e^t}{t^t} = \lim_{t\to\infty} \frac{\sqrt{2\pi t}}{e^t}$$

Stirling's approximation: $t! \sim \sqrt{2\pi t}\left(\dfrac{t}{e}\right)^t$

Number of Permutations vs Number of Functions

Since a function $F \in \mathsf{Perm}_n$ is bijective, it must be **invertible**

$$F^{-1} \text{ exists and } F(x) = y \iff F^{-1}(y) = x$$

What's the (asymptotic) proportion of functions in Func_n that are also permutations (i.e., invertible)?

$$\lim_{n\to\infty} \frac{|\mathsf{Perm}_n|}{|\mathsf{Func}_n|} = \lim_{n\to\infty} \frac{(2^n)!}{2^{n2^n}} \underset{t = 2^n}{=} \lim_{t\to\infty} \frac{t!}{t^t} = \lim_{t\to\infty} \frac{\sqrt{2\pi t} \cdot t^t/e^t}{t^t} = \lim_{t\to\infty} \frac{\sqrt{2\pi t}}{e^t} = 0$$

Stirling's approximation: $t! \sim \sqrt{2\pi t}\left(\dfrac{t}{e}\right)^t$

Asymptotically, almost no function in Func_n is a permutation!

Keyed permutations

A **keyed permutation** is a keyed function $F : \{0,1\}^{\ell_{key}(n)} \times \{0,1\}^{\ell_{in}(n)} \to \{0,1\}^{\ell_{out}(n)}$ such that:

- $\ell_{in}(n) = \ell_{out}(n)$ (this quantity is called the **block length**); and
- For every $k \in \{0,1\}^{\ell_{key}(n)}$, the function $F_k(x) = F(k,x)$ is a permutation

Keyed permutations

A **keyed permutation** is a keyed function $F : \{0,1\}^{\ell_{key}(n)} \times \{0,1\}^{\ell_{in}(n)} \to \{0,1\}^{\ell_{out}(n)}$ such that:

- $\ell_{in}(n) = \ell_{out}(n)$ (this quantity is called the **block length**); and
- For every $k \in \{0,1\}^{\ell_{key}(n)}$, the function $F_k(x) = F(k,x)$ is a permutation

A keyed permutation is **efficient** if:

- There is a polynomial-time algorithm that computes $F(x)$ given x; and
- There is a polynomial-time algorithm that computes $F^{-1}(y)$ given y

Pseudorandom permutations, formal definition

Definition: An efficient, length preserving, keyed function $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is a **pseudorandom permutation** if for all probabilistic polynomial-time distinguishers D, there is a negligible function ε such that:

$$\left| \Pr[D^{F_k(\cdot)}(1^n) = 1] - \Pr[D^{f(\cdot)}(1^n) = 1] \right| \leq \varepsilon(n)$$

Probability over the randomness of the distinguisher and the choice of k

Probability over the randomness of the distinguisher and the uniform choice of $f \in \mathsf{Perm}_n$

Pseudorandom permutations, formal definition

Definition: An efficient, length preserving, keyed function $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is a **pseudorandom permutation** if for all probabilistic polynomial-time distinguishers D, there is a negligible function ε such that:

$$\left|\ \Pr[D^{F_k(\cdot)}(1^n) = 1] - \Pr[D^{f(\cdot)}(1^n) = 1]\ \right| \leq \varepsilon(n)$$

Probability over the randomness of the distinguisher and the choice of k

Probability over the randomness of the distinguisher and the uniform choice of $f \in \mathsf{Perm}_n$

Intuitition: a keyed permutation is pseudorandom permutation if no polynomial-time algorithm can distinguish it from a random permutation

Pseudorandom permutations

Recall that (asymptotically) almost no function in Func_n is a permutation

Nevertheless:

- As soon as $\ell_{in}(n) \geq n$, a PRP is indistinguishable (in polynomial time, with non-negligible gap) from PRF
- Since a PRF is indistinguishable from a random function, this implies that PRPs with $\ell_{in}(n) \geq n$ are also indistinguishable from random functions!

Strong pseudorandom permutations

Sometimes we need even even “stronger” functions than pseudorandom permutation

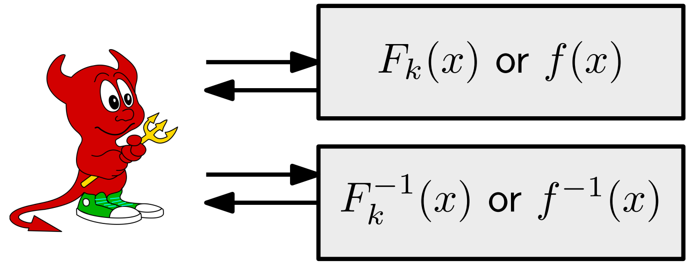
The adversary might be able to exploit the fact that a pseudorandom permutation is invertible to gain a non-negligible advantage

Strong pseudorandom permutations

Sometimes we need even even "stronger" functions than pseudorandom permutation

The adversary might be able to exploit the fact that a pseudorandom permutation is invertible to gain a non-negligible advantage

We define **strong** pseudorandom permutations that are indistinguishable from random permutation even if the adversary has oracle access to **both** the permutation and its inverse

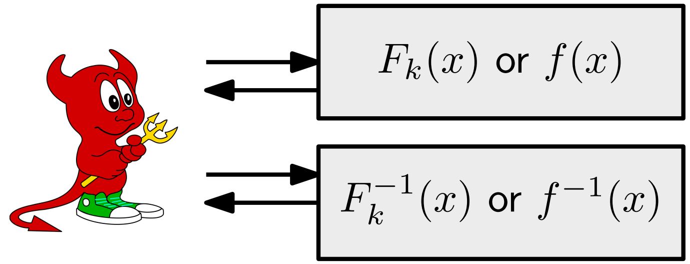


Strong pseudorandom permutations

Sometimes we need even even “stronger” functions than pseudorandom permutation

The adversary might be able to exploit the fact that a pseudorandom permutation is invertible to gain a non-negligible advantage

We define **strong** pseudorandom permutations that are indistinguishable from random permutation even if the adversary has oracle access to **both** the permutation and its inverse



Definition: An efficient, length preserving, keyed function $F : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ is a **strong pseudorandom permutation** if for all probabilistic polynomial-time distinguishers D, there is a negligible function ε such that:

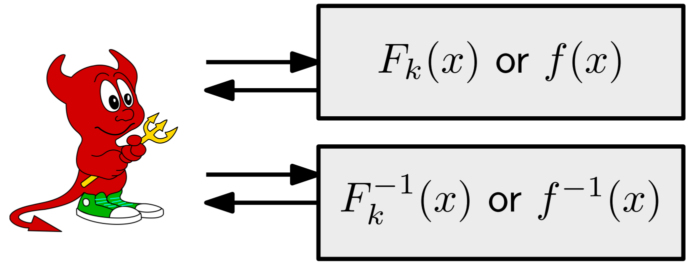
$$\left|\ \Pr[D^{F_k(\cdot),F_k^{-1}(\cdot)}(1^n) = 1] - \Pr[D^{f(\cdot),f^{-1}(\cdot)}(1^n) = 1]\ \right| \leq \varepsilon(n)$$

Strong pseudorandom permutations

Sometimes we need even even "stronger" functions than pseudorandom permutation

The adversary might be able to exploit the fact that a pseudorandom permutation is invertible to gain a non-negligible advantage

We define **strong** pseudorandom permutations that are indistinguishable from random permutation even if the adversary has oracle access to **both** the permutation and its inverse



Definition: An efficient, length preserving, keyed function $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is a **strong pseudorandom permutation** if for all probabilistic polynomial-time distinguishers D, there is a negligible function ε such that:

$$\left| \Pr[D^{F_k(\cdot), F_k^{-1}(\cdot)}(1^n) = 1] - \Pr[D^{f(\cdot), f^{-1}(\cdot)}(1^n) = 1] \right| \leq \varepsilon(n)$$

Strong pesudorandom permutations