Chapter 7 | Theoretical Constructions of Symmetric-Key Primitives

Wednesday, September 27, 2023 9:22 AM

Story:

- In chapter 3,
  we introduced the notion of pseudorandomness
  and
  defined some basic crypto primitives
  including
  PRGs, PRFs and PRP (pseudorandom permutations).

  We showed in Chapter 3 and 4
  that these primitives serve as the
  building blocks for all private-key crypto

  As such, it is of great improtance to
  understand these from a theoretical point of view

  In this chapter
  we formally introduce the concept of
  one-way functions—functions that are
  easy to compute but
  hard to invert
  and how PRGs PRFs and PRPs can be constructed
  from the sole assupmtion that
  one-way functions exist
  (This is not quite true
  since we are for the most part going to rely on
  one-way *permutations* in this chapter
  But it is known that one-way functions suffice.)

  Morevore
  we'll see that one-way functions are
  necessary for "non-trivial" private key crypto.

  i.e. the existence of one-way functions

  iff

  the existince of all (non-trivial) private-key cryptography.

The constructions we show in this chapter
should be viewed as complementary to the
constructions of stream ciphers and block ciphers
discussed in the previous chapter (DID NOT READ □).

The focus of the previous chapter was
how various crypto primitives are currently realised in practice
and to introduce some basic approaches and design principles
that are used.

Somewhat dissappointing was the fact that
none of the constructions we showed
could be proven secure
under any weaker (i.e. more reasonable) assumptions.

In contrast
in the present chapter we will
prove that it is possible to construct
PRPs starting from the very mild assumption that
one-way functions exist.

This assumption is more palatble than
say
assuming that AES is a pseudorandom permutation
both
because it is a qualitatively weaker assumption and
also because
we have a number of candidate,
number-theoretic one-way functions
that have been studied for

many years
even before the advent of cryptography
(see the very beginning of Chapter 6 for further discussion on this point).

The downside however is
that the constructinos we show here are
all far less efficient than
those of Chapter 6 and thus
are not actually used.

It remains an important challenge
for cryptographers to "bridge this gap" and
develop provably secure constructions of
pseudorandom generators, functions, and
permutations whose efficiency is
comparable to the best available stream
cipher and block ciphers.

Collision resistant hash functions

In contrast to the previous chapter
here we do not consider collision-resistant hash functions.

The reason is that
constructions of such hash functions from
one-way functions are unknown and
in fact
there is evidence suggesting
that such constructions are impossible.

We will turn to provable constructions of collision-resistant hash function
based on specific number theoretic assumptinos
in Section 8.4.2

<

Pseudorandom states and
Collision Resistant "Quantum
Hash function"

7.1 One-Way Functions

Story:

- In this section we formally define one-way functions
and then briefly discuss candidates
that satisfy this definition.
    - We see more examples of conjectured OWFs in Ch 8
    - We next introduce the notion of
    hard-core predicates
    which can be viewed as
    encapsulating the hardness of inverting a
    one-way function and
    will be used extensively in the
    constructions that follow in subsequent sections.

7.1.1 Definition

- A OWF $f: \{0,1\}^* \to \{0,1\}^*$ is
(a) easy to compute
(b) yet hard to invert.
    - The condition (a) is easy to formalise
    we simply require that f be computable in poly time.
    - We are ultimately interested in building schemes
    that are hard for
    a probabilistic poly time adversary to break (except with negl prob).
    - Therefore we formalise the condition (b) as
    it be infeasible for any PPT algorithm to invert f
    i.e. find a preimage of a given value of y
    (except with negligible probability).

A technical point is that
this probability is taken over
an experiment in which
y is generated by choosing a
uniform element x of the domain of f
and then setting $y := f(x)$

(rather than choosing y uniformly from the range of f).

The reason for this should become clear
from the constructinos we will see
in the remainder of the chapter.

- Let $f: \{0,1\}^* \to \{0,1\}^*$ be a function.
  Consider the following experiment for
  any algorithm $\mathcal{A}$ and any value n for the security parameter:

$\text{Invert}_{\mathcal{A},f}(n)$

$\mathcal{C}$ $\mathcal{A}$

1. $x \leftarrow \{0,1\}^n$
   $y = f(x)$
2. $1^n, y \rightarrow$
   $\leftarrow x'$
3. out 1 if $f(x') = y = f(x)$
   0 else

We stress that $\mathcal{A}$ need not
find the original pre-image x

it sufficies for $\mathcal{A}$ to find any value x'
for which $f(x') = y = f(x)$.

We give the security parameter 1^n to $\mathcal{A}$
in the second step to stress that
$\mathcal{A}$ may run in time poly in
the security prameter n
regardless of the length of y.

Definition 7.1:
A function $f: \{0,1\}^* \to \{0,1\}^*$ is one-way if
the following two conditions hold:

1. (Easy to compute)
   There exists a poly-time algorithm M_f computing f
   i.e. $M_f(x) = f(x)$ for all x.

2. (Hard to invert)
   For every PPT algorithm $\mathcal{A}$
   there is a negligible function negl such that
   $\Pr[\text{Invert}_{\mathcal{A},f}(n) = 1] \leq \text{negl}(n)$.

Notation.
In this chapter we will often make
the probability space more explicit
by subscripting (part of) it
in the probability notation.

For example
we can succinctly express the
second requirement ni th edefinition above
as follows:
For every PPT algorithm $\mathcal{A}$,
there is a negligible function negl such that

$\Pr_{x \leftarrow \{0,1\}^n}[\mathcal{A}(1^n, f(x)) \in f^{-1}(f(x))] \leq \text{negl}(n)$.

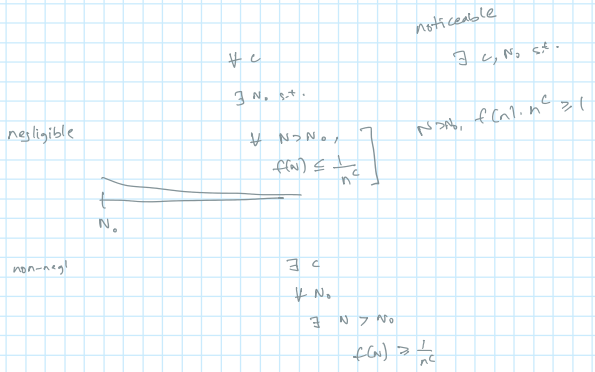
The probabiliy above
is also taken over the
randomness used by $\mathcal{A}$
which here is left implicit.

Successful inversion of one-way functions.

A function that is not one-way is
not necessarily easy to invert all the time
(or even "often").

Rather,
the converse of the second condition of
Definition 7.1 is that
there exists a PPT algorithm $\mathcal{A}$
and a non-negligible function γ
such that
$\mathcal{A}$ inverts $f(x)$ with probability at least $\gamma(n)$

(where the probability is taken over
uniform choice of $x \in \{0,1\}^n$
and
the randomness of $\mathcal{A}$)



This means
in turn
that there exists a positive polynomial $p(\cdot)$
such that
for *infinitely many values of* n,
algorithm $\mathcal{A}$ inverts f with probability at least $1/p(n)$.

Thus
if there exists an $\mathcal{A}$ that inverts f
with probability n^{-10}
for all even values of n
(but always fails to invert f when n is odd),
then f is not one-way
even though $\mathcal{A}$ only succeeds on
half the values of n and
only succeeds with probability n^{-10}
(for values of n
where it succeeds at all).

Exponential-time inversion.
Any one-way function
can be inverted at any point y in exponential time
by simply trying all values $x \in \{0,1\}^n$ until
a value x is found such that $f(x) = y$.

Thus
the existence of one-way functions is
inherently an
assumption about
computational complexity and
computational hardness.

i.e.
it concerns a problem that can be solved
in principle but
is assumed to be hard to solve efficiently.

One-way permutations.
We will often be interested in
one-way functions
with additional structural properties.

We say a function f is *length preserving* if
$|f(x)| = |x|$ for all x.

A one-way function that is
(a) length preserving
and
(b) one-to-one is called a
one-way permutation.

If f is a one-way permutation
then any value y
has unique preimage $x = f^{-1}(y)$.

Neverthless
it is still hard to find x in poly time.

One-way function/permutation families

The above definitions of one-way functions
and permutations are convenient
in that they consider a single function
over an infinite domain and range.

However
most candidate one-way functions and permutatinos
don't fit neatly into this framework.

Instead,
there's an algorithm that generates some
set I of parameters which
define a function f_I;
one wayness here means
essentially that
f_I sholud be one way with
all but negligible probability (over the choice of I)

Because
each value of I defines a different function
we now refer to *families* of
one-way functions (resp. permutations).

Definition 7.2

A tuple $\Pi = (\text{Gen}, \text{Samp}, f)$ of PPT algorithms is
a *function family* if the following hold:

1. The
parameter-generation algorithm Gen
on input 1^n
outputs pramaters I (with $|I| > n$).

Each value of I output by Gen defines
$\mathcal{D}_I$ and $\mathcal{R}_I$
that constitute the domain and range
(resp.)
for a function f_I.

2. The sampling algorithm Samp
on
input I,
outputs
a uniformly distributed element of $\mathcal{D}_I$

3. The deterministic
evaluation algorithm f
on input I and $x \in \mathcal{D}_I$
outputs an element $y \in \mathcal{R}_I$.

We write this as $y := f_I(x)$.

Π is a permutation family if
for each value I output by $\text{Gen}(1^n)$
that
(a) $\mathcal{D}_I = \mathcal{R}_I$
and
(b) the function $f_I: \mathcal{D}_I \to \mathcal{D}_I$ is one-to-one (equivalently, in this case a bijection).

Let Π be a function family.

What follows is the natural analogue of the experiment introduced earlier.

The inverting Experiment

$\text{Invert}_{A,\Pi}(n)$:

1. $I \leftarrow Gen(1^n)$

$x \leftarrow Samp(I)$

(samples from D_I uniformly)

$y := f_I(x)$

2. $I, y \rightarrow$

$\leftarrow x'$

3. out 1 if $f_I(x') = y$

0 else

Definition 7.3

A function/permutation family $\Pi = (\text{Gen}, \text{Samp}, f)$
is one-way if for all PPT algorithms $\mathcal{A}$
there is a negligible function
negl such that

$$\Pr[\text{Invert}_{\mathcal{A},\Pi}(n) = 1] \leq \text{negl}(n).$$

Story:

Throughout this chapter
we work with OWF/OWP over an infinite domain
(as in Definition 7.1)
rather than worknig with
families of OWFs/OWPs.

This is primarily for convenience
(does not significantly affect any of the results; Ex 7.7).

7.1.2 Candidate One-Way Functions

Story:

- One-way functions are of interest only if they exist.
  - We do not know how to prove they exist unconcditionally
    (this would be a major breakthrough in complexity theory)

    so we must conjecture/assume their existence.
  - Such a conjecture is based on the fact thta
    several natural computational problems
    have receieved much attention
    and yet
    have no PPT algorithm for solving them.
  - Perhaps th emsot famous such problem
    is integer fctorisation
    i.e. finding the prime factors of a large integer.
  - It is easy to multiply two numbers
    and obtain their product
    but difficult to take a number
    and find its factors.
  - This leads us to define the function
    $f_{mult}(x, y) = x \cdot y$.
    - If we don't place any restriction on
      the lengths of x and y
      then f_{mult} is easy to invert

      with high prob. xy is even
      and so $(2, \frac{xy}{2})$ is an iverse
    - this issue can be addressed
      by restricting the domain of f_{mult}
      to equal length primes x and y
    - Idea discussed again in Section 8.2
- Another candidate OWF

< Kishor: Check connection NP completeness

- Idea discussed again in Section 8.2

- Another candidate OWF
  not relying directly on number theory
    is based on the
  subset-sum problem and is defined by

  $$f_{ss}(x_1 \dots x_n, J) = \left(x_1 \dots x_n, \left[\sum_{j \in J} x_j \bmod 2^n\right]\right)$$

  where each x_i is an n-bit string
    intrepreted as an integer and
  J is an n-bit string interpreted as
    specifying a subset of $\{1 \dots n\}$.

  Inverting f_{ss} on an output
    $(x_1 \dots x_n, y)$ requires finding a subset
    $J' \subseteq \{1 \dots n\}$
      such that

  $$\sum_{j \in J'} x_j = y \bmod 2^n$$

  Students who have studied NP-completeness
    may recall that this problem NP-complete.

  But even $P \neq NP$
    does not imply that f_{ss} is one way:

    $P \neq NP$ wolud mean thta
      every PPT algorithm
    fails to solve the subset sum problem
      on *at least* one input
    whereas for f_{ss} to be a OWF
      it is required that every PPT algorithm
    fails to solve the subset sum problem
      (at lesat for certain parameters)
        almost always.

  Thus our belief that the
    function above is one-way is
      based on the lack of known algorithms
    to solve this problem even with "small" probability
      on random inputs
    and not merely on the fact that the problem is NP complete.

< Kishor: Check connection NP completeness

- We conclude by showing
    a family of *permutations* that is
      believed to be one-way

  - Let Gen be a PPT algorithm:
      input: 1^n
      output:
        n-bit prime p and
        $g \in \{2 \dots p-1\}$ (a special element).

    Require: the element g should be a generator of $\mathbb{Z}_p^*$

  - Let Samp be an algorithm that
      Input: p, g (numbers); (me: g seems redundant here)
      output: $x \in \{1 \dots p-1\}$.

  - Definition:
      $f_{p,g}(x) = [g^x mod\ p]$

        (assertion: f_{pg} can be computd efficiently,
          follows from the results in Appendix B.2.3)

  - Claims:
    - It can be shown that this function is one-to-one
        and thus a permutation.
    - The presumed difficulty of inverting this funtion
        is based on the conjectured hardness
      of the discrete-log problem
        (We'll say more about this in Section 8.3)

- Remarks
  - Very efficient OWF can be obtained from

practical crypto constructions such as
SHA1 or AES under the assumption that
they are collision resistant
or
pseudorandom permutation
respectively;

- Technically speaking
they cannot satisfy the defintiion of OWFs since
they have fixed length i/o
and so one cannot look at their asymptotic behaviour
Nonetheless,
it's plausible to conjecture they are OW in a concrete sense.

7.1.3 Hard-core Predicates

- Story:
By definition
a OWF is hard to invert.

- Stated differently:
  - given $y = f(x)$
  the value x cannot be computed in its entirety
  by any PPT algorithm
  (except with negilgible prob; we ignore this here).

  - One might get the impression that
  nothing about x can be determined from
  $f(x)$ in poly time.

  - This is *not* necessarily the case

  - Indeed, it is possible for $f(x)$ to "leak" a lot of information about
  x even if f is one-way.

  - For a trivial example
  let g be a one-way function and define
  $f(x_1, x_2) := (x_1, g(x_2))$
  where $|x_1| = |x_2|$.

  - It is easy to show that f is also a OWF
  (this is straightforward)
  even though it reveals half its input.

- For our applications
we will need to identify a specific piece of inforrmation
about x that is "hidden" by $f(x)$.

- This motivates the notion of a "hardcore predicate"
  - A hard-core predicate $\mathrm{hc}: \{0,1\}^* \to \{0,1\}$ of a function f
  has the property that $\mathrm{hc}(x)$ is
  hard to compute with probability
  significantly better than $1/2$ given $f(x)$.

  - Since hc is a boolean function
  it is always possible to compute $\mathrm{hc}(x)$
  with probability $1/2$ by random guessing.

Formally:

Definition 7.4

A function $\mathrm{hc}: \{0,1\}^* \to \{0,1\}$ is
a hard-core predicate of a function f if

hc can be computed in poly time and

for every PPT algorithm $\mathcal{A}$

there is a negl such that

$$\Pr_{x \leftarrow \{0,1\}^n}\left[\mathcal{A}(1^n, f(x)) = \mathrm{hc}(x)\right] \leq \frac{1}{2} + \mathrm{negl}(n)$$

where the probability is taken over the uniform chocie of x in $\{0,1\}^n$
and
the randomness of $\mathcal{A}$.

Remarks:

- We stress that $\mathrm{hc}(x)$ is efficiently computable given x
  (since the function hc can be computed in PT).
  - The defintiion requires that $\mathrm{hc}(x)$
    is hard to compute given $f(x)$
- The above definition does not require
  f to be a OWF/OWP.

  if f is a permutation
  however
  then it cannot have a hard-core predicate
  unless it is one-way.

  (Exercise 7.13) < Exercise 7.13

Simple ideas don't work.

- Consider the predicate
  $\mathrm{hc}(x) := \bigoplus_{i=1}^{n} x_i$
  where $x_1 \ldots x_n$ denotes the bits of x

  One might hope that this is a hard-core predicate of
  any OWF f:
  if f cannot be inverted
  then $f(x)$ must hide at least
  one of th ebits x_i of its preimage x
  which would seem to imply that the
  XOR of all the bits of x is hard to compute.

  Despite its appeal
  this argument is incorrect.

  To see this
  let g be a OWF and define
  $f(x) := (g(x), \bigoplus_i x_i)$

  It is not hard to show that f is OW
  (suppose $\mathcal{A}$ inverts f; feed it $g(x)$ and simply guess the
  second input; use both answers x', x'' produced by $\mathcal{A}$
  and check if $g(x') = g(x)$ or $g(x'') = g(x)$)

  However
  it is clear that $f(x)$ does not hid the value of $\mathrm{hc}(x) = \bigoplus_i x_i$

  because this is part of its output

  therefore $\mathrm{hc}(\mathrm{x})$ is not a hard-core predicate of f.

  Extending this,
  one can show that for any fixed predicate hc
  there is always a OWF f
  for which hc is not a hard-core predicate of f.

Trivial hard-core predicates.

- Some functions have "trivial" hard-core predicates.
  E.g. let f be the function that drops the last bit of its input

  i.e. $f(x_1 \ldots x_n) = x_1 \ldots x_{n-1}$

  It is hard to determine x_n given $f(x)$
  since $\mathrm{x_n}$ is independent of the output

  thus, $\mathrm{hc}(x) = x_n$ is a hard-core predicate of f.
- However, f is not one-way
- When we use hard-core predicates for our constructions
  for our constructinos

it will become clear why trivial hard-core predicates
this sort are of no use.

7.2 From One-Way Functions to Pseudorandomness

Story:

- The goal of this chapter is to show how to construct
  PRGs, PRF/PRPs
  from
  OWF/OWPs

  (pseudorandom generators
  functions
  and
  permutations
  based on any OWF/OWP).

  - In this section
    we give an overview of these constructions.

  - Details are given in the sections that follow.

A hard-core predicate from any one-way function

Story:

The first step is to show that
a hard-core predicate exists for
any OWF.

Actually
it remains open whether this is true

We show something weaker that suffices for our purposes.

i.e. we show that given a OWF f
we can construct a *different* OWF g
along with a hard-core predicate of g.

Theorem 7.5 (Goldreich-Levin theorem).
Assume one-way functions (resp. permutations) exist.

Then there exists
a one-way function (resp. permutation) g
and
a hard-core predicate hc of g.

Construction:

- Let f be a one-way function.
  Functions g and hc are constructed as follows:

  set $g(x,r) := (f(x),r)$ for $|x| = |r|$

  and define

  $\mathrm{hc}(x,r) := \bigoplus_i x_i \cdot r_i$.

  Here, x_i denotes the ith bit of x (similarly for r).

- NB:
  if r is uniform
  then $\mathrm{hc}(x,r)$ outputs the XOR
  of a random subset of the bits of x

  (When $r_i = 1$ the bit x_i is included in the XOR
  otherwise it is not).

Story:

- The Goldreich-Levin theorem, essentially states,
  that if f is a OWF then
  $f(x)$ hides the XOR of a *random subset* of the bits of x.

Pseudorandom generators from one-way permutations.

- The next step is to show
  a hard-core predicate of a one-way *permutation*
  can be used to construct a pseudorandom generator
  (It is known that a hard-core predicate of
  a OW *function* suffices
  but the proof is extremely complicated and
  beyond th escope of this book).

- Specifically, we show:

Theorem 7.6
Let
- f be a OW permutation and
- hc be a hard-core predicate of f.

Then
$G(s) := f(s)||\text{hc}(s)$
is a pseudorandom generator
with expansion factor $\ell(n) = n + 1$.

Story:
- As intuition for why G as defined in the theorem
  constitutes a PRG
  note first that the initial n bits of the output of $G(s)$
  (i.e. the bits of $f(s)$) are
  truly uniformly distributed when s is uniformly distributed
  by virtue of the fact that f is a permutation.

  - Next
    the fact that hc is a hard-core predicate of f
    means that $\text{hc}(s)$ "looks random"
    i.e. is pseudorandom
    even given $f(s)$
    (assuming again that s is uniform).

  - Putting these observations together
    we see that the entire output of G is pseudorandom.

Pseduorandom generators with arbitrary expansion.

Story:
- The existence of a PRG that stretches its seed
  by even a single bit (as we have jsut seen)
  is already highly non-trivial.

  - But for applications
    (e.g. for efficient encryption of large messages as in Section 3.3)
    we need a pseudorandm generator with
    much larger expansion.

  - Fortunately, one can obtain any poly expansion factor we want.

Theorem 7.7

If there exists a PRG (pseudorandom generator) with expansion factor $\ell(n) = n + 1$
then for any polynomial poly there exists a PRG
with expansion factor $\text{poly}(n)$.

Story:
- We conclude that pseudorandom generators with arbitrary (poly)
  expansion can be constructed from any one-way permutation.

Pseudorandom functions/permutations from pseudorandom generators.

- *Pseudorandom generators* suffice for
  constructing EAV-secure private-key encryption schemes

- For achieving CPA-secure privatake-key encryption
  (not to mention message authentication codes), however,
  we relied on *pseudo-random functions*.

- The following shows that the latter can be obtained from the former

Ciphertext only
Known plaintext attack [Eav]
CPA
CCA

Theorem 7.8

If there exists a pseudorandom generator with expansion factor $\ell(n) = 2n$
then
there exists a pseudorandom function.

Story:

- In fact we can do even more:

Theorem 7.9

If there exists a PRF, then there exists a strong pseudorandom permutation.

Story:

- Combining all the above theorems
  as well as the results of Chapter 3 and 4
  we have the following corollaries:

Corollary 7.10

Assuming the existence of one-way permutations
there exist

- pseudorandom generators with any poly expansion factor,
- PRFs
- strong pseudorandom permutations.

Corollay 7.11

Assuming the existence of one-way permutations
there exist

- CCA-secure private-key encryption schemes and
- secure message authentication codes.

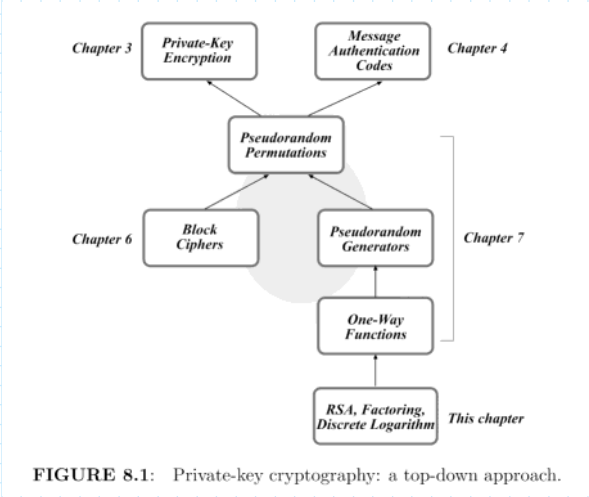


FIGURE 8.1: Private-key cryptography: a top-down approach.

Story:

- As noted earlier
  it is possible to obtain all these results
  based sollely on the existence of OWFs.

7.3 Hard-Core Predicates from OWFs

Theorem 7.12

Let f be a OWF and define
g by $g(x,r) \coloneqq (f(x),r)$ where $|x| = |r|$.

Define gl$(x,r) \coloneqq \bigoplus_{i=1}^{n} x_i \cdot r_i$
where $x = x_1 \dots x_n$ and
$r = r_1 \dots r_n$.

Then gl is a hard-core predicate of g.

Story:

Due to the complexity of the proof
we prove three successively stronger results
culminating in what is claimed in the theorem.

DEFINITION 3.28 *Let $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be an efficient, length-preserving, keyed permutation. F is a* strong pseudorandom permutation *if for all probabilistic polynomial-time distinguishers D, there exists a negligible function* negl *such that:*

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

where the first probability is taken over uniform choice of $k \in \{0,1\}^n$ and the randomness of D, and the second probability is taken over uniform choice of $f \in \mathsf{Perm}_n$ and the randomness of D.

< Looks like Theorem 7.5 explicitly

Goldreich-Levin

7.3.1 A simple case

Story:

- We first show that
  if there exists a poly time adversary A
  that always correctly computes $\text{gl}(x,r)$
  given
  $g(x,r) = (f(x),r)$
  then
  it is possible to invert f in poly time.

- Given the assumption that f
  is a OWF, it follows that
  no such adversary A can exist.

Proposition 7.13

Let f and gl be as in Theorem 7.12.
If there exists a poly time algorithm A such that
$A(f(x),r) = \text{gl}(x,r)$ for all n and all $x, r \in \{0,1\}^n$
then there exists a ploy-time algoirthm A' such that
$A'(1^n, f(x)) = x$ for all n and all $x \in \{0,1\}^n$.

Proof

We constuct A' as follows:

- $A'(1^n, y)$
  - computes $x_i := A(y, e_i)$
    (here $e_i = (00 \dots 010 \dots 00)$ at the ith position, it has 1; zero otherwise)
  - outputs $x = (x_1 \dots x_n)$

- NB: A' runs in poly time

- In the execution of $A'(1^n, f(\hat{x}))$
  the value x_i computed by A' satisfies

$$x_i = A(f(\hat{x}), e^i) = \text{gl}(\hat{x}, e^i) = \bigoplus_{j=1} \hat{x}_j \cdot e_{ij} = \hat{x}_i$$

- Clearly, $\hat{x} = x$ (me: forget about the hats; doesn't help here)

□

Story:

- If f is one-way
  it is impossible for any PPT algorithm to invert f
  with non-negl prob.

  - Thus
    we conclude that
    there is no PPT algorithm that always correctly computes
    $\text{gl}(x,r)$ from $(f(x),r)$.

- This is arather weak result that is
  very far from our ultimate goal of showing that
  $\text{gl}(x,r)$ cannot be computed (wp significantly better than $1/2$)
  given $(f(x),r)$.

7.3.2 A more involved case

Story:

- We now show that it is hard
  for any PPT algorithm A
  to compute $\text{gl}(x,r)$ from $(f(x),r)$
  with prob significantly better than $3/4$.

- We will again show that any such A
  would imple the existence of a poly-time algoithm A'
  that inverts f with non-negl prob

- Notice that the strategy in the proof of Prop 7.13
  fails here
  because it may be that A *never* succeeds when $r = e_i$
  (although it may succeed, say, on all other values of r)

- Furthermore, in the present case A' does not know
  if the result $A(f(x), r)$ is equal to $\mathrm{gl}(x, r)$ or not.

---the only thing A' knows is that
  with high prob, algorithm A is correct.

This further complicates the proof.

Proposition 7.14
Let f and gl be as in Theorem 7.12.
If there exists a PPT algorithm A
and
a polynomial p such that

$$\Pr_{x,r \leftarrow \{0,1\}^n}\left[A(f(x), r) = gl(x, r)\right] \geq \frac{3}{4} + \frac{1}{p(n)}$$

for infinitely many values of n,
then
there exists a PPT algorithm A' such that

$$\Pr_{x \leftarrow \{0,1\}^n}\left[A'(1^n, f(x)) \in f^{-1}(f(x))\right] \geq \frac{1}{4 \cdot p(n)}$$

for infinitely many values n.

Proof

- The main observation underlying th eproof of this proposition is
  that for every $r \in \{0,1\}^n$
  the values
  $\mathrm{gl}(x, r \oplus e_i)$ and
  $\mathrm{gl}(x, r)$
  together can b eused t ocompute the ith bit of x.

- This is true because

$$gl(x, r) \oplus gl(x, r \oplus e_i)$$

$$= \left(\bigoplus_{j=1}^{n} x_j \cdot r_j\right) \oplus \left(\bigoplus_{j=1}^{n} x_j \cdot (r_j \oplus e_{ij})\right)$$

$$= x_i \cdot r_i \oplus (x_i \cdot \bar{r}_i)$$

$$= x_i$$

where $\bar{r}_i$ is the complement of r_i and
the second equality is due to the fact that for $j \neq i$
the value $x_j \cdot r_j$ appears in both sums
and so is cancelled out.

- The above demonstrates that if A answers correctly on both
  $(f(x), r)$ and $(f(x), r \oplus e_i)$
  then
  A' can correctly compute x_i.

- Unfortunately, A' does not know
  when A answers correctly (and when it does not).

- For this reason,
  A' will use multipiple random values of r
  using each one to obtain an estimate of x_i
  and will then take the estimate occuring a
  majority of the time
  as its final guess for x_i.

- As a preliminary step
  we show that for many x's the probability that A
  answers correctly for both $(f(x), r)$ and $(f(x), r \oplus e_i)$
  when r is uniform
  is sufficiently high.

  This allows us to fix x and then
  focus solely on uniform choice of r
  which makes the analysis easier.

Claim 7.15

Let n be such that

$$\Pr_{x,r \leftarrow \{0,1\}^n}\left[A(f(x), r) = g1(x, r)\right] \geq \frac{3}{4} + \frac{1}{p(n)}$$

Then there exists a set $S_n \subseteq \{0,1\}^n$ of size at least $\frac{1}{2p(n)} \cdot 2^n$
such that for every $x \in S_n$ it holds that

$$\Pr_{r \leftarrow \{0,1\}^n}\left[A(f(x), r) = g1(x, r)\right] \geq \frac{3}{4} + \frac{1}{2p(n)}.$$

Proof:

Let $\epsilon(n) = 1/p(n)$ and

define $S_n \subseteq \{0,1\}^n$ to be the set of all x's for which

$$\Pr_{r \leftarrow \{0,1\}^n}\left[A(f(x), r) = g1(x, r)\right] \geq \frac{3}{4} + \frac{\epsilon(n)}{2}.$$

We have

$$\Pr_{x,r \leftarrow \{0,1\}^n}\left[A(f(x), r) = g1(x, r)\right] = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \Pr_{r \leftarrow \{0,1\}^n}\left[A(f(x), r) = g1(x, r)\right]$$

$$= \frac{1}{2^n} \sum_{x \in S_n} \underbrace{\Pr_{r \leftarrow \{0,1\}^n}[\cdots]}_{\leq 1} + \sum_{x \notin S_n} \underbrace{\Pr_{r \leftarrow \{0,1\}^n}[\cdots]}_{\leq \frac{3}{4} + \frac{\epsilon}{2}}$$

$$\leq \frac{|S_n|}{2^n} + \underbrace{\frac{1}{2^n}}_{\leq 1} \cdot \sum_{x \notin S_n} \left(\frac{3}{4} + \frac{\epsilon}{2}\right)$$

$$\leq \frac{|S_n|}{2^n} + \left(\frac{3}{4} + \frac{\epsilon}{2}\right)$$

$\frac{3}{4} + \epsilon(n) \leq$

$\because \quad \frac{3}{4} + \epsilon(n) \leq \Pr_{x,r \leftarrow \{0,1\}^n}\left[A(f(x), r) = g1(x, r)\right]$

$$\frac{3}{4} + \frac{\epsilon(n)}{2} \leq \frac{|S_n|}{2^n} + \frac{3}{4} + \frac{\epsilon}{2}$$

$$\Rightarrow \quad |S_n| \geq \frac{\epsilon(n)}{2} \cdot 2^n$$

□

Story:

- The following now follows as an easy consequence.

Claim 7.16

Let n be such that

$$\Pr_{x,r \leftarrow \{0,1\}^n}\left[A(f(x),r) = gl(x,r)\right] \geq \frac{3}{4} + \frac{1}{p(n)}$$

Then there exists a set $S_n \subseteq \{0,1\}^n$ of size at least $\frac{1}{2p(n)} \cdot 2^n$
such that for every $x \in S_n$ and
every i it holds that

$$\Pr_{r \leftarrow \{0,1\}^n}\left[A(f(x),r) = gl(x,r) \wedge A(f(x), r \oplus e_i) = gl(x, r \oplus e_i)\right] \geq \frac{1}{2} + \frac{1}{p(n)}$$

Proof.

- Let $\epsilon(n) = 1/p(n)$ and
take S_n to be the set guaranteed by the previous claim.

- For any $x \in S_n$ we have that

$$\Pr_{r \leftarrow \{0,1\}^n}\left[A(f(x), r) \neq gl(x,r)\right] \leq \frac{1}{4} - \frac{\epsilon(n)}{2}$$

- Fix $i \in \{1 \ldots n\}$.
  - if r is uniformly distributed,
  then so is $r \oplus e_i$ and thus

$$\Pr_{r \leftarrow \{0,1\}^n}\left[A(f(x), r \oplus e_i) \neq gl(x, r \oplus e_i)\right] \leq \frac{1}{4} - \frac{\epsilon(n)}{2}$$

- We are interested in lower bonuding the prob that
  A outputs the correct answer for both $\mathrm{gl}(x, r)$ and $\mathrm{gl}(x, r \oplus e_i)$;
  equivalently,
  we want to upper bound the probability that A fails
  to output the correct answer in *either* of these cases.

  Note that r and $r \oplus e_i$ are not independent
  so we cannot just multiply the probabilities of failures.

  However,
  we can apply the union (see Prop A7) and sum the probabilities of failure.

  That is
  the probability that A is *incorrect* on either $\mathrm{gl}(x, r)$ or $\mathrm{gl}(x, r \oplus e_i)$
  is at most

$$\left(\frac{1}{4} - \frac{\epsilon(n)}{2}\right) + \left(\frac{1}{4} - \frac{\epsilon(n)}{2}\right) = \frac{1}{2} - \epsilon(n)$$

  and so A is correct on *both* $\mathrm{gl}(x, r)$ and $\mathrm{gl}(x, r \oplus e_i)$
  with probability *at least* $\frac{1}{2} + \epsilon(n)$.

  This proves the claim.

□

Story:

- For the rest of the proof
  we set $\epsilon(n) = 1/p(n)$ and consider only those values of n for which

$$\Pr_{x,r \leftarrow \{0,1\}^n}\left[A(f(x),r) = gl(x,r)\right] \geq \frac{3}{4} + \epsilon(n). \qquad [7.1]$$

  - The previous claim states that
    for an $\frac{\epsilon(n)}{2}$ fraction of inputs x and
    for any i
    algorithm A answers correctly on both
    $(f(x), r)$ and $(f(x), r \oplus e_i)$ with
    probability at least $\frac{1}{2} + \epsilon$

over uniform choice of r.

And from now on, we focus only on such values of x.

- We construct a PPT algorithm A' that inverts $f(x)$ with prob at least $1/2$ when $x \in S_n$.

  - This suffices to prove Prop 7.14
    since then, for infinitely many ns,

$$\Pr_{x \leftarrow \{0,1\}^n}\left[A'(1^n, f(x)) \in f^{-1}(f(x))\right]$$

$$\geq \underbrace{\Pr_{x \leftarrow \{0,1\}^n}\left[A'(1^n, f(x)) \in f^{-1}(f(x)) \mid x \in S_n\right]}_{\geq \frac{1}{2}} \cdot \underbrace{\Pr_{x \leftarrow \{0,1\}^n}\left[x \in S_n\right]}_{\frac{\epsilon}{2}}$$

Recall: $\frac{|S_n|}{2^n} = \frac{\epsilon}{2}$

$$= \frac{1}{4p(n)}.$$

- Algorithm A' given as input 1^n and y works as follows:

1. For $i = 1 \ldots n$ do
   - Repeatedly,
     choose a uniform $r \in \{0,1\}^n$ and
     compute $A(y,r) \oplus A(y, r \oplus e_i)$ as an
     "estimate" for the ith bit of the preimage of y.
   - After doing this sufficiently many times (detailed below)
     let x_i be the "estimate" that occurs a majority of the time.
2. Output $x = x_1 \ldots x_n$.

We sketch an analysis of the probability that A' correctly inverts its given input y
(we allow ourselves to be a bit laconic
since a full proof for the more difficult case is given in the following section)

- Say $y = f(\hat{x})$ and
  recall that we assume here that n is such that Eq 7.1 holds
  and
  $\hat{x} \in S_n$.

- Fix some i.

- The previous claim implies that the estimate $A(y,r) \oplus A(y, r \oplus e_i)$
  equals $\mathrm{gl}(\hat{x}, e_i)$ with prob at least $\frac{1}{2} + \epsilon$
  over the choice of r.

  - By obtaining enough estimates and
    letting x_i be the majority value <---- x_i is a random variable
    A' can ensure that x_i equals $\mathrm{gl}(\hat{x}, e_i)$ with prob at least $1 - \frac{1}{2n}$.

    < So the full string
    one should be able to recover
    with prob $1 - \frac{n}{2n} = \frac{1}{2}$.

  - Of course
    we need to ensure that poly many estimates are enough.

  - Fortunately
    since $\epsilon(n) = 1/p(n)$ for some poly p and
    an independent value of r is used for each estimate,

    the Chernoff bound shows that poly many estimates suffice.

  - Putting it together:
    we have that for each i the value x_i computed by A' is
    incorrect with probability at most $\frac{1}{2n}$.

A union bound thus shows that A' is
incorrect for *some* i with probability at most $n \cdot \frac{1}{2n} = \frac{1}{2}$.

That is, A' is correct for all i—and thus correctly inverts y—with prob
at least $1 - \frac{1}{2} = \frac{1}{2}$.

This completes the proof of Prop 7.14

□

Story:

- A corollary of Prop 7.14 is that
  if f is a OWF then
  for any poly-time algorithm A
  prob that A correctly guesses $\text{gl}(x,r)$
  when given $(f(x),r)$
  is at most negligibly more than $3/4$.

< PPT should also be fine
(as far as I can tell)

7.3.3 The Full Proof

Story:

- We assume familiarity with the simplified proofs
  in the previous sections, and
  build on the ideas developed there.
  - We rely on some terminology and
    standard results from
    prob theory discussed in Appendix A.3
  - We prove the following
    which implies Theorem 7.12

THEOREM 7.12 *Let f be a one-way function and define g by $g(x,r) \stackrel{\text{def}}{=} (f(x),r)$, where $|x| = |r|$. Define $\text{gl}(x,r) \stackrel{\text{def}}{=} \bigoplus_{i=1}^{n} x_i \cdot r_i$, where $x = x_1 \cdots x_n$ and $r = r_1 \cdots r_n$. Then gl is a hard-core predicate of g.*

Proposition 7.17

Let f and gl be as in Theorem 7.12.

If there exists a PPT algorithm A and a poly p such that

$$\Pr_{x,r \leftarrow \{0,1\}^n}\left[A(f(x),r) = \text{gl}(x,r)\right] \geq \frac{1}{2} + \frac{1}{p(n)}$$

for infinitely many values of n,
then
there exsits a PPT algorithm A' and a poly p' such that

$$\Pr_{x,r \leftarrow \{0,1\}^n}\left[A'(1^n, f(x)) \in f^{-1}(f(x))\right] \geq \frac{1}{p'(n)}$$

for infinitely many values of n.

Proof.

Once again
we set $\epsilon(n) = 1/p(n)$ and
consider only those values of n
for which

$$\Pr_{x,r \leftarrow \{0,1\}^n}\left[A(f(x),r) = \text{gl}(x,r)\right] \geq \frac{1}{2} + \frac{1}{p(n)}$$

Story:

- The following is analogous to Claim 7.15 and is proved in the same way.

Claim 7.18
Let n be such that

$$\Pr_{x,r \leftarrow \{0,1\}^n}\left[A\left(f(x),r\right) = \mathrm{gl}(x,r)\right] \geq \frac{1}{2} + \epsilon(n)$$

Then, there exsits a set $S_n \subseteq \{0,1\}^n$ of size at least $\frac{\epsilon}{2} \cdot 2^n$ such that for every $x \in S_n$ it holds that

$$\Pr_{r \leftarrow \{0,1\}^n}\left[A\left(f(x),r\right) = \mathrm{gl}(x,r)\right] \geq \frac{1}{2} + \frac{\epsilon(n)}{2}$$

<crucially, the x dependence
is removed and yet the size of
S is a fraction $\frac{\epsilon}{2}$ of the
total set of strings

(recall: the proof from the previous time goes through for essentially any constant
(did not have to be $3/4$ or even $1/2$).

Story:

- If we start by trying to prove an analogue of Claim 7.16
  the best one can claim here is that
  when $x \in S_n$, one has

$$\Pr_{r \leftarrow \{0,1\}^n}\left[A\left(f(x),r\right) = \mathrm{gl}(x,r) \bigwedge A\left(f(x),r \oplus e_i\right) = \mathrm{gl}(x,r \oplus e_i)\right] \geq \epsilon(n)$$

  for any i.

  - Thus,
    if w etry to use $A(f(x),r) \oplus A(f(x),r \oplus e_i)$
    as an estimate for x_i
    all we can claim is that
    this estimate will be correct
    with probability at least ϵ
    which may not be better than taking a random guess!

    We cannot claim that flipping the result
    gives a good estimate either.
    (i.e. $\neg$ of the estimate would also be a bad estimate)

- Instead,
  we design A' so that
  it computes $\mathrm{gl}(x,r)$ and $\mathrm{gl}(x,r \oplus e_i)$ by
  invoking A only once.

  We do this by having A' run $A(f(x),r \oplus e_i)$
  and having A' simply
  "guess" the value $\mathrm{gl}(x,r)$ itself.

  The naive way to do this
  would be to choose the rs independently
  (as before)
  and have A' make an independent guess of
  $\mathrm{gl}(x,r)$ for each value of r.

  But then
  the probability that
  all such guesses are correct—which, as we will see, is necessary
  if A' is to output the correct inverse—
  would be negligible because poly many r's are used.

  (current understanding:
  Draw many rs
  for each r
  compute $A(f(x),r \oplus e_i)$ and guess $\mathrm{gl}(x,r)$
  Since there are many rs
  guessing gl for each correctly
  would happen with negligible probbaility.

  As we will see,
  the guesses must all be correct
  for A' to produce the correct inverse)

  The crucial observation of the present proof is that
  A' can generate the r's in a
  pairwise independent manner and
  make its guesses in a particular way
  so that with
  non-negl probability
  as all its guesses are correct.

Specifically, in order to generate m values of r
we have A' select
$\ell = \log(m+1)$
independent and uniformly distributed strings
$s^1 \ldots s^\ell \in \{0,1\}^n$

(To generate m samples
it first samples ℓ many ss of length n
where ℓ is the number of bits
needed to store m)

Then
for every non-empty subset $I \subseteq \{1, \ldots \ell\}$
we set $r^I := \bigoplus_{i \in I} s^i$.

Since there are $2^\ell - 1$ nonempty subsets
(2 choices for each element; remove the null set)
this defines a collection of
$2^{\log(m+1)} - 1 \geq m$ strings.
(for us it is equal but take ceiling of log).

[Intuition: sample ℓ strings;
produce a new string by XORING a subset $I \subseteq \{1 \ldots \ell\}$
and this will allow you to output
$\sim 2^\ell$ that are "independent", todo check
No, only pairwise; read below]

Since there are $2^\ell - 1$ nonempty subsets
this defines a collection of $2^{\log m+1} - 1 \geq m$ strings.

The strings are not independent but
they *are* pairwise independent.

To see this
notice that for every two subsets $I \neq J$
there is an index $j \in I \cup J$
such that $j \notin I \cap J$.

Without loss of generality
assume $j \notin I$.

Then, the value of s^j is uniform and
independent of the value of r^I (highlighte above).

Since s^j is included in the XOR that defines r^J
this implies that
r^J is uniform and indepnedent of r^I as well.

We now have the following two important observations

1. Given $\mathrm{gl}(x, s^1) \ldots \mathrm{gl}(x, s^\ell)$
it is possible to compute $\mathrm{gl}(x, r^I)$
for every subset $I \subseteq \{1 \ldots \ell\}$.

This is because
$\mathrm{gl}(x, r^I) =$

$$\mathrm{gl}\left(x, \bigoplus_{i \in I} \underbrace{s^i}_{n\text{-bit}}\right)$$

s^is first, then XOR

$$= \bigoplus_{j=1}^{n} \left[x[j] \oplus \left(\bigoplus_{i \in I} s^i \right)[j] \right]$$

$$= \bigoplus_{i \in I} \left(\bigoplus_{j=1}^{n} x[j] \oplus s^i[j] \right)$$

xor first, then s^is

$$= \bigoplus_{i \in I} \mathrm{gl}(x, s^i)$$

2. If A' simply guesses the values of
$\mathrm{gl}(x, s^1) \ldots \mathrm{gl}(x, s^\ell)$
by choosing a uniform bit for each,
then *all* guseses will be correct
with probability $1/2^\ell$.

If m is polynomial in the security parameter n
then $1/2^\ell$ is not negligible
and so
with non-neglible probability A'
correctly guesses all the values
$\mathrm{gl}(x, s^1) \ldots \mathrm{gl}(x, s^\ell)$.

Combining the above
yields a way of obtaining $m = \text{poly}(n)$ uniform and
pairwise-independ independent strings $\{r^l\}$
along with *correct* values for $\{\text{gl}(x, r^l)\}$
with non-negligible probability.

These values can then be used
to compute x_i
in the same way, as in the proof of Proposition 7.14.

Details follow:

The inversion algorithm A'.

We now provide
a full description of an algorithm A'
that receives inputs $1^n, y$
and tries to compute an inverse of y.

The algorithm proceeds as follows:

1. Set $\ell := \log\left(\frac{2n}{\varepsilon^2}\right) + 1$
2. Choose uniform, independent $s^1 \ldots s^\ell \in \{0,1\}^n$
and $\sigma^1 \ldots \sigma^\ell \in \{0,1\}$.

3. For every non-empty subset
$I \subseteq \{1, \ldots \ell\}$
compute $r^I := \bigoplus_{i \in I} r^i$ & $\sigma^I := \bigoplus_{i \in I} \sigma^i$
(I'm treating $s^i := r^i$ equivalently. s^i)
(intuitively, this is the guess for $A(y, r^I)$.)

4 For $i = 1, \ldots n$, do the following
(a) If non-empty $I \subseteq \{1, \ldots \ell\}$ set
$x_i^I := \sigma^I \oplus A(y, r^I \oplus e^i)$

(b) Set $x_i := \text{majority}_I \{x_i^I\}$
(i.e. take the bit that appeared a majority of the times)

5. Output $x = x_1 \ldots x_n$

[Boddu: Now we'll see why ℓ was chosen to be what it was chosen to be

- IT remains to compute th eprobability that
A' outputs $x \in f^{-1}(y)$.
  - [boring qualification on y, n]
  As in the proof of Proposition 7.14
  we focus only on n as in Claim 7.18 and
  assume $y = f(\hat{x})$ fro some $\hat{x} \in S_n$.
  - Each σ^i represents a "guess" for the
  value of $\text{gl}(\hat{x}, s^i)$.
  - As noted earlier,
  with non-negl probabiity
  all these guesses are correct.

  We show that conditioned on this event
  A' outputs $x = \hat{x}$ with probability at least $1/2$.

- Assume $\sigma^i = \text{gl}(\hat{x}, s^i)$ for all i.
  - Then, $\sigma^I = \text{gl}(\hat{x}, r^I)$ $\forall$ I.
  ($r^I = \bigoplus_{i \in I} s^i$)
  - Fix an index $i \in \{1, \ldots n\}$ &

consider the prob. that

A' obtains the correct value

$x_i = \hat{x}_i$.

- For any non-empty I

we have $A(y, x^I \oplus e^i) = g(\hat{x}, x^I \oplus e^i)$

with prob. at least $\frac{1}{2} + \frac{\epsilon}{2}$

(over the choice of ℓ).

($\because$ $\hat{x} \in S_\ell$ & $x^I \oplus e_i$ is uniformly distributed)

- Thus, for any non-empty subset I,

we have

$$\Pr\left[x_i^I = \hat{x}_i \right] \geq \frac{1}{2} + \frac{\epsilon}{2}$$

($\because$ we already conditioned on the other "guess" being correct;

recall the def^n of x^I from the alg)

- Moreover, the $\{x_i^I\}_{I \subseteq \{1,\dots,\ell\}}$ are pairwise indep.

$\because$ the $\{x^I\}_{I \subseteq \{1,\dots,\ell\}}$

(ℓ here $\{x^I \oplus e^i\}_{I \subseteq \{1,\dots,\ell\}}$) are

pairwise independent.

$f(r), f(r')$

and you know r, r' independent

then

$f(r), f(r')$ are also independent

- Since x_i is defined to be the value

that occurs a majority of the

times among the $\{x_i^I\}_{I \subseteq \{1,\dots,\ell\}}$

one can apply Prop. A.13

to obtain

$$\Pr\left[x_i \neq \hat{x}_i \right] \leq \frac{1}{4 \cdot \left(\frac{\epsilon}{2}\right)^2 \cdot \left(2^\ell - 1\right)}$$

($\because$ we were correct in 1 trial w.p. $\frac{1}{2} + \frac{\epsilon}{2}$; # sampled; recall: $\ell = \log \frac{2n}{\epsilon^2} + 1$; $\epsilon \to 0$)

$$\leq \frac{1}{4 \cdot \left(\frac{\epsilon}{2}\right)^2 \cdot \frac{2n}{\epsilon^2}}$$

$$= \frac{1}{2n}.$$

PROPOSITION A.13 *Fix $\varepsilon > 0$ and $b \in \{0,1\}$, and let $\{X_i\}$ be pairwise-independent, 0/1-random variables for which $\Pr[X_i = b] \geq \frac{1}{2} + \varepsilon$ for all i. Consider the process in which m values $X_1, \dots, X_m$ are recorded and X is set to the value that occurs a strict majority of the time. Then*

$$\Pr[X \neq b] \leq \frac{1}{4 \cdot \varepsilon^2 \cdot m}.$$

PROOF Assume $b = 1$; by symmetry, this is without loss of generality. Then $\mathsf{Exp}[X_i] = \frac{1}{2} + \varepsilon$. Let X denote the strict majority of the $\{X_i\}$ as in the proposition, and note that $X \neq 1$ if and only if $\sum_{i=1}^m X_i \leq m/2$. So

$$\begin{aligned}
\Pr[X \neq 1] &= \Pr\left[\sum_{i=1}^m X_i \leq m/2\right] \\
&= \Pr\left[\frac{\sum_{i=1}^m X_i}{m} - \frac{1}{2} \leq 0\right] \\
&= \Pr\left[\frac{\sum_{i=1}^m X_i}{m} - \left(\frac{1}{2} + \varepsilon\right) \leq -\varepsilon\right] \\
&\leq \Pr\left[\left|\frac{\sum_{i=1}^m X_i}{m} - \left(\frac{1}{2} + \varepsilon\right)\right| \geq \varepsilon\right].
\end{aligned}$$

Since $\mathsf{Var}[X_i] \leq 1/4$ for all i, applying the previous corollary shows that $\Pr[X \neq 1] \leq \frac{1}{4\varepsilon^2 m}$ as claimed. ■

- The above holds for all i,

so by applying a union bound

we see that

$\Pr$ $x_i \neq \hat{x}_i$ for some i,

is at most $\frac{1}{2}$.

($\because \sum_n \frac{1}{2n} = \frac{1}{2}$)

i.e. $(x_i = \hat{x}_i \ \forall i)$

π w.p. $\geq \frac{1}{2}$

$\left(\because \sum_{i=1}^{n} \frac{1}{2n} = \frac{1}{2}\right)$

i.e. $(x_i = \hat{x}_i \ \forall i)$

$\Updownarrow$ w.p. $\geq \frac{1}{2}$

$x = \hat{x}$

- Putting everything together,

  Let n be as in Claim 7.18.2

  $y = f(\hat{x})$.

  With prob. at least $\frac{\epsilon}{2}$

  we have $\hat{x} \in S_n$.

  All guesses σ_i are correct w.p. at least

  $$\frac{1}{2^\ell} \geq \frac{1}{2 \cdot \left(\frac{2n}{\epsilon^2} + 1\right)} > \frac{\epsilon^2}{5n}$$

  for a large enough n.

- conditioned on both the above,

  A' outputs $x = \hat{x}$ with prob. at least $1/2$.

- Thus, the overall prob. with which A' inverts

  is at least $\left(\frac{\epsilon^2}{5n}\right)\left(\frac{\epsilon}{2}\right) \cdot \frac{1}{2} = \frac{\epsilon^3}{20n} = \frac{1}{20 \cdot p^3}$

  for infinitely many n's.

□

7.4 Constructing Pseudorandom Generators

Story:

- We first show how to construct
  pseudorandom generators
  that stretch their input by a single bit
  under the assumption that one-way *permutations* exist.

- We then show
  how t oextend this to obtain
  any polynomial expansion factor.

7.4.1 Pseudorandom Generators with Minimal Expansion

Let f be a one-way permutation with hard-core predicate hc.

Story:

- thsi means that $\text{hc}(s)$ "looks random" given $f(s)$
  when s is uniform.

- Furthermore
  since f is a permutation
  $f(s)$ itself is uniformly distributed
  (apply a permutatino to a uniformly distributed value
  yields a uniformly distributed value).

- So if s is a uniform n-bit string
  the $(n+1)$-bit string $f(s)||\text{hc}(s)$ consists of
  a uniform n bit string plus
  an additional bit
  that looks uniform,
  even conditioned on the initial n bits;
  in other words

this $(n+1)$-bit string is a *pseudorandom*.

- Thus,
  the algorithm G defined by
  $G(s) = f(s)||\text{hc}(s)$
  is a pseudorandom generator.

Theorem 7.19

Let f be a one-way permutation with hard-core predicate hc.
Then,
algorithm G defined by $G(s) = f(s)||\text{hc}(s)$ is a pseudorandom generator
with expansion factor $l(n) = n+1$.

Proof

Let D be a PPT algorithm.

Strategy: We prove there is a negligible function negl such that

$$\Pr_{x \leftarrow \{0,1\}^{n+1}}[D(x)=1] - \Pr_{s \leftarrow \{0,1\}^n}[D(G(s))=1] \leq \text{negl}(n) \qquad (7.3)$$

and a similar argument shows there's a negligible functino negl' such that

$$\Pr_{s \leftarrow \{0,1\}^n}[D(G(s))=1] - \Pr_{x \leftarrow \{0,1\}^{n+1}}[D(x)=1] \leq \text{negl}'(n)$$

Rough

$$\left[\frac{1}{2}\Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|0)=1 \mid x'=0]\right] + \frac{1}{2}\Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|1)=1 \mid x'=1]$$

Observe:

$$\Pr_{x\leftarrow\{0,1\}^{n+1}}[D(x)=1] = \Pr_{x\leftarrow\{0,1\}^n,\, x'\leftarrow\{0,1\}}[D(x\|x')=1]$$

$$= \Pr_{s\leftarrow\{0,1\}^n,\, x'\leftarrow\{0,1\}}[D(f(s)\|x')=1]$$

$$= \Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|0)=1] \cdot \Pr[\text{hc}=0] + \Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|1)=1] \cdot \Pr[\text{hc}=1]$$

$$= A \cdot \Pr[\text{hc}=0] + \Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\text{hc}(s))=1 \mid \text{hc}=0] \cdot \Pr[\text{hc}=0] + \Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\overline{\text{hc}}(s))=1 \mid \overline{\text{hc}}=0] \cdot \Pr[\overline{\text{hc}}=0]$$

$$B = \Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|1)=1] = \Pr[\text{hc}=1] \cdot \Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\text{hc}(s))=1 \mid \text{hc}=1] + \Pr[\overline{\text{hc}}=1] \cdot \Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\overline{\text{hc}}(s))=1 \mid \overline{\text{hc}}=1]$$

$$\Pr_{f,\text{hc}}[D(f\|\text{hc})=1]$$

$$P[A] = P[A|B]P[B] + P[A|\neg B]P[\neg B]$$

$$\frac{1}{2}A + \frac{1}{2}B = \frac{1}{2}\left[\Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\text{hc}(s))=1] + \Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\overline{\text{hc}}(s))=1]\right]$$

$$= \frac{1}{2}\cdot\Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\text{hc}(s))=1] + \frac{1}{2}\Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\overline{\text{hc}}(s))=1]$$

using the fact that f is a permutation (for the second equality)
and that a uniform bit r' equals $\text{hc}(s)$ with prob. exactly $1/2$ for the "third" equality.

Since

$$\Pr_{s\leftarrow\{0,1\}^n}[D(G(s))=1] = \Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\text{hc}(s))=1]$$

(by def^n of G)

this means that Eq. 7.3 is equivalent to

$$\frac{1}{2}\cdot\left(\Pr_{s\leftarrow\{0,1\}^n}[D(f\|\overline{hc})=1]-\Pr_{s\leftarrow\{0,1\}^n}[D(f\|hc)=1]\right)\leq negl$$

Me:

- Of course, we had to *prove* Eq 7.3 holds
- To see this,
  observe that if the equation above (which is equiv. to Eq. 7.3) does not hold
  then one can distingusih between hc and $\overline{hc}$
- I am skipping the details for now.

Consider the following algorithm $\mathcal{A}$ that is given as input a value $y = f(s)$ and tries to predict the value of $\mathsf{hc}(s)$:

1. Choose uniform $r' \in \{0,1\}$.
2. Run $D(y\|r')$. If D outputs 0, output r'; otherwise output $\bar{r}'$.

Clearly $\mathcal{A}$ runs in polynomial time. By definition of $\mathcal{A}$, we have

$$\begin{aligned}
&\Pr_{s\leftarrow\{0,1\}^n}[\mathcal{A}(f(s))=\mathsf{hc}(s)]\\
&=\frac{1}{2}\cdot\Pr_{s\leftarrow\{0,1\}^n}[\mathcal{A}(f(s))=\mathsf{hc}(s)\mid r'=\mathsf{hc}(s)]\\
&\quad+\frac{1}{2}\cdot\Pr_{s\leftarrow\{0,1\}^n}[\mathcal{A}(f(s))=\mathsf{hc}(s)\mid r'\neq\mathsf{hc}(s)]\\
&=\frac{1}{2}\cdot\left(\Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\mathsf{hc}(s))=0]+\Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\overline{\mathsf{hc}}(s))=1]\right)\\
&=\frac{1}{2}\cdot\left(\left(1-\Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\mathsf{hc}(s))=1]\right)+\Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\overline{\mathsf{hc}}(s))=1]\right)\\
&=\frac{1}{2}+\frac{1}{2}\cdot\left(\Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\overline{\mathsf{hc}}(s))=1]-\Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\mathsf{hc}(s))=1]\right).
\end{aligned}$$

Since hc is a hard-core predicate of f, it follows that there exists a negligible function negl for which

$$\frac{1}{2}\cdot\left(\Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\overline{\mathsf{hc}}(s))=1]-\Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\mathsf{hc}(s))=1]\right)\leq\mathsf{negl}(n),$$

as desired. ■

□

7.4.2 Increasing the Expansion Factor

Story:

- We now show that the expansion factor
  of a pseudorandom generator can be increased
  by any desired (poly) amount.

- This means that
  the previous construction
  with expansion factor $l(n) = n + 1$
  suffices for constructing a pseudorandom generator
  with arbitrary (poly) expansion factor.

Theorem 7.20

If there exsits a pseudorandom generator G with
expansion factor $n + 1$
then for any polynomial poly
there exists a pseudorandm generator $\hat{G}$
with expansion factor $\mathrm{poly}(n)$.

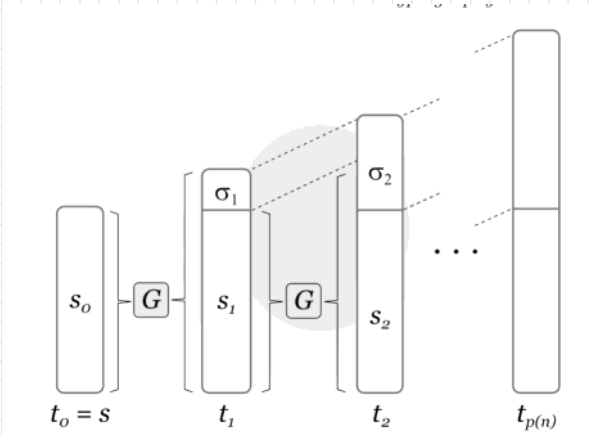
Proof

Story:

- We first consider constructing a PRG $\hat{G}$ that
  outputs $n + 2$ bits.

Definition:
$\hat{G}$

Input: inital seed $s \in \{0,1\}^n$
Outputs:
$t_1 := G(s)$, to obtain $n+1$ pseudorandm bits
the initial n bits of t_1 are then used again
as a seed for G
the resulting $n+1$ bits,
concatenated with the final bit t_1
yield the $(n+2)$ bit output.



Story:
- The second application of G usus a pseudorandom seed
rather than a random one.

- The proof of security we give next
shows that this does not impact
the pseudorandomness of the output.

Goal:
Prove that $\hat{G}$ is a PRG.

Sub-Proof:
Define three sequences of distributions

$\{H^0_n\}_{n=1,\dots}$, $\{H^1_n\}_{n=1,\dots}$ & $\{H^2_n\}_{n=1,\dots}$

where each of H^0_n H^1_n & H^2_n is

a distribution over strings of length $n+2$.

In distribution H_n^0
a uniform string $t_0 \in \{0,1\}^n$ is chosen
and the output is $\hat{G}(t_0)$.

In distribution H_n^1
a uniform string $t_1 \in \{0,1\}^{n+1}$ is chosen
and parsed as $s_1||\sigma_1$
(where s_1 are the initial n bits of t_1 and
σ_1 is the final bit).
and the output is $G(s_1)||\sigma_1$

In distribution H_n^2
the output is $t_2 \in \{0,1\}^{n+2}$
(chosen uniformly)

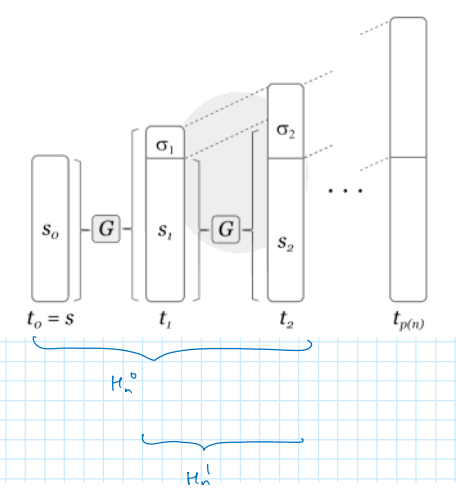
Notation: Denote by $t_2 \leftarrow H_n^i$
sampling t_2 from distribution H_n^i

Showing these distributions are indistinguishable

Story:
- Fix a PPT distinguisher D.

Claim:

Recall:
$\hat{G}$ goes from n to $n+2$



Story:

- Fix a PPT distinguisher D.

Claim:

There's a negligible functino such that

$$\left| \Pr_{t_1 \leftarrow H_n^0}[D(t_1)=1] - \Pr_{t_2 \leftarrow H_n^1}[D(t_2)=1] \right| \leq \text{negl}'(n) \qquad [7.4]$$

Proof of the claim:

To see this

consider the PPT distinguisher D'

- Input: $t_1 \in \{0,1\}^{n+1}$
  parses it as $s_1 || \sigma_1$
  (s_1 has length n)
- Compute: $t_2 := G(s_1)||\sigma_1$
- Output: $D(t_2)$

Clearly, D' runs in poly time.

Observe:

- If t_1 is uniform,
  the distribution on t_2 gneerated by D' is
  exactly that of H_n^1
  (essentially by definition).
  Thus

$$\Pr_{t_1 \leftarrow \{0,1\}^{n+1}}[D'(t_1)=1] = \Pr_{t_2 \leftarrow H_n^1}[D(t_2)=1]$$

- If $t_1 = G(s)$ for uniform $s \in \{0,1\}^n$
  the distirbution on t_2 generated by D'
  is exactly that of H_n^0.

  That is

$$\Pr_{s \leftarrow \{0,1\}^n}[D'(G(s))=1] = \Pr_{t_2 \leftarrow H_n^0}[D(t_2)=1].$$

Pseudorandomness of G implies that

there is a negligible function negl′ with

$$\left| \Pr_{s \leftarrow \{0,1\}^n}[D'(G(s))=1] - \Pr_{t_1 \leftarrow \{0,1\}^{n+1}}[D'(t_1)=1] \right| \leq \text{negl}'(n)$$

and this immeditaely yields 7.4

(proves the claim).

□

Story;

- We next claim the following:

Claim: There is a negligible function

negl″ such that

$$\left| \Pr_{t_1 \leftarrow H_n^1}[D(t_2)=1] - \Pr_{t_2 \leftarrow H_n^2}[D(t_2)=1] \right| \leq \text{negl}''(n)$$

Proof of the claim:

Defin: Distinguisher D''

Input: $w \in \{0,1\}^{n+1}$

Procedure: Choose uniform $\sigma_1 \in \{0,1\}$

Defin $t_2 := w||\sigma_1$

Outputs: $D(t_2)$

Observations:

- If w is uniform then so is t_2. Thus

$$\Pr_{w \leftarrow \{0,1\}^{n+1}}[D''(w)=1] = \Pr_{t_2 \leftarrow H_n^2}[D(t_2)=1]$$

- If $w = G(s)$ for uniform $s \in \{0,1\}^n$ then
  t_2 is distributed exactly according to H_n^1, thus

$$\Pr_{s \leftarrow \{0,1\}^n}\left[D^{n+1}(G(s))=1\right] = \Pr_{t_1 \leftarrow H_n^1}\left[D(t_1)=1\right]$$

As before
pseudorandomness of G implies Eq. 7.5
(and completes the claim)

□

Final argument (for the $n \to n+2$ case)

Putting everything together, we have

$$\left|\Pr_{s \leftarrow \{0,1\}^n}\left[D(\hat{G}(s))=1\right] - \Pr_{x \leftarrow \{0,1\}^{n+2}}\left[D(x)=1\right]\right|$$

$$= \left|\Pr_{t_1 \leftarrow H_n^0}\left[D(t_1)=1\right] - \Pr_{t_2 \leftarrow H_n^2}\left[D(t_2)=1\right]\right|$$

$$\leq \left|\cdots H_n^0 - \cdots H_n^1\right| + \left|\cdots H_n^1 - \cdots H_n^2\right|$$

$$\leq \mathrm{negl}' + \mathrm{negl}'' = \mathrm{negl}'''$$

Since D was an arbitrary PPT distinguisher
this proves that $\hat{G}$ is a pseudorandom generator.

The general case:

- It should follow analogously
- Can only do this poly many times (else sum of negligble functinos would not stay neglgible)

Putting it all together

- Let f be a one-way permutation
- Taking the pseudorandom generator
  with expansion factor $n+1$ from Theorem 7.19
  and
  increasing the expansion factor to $n+l$
  using the approach from the proof of Theorem 7.20
  we obtain the following PRG $\hat{G}$:

$$\hat{G}(s) := f^{(l)}(s) \| \mathrm{hc}(f^{(l-1)}(s)) \| \ldots \| \mathrm{hc}(s)$$

where $f^i(s)$ refers to i-fold iteration of f.

- Note:
  $\hat{G}$ uses l evalutainos of f
  and generates one pseudorandom bit per evaluation using the
  hard-core predicate hc.

Connection to stream ciphers.

- Recall from Section 3.3.1 that
  a stream cihper (without an initial vector)
  is defined by algorithms init, GetBits
  where $init$ takes a seed $s \in \{0,1\}^n$
  and returns initial state st
  and GetBits takes as input
  the current state st and
  outputs a bit σ
  and updated state st'

The construction $\hat{G}$ from the preceding proof
fits nicely in this paradigm
Take init to b ethe trivial algorithm
that outputs st $= s$
and define
GetBits(st) t ocompute
G(st)
and parse the result as st′||σ

updated state st′. (If we use this stream cipher to generate $p(n)$ output bits starting from seed s, then we get exactly the final $p(n)$ bits of $\hat{G}(s)$ in reverse order.) The preceding proof shows that this yields a pseudorandom generator.

Hybrid arguments

Hybrid arguments. A *hybrid argument* is a basic tool for proving indistinguishability when a basic primitive is (or several different primitives are) applied multiple times. Somewhat informally, the technique works by defining a series of intermediate "hybrid distributions" that bridge between two "extreme distributions" that we wish to prove indistinguishable. (In the proof above, these extreme distributions correspond to the output of $\hat{G}$ and a random string.) To apply the proof technique, three conditions should hold. First, the extreme distributions should match the original cases of interest. (In the proof above, H_n^0 was equal to the distribution induced by $\hat{G}$, while $H_n^{p(n)}$ was the uniform distribution.) Second, it must be possible to translate the capability of distinguishing consecutive hybrid distributions into breaking some underlying assumption. (Above, we essentially showed that distinguishing H_n^i from H_n^{i+1} was equivalent to distinguishing the output of G from random.) Finally, the number of hybrid distributions should be polynomial. See also Theorem 7.32.

Ch 7 (cont.) | Section 7.5 onwards

Wednesday, October 18, 2023 10:02 AM

7.5 Constructing Pseudorandom Functions

Story

- We show how to construct a PRF from any (length doubling) PRG.
  - Recall
    a PRF is an efficiently computable
    keyed function F
    that is indistinguishable from
    a truly random function
    (in the sense described in Section 3.5.1)
  - For simplicity
    restrict here to the case where F is length preserving
    i.e. $k \in \{0,1\}^n$ the function F_k maps n-bits inputs to n-bit outputs.
  - A length preserving PRF can be viewed
    (informally) as a PRG with expansion factor $n \cdot 2^n$

    Given such a PRG G
    one could define $F_k(i)$ (for $0 \leq i < 2^n$)
    to be the ith n-bit block of $G(k)$.

    The reason this doesn't work
    is that F must be efficiently computable
    there are exponentially many blocks
    and we need to a way to comput ethe ith block
    without having to compute all other blocks.

- We will do this by computing
  "blocks" of the output by walking down a binary tree.
  - We exemplify the construction by
    first showing a PRF taking 2 bit inputs.
    - Let G be a PRG with expansion factor $2n$
    - If we use G as in the proof of Theorem 7.20
      we can obtain a PRG $\hat{G}$ with expansion factor $4n$
      that uses 3 invocations.
    - (We produce n additional pseudorandom bits,
      each time G is applied)

THEOREM 7.20 *If there exists a pseudorandom generator G with expansion factor $n+1$, then for any polynomial* poly *there exists a pseudorandom generator $\hat{G}$ with expansion factor* poly(n).

    - If we define $F'_k(i)$ (where $0 \leq i < 4$ and i is encoded as a 2-bit binary string)
      to be the ith block of $\hat{G}(k)$
      then computation of $F'_k(3)$
      would require computing all of $\hat{G}$
      and hence 3 invocations of G.

- We show how to construct a PRF F
  using only two invocations of G
  on any input.
  - Let G_0 and G_1 be functions
    denoting the first and second halves of
    the output of G
    i.e. $G(k) = G_0(k)||G_1(k)$
    where $|G_0(k)| = |G_1(k)| = |k|$.
  - Define F as follows:
    $F_k(ab) = G_b(G_a(k))$

$F_k(00) = G_0(G_0(k))$ $F_k(10) = G_0(G_1(k))$

where $|G_0(k)| = |G_1(k)| = |k|$.

- Define F as follows:

  $F_k(ab) = G_b(G_a(k))$

  $a, b \in \{0,1\}$

$$F_k(00) = G_0(G_0(k)) \qquad F_k(10) = G_0(G_1(k))$$
$$F_k(01) = G_1(G_0(k)) \qquad F_k(11) = G_1(G_1(k)).$$

- We claim that the four strings above are
  pseudorandm *even when viewed together*
  (this suffices to prove that F is pseudorandom).

- Intutively
  this is because
  $G_0(k)||G_1(k) = G(k)$
  is pseudorandom and hence
  indistinguishable from a uniform $2n$ bit string
  $k_0||k_1$.

- But then

$$G_0(G_0(k)) \,\|\, G_1(G_0(k)) \,\|\, G_0(G_1(k)) \,\|\, G_1(G_1(k))$$

  is indinguishable from

$$G_0(k_0) \,\|\, G_1(k_0) \,\|\, G_0(k_1) \,\|\, G_1(k_1) = G(k_0) \,\|\, G(k_1).$$

- Since G is a PRG
  the above is indistuinguishable from a $4n$-bit
  string.

- A formal proof uses a hybrid argument.

- Generalising this idea
  we can obtain a PRF on n bit inputs
  by defining
  $F_k(x) = G_{x_n}(\dots G_{x_1}(k) \cdots)$
  where $x = x_1 \dots x_n$

  $x_i \in \{0,1\}$

  (recall: $G(k)$ is parsed as $G_0(k)||G_1(k)$)

Construction 7.21

Let G be a PRG with expansion factor $l(n) = 2n$,
and
define G_0, G_1 as above.

For $k \in \{0,1\}^n$,
define the function $F_k: \{0,1\}^n \to \{0,1\}$ as:

$F_k(x_1 \dots x_n) = G_{x_n}(\dots G_{x_1}(k) \cdots)$.

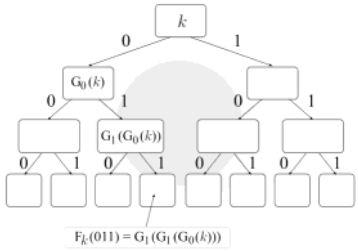
- The intuition for why this function is pseudorandom
  is the same as before
  but the formal proof is complicated by
  the fact that there are now exponentially many inputs to consider.

- It is useful to view this construction as defining
  (for each key $k \in \{0,1\}^n$)
  a complete binary tree of depth n
  in which each node contains an n-bit value.

(see figure 7.2 in which $n = 3$)

(for each key $k \in \{0,1\}^n$)
a complete binary tree of depth n
in which each node contains an n-bit value.

(see figure 7.2 in which $n = 3$)



- The root has value k
  and for every internal node with value k'
  its left child has value $G_0(k')$ and its right child has value $G_1(k')$

  The result $F_k(x)$ for $x = x_1 \dots x_n$
  is then defined to be the value
  on the leaf node reached
  by traversing the tree accordnig to the bits of x
  (where $x_i = 0$ means left, 1 means right)

- The function is only defined for inputs of length n
  and thus only values on the leaves are ever output;
  (i.e. don't need to go deeper)

- The size of the tree is exponential in n
  Yet,
  to compute $F_k(x)$
  the entire tree needn't be constructed
  or stored—only n evaluations of G are needed.

Theorem 7.22

If G is a PRG with expansion factor $l(n) = 2n$
then Construction 7.21
is a PRF.

Proof.

$$\left| \Pr_{k \leftarrow K}\left[D^{F_k(\cdot)} = 1 \right] - \Pr_{f \text{ random function}}\left[D^{f(\cdot)} = 1 \right] \right| \leq \text{negl}$$

We first show the following:

Claim: for any poly t
it is infeasible to distinguish
$t(n)$ uniform $2n$-bit strings from
$t(n)$ pseudorandom strings;
i.e. for any polynomial t and
any PPT algorithm A
the following is negligible:

$$\left| \Pr\left[A(r_1 \| \dots \| r_{t(n)}) = 1 \right] - \Pr\left[A(G(s_1) \| \dots \| G(s_{t(n)})) = 1 \right] \right|$$

where the first prob is over uniform choices of $r_1 \dots r_{t(n)} \in \{0,1\}^{2n}$
and the second is over uniform choices of $s_1 \dots s_{t(n)} \in \{0,1\}^n$

Proof of claim:
The proof is by a hybrid argument.
[skipping writing]

idea: consider the "hybrid"
$A(r_1, \dots r_{j-1}, w, G(s_{j+1}), \dots G(s_{t(n)}))$

Call this distribution G_n^i when w is uniformly chosen

Argue that all $G_n^i \approx G_n^{i+1}$ for all i for computationally bounded observers

Story:

- Returning to the crux of the proof
  we now show that F as in Construction 7.21
  is a PRF.

Let:
D be an arbitrary PPT distinguisher
given 1^n as input.

Story:

- We show that D
  cannot distinguish between the case when
  it is given oracle access to
  a function that is equal to F_k (for uniform k)
  or
  a function chosen uniformly from Func_n

- To do so,
  we use another hybrid argument.

- Here,
  we define a sequence of distributions
  over the values at the leaves of a complete binary tree
  of depth n.

- By associating each leaf with a string of length n
  as in Construction 7.21
  we can equivalently view these
  as distributions over functions
  mapping n bit inputs to n bit outputs.

Definition H_n^i

- For any n and $0 \leq i < n$
  let H_n^i be the following distribution
  over the values at the leaves of a binary tree of depth n

  - Choose values for the nodes at elevel i
    independently and uniformly from $\{0,1\}^n$

  - Then for every node at level i or below
    with value k,
    its left child is assigned $G_0(k)$
    and right child is assigned $G_1(k)$.

- NB:
  - Note that H_n^n corresponds to the distribution in which
    all values at the leaves are
    chosen uniformly and independently
    and thus
    corresponds to
    choosing a uniform function from Func_n
    whereas
    H_n^0 corresponds to choosing a uniform key k
    in Construction 7.21
    (since only the root in that case is chosen uniformly).

i.e.

$$\left| \Pr_{k \leftarrow \{0,1\}^n}\left[D^{F_k(\cdot)}(1^n)=1\right] - \Pr_{f \leftarrow \text{Func}_n}\left[D^{f(\cdot)}(1^n)=1\right] \right|$$

$$= \left| \Pr_{f \leftarrow H_n^0}[\ldots] - \Pr_{f \leftarrow H_n^n}[\ldots] \right| \qquad (7.13)$$

- Story: We show that Eq 7.13 is negligible (completing the proof)

- Let $t = t(n)$ be a poly upper bound on the number of queries D makes to its oracles
  on input 1^n.

  - Define a distinguisher A that
    tries to distinguish
    t(n) uniform $2n$ bit strings
    from
    t(n) pseudorandom strings.

Distinguisher A:

A is given as input $2n \cdot t(n)$ bit string
$w_1, \ldots, w_{t(n)}$.
(ws are $2n$ bit strings)

1. Uniformly sample $j \leftarrow \{0, \ldots n-1\}$
   In what follows
   A (implicitly)
   maintains a binary tree of depth n
   with n bit values at (a subset of the)
   internal nodes at depth $j+1$ and below.

2. Run $D(1^n)$.
   When D makes oracle query $x = x_1 \ldots x_n$,
   look at the prefix, $x_1 \ldots x_j$.

   There are two cases:

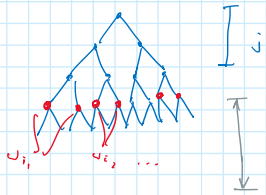
   - If D has never made a query with this prefix before
     then use $x_1 \ldots x_j$ to reach a node on the jth level of the tree.

     Take the next unused $2n$ bit string w
     (for each query made by D, use the next available w
     recall: the input to A was $t(n)$ many $2n$ bit strings w)

     and
     set the value of the left child of node v to the left half of w
     and
     the value of the right child of v to the righht half of w.

   - If D has made a query with prefix $x_1 \ldots x_j$ before
     then
     node $x_1 \ldots x_{j+1}$ has
     already been assigned value.

     Using the value at node $x_1 \ldots x_{j+1}$,
     compute the value at the leaf
     corresponding to $x_1 \ldots x_n$ as in
     Construction 7.21
     and
     return this value to D



3. When the execution of D is done
   output the bit returned by D.

Story:
- A runs in poly time
- NB: It is important here that A does not need to store
  the entire binary tree of exp size.
  - Instead, it "fills in" the values of at most $2t(n)$ nodes
    in the tree.

Argument:
Say A chooses $j = j^*$.

Observe that:
1. If A's input is a uniform $2n \cdot t(n)$ bit string
   then the answers it gives to D are distributed exactly as if
   D were interacting with a function
   chosen from distribution $H_n^{j^*+1}$.

   This holds because the values of the nodes at level j^*+1
   of the tree are uniform and indepnedent.

2. If A's input consists of $t(n)$ pseudorandom strings—
   i.e. $w_i = G(s_i)$ for uniform seed s_i—
   then the answers it gives to D
   are distributed exactly as if Dwere interacting
   with a function chosen from distribution $H_n^{j^*}$.

   This holds because the values of the nodes at level j^* of the tree
   (namely the s-values)
   are uniform and indepnedent
   (these s values are unknown to A

but this makes no difference).

Proceeding as before
one can show that

$$\left| \Pr\left[A(1^n, y_1 \| \dots \| y_{\ell(n)}) = 1\right] - \Pr\left[A(G(s_1) \| \dots \| G(s_{\ell(n)})) = 1\right] \right| \qquad \text{(Eq. 7.14)}$$

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

We have shown earlier
that Eq 7.14 must be negligible.

Thus, this implies Eq 7.13 must be negilgible as well.

□

7.6 Constructing (Strong) Pseduorandom Permutatinos

Story:

- We next show how pseudorandom permutations
and *strong* pseudorandom permutations
can be constructed from any pseudorandom function.

- Recall from Section 3.5.1
that a pseudorandom permutation
is a PRF that is also **efficiently invertible**
while a *strong* pseudorandom permutation
is additionally hard to distinguish
from a random permutation
even by an adversary
given accoss to both
the permutation and *its inverse*.

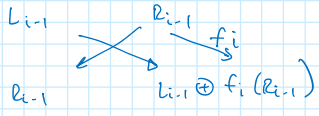
Fiestel networks revisited.

- A feistel network
introduced in Section 6.2.2
provides a way of constructing an invertible function
from an arbitrary set of functions.

- A Feistel networks operatos in a series of rounds:
  - Input to the ith round
  is a string of length $2n$,
  divided into two n bit halvies
  L_{i-1} and R_{i-1} (the "left half" and "right half" resp.)

  - The output of the ith round
  is the $2n$ bit string (L_i, R_i) where

$$L_i := R_{i-1} \text{ and } R_i := L_{i-1} \oplus f_i(R_{i-1})$$



for some efficiently computable
(but not necessarily invertible) function f_i
mapping n-bit inputs to n-bit outputs.

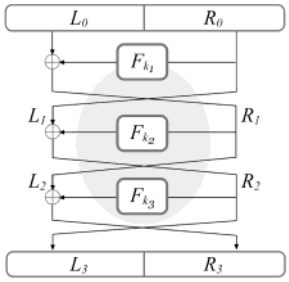


FIGURE 7.3: A three-round Feistel network, as used to construct a pseudorandom permutation from a pseudorandom function.

- We denote by $\text{Feistel}_{f_1 \ldots f_r}$ the r-round Feistel network using functions $f_1 \ldots f_r$

  (i.e. $\text{Feistel}_{f_1 \ldots f_r}(L_0, R_0)$ outputs the $2n$ bit string (L_r, R_r).

- We saw in Section 6.2.2 that $\text{Feistel}_{f_1 \ldots f_r}$ is an efficiently invertible permutation regardless of the $\{f_i\}$.

e.g. (invert) from $R_{i-1}, L_{i-1} \oplus f_i(R_{i-1})$ produce L_{i-1}, R_{i-1}

start with $f_i(R_{i-1})$ ∵ f_i is known

$\oplus$ xor with $L_{i-1} \oplus f_i(R_{i-1})$ to get L_{i-1}.

- We can define a keyed permutation by using a by using a Feistel network in which the $\{f_i\}$ depned on a key

  - E.g.

    let $F: \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be a PRF and define the keyed permutation F^1 as

    $$F_k^1(x) := \text{Feistel}_{F_k}(x)$$

    NB: F_k^1 has an n bit key and maps $2n$ bit inputs to $2n$ bit outputs.

    i.e. $F^1: \{0,1\}^{3n} \to \{0,1\}^{2n}$

  - Is F^1 pseudorandom?
    - A little thought shows that it is decidedly *not*.

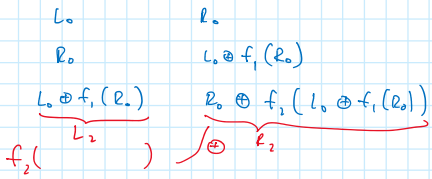
  - For any key $k \in \{0,1\}^n$ the first n bits of the output of F_k^1 (i.e. L_1) are equal to the last n bits of the input (i.e. R_0) something that occurs with only negligible probabbility for a random function.

  - Trying again, define $F^2: \{0,1\}^{2n} \times \{0,1\}^{2n} \to \{0,1\}^{2n}$ as fallows:

    $$F_{k_1,k_2}^2(x) := \text{Feistel}_{F_{k_1},F_{k_2}}(x)$$

    (Note that k_1 and k_2 are independent keys).

    Unfortunately, F^2 is not pseudorandom either (we are asked to prove in Exercise 7.16)



$f_2(L_2) \oplus R_2 = R_0$

should not hold for a random f^n...

- Given this it may be somewhat surprising that a *three*-round Feistel network *is* pseudorandom.

  - Define the keyed permutation F^3 taking a key of length $3n$ and mapping $2n$-bit inputs to $2n$ bit outputs as follows:

    $$F_{k_1,k_2,k_3}^3(x) := \text{Feistel}_{F_{k_1},\ldots F_{k_3}}(x)$$

where $k_1 \ldots k_3$ are indepnedent.

THEOREM 7.23

If F is a PRF
then F^3 is a pseudorandom permutation.

Proof:

- Let D be a PPT distinguisher.
  In the remainder of the proof
  we show the following is negligible:

$$\left|\Pr\left[D^{Feistel_{f_1 \ldots f_3}(\cdot)}(1^n) = 1\right] - \Pr\left[D^{\pi(\cdot)}(1^n) = 1\right]\right|$$

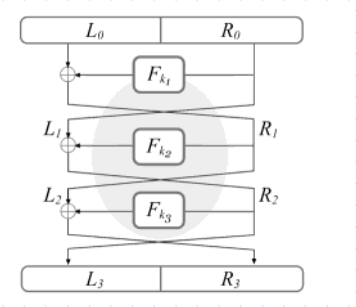
  where the first probability is taken over uniform
  and independent choice of
  f_1, f_2, f_3 from Func_n
  and
  the second probability is taken over uniform choice
  of π from Perm_{2n}.

- Fix some value for the security parameter n
  and
  let $q = q(n)$ denote
  a polynomial upper bound on the number of oracle queries
  made by D.

- We assume without loss of generality
  that D never makes the same oracle query twice.

- Story:
  Focussing on D's interaction with
  $\mathrm{Fiestel}_{f_1 \ldots f_3}(\cdot)$

  let (L_0^i, R_0^i) denote the
  ith query D makes to its oracle
  and
  let $(L_1^i, R_1^i) \ldots (L_3^i, R_3^i)$
  denote intermediate values
  after rounds 1 to 3,
  resp. that result from the query.



- Note that D chooses (L_{i0}, R_{i0}) and sees the result (L_{i3}, R_{i3})
  but does not directly observe
  L_{i1}, R_{i1} or L_{i2}, R_{i2}

  Me: Using subscripts only to make typing easier.

- Definition:
  There's a collision at R_1 if
  $R_{i1} = R_{j1}$ for some distinct i, j.

  (collision at the "first round")
  (i and j are query)

- Claim: We first
  prove that a collision at R_1 occurs
  with only negligible probability.

  Proof:

  - Consider any fixed $i \neq j$
  - If $R_{i0} = R_{j0}$ then $L_{i0} \neq L_{j0}$
    (because queries are assumed to be distinct)

    But then,
    $R_{i1} = L_{i0} \oplus f_1(R_{i0}) \neq L_{j0} \oplus f_1(R_{j0}) = R_{j1}$

    (because parity with the same string cannot map two strings to one)

    [THUS, there's no collision in this case; $R_{i1} \neq R_{j1}$]

  - If $R_{i0} \neq R_{j0}$ then
    $f_1(R_{i0})$ and $f_1(R_{j0})$ are uniform and independent

    so

    $\Pr[L_{i0} \oplus f_1(R_{i0}) = L_{j0} \oplus f_1(R_{j0})]$
    $= \Pr[f_1(R_{j0}) = L_{i0} \oplus f_1(R_{i0}) \oplus L_{j0}]$
    $= 2^{-n}$.

L_{i0} R_{i0}

L_{i1} R_{i0} $L_{i0} \oplus f_1(R_{i0})$ $= R_{i1}$

L_{i2} $L_{i0} \oplus f_1(R_{i0})$ $R_{i0} \oplus f_2(L_{i0} \oplus f_1(R_{i0}))$ $= R_{i2}$

L_{i3} $R_{i0} \oplus f_1(L_{i0} \oplus f_1(R_{i0}))$ $L_{i0} \oplus f_1(R_{i0}) \oplus f_2(R_{i0} \oplus f_2(L_{i0} \oplus f_1(R_{i0})))$ $= R_{i3}$

  - Taking a union bound over all distinct i, j gives

    $\Pr[\text{collision at } R_1] \leq q^2/2^n$.

□

M: $\Pr[S \leftarrow x : x \leftarrow U(1^n)] = \frac{1}{2^n} + \text{negl}$

M': $\Pr[S \leftarrow x : x \leftarrow \{0,1\}^n] = \frac{1}{2^n}$ ←

$|D(M, M')| \leq \text{negl}$.

Story:

- Suppose there's a collision at R_2
  if $R_{i2} = R_{j2}$ for some distinct i, j.
  - We prove that
    conditioned on no collision at R_1
    the probability of a collision at R_2 is negligible.
  - The analysis is as above:

consider any fixed i, j and note that if
there is no collision at R_1 then
$R_{i1} \neq R_{j1}$.

- Thus $f_2(R_{i1})$ and $f_2(R_{j1})$ are uniform and independent and therefore

$$\Pr\left[L_{i1} \oplus f_2(R_{i1}) = L_{j1} \oplus f_2(R_{j1}) \middle| \text{no collision at } R_1\right] = 2^{-n}.$$

(note that f_2 is independent of f_1, so calculations are easy).

- The union bound then gives

$$\Pr[\text{collision at } R_2 | \text{no collision at } R_1] \leq q^2/2^n.$$

NB: $L_{i3} = R_{i2} = L_{i1} \oplus f_2(R_{i1})$
thus conditioned on there being no collision
at R_1
the values $L_{13} \ldots L_{q3}$ are all independent and uniformly distributed in $\{0,1\}^n$.

Story:

- One can condition on there being on collisions in $L_{13} \ldots L_{q3}$
and they uniform in $\{0,1\}^n$ (conditioned on no collision),
i.e. uniform over all non-colliding sequences of length q.

- Similarly,
$R_{i3} = L_{i2} \oplus f_3(R_{i2})$

thus
conditioned on no collisions at R_2
the values $\mathrm{R}_{13} \ldots \mathrm{R}_{\mathrm{q3}}$ are uniformly distirbuted in $\{0,1\}^n$
independent of each other as well as $L_{13} \ldots L_{q3}$.

Summary:

- When querying F^3 (with uniform round functions)
on a series of q distinct inputs,
except with negligible probability
the output values $(L_{13}, R_{13}) \ldots (L_{q3}, R_{q3})$ are
distributed such that $\{L_{i3}\}$ are uniform and independent but distinct
and
the $\{R_{i3}\}$ are uniform and independent n-bit values.

< TODO: clarify

- In contrast,
when querying a random permutation on a series of q distinct inputs
the output values
$(L_{13}R_{13}) \ldots (L_{q3}R_{q3})$ are
uniform and independent
but
distinct $2n$ bit values.

- The best distinguishing attack for D, then
is t oguess that it is interacting with a random permutation
if $L_{i3} = L_{j3}$ for some distinct i, j.

- But this happens with negilgible probability in that case.

- This can be turned into a formal proof.

□

Story:

- F^3 is *not a strong* pseudorandom permutation
as you are asked to demonstrate in Exercise 7.17.

- Fortunately,
adding a fourth round *does* yield a strong pseduorandom permutation.

- The details are given as Construction 7.24

Theorem 7.25

If F is pseudorandom function then
Construction 7.24 is
a strong pseudorandm permutation that
maps $2n$-bit inputs to $2n$ bit outputs
(and uses a $4n$ bit key)

CONSTRUCTION 7.24

Let F be a keyed, length-preserving function. Define the keyed permutation $F^{(4)}$ as follows:

- **Inputs:** A key $k = (k_1, k_2, k_3, k_4)$ with $|k_i| = n$, and an input $x \in \{0,1\}^{2n}$ parsed as (L_0, R_0) with $|L_0| = |R_0| = n$.
- **Computation:**
  1. Compute $L_1 := R_0$ and $R_1 := L_0 \oplus F_{k_1}(R_0)$.
  2. Compute $L_2 := R_1$ and $R_2 := L_1 \oplus F_{k_2}(R_1)$.
  3. Compute $L_3 := R_2$ and $R_3 := L_2 \oplus F_{k_3}(R_2)$.
  4. Compute $L_4 := R_3$ and $R_4 := L_3 \oplus F_{k_4}(R_3)$.
  5. Output (L_4, R_4).

7.7 Assumptions for Private-Key Cryptography

Story:
- We have shown that < Goldreich Levin
  (1) if there exists one-way permutations
  then
  there exist pseudorandom generators
  (hard-core bit)

  (2) if there exist PRGs
  then
  there exist PRFs
  (tree)

  (3) if there exist PRF
  then
  there exist (strong) pseudorandom permutations.

- Although we did not prove it here
  it is possible to construct PRGs from OWF (we did it from permutations).

- We thus have the following fundamental theorem

Theorem 7.26
If OWFs exist,
then so do PRGs, PRFs and strong PRPs.

All the private key schemes we have studied in Ch 3 and 4
can be constructed from PRGs/PRFs. We thus have

Theorem 7.27
If OWFs exist
then so do CCA secure private-key encryption
schemes and
secure MACs.

Story:

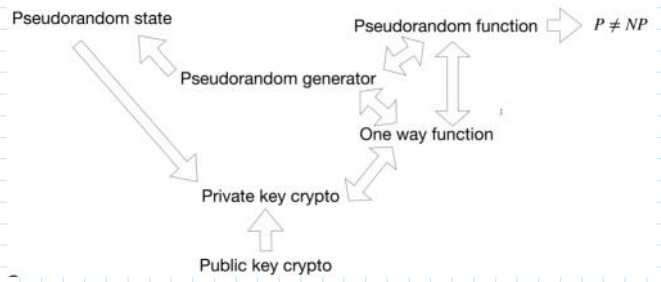
- i.e OWFs are sufficient
  for all private-key cryptography.

- Here we show that OWFs are also necessary.

Pseudorandomness implies OWFs.

We begin by showing that PRGs imply the existence of OWFs.

Proposition 7.28
If a PRG exists, then so does a OWF.



Proof.
Let:
G be a PRG with expansion factor $\ell(n) = 2n$.

(By Theorem 7.20,
we know that
the existence of a PRG implies the existence of
one with this expansion factor)

We show that G itself is one-way.

Efficient computability is straightforward
(since G can be computed in poly time).

We show that
the ability to invert G can be translated into
the ability to distinguish
the output of G from uniform.

Intuitively,
this holds because the ability to invert G
implies
the ability to find the seed used by the generator.

Let $\mathcal{A}$ be an arbitrary PPT algorithm (which is supposed to invert G)

We show that
$\Pr[\text{Invert}_{A,G}(n) = 1]$ is negligible (see game Definition 7.1).

To see this [by showing $\mathcal{A}$ can distinguish
consider the following PPT distingusher D:
on input a string $w \in \{0,1\}^{2n}$
run $\mathcal{A}(w)$ to obtain output s.

If $G(s) = w$ then output 1
otherwise
output 0.

We now analyse the behaviour of D.
First consider the probability that D outputs 1
when its input string w is uniform.

Since there are at most 2^n values in th erange of G
(namely, the values $\{G(s)\}_{s\in\{0,1\}^n}$,
the probability that w is in the range of G
is at most $\frac{2^n}{2^{2n}} = 2^{-n}$.

When w is not in the range of G
it is impossible for $\mathcal{A}$ to compute an inverse of w
and thus impossible for D to output 1.

We conclude that
$\Pr_{w\leftarrow\{0,1\}^{2n}}[D(w) = 1] \leq 2^{-n}$.

On the other hand
if $w = G(s)$ for a seed $s \in \{0,1\}^n$

- Suppose the PRG is not a one-way function
  - i.e. suppose the PRG is invertible
    - (Suppose the PRG can be inverted by $\mathcal{A}$)
  - then, the PRG ceases to be a PRG
    it can be distinguished from a random string.

- So, the strategy is to start with $\mathcal{A}$
  and construct a distinguisher D
  that can distinguish the output of the PRG from a uniform string.

chosen uniformly at random then
$\mathcal{A}$ computes a correct inverse
(and so D outputs 1)
with probability exactly equal to $\Pr[\text{Invert}_{A,G}(n) = 1]$.

Thus

$$\left|\Pr_{w\leftarrow\{0,1\}^{2n}}[D(w) = 1] - \Pr_{s\leftarrow\{0,1\}^{n}}[D(G(s)) = 1]\right|$$
$$\geq \Pr[\text{Invert}_{A,G}(n) = 1] - 2^{-n}$$

Since G is a PRG the above must be negligible.

Since 2^{-n} is negligible
it implies $\Pr[\text{Invert}_{A,G}(n) = 1]$
is negligible as well
and
so G is one-way.

□

Non-trivial private-key encryption implies one-way functions.

Story:

- Propopsition 7.28 (above) does not imply that
one-way functions are needed for
constructing secure private key encryption schemes
since
it may be possible to construct the latter
without relying on a pseudorandom generator.

- Furthermore
it *is* possible to construct
perfectly secret encryption schemes (see Ch 2)
as long as the plaintext is no longer than the key.

- Thus a proof that secure private-key encryption
implies one-way functions requires more care.

Proposition 7.29
If there exists an EAV-secure private-key encryption scheme
that encrypts messages twice as long as its key
then a OWF exists.

< Kishor:
why twice?
Could we say something in the cas
of n to $n + 1$?
i.e. key of size n
messages of size $n + 1$

Proof.

Let $\Pi = (\text{Enc}, \text{Dec})$
be a private key encryption scheme
that has indistinguishable encryptions
in the presence of an eavesdropper
and
encrypts messages of length $2n$
when the key has length n.

(We assume for simplicity that
the key is chosen uniformly)

Say that when an n bit key is used
Enc uses at most $\ell(n)$ bits of randomness.

Denote the encryption of a message m
using a key k and randomness r by $\text{Enc}_k(m; r)$.

Define the following function f:

$$f(k, m, r) := \text{Enc}_k(m; r)||m$$

where $|k| = n, |m| = 2n, |r| = \ell(n)$

We claim that f is a OWF.

Clearly
it can be efficiently computed.

We show that
it is hard to invert.

For any A PPT algorithm
we show that $\Pr[\text{Invert}_{A,f}(n) = 1]$ is negligible.

Consider the following PPT adversary A'
attacking private-key encryption scheme Π
(i.e. in experiment $\text{PrivK}^{\text{eav}}_{\Pi,A'}(n)$).

$A'(1^n)$ acts as follows:

1. Choose uniform $m_0, m_1 \leftarrow \{0,1\}^{2n}$
   and output them.
   Recieve and return a challenge ciphertext c.

2. Run $A(c||m_0)$ to obtain
   (k', m', r').

   If $(k', m,' r') = c||m_0$
   output 0
   else
   output 1

< Here
A takes as input elemnts in the codomain of f
this in turn is of the form ciphretxet|message

We now analyse the beahivour of A'.

When c is an encryption of m_0
then c||m_0 is distributed exactly as
$f(k, m_0, r)$ for uniform k, m_0 and r.

Therefore
A outputs a valid invers of $c||m_0$
(and hence A' outputs 0)
with prob exactly equal to $\Pr[\text{Invert}_{A,f}(n) = 1]$.

On the other hand
when c is an encrypton of m_1
then c independent of m_0

For any fixed value of the challenge
ciphertext c
there are at most
2^n possible messages
(one for each possible key)
to which c can correspond.

Since m_0 is a uniform $2n$-bit string
this means the prob there exists some key k
for which
$\text{Dec}_k(c) = m_0$ is at most $\frac{2^n}{2^{2n}} = 2^{-n}$.

This gives an upper bound on the probability with which
A can possibly output a valid inverse of
$c||m_0$ under f
and hence
an upper bound on the probability with which
A' outputs 0 in that case.

Putting the above together, we have

$$\Pr\left[\text{PrivK}^{\text{eav}}_{\Pi,A'}(n) = 1\right]$$
$$= \frac{1}{2} \cdot \Pr[A' \text{ outputs } 0 \mid b = 0] + \frac{1}{2} \cdot \Pr[A' \text{ outputs } 1 \mid b = 1]$$

< recall: b is $c = Enc_k(m_b)$

$$
\begin{aligned}
&\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\Pi,\mathcal{A}'}(n)=1\right] \\
&= \frac{1}{2}\cdot \Pr\left[\mathcal{A}' \text{ outputs } 0 \mid b=0\right] + \frac{1}{2}\cdot \Pr\left[\mathcal{A}' \text{ outputs } 1 \mid b=1\right] \\
&\geq \frac{1}{2}\cdot \Pr[\mathsf{Invert}_{\mathcal{A},f}(n)=1] + \frac{1}{2}\cdot\left(1-2^{-n}\right) \\
&= \frac{1}{2} + \frac{1}{2}\cdot\left(\Pr[\mathsf{Invert}_{\mathcal{A},f}(n)=1] - 2^{-n}\right).
\end{aligned}
$$

< recall: b is $c = Enc_k(m_b)$

Security of Π means that $\Pr\left[\mathrm{PrivK}_{\Pi,\mathcal{A}'}(n)=1\right] \leq \frac{1}{2} + \mathrm{negl}(n)$
and that in turn means
$\Pr[\mathrm{Invert}_{\mathcal{A},f}(n)=1]$ is also negligible,
completing the proof that f is one-way.

□

Message authentication codes imply one-way functions

- Claim: It is also true that
  message authentication codes satisfying Definition 4.2
  imply
  the existence of one-way functions.

Story:

- As in the case of private-key encryption
  a proof of this fact is somewhat subtle
  because
  unconditional MACs *do* exist when
  there is an apriori bound on the number of
  messages that will be authenticated.

  (see section 4.6)

- Thus,
  a proof relies on the fact that
  Definition 4.2 requires security
  even when
  the adversary sees the authentication tags
  of an *arbtirary* (poly) number of messages.

- The proof is involved so we skip it.

Discussion.

- We conculde that the existince of one-way functions
  is necessary and sufficient
  for all (non-trivial) private-key cryptography.

- In other words
  OWFs are a minimal assumption
  as far as private-key cryptography is concerned.

- Interestingly
  this appears not to be the case for hash functions
  and public-key encryptions
  where one-way functions are known to be necesary
  but are not known (or believed) to be sufficient.

7.8 Computational Indistinguishability

Story:

- The notion of computational indistinguishability is

central to the theory of cryptography
and it underlies
what we have seen in Ch 3 and this chapter.

- Informally,
  two probability distributions are
  computationally indistinguishable
  if no efficient algorithm can
  tell them apart
  (or distinguish them).

- In more detail
  consider two distributions
  X and Y over strings of some length ℓ

  i.e. X and Y each assign some probability
  to every string $\{0,1\}^{\ell}$

  - When we say that some algorithm D
    cannot distinguish these two dsitributinos
    we mean that D cannot tell whether
    it is given a string from
    distribution X or
    whether it is given a string
    sampled from dstributino Y.

  - Put differently
    if we imagen D outputting 0
    when it belives its input was sampled from X
    and 1 when it believes it was sampled from Y

    then
    the probability that D outputs 1
    should be roughly the same
    regardless of whether D is provided with a sample from X
    or from Y.

    In other words, we want

$$\left| \Pr_{s \leftarrow X}[D(s)=1] - \Pr_{s \leftarrow Y}[D(s)=1] \right| \text{ to be small.}$$

  - This should be reminiscent of the way
    we defined pseudorandom generators
    and
    iundeed
    we will soon formally redefine the notion
    of a pseudorandom generator
    using this terminology.

- The formal definition of computational indistinguishabalitiy
  refers to *probability ensembles*
  which are infinite sequneces of
  probability distributinos.

  (this formalism is necessary for a meaningful asymptotic approach).

  - Although the notion can be generalised
    for our purposes
    we consider probability ensembles ivn which
    the underlying distributinos are
    indexed by natural numbers.

  - If for every natural number n
    we have a distributino X_n
    then $\mathcal{X} = \{X_n\}_{n \in \mathbb{N}}$
    is a **probability ensemble**.

- It is often the case that
  $X_n = Y_{t(n)}$
  for some function t
  in which case we write
  $\{Y_{t(n)}\}_{n\in\mathbb{N}}$ in place of $\{X_n\}_{n\in\mathbb{N}}$

- We will only be interested in
  efficiently samplable
  probability ensembles.

  - An ensemble $\mathcal{X} = \{X_n\}_{n\in\mathbb{N}}$ is efficiently sampleable
    if there is a PPT algorithm S
    such that the random variables
    $S(1^n)$ and X_n are
    identically distributed.

  - That is,
    algorithm S is an efficient way of sampling $\mathcal{X}$.

- We can now
  formally define what it means for
  two ensembles to be
  computationally indistinguishable.

Definition 7.30

Two probability ensembles $\mathcal{X} = \{X_n\}_{n\in\mathbb{N}}$ and $\mathcal{Y} = \{Y_n\}_{n\in\mathbb{N}}$
are **computationally indistinguishable**,
denoted by

$$\mathcal{X} \overset{c}{\equiv} \mathcal{Y}$$

if for every PPT distingusiher D there exists a negligible function negl
such that

$$\left| \Pr_{x\leftarrow X_n}[D(1^n, x)=1] - \Pr_{y\leftarrow Y_n}[D(1^n, y)=1] \right| \le \mathrm{negl}(n)$$

Story

- In the definition, D is given the unary input 1^n
  so that it can run in time poly in n.

  - This is important
    when the outputs of X_n and Y_n may have
    length less than n.

  - As shorthand
    in probability expressions
    we will sometimes write X
    as a placeholder for a random sample from
    distribution X.

  i.e.
  we would write $\Pr[D(1^n, X_n) = 1]$
  in place of writing
  $\Pr_{x\leftarrow X_n}[D(1^n, x)=1]$.

Claim:

The relation of computational indistinguishability is transitive:

$$\mathcal{X} \overset{c}{\equiv} \mathcal{Y} \;\&\; \mathcal{Y} \overset{c}{\equiv} \mathcal{Z} \Rightarrow \mathcal{X} \overset{c}{\equiv} \mathcal{Z}$$

Pseudorandomness and pseudorandom generators.

- Pseudorandomness is
  just a special case of computational indistingusihability.
  - For any integer ℓ
    let U_ℓ denote the uniform distribution over $\{0,1\}^\ell$.
  - We can define a PRG as follows:

Definition 7.31

Let $\ell(\cdot)$ be a polynomial
let G be a (deterministic) poly time algorithm where
for all s it holds that $|G(s)| = \ell(|s|)$.

We say that G is a **pseudorandom generator** if the following two conditions hold:

1. (Expansion): For every n it holds that $\ell(n) > n$.
2. (pseudorandomness):
   The ensemble $\{G(U_n)\}_{n\in\mathbb{N}}$
   is computationally indistinguishable from
   the ensemble $\{U_{\ell(n)}\}_{n\in\mathbb{N}}$.

Story:

- Many of the other defintiions and assumptinos in this book
  can also be cas as special cases
  or
  variants of computational indistinguishability.

Multiple samples.

An important theorem regarding computational indistinguishability
is that
poly many samples of (efficiently sampleable)
computationally indistinguishable ensembles
are also computationally indistinguishable.

Theorem 7.32

Let $\mathcal{X}$ and $\mathcal{Y}$ be efficiently samplable probbaility ensembles
that are computationally indistguishable.

Then,
for every polynomial p
the ensemble

$$\overline{X} = \{(X_n^{(1)}, \ldots X_n^{(p(n))})\}_{n\in\mathbb{N}}$$

is computationally indistinguishable from the ensemble

$$\overline{Y} = \{(Y_n^{(1)}, \ldots Y_n^{(p(n))})\}_{n\in\mathbb{N}}$$

different samples from the same distribution Y.

Story:

- For example
  let G be a PRG with expansion factor $2n$
  in which case the ensembles $\{G(U_n)\}_{n\in\mathbb{N}}$ and $\{U_{2n}\}_{n\in\mathbb{N}}$
  are computationally indistinguishable.

- In the proof of Theorem 7.22
  we showed that for any polynomial t
  the ensembles

$$\{(\underbrace{G(U_n), \ldots G(U_n)}_{t(n)})\}_{n\in\mathbb{N}} \quad \& \quad \{(\underbrace{U_{2n}, \ldots U_{2n}}_{t(n)})\}_{n\in\mathbb{N}}$$

are also computationally indistinguishable.

Theorem 7.32 is proved by a hybrid argument
in exactly the same way.

References and Additional Reading

- OWF—Diffe Hellman, Yao
- Hard core predicates
  Blum and Micali
  exsitence for every OWF
  Goldreich and Levin
- First PRG construction
  Blum and Micali
- PRG from any one-way permutation
  Yao
- PRG from any OWF
  Håstad et al
- PRFs were defined and constructed by
  Goldreich Goldwasser and Micali
  - Extension to (strong) PRPs
    by Luby and Rackoff
- Fact that OWF are a necessary assumption
  for most of private key crypto was shown in [93]
- Goldreich's book [75] for more detail