

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 importance 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 assumption 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.)

Moreover
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 existence 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 disappointing 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 palatable 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 constructions 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.

Pseudorandom states and
Collision Resistant "Quantum
Hash function"

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

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 $f: \{0,1\}^* \rightarrow \{0,1\}^*$ 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).

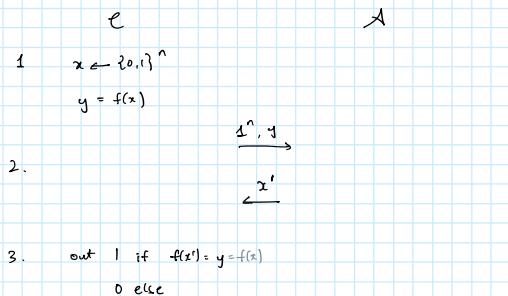
A technical point is that
this probability is taken over
an experiment in which
 y is generated by choosing a
uniform element x of the domain of f
and then setting $y := f(x)$

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

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

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

Invert _{A, f} (n)



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

it suffices 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 parameter n
regardless of the length of y .

Definition 7.1:

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

- (Easy to compute)**
There exists a poly-time algorithm M_f computing f
i.e. $M_f(x) = f(x)$ for all x .
- (Hard to invert)**
For every PPT algorithm \mathcal{A}
there is a negligible function negl such that
 $\Pr[\text{Invert}_{A,f}(n) = 1] \leq \text{negl}(n)$.

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 in the definition 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 probability 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 $x = f^{-1}(y)$.

$$\begin{aligned} & \exists c \quad \forall n \geq n_0 \quad \Pr_{x \sim \{0,1\}^n} [f(x) = y] \geq \frac{1}{n^c} \\ & \exists c \quad \forall n > n_0 \quad f(x) \leq \frac{1}{n^c} \quad \text{negligible} \\ & \exists c \quad \forall n > n_0 \quad f(x) \geq \frac{1}{n^c} \quad \text{non-neg} \end{aligned}$$

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

One-way function/permuation 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 permutations 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 should 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 parameters 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 $f_I: \mathcal{D}_I \rightarrow \mathcal{R}_I$ 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

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

ℓ

\mathcal{A}

1. $I \leftarrow \text{Gen}(1^r)$
 $x \leftarrow \text{Samp}(I)$
 (samples from D_I)
 $y := f_x(x)$

2.

```

    graph LR
        I[I] --> x[x]
        x -- f --> y[y]
        y -- "f^{-1}" --> x
    
```

3. out 1 if $f(x) = y$
 0 else

Definition 7.3

A function/permuation 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 working 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 unconditionally (this would be a major breakthrough in complexity theory)

so we must conjecture/assume their existence.

- Such a conjecture is based on the fact that several natural computational problems have received much attention and yet

have no PPT algorithm for solving them.

- Perhaps the most famous such problem is integer factorisation 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 $f_{\text{mult}}(x, y) = x \cdot y$.

- 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 inverse

- 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

- 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
- ◀ Kishor: Check connection NP completeness
- $$f_{ss}(x_1 \dots x_n, J) = \left(x_1 \dots x_n, \left[\sum_{j \in J} x_j \bmod 2^n \right] \right)$$
- where each x_i is an n -bit string
interpreted 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:
- $P \neq NP$ would mean that
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 least 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.
- 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:

$$f_{p,g}(x) = [g^x \bmod p]$$

(assertion: $f_{p,g}$ can be computed 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 function
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 definition 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 negligible 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 information about x that is "hidden" by $f(x)$.
- This motivates the notion of a "hardcore predicate"
 - A hard-core predicate $hc: \{0,1\}^* \rightarrow \{0,1\}$ of a function f has the property that $hc(x)$ 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 $hc(x)$ with probability $1/2$ by random guessing.

Formally:

Definition 7.4

A function $hc: \{0,1\}^* \rightarrow \{0,1\}$ 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

$$\Pr_{x \in \{0,1\}^n} [\mathcal{A}(i^n, -f(x)) = hc(x)] \leq \frac{1}{2} + negl(n)$$

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

Remarks:

- We stress that $hc(x)$ is efficiently computable given x
 (since the function hc can be computed in PT).

◦ The definition requires that $hc(x)$
 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
 $hc(x) := \bigoplus_{i=1}^n x_i$
 where $x_1 \dots x_n$ 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 the bits 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 hide the value of $hc(x) =$
 $\bigoplus_i x_i$
 because this is part of its output
 therefore $hc(x)$ 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. $f(x_1 \dots x_n) = x_1 \dots x_{n-1}$

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

thus, $hc(x) = x_n$ is a hard-core predicate of f .

- However, f is not one-way

- When we use hard-core predicates for our constructions
 for our constructions

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

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

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

- NB:

if r is uniform
then $hc(x, r)$ 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 the scope 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

$G(s) := f(s) \parallel hc(s)$
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 $hc(s)$ "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.

Pseudorandom generators with arbitrary expansion.

Story:

- The existence of a PRG that stretches its seed
by even a single bit (as we have just 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 pseudorandom 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 private-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.

DEFINITION 3.28 Let $F : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$ 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:

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

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 $f \in \text{Perm}_n$ and the randomness of D .

Corollary 7.11

Assuming the existence of one-way permutations
there exist

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

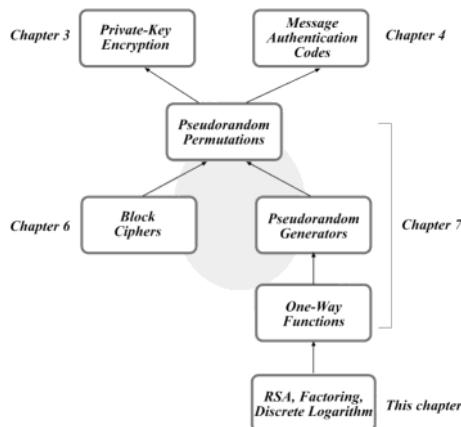


FIGURE 8.1: Private-key cryptography: a top-down approach.

Story:

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

7.3 Hard-Core Predicates from OWFs

Theorem 7.12

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

< Looks like Theorem 7.5 explicitly

Goldreich-Levin

Define $gl(x, r) := \bigoplus_{i=1}^n x_i \cdot r_i$
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.

7.3.1 A simple case

Story:

- We first show that if there exists a poly time adversary A that always correctly computes $g(x, r)$ 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 g be as in Theorem 7.12.

If there exists a poly time algorithm A such that

$$A(f(x), r) = g(x, r) \text{ for all } n \text{ and all } x, r \in \{0,1\}^n$$

then there exists a poly-time algorithm A' such that

$$A'(1^n, f(x)) = x \text{ for all } n \text{ and all } x \in \{0,1\}^n.$$

Proof

We construct A' as follows:

- $A'(1^n, y)$
 - computes $x_i := A(y, e_i)$
(here $e_i = (00 \dots 010 \dots 00)$ at the i th 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

$$\begin{aligned} x_i &= A(f(\hat{x}), e^i) \\ &= g(\hat{x}, e^i) \\ &\stackrel{\text{def}}{=} \sum_{j=1}^n \hat{x}_j \cdot e_{ij} \\ &= \hat{x}_i. \end{aligned}$$

- 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 $g(x, r)$ from $(f(x), r)$.
- This is a rather weak result that is very far from our ultimate goal of showing that $g(x, r)$ 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 $g(x, r)$ from $(f(x), r)$ with prob significantly better than 3/4.

- We will again show that any such A would implement the existence of a poly-time algorithm 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 $g(x, r)$ 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 g be as in Theorem 7.12.

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

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

for infinitely many values of n ,

then

there exists a PPT algorithm A' such that

$$\Pr_{x \leftarrow \{0,1\}^n} [A'(f(x), r) \in f^{-1}(f(x))] \geq \frac{1}{2 \cdot p(n)}$$

for infinitely many values n .

Proof

- The main observation underlying the proof of this proposition is that for every $r \in \{0,1\}^n$ the values $g(x, r \oplus e_i)$ and $g(x, r)$ together can be used to compute the i th bit of x .
- This is true because

$$\begin{aligned} & g(x, r) \oplus g(x, r \oplus e_i) \\ &= (\bigoplus_{j \neq i} x_j \cdot r_j) \oplus (\bigoplus_{j \neq i} x_j \cdot (r_j \oplus e_{ij})) \\ &\quad (\cancel{x_i \cdot r_i} \oplus \cancel{x_i \cdot r_i} \oplus \dots \oplus \cancel{x_i \cdot r_i}) \oplus \\ &\quad (\cancel{x_i \cdot r_i} \oplus \dots \oplus \cancel{x_i \cdot r_i} \oplus \dots \oplus \cancel{x_i \cdot r_i}) \\ &= x_i \cdot r_i \oplus (x_i \cdot \bar{r}_i) \\ &= x_i \end{aligned}$$

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 multiple random values of r using each one to obtain an estimate of x_i and will then take the estimate occurring 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

$$\Pr_{x \in \{0,1\}^n} [A(f(x), x) = g(x, x)] \geq \frac{3}{4} + \frac{1}{p(n)}$$

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

$$\Pr_{x \in \{0,1\}^n} [A(f(x), x) = g(x, x)] \geq \frac{3}{4} + \frac{1}{2p(n)}.$$

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

$$\Pr_{x \in \{0,1\}^n} [A(f(x), x) = g(x, x)] \geq \frac{3}{4} + \frac{\epsilon(n)}{2}.$$

We have

$$\begin{aligned} \Pr_{x \in \{0,1\}^n} [A(f(x), x) = g(x, x)] &= \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \Pr_{x \in S_n} [A(f(x), x) = g(x, x)] \\ &= \frac{1}{2^n} \sum_{x \in S_n} \Pr_{x \in \{0,1\}^n} [\dots] + \\ &\quad \sum_{x \notin S_n} \Pr_{x \in \{0,1\}^n} [\dots] \\ &\leq \frac{|S_n|}{2^n} + \frac{1}{2^n} \cdot \sum_{x \notin S_n} \left(\frac{3}{4} + \frac{\epsilon(n)}{2} \right) \\ &\leq \frac{|S_n|}{2^n} + \left(\frac{3}{4} + \frac{\epsilon(n)}{2} \right) \end{aligned}$$

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

$$\therefore \frac{3}{4} + \epsilon(n) \leq \Pr_{x, x \in \{0,1\}^n} [A(f(x), x) = g(x, x)]$$

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

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

□

Story:

- The following now follows as an easy consequence.

Claim 7.16

Let n be such that

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

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

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

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

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

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

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

- We are interested in lower bounding the prob that
A outputs the correct answer for both $g(x, r)$ and $g(x, r \oplus e_i)$;
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 $g(x, r)$ or $g(x, r \oplus e_i)$
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 $g(x, r)$ and $g(x, r \oplus e_i)$
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

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

- 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 n ,

$$\begin{aligned}
 & \Pr_{x \leftarrow \{0,1\}^n} [A'(1^n, f(x)) \in f^{-1}(f(x))] \\
 & \geq \Pr_{x \leftarrow \{0,1\}^n} [A'(1^n, f(x)) \in f^{-1}(f(x)) \mid x \in S_n] \cdot \Pr_{x \leftarrow \{0,1\}^n} [x \in S_n] \\
 & \geq \frac{1}{2} \cdot \frac{\epsilon}{2} = \frac{\epsilon}{4} \\
 & = \frac{1}{4p(n)}.
 \end{aligned}$$

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

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

- For $i = 1 \dots n$ 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 i th 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.
- Output $x = x_1 \dots x_n$.

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 $gl(\hat{x}, e_i)$ 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 $gl(\hat{x}, e_i)$ 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 $g(x, r)$ 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 $gl(x, r) \stackrel{\text{def}}{=} \bigoplus_{i=1}^n x_i \cdot r_i$, 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

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

for infinitely many values of n ,
then

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

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

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

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

Story:

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

Claim 7.18

Let n be such that

$$\Pr_{x \in \{0,1\}^n} [A(f(x), x) = g(x, x)] \geq \frac{1}{2} + \epsilon n$$

Then, there exists 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

$$\Pr_{x \in S_n} [A(f(x), x) = g(x, x)] \geq \frac{1}{2} + \frac{\epsilon n}{2}$$

<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:

- o 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

$$\Pr_{x \in S_n} [A(f(x), x) = g(x, x) \wedge A(f(x), x \oplus e_i) = g(x, x \oplus e_i)] \geq \epsilon n$$

for any i .

- Thus, if we try 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)

- o Instead, we design A' so that it computes $g(x, r)$ and $g(x, r \oplus e_i)$ 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 $g(x, r)$ itself.

The naive way to do this would be to choose the r s independently (as before) and have A' make an independent guess of $g(x, r)$ 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 r s
for each r
compute $A(f(x), r \oplus e_i)$ and guess $g(x, r)$
Since there are many r s
guessing $g(x, r)$ for each correctly
would happen with negligible probability.)

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

$$s^1 \dots s^\ell \in \{0,1\}^n$$

(To generate m samples

it first samples ℓ many s s of length n
where ℓ is the number of bits
needed to store m)

Then

for every non-empty subset $I \subseteq \{1, \dots, \ell\}$

$$\text{we set } r^I := \bigoplus_{i \in I} s^i.$$

Since there are $2^\ell - 1$ nonempty subsets

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

[Intuition: sample ℓ strings;

produce a new string by XORING a subset $I \subseteq \{1, \dots, \ell\}$

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 $2^{\log(\ell+1)} - 1 \geq m$ 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 (highlighted above).

Since s^j is included in the XOR that defines r^J

this implies that

r^J is uniform and independent of r^I as well.

We now have the following two important observations

1. Given $gl(x, s^1) \dots gl(x, s^\ell)$

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

This is because

$$gl(x, r^I) =$$

$$\begin{aligned} & gl(x, \bigoplus_{i \in I} s^i) \\ & \quad \stackrel{i \in I}{\underbrace{}}_{\text{first, then XOR}} \\ &= \bigoplus_{j=1}^{\ell} [x[j] \oplus (\bigoplus_{i \in I} s^i)[j]] \\ &= \bigoplus_{i \in I} \left(\bigoplus_{j=1}^{\ell} [x[j] \oplus s^i[j]] \right) \\ &= \bigoplus_{i \in I} gl(x, s^i) \end{aligned}$$

2. If A' simply guesses the values of

$gl(x, s^1) \dots gl(x, s^\ell)$
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-negligible probability A'

correctly guesses all the values
 $gl(x, s^1) \dots gl(x, s^\ell)$.

Combining the above

yields a way of obtaining $m = \text{poly}(n)$ uniform and pairwise-independent 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 $\ell := \log\left(\frac{2n}{\epsilon^2}\right) + 1$
2. Choose uniform, independent $s^1 \dots s^\ell \in \{0,1\}^n$ and $\sigma^1 \dots \sigma^\ell \in \{0,1\}$.
3. For every non-empty subset $I \subseteq \{1, \dots, \ell\}$,
compute $x^I := \bigoplus_{i \in I} x^i$ and $\sigma^I := \bigoplus_{i \in I} \sigma^i$.
f'm trying to find x^I equivalently.
intuitively, this is the guess for $A(y, x^I)$.
4. For $i = 1, \dots, n$, do the following
 - (a) If non-empty $I \subseteq \{1, \dots, \ell\}$ w/ $x^I \neq x^{\bar{I}}$, set $x_i^I := x^I \oplus A(y, x^I \oplus e^i)$
 - (b) Set $x_i := \text{majority}_I \{x_i^I\}$
e.g. take the bit that appeared a majority of the time!
5. Output $x = x_1 \dots x_n$

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

- It remains to compute the probability 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 $y = f(\hat{x})$ for 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 probability 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, $\sigma^I = \text{gl}(\hat{x}, x^I) \neq I$.
 $x^I = \bigoplus_{i \in I} s^i$
 - Fix an index $i \in \{1, \dots, n\}$ &

consider the prob. that

x obtained the correct value

$$x_i = \hat{x}_i.$$

- For any non-empty I

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

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

(on the choice of y).

($\because \hat{x} \in S_n$ & $x^I \oplus e^i$ is uniformly distributed)

- Thus, for any non-empty subset I ,

we have

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

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

recall the defn of x^I from the alg.)

- Moreover, the $\{x_i^I\}_{I \subseteq \{1, \dots, n\}}$ are pairwise independent

\therefore the $\{x^I\}_{I \subseteq \{1, \dots, n\}}$

(hence $\{x^I \oplus e^i\}_{I \subseteq \{1, \dots, n\}}$) 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 time among the $\{x_i^I\}_{I \subseteq \{1, \dots, n\}}$ one can apply Prop. A.13 to obtain

$$\begin{aligned} \Pr[x_i \neq \hat{x}_i] &\leq \frac{1}{4 \cdot \left(\frac{\epsilon}{2}\right)^2 \cdot (2^n - 1)} \\ &\quad \text{# sampled} \\ &\quad \text{recall: } l = \log_2 \frac{2^n}{\epsilon^2} + 1 \\ &\quad \text{# sampled} \\ &\leq \frac{1}{4 \cdot \left(\frac{\epsilon}{2}\right)^2 \cdot \frac{2^n}{\epsilon^2}} \\ &= \frac{1}{2n}. \end{aligned}$$

- 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 $\frac{1}{2}$.

$$\left(\because \sum_i \frac{1}{2n} = \frac{1}{2} \right)$$

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

u.p. $\geq \frac{1}{2}$

PROPOSITION A.13 Fix $\epsilon > 0$ and $b \in \{0, 1\}$, and let $\{X_i\}$ be pairwise-independent, 0/1-random variables for which $\Pr[X_i = b] \geq \frac{1}{2} + \epsilon$ 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

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

PROOF Assume $b = 1$; by symmetry, this is without loss of generality. Then $\Pr[X_i = 1] = \frac{1}{2} + \epsilon$. 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

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

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

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

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

\mathbb{P} w.p. $\geq \frac{1}{2}$
 $x = \hat{x}$

- Putting everything together,

Let n be as in claim 7.18 &

$$y = f(x).$$

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

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

All guess σ^i are correct w.p. at least

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

for a large enough n .

- Conditioned on both the above,

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

- Thus, the overall prob. with which λ' succeeds

$$\geq \text{at least } \left(\frac{\epsilon^2}{5n} \right) \left(\frac{1}{2} \right) = \frac{\epsilon^3}{20n} = \frac{1}{20n^{1/3}}$$

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 to extend 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:

- this means that $hc(s)$ "looks random" given $f(s)$ when s is uniform.
- Furthermore since f is a permutation $f(s)$ itself is uniformly distributed (apply a permutation to a uniformly distributed value yields a uniformly distributed value).
- So if s is a uniform n -bit string the $(n+1)$ -bit string $f(s)||hc(s)$ 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 $G(s) = f(s) \parallel hc(s)$ is a pseudorandom generator.

Theorem 7.19

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

Then,

algorithm G defined by $G(s) = f(s) \parallel hc(s)$ is a pseudorandom generator with expansion factor $l(n) = n + 1$.

Proof

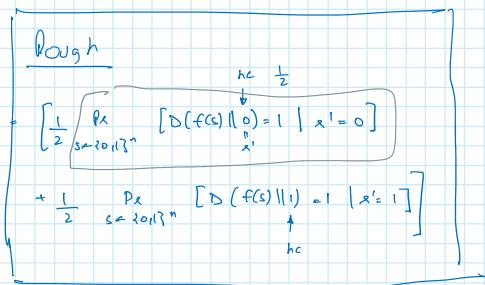
Let D be a PPT algorithm.

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

$$\Pr_{\substack{x \in \{0,1\}^{n+1} \\ s \in \{0,1\}^n}} [\Delta(x)=1] - \Pr_{\substack{s \in \{0,1\}^n \\ x' \in \{0,1\}^n}} [\Delta(G(s))=1] \leq \text{negl}(n) \quad (7.3)$$

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

$$\Pr_{\substack{s \in \{0,1\}^n \\ x' \in \{0,1\}^n}} [\Delta(G(s))=1] - \Pr_{\substack{s \in \{0,1\}^n \\ x' \in \{0,1\}^n}} [\Delta(x')=1] \leq \text{negl}'(n)$$



Observe:

$$\Pr_{\substack{x \in \{0,1\}^{n+1} \\ s \in \{0,1\}^n}} [\Delta(x)=1] = \Pr_{\substack{s \in \{0,1\}^n, x' \in \{0,1\}^n}} [\Delta(x \parallel x')=1]$$

$$= \Pr_{\substack{s \in \{0,1\}^n, x' \in \{0,1\}^n}} [\Delta(f(s) \parallel x')=1]$$

$$= \Pr_{\substack{s \in \{0,1\}^n}} [\Delta(f(s)) \parallel x' = 1] = \Pr_{\substack{s \in \{0,1\}^n}} [\Delta(f(s)) \parallel hc(s) = 1 | hc = 0]$$

$$= \Pr_{\substack{s \in \{0,1\}^n}} [\Delta(f(s)) \parallel hc(s) = 1 | hc = 0] + \Pr_{\substack{s \in \{0,1\}^n}} [\Delta(f(s)) \parallel hc(s) = 1 | hc = 1]$$

$$= \Pr_{\substack{s \in \{0,1\}^n}} [\Delta(f(s)) \parallel hc(s) = 1 | hc = 0] + \Pr_{\substack{s \in \{0,1\}^n}} [\Delta(f(s)) \parallel hc(s) = 1 | hc = 1]$$

$$\Pr_{\substack{s \in \{0,1\}^n \\ x' \in \{0,1\}^n}} [\Delta(f(s)) \parallel x' = 1]$$

$$\Pr_{\substack{s \in \{0,1\}^n \\ x' \in \{0,1\}^n}} [\Delta(f(s)) \parallel hc(s) = 1 | hc = 0] + \Pr_{\substack{s \in \{0,1\}^n \\ x' \in \{0,1\}^n}} [\Delta(f(s)) \parallel hc(s) = 1 | hc = 1]$$

$$\Pr_{\substack{s \in \{0,1\}^n \\ x' \in \{0,1\}^n}} [\Delta(f(s)) \parallel hc(s) = 1 | hc = 0] + \Pr_{\substack{s \in \{0,1\}^n \\ x' \in \{0,1\}^n}} [\Delta(f(s)) \parallel hc(s) = 1 | hc = 1]$$

$$\frac{1}{2} A + \frac{1}{2} B = \frac{1}{2} \left[\Pr_{\substack{s \in \{0,1\}^n}} [\Delta(f \parallel hc) = 1] + \Pr_{\substack{s \in \{0,1\}^n}} [\Delta(f \parallel \bar{hc}) = 1] \right]$$

$$= \frac{1}{2} \cdot \Pr_{\substack{s \in \{0,1\}^n}} [\Delta(f(s) \parallel hc(s)) = 1] + \frac{1}{2} \Pr_{\substack{s \in \{0,1\}^n}} [\Delta(f(s) \parallel \bar{hc}(s)) = 1]$$

using the fact that f is a permutation (for the second equality)

and that a uniform bit r' equals $hc(s)$ with prob. exactly $1/2$ for the "third" equality.

Since

$$\Pr_{\substack{s \in \{0,1\}^n}} [\Delta(x(s)) = 1] = \Pr_{\substack{s \in \{0,1\}^n}} [\Delta(f(s)) \parallel hc(s) = 1]$$

(by defn of x)

This means that Eq. 7.3 is equivalent to

$$\frac{1}{2} \cdot \left(\Pr_{s \in \{0,1\}^n} [\Delta(f \parallel \text{hc}) = 1] - \Pr_{s \in \{0,1\}^n} [\Delta(f \parallel \bar{\text{hc}}) = 1] \right) \leq \text{negl}$$

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 distinguish between hc and $\bar{\text{hc}}$
- 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 $\text{hc}(s)$:

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

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

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

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

$$\frac{1}{2} \cdot \left(\Pr_{s \in \{0,1\}^n} [D(f(s) \parallel \bar{\text{hc}}(s)) = 1] - \Pr_{s \in \{0,1\}^n} [D(f(s) \parallel \text{hc}(s)) = 1] \right) \leq \text{negl}(n),$$

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 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 $\text{poly}(n)$.

Proof

Story:

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

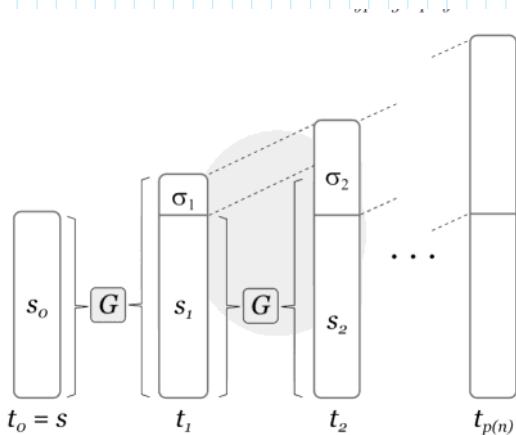
Definition:

\hat{G}

Input: initial seed $s \in \{0,1\}^n$

Outputs:

- $t_1 := G(s)$, to obtain $n + 1$ pseudorandom 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 uses 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

$$\{H_n^0\}_{n=1}^{\infty}, \quad \{H_n^1\}_{n=1}^{\infty}, \quad \text{and } \{H_n^2\}_{n=1}^{\infty}$$

where each of $H_n^0, H_n^1, H_n^2 \in \mathcal{D}_{n+2}$

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 $\hat{G}(t_0)$.

Recall:

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

In distribution H_n^1

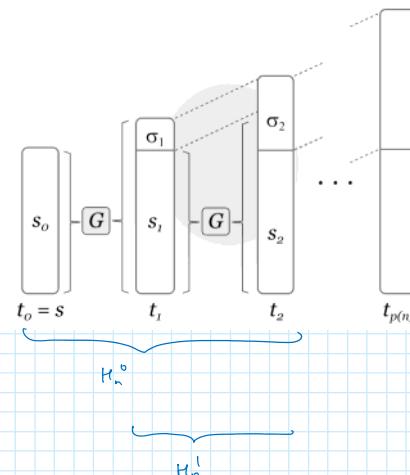
a uniform string $t_1 \in \{0,1\}^{n+1}$ is chosen
and parsed as $s_1 || \sigma_1$
(where s_1 are the initial n bits of t_1 and
 σ_1 is the final bit).
and the output is $G(s_1) || \sigma_1$

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:

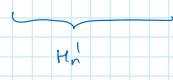
Story:

- Fix a PPT distinguisher D .

Claim:

There's a negligible function such that

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



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 generated by D' is
exactly that of H_n^1
(essentially by definition).

Thus

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

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

That is

$$\Pr_{s \leftarrow \{0,1\}^n} [\Delta'(G(s)) = 1] = \Pr_{t_1 \leftarrow H_n^0} [\Delta(t_1) = 1].$$

Pseudorandomness of G implies that
there is a negligible function negl' with

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

and this immediately yields 7.4
(proves the claim).

□

Story:

- We next claim the following:

Claim: There is a negligible function
 negl'' such that

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

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

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

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

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

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

□

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

Putting everything together, we have

$$\begin{aligned} & \left| \Pr_{\substack{s \leftarrow \{0,1\}^n \\ t_1 \leftarrow H_n^1}} [D(\hat{G}(s)) = 1] - \Pr_{\substack{s \leftarrow \{0,1\}^{n+2} \\ t_2 \leftarrow H_n^2}} [D(s) = 1] \right| \\ &= \left| \Pr_{\substack{s \leftarrow \{0,1\}^n \\ t_2 \leftarrow H_n^2}} [D(t_2) = 1] - \Pr_{\substack{s \leftarrow \{0,1\}^n \\ t_2 \leftarrow H_n^2}} [D(t_2) = 1] \right| \\ &\leq \left(\dots H_n^0 - \dots H_n^1 \right) + \left(\dots H_n^1 - \dots H_n^2 \right) \\ &\leq \text{negl}' + \text{negl}'' = \text{negl}''' \end{aligned}$$

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 negligible functions would not stay negligible)

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} :

$$\hat{G}(s) := f^l(s) \parallel hc(f^{l-1}(s)) \parallel \dots \parallel hc(s)$$

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

- Note:
 \hat{G} uses l evaluations 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 cipher (without an initial vector)
is defined by algorithms $\text{init}, \text{GetBits}$
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 be the trivial algorithm that outputs $st = s$
and define
GetBits(st) to compute
 $G(st)$
and parse the result as $st' || \sigma$

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.