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.

<
Pseudorandom states and
Collision Resistant "Quantum
Hash function"

We will turn to provable constructions of collision-resistant hash function
based on specific number theoretic assumptinos
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 $f: \{0,1\}^* \to \{0,1\}^*$ 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 $y \coloneqq f(x)$
(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:

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

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

1 $\quad x \leftarrow \{0,1\}^n$
$\quad y = f(x)$

2. $\quad \xrightarrow{1^n,\, y}$
$\quad \xleftarrow{x'}$

3. $\quad$ out 1 if $f(x') = y = f(x)$
$\quad$ 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
   $\Pr[\mathrm{Invert}_{A,f}(n) = 1] \leq \mathrm{negl}(n)$.

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

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

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

Successful inversion of one-way functions.

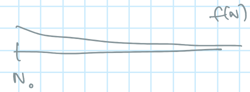
A function that is not one-way is
not necessarily easy to invert all the time
(or even "often").

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 $\mathcal{A}$)

noticeable
$\exists\, c, N_0$ s.t.
$N > N_0$, $f(n) \cdot n^c \geq 1$

negligible
$\forall c$
$\exists N_0$ s.t.
$\forall N > N_0$,
$f(N) \leq \frac{1}{n^c}$



non-negl
$\exists c$
$\forall N_0$
$\exists N > N_0$
$f(N) \geq \frac{1}{n^c}$

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 $\Pi = (\mathrm{Gen}, \mathrm{Samp}, f)$ 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 $y \coloneqq f_I(x)$.

Π is a permutation family if
for each value I output by $\mathrm{Gen}(1^n)$
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

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

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

1. $I \leftarrow \mathrm{Gen}(1^n)$

   $x \leftarrow \mathrm{Samp}(I)$

   (samples uniformly from D_I)

   $y := f_I(x)$

2. $\xrightarrow{I,\, y}$

   $\xleftarrow{x'}$

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

   0 else

Definition 7.3

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

$$\Pr\left[\mathrm{Invert}_{\mathcal{A},\Pi}(n) = 1\right] \le \mathrm{negl}(n).$$

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

  < Kishor: Check connection NP completeness

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

  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.

- 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
    $g \in \{2 \ldots p-1\}$ (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: $x \in \{1 \ldots p-1\}$.
  - Definition:
    $f_{p,g}(x) = [g^x \bmod p]$

    (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
      $f(x_1, x_2) \coloneqq (x_1, g(x_2))$
      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 $\mathrm{hc}\colon \{0,1\}^* \to \{0,1\}$ 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 $\mathrm{hc}\colon \{0,1\}^* \to \{0,1\}$ 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

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

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
$\mathrm{hc}(x) \coloneqq \bigoplus_{i=1}^{n} x_i$
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 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
$f(x) \coloneqq (g(x), \bigoplus_i x_i)$

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. $f(x_1 \dots x_n) = x_1 \dots x_{n-1}$

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 $g(x, r) \coloneqq (f(x), r)$ for $|x| = |r|$

  and define

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

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
$G(s) \coloneqq f(s)||\mathrm{hc}(s)$
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 $\mathrm{hc}(s)$ "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:

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.

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.

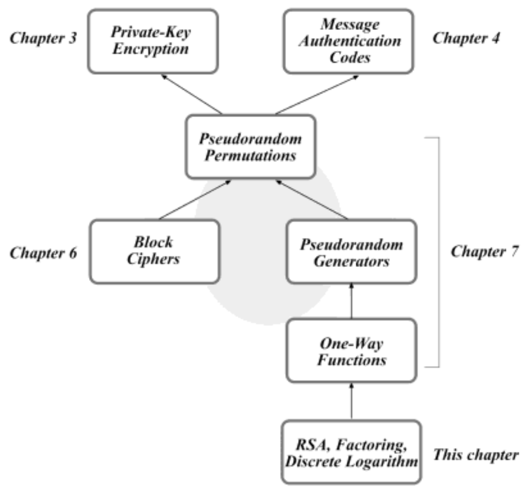


FIGURE 8.1: Private-key cryptography: a top-down approach.

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

Then gl is a hard-core predicate of g.

< Looks like Theorem 7.5 explicitly

Goldreich-Levin

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 $\mathrm{gl}(x,r)$
  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
$A(f(x),r) = \mathrm{gl}(x,r)$ 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 $x_i \coloneqq A(y, e_i)$
    (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

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

- 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
    $\text{gl}(x, r)$ 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 $\text{gl}(x, r)$ 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

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

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

$$(\cancel{x_1 \cdot r_1} \oplus \cancel{x_2 \cdot r_2} \oplus \dots \oplus \cancel{x_n \cdot r_n}) \oplus$$
$$(\cancel{x_1 \cdot r_1} \oplus \dots \oplus x_i \cdot \bar{r}_i \oplus \dots \oplus \cancel{x_n \cdot r_n})$$

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

$$= x_i$$

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 multpiple 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

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

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 $x \in S_n$ it holds that

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

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

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

We have

$$\frac{3}{4} + \epsilon(n) \leq \Pr_{x,r \leftarrow \{0,1\}^n}\left[\mathcal{A}(f(x), r) = gl(x,r)\right] = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \Pr_{r \leftarrow \{0,1\}^n}\left[\mathcal{A}(f(x), r) = gl(x,r)\right]$$

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

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

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

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

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

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

□

Story:

- The following now follows as an easy consequence.

Claim 7.16

Let n be such that

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

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

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

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

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

- We are interested in lower bonuding the prob that
  A outputs the correct answer for both gl(x, r) and gl(x, r ⊕ 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 $i = 1 \dots n$ 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 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 $x = x_1 \dots x_n$.

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

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

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

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

Story:

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

Claim 7.18
Let n be such that

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

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

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

<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

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

  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 $\text{gl}(x, r)$ and $\text{gl}(x, r \oplus e_i)$ 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 $\text{gl}(x, r)$ itself.

The naive way to do this
would be to choose the rs independently
(as before)
and have A' make an independent guess of
$\text{gl}(x, r)$ 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 $\text{gl}(x, r)$
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
$s^1 \dots s^\ell \in \{0,1\}^n$

(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 $I \subseteq \{1, \dots \ell\}$
we set $r^I \coloneqq \bigoplus_{i \in I} s^i$.

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 $I \subseteq \{1 \dots \ell\}$
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 $2^{\log m+1} - 1 \geq m$ 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 $\mathrm{gl}(x, s^1) \dots \mathrm{gl}(x, s^\ell)$

   it is possible to compute $\mathrm{gl}(x, r^I)$

   for every subset $I \subseteq \{1 \dots \ell\}$.

   This is because

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

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

   s^is first,

   then XOR

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

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

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

   XOR first,

   then s^is

2. If A' simply guesses the values of

   $\mathrm{gl}(x, s^1) \dots \mathrm{gl}(x, s^\ell)$

   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

   $\mathrm{gl}(x, s^1) \dots \mathrm{gl}(x, s^\ell)$.

Combining the above

yields a way of obtaining $m = \mathrm{poly}(n)$ uniform and

pairwise-independ independent strings $\{r^I\}$

along with *correct* values for $\{\mathrm{gl}(x, r^I)\}$

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 $s^I := \bigoplus_{i \in I} s^i$ &
   $\sigma^I := \bigoplus_{i \in I} \sigma^i$

   I'm treating $s^i =: s^i$ equivalently.

   intuitively, this is the guess for $A(y, s^I)$.

4. For $i = 1, \dots n$, do the following
   (a) $\forall$ non-empty $I \subseteq \{1, \dots \ell\}$ set
   $$x_i^I := \sigma^I \oplus A(y, s^I \oplus e^i)$$
   (b) Set $x_i := \text{majority}_I \{x_i^I\}$
   (i.e. take the bit that appeared a majority of the times)

5. Output $x = x_1 \dots x_n$

[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, $\sigma^I = \text{gl}(\hat{x}, s^I)$ $\forall$ I.
    $s^I = \bigoplus_{i \in I} s^i$
  - Fix an index $i \in \{1, \dots n\}$ &
    consider the prob. that
    A' obtains the correct value
    $x_i = \hat{x}_i$.
  - For any non-empty I
    we have $A(y, s^I \oplus e^i) = \text{gl}(\hat{x}, s^I \oplus e^i)$
    with prob. at least $\frac{1}{2} + \frac{\epsilon}{2}$

(over the choice of r).

($\because$ $\hat{s} \in S_n$ & $r^I \oplus e_i$ is uniformly distributed)

- Thus, for any non-empty subset I, we have

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

($\because$ we already conditioned on the other "guess" being correct;
recall the defn of x^I from the alg)

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

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

(& hence $\{r^I \oplus e^i\}_{I \subseteq \{1,\dots,\ell\}}$) 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

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. ■

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

$\because$ we were correct in I trial w.p. $\frac{1}{2} + \frac{\epsilon}{2}$

recall: $\ell = \log \frac{2n}{\epsilon^2} + 1$

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

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

- The above holds for all i, so by applying a union bound we see that

$\Pr_r$ $x_i \neq \hat{x}_i$ for some i, is at most $\frac{1}{2}$.

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

i.e. $(x_i = \hat{x}_i \;\; \forall i)$ w.p. $\geq \frac{1}{2}$

$\Updownarrow$

$x = \hat{x}$

- 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 guess 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 $\left(\frac{\epsilon^2}{5n}\right)\left(\frac{\epsilon}{2}\right)\cdot\frac{1}{2} = \frac{\epsilon^3}{20n} = \frac{1}{20np^3}$

for infinitely many ns.

□