CS 282A/MATH 209A: Foundations of Cryptography — Prof. Rafail Ostrosky

Lecture 3

Lecture date: Mon/Wed, 24/26 of January, 2005 — *Scribe: Urwin, Karimi, Cheng*

1 Hard-Core Bits

1.1 Introduction to Hard-Core Bits

In this lecture our goal is to discuss Hard-Core Bits and draw conclusions from definitions presented along the way. Hard-Core Bits were defined by Blum and Micali in 1982. Informally, a Hard-Core Bit $B(\cdot)$ of a one-way function $f(\cdot)$ is a bit which is as hard to compute as it is to invert f. Blum and Micali showed that a particular number theoretic function (which is believed to be one-way) has a Hard-Core Bit. It was later shown that all (padded) one-way functions have a Hard-Core Bit. We conclude by presenting this proof (due to Goldreich and Levin 1989).

Motivating example (due to Manuel Blum): Consider the problem of gambling on the outcome of a random coin flip with an adversary over a telephone line. If you bet on heads and allow the adversary to flip the coin and then inform you of the outcome, he may cheat and say tails without even bothering to flip the coin. Now suppose that after losing quite a bit of money, you decide to play a more sophisticated game in which both you and the adversary select a random bit and you win if the XOR of the two bits is 1. Unfortunately, it is still unsafe to transmit your random bit in the clear to an un-trustworthy adversary, for your adversary can always cheat by claiming that it selected the same bit.

To keep from being swindled further, you decide on the following commitment protocol to play the game described above fairly. You begin by sending the adversary your bit in a locked safe, then the adversary sends you its bit in the clear, and finally you send the adversary the combination to the safe. Both of you then compute the XOR of the two bits certain that the other party had no unfair advantage playing the game. We use this analogy to motivate the idea that it may be possible to send a commitment of a secret bit to an adversary, without revealing any information as to the value of that bit. Our objective is to develop such a legitimate commitment protocol based on one-way functions.

Assume that we have a one-way function. One (unfair) strategy would be to commit to b by sending $b \oplus x_3$ with $f(x)$. The flaw with this strategy is that the player can cheat, since $f(x)$ might not have unique inverses. In particular, suppose $f(x)$ has inverses x_1 and x_2 such that the third bit of x_1 and x_2 differ. Then once the adversary presents its random bit

in the clear, the player can choose to transmit either x_1 or x_2 to the adversary, and clearly will choose to transmit the one which results in a payoff.

What if we assume much more, i.e. that we have a 1-1, length-preserving one-way function? The sender can no longer cheat in the manner described above, but the receiver may still be able to cheat. Just because $f(x)$ is hard to invert does not necessarily mean that any individual bit of $f(x)$ is hard to invert. As an example, suppose we have a one-way function $f(x)$ and another function $g(x) = g(b_1, b_2, b_3, x_4, x_5, \ldots, x_n) = b_1 b_2 b_3 f(x_4, x_5, \ldots, x_n)$. Now since f is one-way, g is also one-way, yet the three highest-order bits of $g(x)$ are simple to invert.

1.2 Bit Commit Protocol

Details

1. *Alice* flips r_0 and locks the result in a safe deposit box.

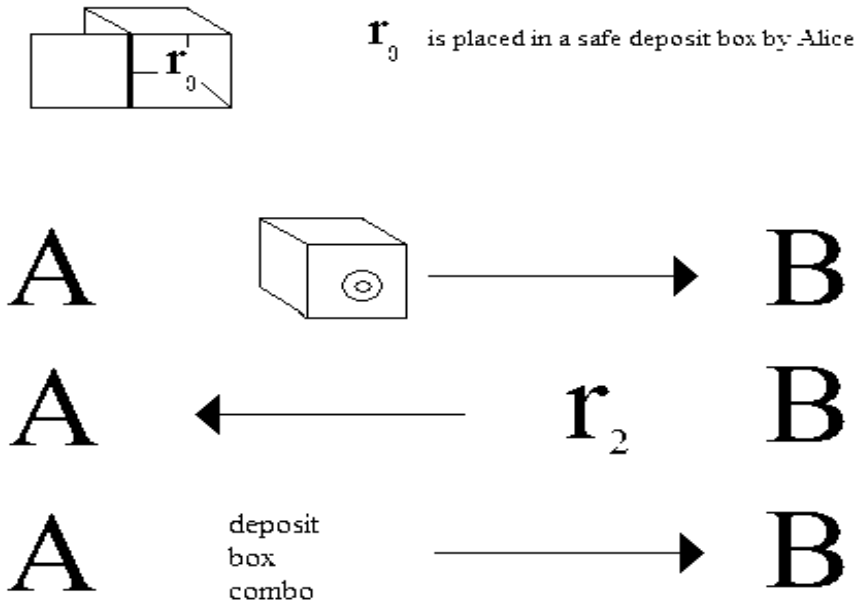
2. The locked safe deposit box with r_0 inside is given to *Bob*.

3. *Bob* in turn flips r_2 in the open and sends to result to *Alice*.

4. *Alice* then sends the deposit box combination for *Bob* to open the box containing the outcome of r_0.

5. *Alice* and *Bob* then exclusive-or the two flips r_0 and r_2 (*ie*) $coin = r_0 \bigoplus r_2$.

Diagram 1 *Alice and Bob wish to flip an unbiased coin*



concluding with the coin $= r_0 \bigoplus r_2$

This example describes what is called a Bit Commitment (BC) Protocol. It is the idea where Alice commits a bit and sends it to Bob without revealing the value of the bit. There are two properties that we wish to have in a BC Protocol.

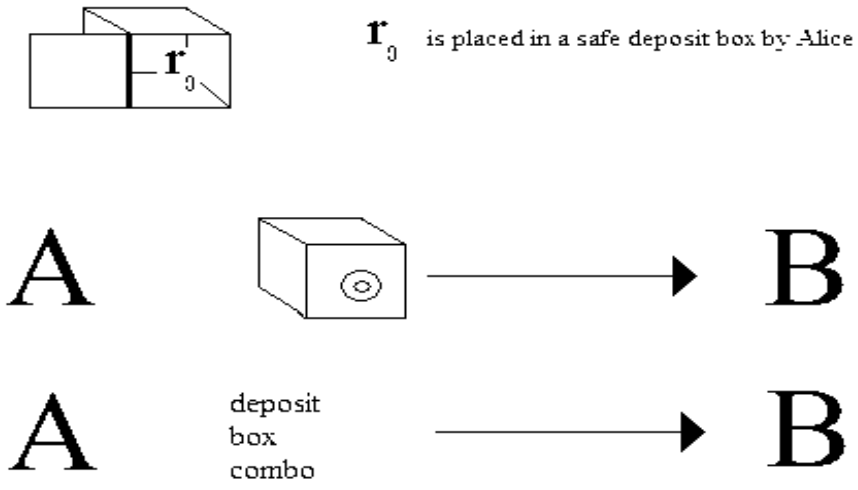
(1) Given the 'box', Bob cannot predict what's 'in it' with probability $\geq \frac{1}{2}$ + a negligible amount.

(2) After committing, Alice cannot change her mind about what is in the 'box'.

This exchange is known as the Commit Phase, where two events take place: hiding and binding. Alice places the outcome of r_0 in the safe deposit box is the hiding event. Hiding provides that Bob can not predict the outcome of r_0 with probability greater than $\frac{1}{2} + \frac{1}{n^c}$. Taking the safe deposit box and sending it to Bob is the binding event. Once Alice commits to the contents of the safe deposit box, she can not change her mind.

The Commit Phase is followed by the De-commit Phase in which Alice sends the combination to Bob.

Diagram 2 *Pictorially, the Commit phase looks like:*



Definition 3 ***Bit Commit Protocol*** -

Commit phase:

hiding - Given a locked safe deposit box, Bob can not predict what is in the box with probability $> \frac{1}{2} + \epsilon$

binding - After the commit phase Alice can not change her mind.

De-Commit phase:

Now Alice tells Bob the combination and he can unlock the safe deposit box and examine the contents.

Going further in depth we see that we need to somehow 'commit' the coin flip that Alice initially sends to Bob. As discussed before, a function that is one-way and 1-1 can still reveal a large part of its input. Instead, we look for some bit of information that is hard to compute. If we can find this bit b, then we can use it to 'commit' Alice's coin flip (i.e. $b \bigoplus r_0$). This bit b is what we call a Hard-Core Bit and we discuss it thoroughly in the next section.

1.3 Definition of a Hard-Core Bit

These examples motivate the following definition of a Hard-Core Bit due to Blum and Micali. Intuitively, a Hard-Core Bit is a bit associated with a one-way function which is as hard to determine as is inverting the one-way function.

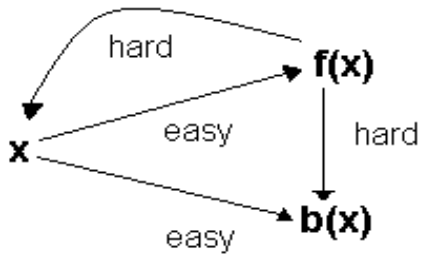
Definition 4 A **Hard-Core Bit** $B(\cdot)$ of a function $f(\cdot)$ is a boolean predicate such that:

- $B(x)$ is easy to compute given x, i.e. in deterministic polytime
- Given $f(x), B(x)$ is hard to guess better than at random:

$$\forall c \; \forall \text{ probabilistic poly-time } A, \text{ there exists } N_c \text{ such that } \forall n > N_c$$

$$Pr_{\{x,\omega\}}[A(f(x)) = B(x)] < \frac{1}{2} + \frac{1}{n^c}$$

where $|x| = n$, and probability is taken over x and coin-flips ω of A.



1.4 Does the existence of a hard-core bit $B(\cdot)$ for a function $f(\cdot)$ implies that f is a one-way function?

We note first that the existence of a Hard-Core Bit for f does not necessarily imply that the corresponding one-way function is hard. As an example, the almost-identity function $I(b, x) = x$ has a hard-core bit b but is not hard-to-invert in the sense that we have defined in previous lectures. However, if no information is lost by the function f, then the existence of a Hard-Core Bit guarantees the existence of a one-way function. We prove a somewhat weaker theorem below.

Theorem 5 If f is a permutation which has a Hard-Core Bit, then f is a one-way function.

Proof Assume f is not one-way. Then there exists a good inverter for f which correctly computes inverses with probability $q > \epsilon(n)$, where probability is taken over x and coin-flips of A. The predictor for the Hard-Core Bit B first attempts to invert f using this good inversion strategy. If it succeeds in inverting f, it knows x, and can compute $B(x)$ in polynomial time. Otherwise, with probability $1 - q$, it fails to invert f, and flips a coin as its guess for $B(x)$. The predictor predicts B correctly with probability

$$q \cdot 1 + (1 - q) \cdot \frac{1}{2} = \frac{1}{2} + \frac{q}{2} \geq \frac{1}{2} + \frac{\epsilon(n)}{2}$$

Therefore f does not have a Hard-Core Bit, proving the contrapositive. ■

1.5 One-way functions have hard-core bits

The next two lectures are devoted to a proof of the following important theorem, first proved in 1989 by Goldreich and Levin, then simplified (by Venkatesan and Rackoff). It says that if $f_1(x)$ is a strong one-way function, then parity of a random subset of bits of x is a Hard-Core Bit:

Theorem 6 [Goldreich, Levin] Let f_1 be a strong one-way function. Let $f_2(x, p) \equiv (f_1(x), p)$, where $|x| = |p| = n$. Then

$$B(x, p) \equiv \sum_{i=1}^{n} x_i p_i \bmod 2$$

is hard-core for f_2.

Notice that a random subset is chosen by choosing p at random. The Hard-Core Bit of x is simply parity of a subset of bits of x, where the subset corresponds to all bits of x where corresponding bits of p are set to one.

Proof outline

The proof that $B(x, p)$ is a Hard-Core Bit will be by contradiction. We begin by assuming that $B(x, p)$ is not a Hard-Core Bit for f_2. That is:

$B(\cdot, \cdot)$ is not hard-core: $\exists A_B \in \text{PPT}$, $\exists c$ such that for infinitely many n,

$$Pr_{\{x,p,\omega\}}[A_B(f_2(x, p)) = B(x, p)] > \frac{1}{2} + \frac{1}{n^c} \equiv \frac{1}{2} + \epsilon(n)$$

where A_B is probabilistic poly-time and probability is taken over x, p and coins ω of A_B

We want to show that we can invert f_2 with noticeable probability, proving that f_2 (and likewise f_1) is not a strong one-way function, i.e.:

f_2 is not a strong one-way function: $\exists A_{f_2} \in$ PPT, $\exists c$ such that for infinitely many n:

$$Pr_{\{x,p,\omega\}}[A_{f_2} \text{ inverts } f_2(x,p)] > \frac{1}{n^c}$$

where A_{f_2} is probabilistic poly-time and probability is taken over x, p and coins of A_{f_2}

We will show how to construct A_{f_2} using A_B as s subroutine.

Preliminaries

First, let us recall some useful definitions and bounds. Recall that a set of random variables is *pairwise independent* if given the value of a single random variable, the probability distribution of any other random variable form the collection is not affected. That is,

Definition 7 Pairwise independence: A set of random variables $X_1, \ldots, X_n$ are pairwise independent if $\forall i \neq j$ and $\forall a, b$:

$$Pr_{\{X_i,X_j\}}[X_i = a \wedge X_j = b] = Pr_{X_i}[X_i = a] \cdot Pr_{X_j}[X_j = b]$$

As an example of pairwise independence, consider distribution of three coins, taken uniformly from: {HHH,HTT,THT,TTH}. It is easy to check that given an outcome of any one of the three coins, the outcome of any other (of the two remaining) coins is still uniformly distributed. Notice however, that the number of *sample points* is small (only 4). On the other hand, for total (i.e. three-wise) independence we need all 8 combinations.

We are sometimes interested in bounding tail probabilities of large deviations. In particular, recall a Chernoff bound which we already used:

Definition 8 Chernoff bound: Let $X_1, \ldots, X_m$ be (totally) independent $0/1$ random variables with common probability $0 < p < 1$, and let $S_m = X_1 + X_2 + \ldots X_m$. Then

$$Pr_{\{X_1,\ldots,X_m\}}[|S_m - pm| > \delta m] \leq 2e^{-\frac{\delta^2 m}{2}}$$

Notice that as a function of m, the error-probability in Chernoff bound drops exponentially fast. In case of *pairwise independence* we have an analogous, Chebyshev bound. Like the Chernoff bound, the Chebyshev bound states that a sum of identically distributed 0/1 random variables deviates far from its mean with low probability which decreases with the number of trials (i.e. m). Unlike the Chernoff bound, in Chebyshev bound the trials need only be pairwise independent, but the probability drops off only polynomialy (as opposed to exponentially) with respect to the number of trials.

Definition 9 Chebyshev bound: Let $X_1, \ldots, X_m$ be pairwise independent 0/1 random variables with common probability $0 < p < 1$, and let $S_m = X_1 + X_2 + \ldots X_m$. Then

$$Pr_{\{X_1,\ldots,X_m\}}[|S_m - pm| > \delta m] \leq \frac{1}{4\delta^2 m}$$

We also implicitly used before a union bound, which simply states that:

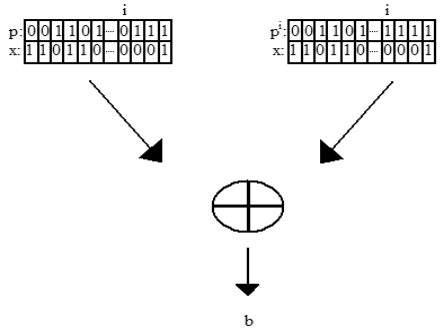
Definition 10 Union Bound: For any two events A and B (which need not be independent), the

$$Pr[A \bigcup B] \leq Pr[A] + Pr[B]$$

Observation 11 *Making only the i^{th} position of p 1 and the rest 0 is unreasonable because p is chosen at random. Instead it is just as effective to only flip one bit of p and xor the results to get the bit of x. Now the i^{th} position in p is flipped and still gives the same result as before:*

$< p \cdot x > \oplus < p^i \cdot x > = i^{th}$ *bit of* x

Pictorially:



Observation 12 *Probability Distribution on 3 bits.*

In this observation, Pairwise Independence can be understood through a simple example. Below are three binary variables b_0, b_1, b_2 with equal distribution (i.e. uniformly random). However, not all three random variables need to be generated. With only two of them the last one can be found using Pairwise Independence.

$$\begin{array}{c} b_0\ b_1\ b_2 \\ 0\ 0\ 0 \\ 0\ 1\ 1 \\ 1\ 0\ 1 \\ 1\ 1\ 0 \end{array}$$

This can be accomplished by taking any two columns and xor-ing them. All possible choices look random when you apply this property over the variable values.

$$\begin{array}{ll} b_0 b_1 \rightarrow b_0 \oplus b_1 & this\ looks\ like\ b_2 \\ b_0 b_2 \rightarrow b_0 \oplus b_2 & this\ looks\ like\ b_1 \\ b_1 b_2 \rightarrow b_1 \oplus b_2 & this\ looks\ like\ b_0 \end{array}$$

Observation 13 *What is the probability that the adversary is right?*

- *Assume all log(n) bits guessed are correct. This happens with probability 1/n.*
- *Compare the family of pairwise independent $< p_*, b_* >$ of size $m = \frac{2n}{\epsilon^2}$*

The family of p's and b's are the entire possible set that can be generated from combinations of a log n set. The log n set (similar to below) can then be combined using Pairwise Independence (as shown above) to get the entire set of p's and b's i.e. all n of them.

$$\begin{array}{rl} p_1 : [\ 10111\ ...\ ] & b_1 \\ p_2 : [\ 01011\ ...\ ] & b_2 \\ \vdots & \vdots \\ p_{logn} : [\ 10001\ ...\ ] & b_{logn} \end{array}$$

With this base set all possible subsets are taken to form the entire family $< p_*, b_* >$. This family represents all possible combinations of p's and the b's they create.

Thus with Pairwise Independence *and* xor *it is possible to produce an n bit array using only* log n *bits.*

Two warmup proofs

To motivate the direction we will be heading for in the full proof, we first consider two scenarios in which the adversary on input $f_1(x)$ and p can guess $B(x,p)$ with probability much greater then a half.

In the following two warmup proofs, we use the following notation to (hopefully) clarify the presentation of the results. Given a string x, we use x^i to denote the string x with the ith bit flipped. We use array notation, $x[j]$, to denote the jth bit of x. Also, when referring to a string in the set of strings P, we use p_k to denote the kth string in the set.

The super-easy proof: Suppose the adversary A_B is able to guess the Hard-Core Bit $B(x,p)$ given $f_2(x,p)$ with probability 1. Then A_B can compute x bit-by-bit in the following manner. To compute $x[i]$, the ith bit of x, choose a random string p, and construct p^i. Since the adversary can compute hard-core bits with certainty, it can compute $b_1 = B(x,p)$ and $b_2 = B(x,p^i)$. By a simple case analysis, $x[i] = b_1 \oplus b_2$. After n iterations of this procedure (i.e. separately for each bit of x), we have the entire string x.

The somewhat easy proof: Now suppose that for every x, the adversary A_B is able to guess the Hard-Core Bit $B(x,p)$ given $f_2(x,p)$ with probability (over p) greater then $\frac{3}{4} + \epsilon(n)$: $Pr_p A_B(f_2(x,p)) = B(x,p) > \frac{3}{4} + \epsilon(n)$. Using the same procedure as in the super-easy proof, i.e. for each $x[i]$, we pick a random p and compute p^i, then guess $B(x,p)$ and $B(x,p^i)$ to help us determine $x[i]$. If we let E_1 and E_2 denote the events that the adversary's guesses for $B(x,p)$ and $B(x,p^i)$ are correct. We know that:

$$Pr_p[E_1 \equiv [A_B(f_2(x,p)) = B(x,p)]] > \frac{3}{4} + \epsilon(n)$$

and

$$Pr_p[E_2 \equiv [A_B(f_2(x,p^i)) = B(x,p^i)]] > \frac{3}{4} + \epsilon(n)$$

But these two events are *not* independent. Our guess for $x[i]$ is correct if both E_1 and E_2 occur (we also happen to get lucky if neither E_1 nor E_2 occur, but we ignore this case). We know that $Pr_p[\neg E_1] = \frac{1}{4} + \epsilon(n)$ and $Pr_p[\neg E_2] = \frac{1}{4} + \epsilon(n)$. Hence, by using union bound:

$$Pr_p[E_1 \wedge E_2] = 1 - Pr_p[\neg E_1 \vee \neg E_2] \geq 1 - \left[\left(\frac{1}{4} - \epsilon(n)\right) + \left(\frac{1}{4} - \epsilon(n)\right)\right] \geq \frac{1}{2} + 2\epsilon(n)$$

By employing tricks we have already seen, we can run the procedure for poly-many random p for each $x[i]$ and take the majority answer, which by a Chernoff bound amplifies the probability of success so that all bits of x can be guessed correctly with overwhelming probability.

1.6 One-way function have hard-core bits: The full proof

Now that we have obtained some insight as to how using a predictor of a Hard-Core Bit can help us to invert, we are ready to tackle the full proof. Therefore, we now assume that we are given an algorithm A_B which can compute the Hard-Core Bit with probability $> \frac{1}{2} + \epsilon(n)$ (over x, p and it coin-flips) and show an algorithm A_f(which uses A_B as a black-box) to invert f with noticeable probability.

The main idea of the proof is as follows: from the somewhat easy proof it is clear that we can not use our predictor twice on the same random string p. However, if for a random p we **guess correctly** an answer to $B(x,p) = b_1$, we can get the i'th bit of x with probability $\frac{1}{2} + \epsilon(n)$ by asking A_B to compute $B(x,p^i) = b_2$ *only once* for this (p, p^i) pair. So if we guess polynomialy many $B(x,p_j) = b_j$ correctly for different random p_j's we can do it. But we can only guess (with non-negligible probability) logarithmic number of totally independent bits. However, as we will see, we *can* guess (with non-negligible probability) a polynomial number of pairwise independent bits, and hence can do it. Now we go into the details.

Eliminating x from probabilities

In the somewhat easy proof, we assumed that the predictor A_B had $> \frac{3}{4} + e(n)$ chances for all x. But our A_B does not have such a guarantee. That is, we are only given that: We wish to say that for a sufficiently "large" fraction of x, we still have a $> \frac{1}{4} + e(n)$ guarantee (only over the choice of p and *coins*) and try to invert only on this fraction of x's. Thus, we begin by formalizing the notion of a *good* x and restrict our attention to adversaries which have reasonable chance of inverting $f_2(x,p)$ only on good x.

Definition 14 A string x is said to be good if $Pr_{p,\omega}[A_B(f(x),p) = B(x,p)] > \frac{1}{2} + \frac{\epsilon(n)}{2}$ where probability is taken over p and coin-flips ω of A_B.

Claim 15 At least an $\frac{\epsilon(n)}{2}$ fraction of x is good.

Proof Suppose not. Then,

$$\begin{aligned} Pr_{x,p,\omega}[A_B(f_2(x,p)) = B(x,p)] &= Pr_{x,p,\omega}[A_B(f_2(x,p)) = B(x,p) | x \text{ is good}] \cdot Pr_x[x \text{ is good}] \\ &\quad + Pr_{x,p,\omega}[A_B(f_2(x,p)) = B(x,p) | x \text{ not good}] \cdot Pr_x[x \text{ not good}] \\ &\leq 1 \cdot \frac{\epsilon(n)}{2} + \left(\frac{1}{2} + \frac{\epsilon(n)}{2}\right) \cdot 1 \\ &= \frac{1}{2} + \epsilon(n) \end{aligned}$$

This yields a contradiction, so the claim holds. ■

Overall strategy

Consider an adversary which attempts to invert $f(x)$ only on the set of good x, and succeeds with probability $> \frac{1}{2}$ on this set. Such an adversary succeeds in inverting $f(x)$ with total probability $\geq \frac{\epsilon(n)}{4}$, which is non-negligible, thereby ensuring that f is not a one-way function. This is exactly what we are going to do.

Our next question is for good x, with what probability does the adversary need to guess each bit $x[j]$ of x correctly in order to ensure that the entire x string is guessed correctly with probability $> \frac{1}{2}$. If the adversary computes each $x[j]$ correctly with probability $1 - \gamma$, then we can upper bound the probability that the adversary's guess for x is incorrect by employing the Union Bound:

$$Pr_\omega[A_{f_1}(f_1(x)) \text{ gets some bit of } x \text{ is wrong}] \leq \sum_{i=1}^{n} Pr_\omega[A_{f_1}(f_1(x)) \text{ gets } i\text{th bit wrong}] \leq n\gamma$$

where ω are coin-flips of A_f.

Setting $\gamma < \frac{1}{2n}$ guarantees that the probability that some bit of x is wrong is less than $\frac{1}{2}$, or equivalently, ensures that A_{f_1} guess is correct with probability $> \frac{1}{2}$. That is, if A_f can get each individual bit of x with probability grater then $(1 - \frac{1}{2n})$ then we can use the same procedure to get all bits of x with probability greater then $\frac{1}{2}$ even *if our method of getting different bits of x is not independent*!

Using pairwise independent p's

Our next goal is to devise a strategy for the adversary to guess each bit $x[j]$ with probability at least $1 - \frac{1}{2n}$. Again, we begin by making an assumption which seems difficult to achieve, prove the result given the far-fetched assumption and then show how to derive the assumption.

Lemma 16 Suppose we are given a collection of $m \equiv \frac{2n}{\epsilon(n)^2}$ *pairwise independent* $p_1, \ldots, p_m$, *where* $1 \leq i \leq m$, $|p_i| = n$ *and every* p_i *is uniformly distributed. Moreover, suppose that for every* i, *we are given a* b_i *satisfying* $x \cdot p_i = b_i$. *Then, for good* x, *we can compute* $x[j]$ *correctly with probability* $\geq 1 - \frac{1}{2n}$ *in polynomial*

Proof The adversary employs the following polytime algorithm.

1. For each $i \in 1, \ldots m$, construct p_i^j by flipping the jth bit of p_i.
2. Compute $b_i^j = x \cdot p_i^j$.
3. Derive a guess for $x[j]$ as in the "somewhat easy" proof: $g_i = b_i^j \oplus b_i$
4. Take the majority answer of all guesses g_i as the guess for $x[j]$.

We are interested in bounding the probability that the majority of our guesses were wrong, in which case our guess for $x[j]$ is also wrong. Define $Y_i = 1$ if g_i was incorrect and $Y_i = 0$ otherwise, and let $Y_m = \sum_{i=1}^{m} y_i$. Thus, we have i. Using Chebyshev:

$$\begin{aligned}
Pr\left[Y_m > \frac{m}{2}\right] &= Pr\left[Y_m - mp > \frac{m}{2} - mp\right] \\
&= Pr\left[Y_m - mp > \left(\frac{1}{2} - p\right)m\right] \\
&\leq Pr\left[|Y_m - mp| > \left(\frac{1}{2} - p\right)m\right] \\
\text{[Chebyshev]} \quad &\leq \frac{1}{4[(\frac{1}{2} - p)^2 m]} \\
&= \frac{1}{4\epsilon(n)^2 m}
\end{aligned}$$

Substituting in for $m = \frac{2n}{\epsilon(n)2}$ ensures that the probability that we misguess $x[j]$ (i.e., that $Pr[Y_m > \frac{m}{2}]$) is at most $\frac{1}{2n}$, proving the claim. ■

Lemma 17 If we are given uniformly distributed completely independent $p_1, \ldots, p_l$ for $l \equiv \lceil \log(\frac{2n}{\epsilon(n)^2}) + 1 \rceil$ together with $b_1, \ldots, b_l$ satisfying $B(x, p_i) = b_i$ then we can construct in polynomial time a pairwise independent uniformly distributed $p_1, \ldots, p_m$, $m \equiv \frac{2n}{\epsilon(n)^2}$ sample of correct equations of the form $B(x, p_i) = b_i$

Proof The proof hinges on the following fact, whose proof we omit because it is a simple case analysis:

Fact 18 Given correct equations $x \cdot p_1 = b_1$ and $x \cdot p_2 = b_2$, then $x \cdot (p_1 \oplus p_2) = (b_1 \oplus b_2)$.

It is easy to see by induction that this fact extends to the case in which there are arbitrarily many b_i and p_i. Therefore, we can generate a large set of new, valid equations by repeatedly

choosing an arbitrary subset of the p_is, XOR them together; XOR the corresponding b_is together to form a new equation of the form, for example, $x \cdot p_{1,3,5,7} = b_{1,3,5,7}$. Since there are $(2^l - 1)$ non-empty subsets of a set of size l, by choosing all possible subsets, the new set is of polynomial size $2^{\log l} = m$ and each new equation is polytime constructible. Furthermore, if we look at the symmetric difference of two different subsets, they are pairwise independent, so the entire set of new equations is pairwise independent. ■

Putting it all together

We now have all the machinery to provide a construction for inverting $f_1(x)$ with noticeable probability given a predictor A_B for predicting $B(x,p)$ with probability summarize it below:

Algorithm $A_{f_1}(y = f_1(x))$:

Step 1: Pick a set $P \equiv \{p_1, \ldots, p_l\}$ uniformly at random, where $|x| = |p_i| = n$ and $l \equiv \lceil \log(\frac{2n}{\epsilon(n)^2}) + 1 \rceil$

Step 2: Compute pairwise independent $\hat{P} \equiv \{\hat{p}_1, \ldots, \hat{p}_m\}$ where $m \equiv 2^l$ and $\hat{P}$ is computed by taking XOR of all possible non-empty subsets of P.

Step 3: For all possible 2^l bit-strings $b_1, \ldots, b_l$ of length l do:

Step 3.1: [Assume that for every $p_i \in P$, $(1 \leq i \leq l)$, $B(x, p_i) = b_i$] From P, and $b_1, \ldots, b_l$ compute for every $\hat{p}_k \in \hat{P}$ bit $\hat{b}_k$, where where $\hat{b}_k$ are computed by taking XOR of the corresponding (to $\hat{p}_k$) b_i's from step 3, and where $\hat{b}_k \equiv B(x, \hat{p}_k)$ for all $1 \leq k \leq m$.

Step 3.2: For j from 1 to n do: [compute all bits $x[j]$ of x]

Step 3.2.1: For every $\hat{p}_k \in \hat{P}$, where $1 \leq k \leq m$ ask A_B to predict $c_k \equiv B(x, \hat{p}_k^j)$. Let $b_{\hat{p}_k} \equiv c_k \bigoplus \hat{b}_k$

Step 3.2.2: define $x[j]$ as the majority of $\hat{b}_{\hat{p}_k}$ from step 3.2.1

Step 3.3: Check if for $z \equiv (x[1], \ldots, x[n])$ after step 3.2 $f_1(z) = y$. If so, **return** z, otherwise go back to step 3.

The adversary randomly selects a set of l strings of length n to form a set P. It then iterates through all possible completions $x \cdot p_i = b_i$, of which there are only $2^{\log l} = m$. For each incorrect completion, the adversary will perform a polynomial amount of useless work which we are not interested in; we focus on the work performed on the correct completion of the set of equations (which we can check in step 3.3). By Lemma 5.10, since the set of l

equations is totally independent, we can construct a pairwise independent set of m equations which are also correct. (step 3.1) Now from Lemma 5.9, this set of m equations suffices to invert $f(x)$ if x is good with probability $> \frac{1}{2}$. Early on, we noticed that the existence of a probabilistic poly-time A_{f_1} which succeeds in inverting $f_1(x)$ for good x with probability $> \frac{1}{2}$ proves that we can invert f with probability greater then $\frac{\epsilon(n)}{4}$, since good x occurs with probability greater then $\frac{\epsilon(n)}{2}$. But if we can invert f_1 with probability greater then $\frac{\epsilon(n)}{4}$, f_1 is not a strong one-way function. This completes the proof of the contrapositive, so we have shown that every one-way function has a Hard-Core Bit. ■

Remark: Instead of searching through all possible 2^l bit strings $b_1, \ldots, b_l$ in step 3, we can just pick *at random* $b_1, \ldots, b_l$ and try it only once. We guessed correctly with probability $\frac{1}{2^l} = \frac{1}{poly}$, hence we will still invert f_1 on good x with $\frac{1}{poly}$ probability.