Three XOR-Lemmas — An Exposition

Oded Goldreich
Department of Computer Science and Applied Mathematics
Weizmann Institute of Science, Rehovot, Israel

First version July 1991
revised November 27, 1995

Abstract

We provide an exposition of three Lemmas which relate general properties of distributions with the exclusive-or of certain bit locations.

The first XOR-Lemma, commonly attributed to U.V. Vazirani, relates the statistical distance of a distribution from uniform to the maximum bias of the xor of certain bit positions. The second XOR-Lemma, due to U.V. Vazirani and V.V. Vazirani, is a computational analogue of the first. It relates the pseudorandomness of a distribution with the difficulty of predicting the xor of bits in particular or random positions. The third Lemma, due to Goldreich and Levin, relates the difficulty of retrieving a string and the unpredictability of the xor of random bit positions. The most notable XOR Lemma – that is the so-called Yao XOR Lemma is not discussed here.

The proofs presented here differ from the proofs presented in the original works. Furthermore, these proofs are believed to be simpler, of wider applicability and yield somewhat better results. Credits for these improved proofs and their presentation are only partially due to author, and are mainly due to several other researchers.

Preface

The existence of ECCC motivated me to revise this five-year old survey and make it widely accesssible. The first two chapters are taken from my old survey (TR-681 of the C.S. Dept. of the Technion, Israel, 1991). For the third chapter, I've used a revision of parts from my book on "Foundations of Cryptography" (fragements of this book are available from ECCC).

As stated in the abstract, Yao's XOR-Lemma is not one of the XOR Lemmas surveyed here. I would like to call the reader's attention to a survey of Yao's XOR-Lemma which has appeared as ECCC TR95-050 (co-authored by Noam Nisan, Avi Wigderson and myself).

Contents

Chapter 1

The Information Theoretic XOR-Lemma

The Information Theoretic XOR-Lemma, commonly attributed to U.V. Vazirani relates two measures of the "randomness" of distributions over n-bit long strings.

- The statistical difference from uniform; namely, the statistical difference (variation difference) between the "target" distribution and the uniform distribution.
- The maximum bias of the xor of certain bit positions; namely, the bias of a 0-1 random variable obtained by taking the exclusive-or of certain bits in the "target" distribution.

It is well known that the statistical difference from uniform is bounded above by 2^n times the maximum bias of the xor's. Several researchers have noticed that the factor in the bound can be improved to $\sqrt{2^n}$. We provide a four line proof of this fact. We also explain the reason for the popularity of the worse bound.
The proof presented here has appeared as an appendix in [2].

1.1 Introduction

Let π be a an arbitrary probability distribution over $\{0,1\}^n$ and let μ denote the uniform distribution over $\{0,1\}^n$ (i.e., $\mu(x) = 2^{-n}$ for every $x \in \{0,1\}^n$). Let $x = x_1 \cdots x_n$ and $N \stackrel{\text{def}}{=} 2^n$. The XOR-Lemma relates two "measures of closeness" of π and μ.

- The statistical difference ("variation difference") between π and μ; namely,

$$\text{stat}(\pi) \stackrel{\text{def}}{=} \sum_x |\pi(x) - \mu(x)|$$

- The "maximum bias" of the exclusive-or of certain bit positions in strings chosen according to the distribution π; namely,

$$\text{maxbias}(\pi) \stackrel{\text{def}}{=} \max_{S \neq \emptyset} |\pi(\{x : \oplus_{i \in S} x_i = 0\}) - \pi(\{x : \oplus_{i \in S} x_i = 1\})|$$

The XOR-Lemma, commonly attributed to U.V. Vazirani [16][1], states that $\text{stat}(\pi) \leq N \cdot \text{maxbias}(\pi)$. The proof is based on viewing distributions as elements in an N-dimensional

[1] The special case where the maxbias is zero appears in Chor et. al. [5]

vector space and observing that the two measures considered by the lemma are merely two norms taken with respect to two different orthogonal bases. Hence, the XOR-Lemma follows from a (more general and quite straightforward) technical lemma which relates norms taken with respect to different orthonormal bases. It turns out that $\mathrm{stat}(\pi) \leq \sqrt{N} \cdot \mathrm{maxbias}(\pi)$. It seems that the previously inferior bound was due to a less careful way of using the same underlying ideas.

As motivation to the XOR-Lemma, we point out that it has been used in numerous works (e.g., Vazirani [16], Naor and Naor [12]). In a typical application, first a upper bound is proved on the maxbias of the constructed distribution and then the XOR-Lemma is applied to derive a bound on the statistical difference from the uniform distribution.

1.2 Preliminaries: the XOR-Lemma and vector spaces

Probability distributions over $\{0,1\}^n$ are functions from $\{0,1\}^n$ to the reals. Such functions form a N-dimensional vector space. The standard basis, denoted K, for this space is the orthonormal basis defined by the "Kroniker functions" (i.e., the Boolean functions $\{k_\alpha : \alpha \in \{0,1\}^n\}$ where $k_\alpha(x) = 1$ if $x = \alpha$). The statistical difference between two distributions equals the norm L-1 of their difference taken in the above K basis. On the other hand, the maxbias of a distribution equals the maximum "Fourier coefficient" of the distribution, which in turn corresponds to the max-norm (norm L-∞) of the distribution taken in a different basis. The basis is defined by the functions $\{b_S : S \subset \{1,2,...,n\}\}$ where $b_S(x) = (-1)^{\Sigma_{i \in S} x_i}$. Note that $b_S(x) = 1$ if the exclusive-or of the bits $\{x_i : i \in S\}$ is 0 and $b_S(x) = -1$ otherwise. The new basis is orthogonal but not orthonormal. We hence consider the normalized basis, denoted F, consisting of the functions $f_S = \frac{1}{\sqrt{N}} b_S$.

Notation: Let B be an orthonormal basis and r an integer. We denote by $\mathbf{N}_r^B(v)$ the norm L-r of v with respect to the basis B. Namely, $\mathbf{N}_r^B(v) = (\sum_{e \in B} \langle e, v \rangle^r)^{(1/r)}$, where $\langle e, v \rangle$ is the absolute value of the inner product of the vectors e and v. We denote by $\mathbf{N}_\infty^B(v)$ the limit of $\mathbf{N}_r^B(v)$ when $r \to \infty$ (i.e., $\mathbf{N}_\infty^B(v)$ is $\max_{e \in B} \langle e, v \rangle$).

Clearly, $\mathrm{stat}(\pi) = \mathbf{N}_1^K(\pi - \mu)$ whereas $\mathrm{maxbias}(\pi) = \sqrt{N} \cdot \mathbf{N}_\infty^F(\pi - \mu)$. Following is a proof of the second equality. Let $\delta(x) = \pi(x) - \mu(x)$. Clearly, $\mathrm{maxbias}(\mu) = 0$ and hence $\mathrm{maxbias}(\pi) = \mathrm{maxbias}(\delta)$. Also $\sum_x \delta(x) = 0$. We get

$$\begin{aligned}
\mathrm{maxbias}(\delta) &= \max_{S \neq \emptyset} |\delta(\{x : b_S(x) = 1\}) - \delta(\{x : b_S(x) = -1\})| \\
&= \max_{S \neq \emptyset} |\sum_x b_S(x) \cdot \delta(x)| \\
&= \sqrt{N} \cdot \max_S |\sum_x f_S(x) \cdot \delta(x)| \\
&= \sqrt{N} \cdot \mathbf{N}_\infty^F(\delta)
\end{aligned}$$

1.3 Proof of the XOR-Lemma

The XOR-Lemma follows from the following technical lemma

Technical Lemma: *For every two orthogonal bases A and B and every vector v*

$$\mathbf{N}_1^A(v) \leq N \cdot \mathbf{N}_\infty^B(v)$$

This technical lemma has a three line proof

EQ (1): For every orthogonal basis A,

$$\mathbf{N}_1^A(v) \leq \sqrt{N} \cdot \mathbf{N}_2^A(v)$$

EQ (2): For every pair of orthonormal bases A and B,

$$\mathbf{N}_2^A(v) = \mathbf{N}_2^B(v)$$

EQ (3): For every orthogonal basis B,

$$\mathbf{N}_2^B(v) \leq \sqrt{N} \cdot \mathbf{N}_\infty^B(v)$$

Hence we get,

XOR-Lemma (Revised): $\mathrm{stat}(\pi) \leq \sqrt{N} \cdot \mathrm{maxbias}(\pi)$.

Proof: By the above

$$\mathrm{stat}(\pi) = \mathbf{N}_1^K(\pi - \mu) \leq N \cdot \mathbf{N}_\infty^F(\pi - \mu) = \sqrt{N} \cdot \mathrm{maxbias}(\pi)$$

■

1.4 Discussion

The inferior bound, $\mathrm{stat}(\pi) \leq N \cdot \mathrm{maxbias}(\pi)$, has been derived by using one of the following two bounds instead of our Technical Lemma

- $\mathbf{N}_1^A(v) \leq \sqrt{N}\mathbf{N}_1^B(v) \leq \sqrt{N} \cdot N\mathbf{N}_\infty^B(v)$. The first inequality is proved similarly to the proof of our Technical Lemma (using $\mathbf{N}_2^B(v) \leq \mathbf{N}_1^B(v)$ instead of EQ (3)). The second inequality is trivial. Each of the two inequalities is tight, but their conjunction is wasteful.
- $\mathbf{N}_1^A(v) \leq N \cdot \mathbf{N}_\infty^A(v) \leq N \cdot \sqrt{N}\mathbf{N}_\infty^B(v)$. The second inequality is proved similarly to the proof of our Technical Lemma (using $\mathbf{N}_\infty^A(v) \leq \mathbf{N}_2^A(v)$ instead of EQ (1)). The first inequality is trivial. Again, each of the inequalities is tight, but their conjunction is wasteful.

Chapter 2

The Computational XOR-Lemma

We provide an exposition of the computational XOR-Lemma. By computational XOR-Lemma we refer to the assertion that a distribution on "short" strings is pseudorandom if and only if the xor of any of its bits is unpredictable. This Lemma was first proved by U.V. Vazirani and V.V. Vazirani. The proof we present here is taken from the paper of Goldreich and Levin. We demonstrate the applicability of the computational XOR-Lemma by using it to construct pseudorandom generators with linear expansion factor which are secure against small (yet linear) bounded space.

2.1 Introduction

This chapter is concerned the relation between two types of computationally restricted tests of randomness. To be more precise, we are concerned with the pseudorandomness of a random variable Y given some partial information represented by an related random variable X. For sake of simplicity we write $X = f(R)$ and $Y = g(R)$ where f and g are fixed functions and R is a random variable uniformly distributed on strings of some length.

Tests of the first type are algorithms which, on input a pair (x, y), output a single bit. We consider the probability that the test outputs 1 given that $x = f(r)$ and $y = g(r)$ where r is selected uniformly and compare it to the probability that the test outputs 1 given that $x = f(r)$ as before and y is selected (independently and) uniformly among the strings of length $|g(r)|$. We call the absolute value of the difference between these two probabilities, the *distinguishing gap* of the test.

Tests of the second type are algorithms which, on input a string $f(r)$, output a single bit. The output is supposed to be the inner-product (mod 2) of the string $g(r)$ with some fixed string β (which is not all-zero). We consider the probability that the algorithm outputs the correct value given that r is selected uniformly. We call the absolute value of the difference between the success probability and the failure probability, the *advantage* of the algorithm. Note that the inner-product (mod 2) of $g(r)$ and β equals the exclusive-or of the bits in $g(r)$ which are located in positions corresponding to the 1 bits of β. Hence, tests of the second type try to predict the xor of bits in $g(r)$ which are in specified bit locations.

Vazirani and Vazirani [18] proved that if the tests are restricted to run in probabilistic polynomial-time and the length of $g(r)$ is logarithmic in the length of $f(r)$ then the two types of tests are equivalent in the following sense: (for every polynomial-time computable functions f and g) there exists a test of the first type with a non-negligible distinguishing gap if and only if there exists a test of the second type with a non-negligible advantage[1]. A different proof has appeared in Goldreich and Levin [8]. The interesting direction is, of course, the assertion that if there exists a test of the first type with a non-negligible distinguishing gap then there exists a test of the second type with a non-negligible advantage[2]. This assertion is hereafter referred to as the *computational xor-lemma.*

The purpose of this chapter is to present a clear proof of the computational xor-lemma and to point out its applicability to other resource bounded machines. Our presentation follows the proof presented in [8], where all obvious details are omitted. Hence, the only advantage of our presentation is in its redundancy.

2.1.1 The Computational XOR-Proposition

To prove the computational xor-lemma, we present a particular algorithm, denoted G, which (given $f(r)$) tries to predict a specified xor of the bits of $g(r)$. The predictor G uses as subroutine a test, T, which (on input $f(r)$ and y) distinguishes a random y from $y = g(r)$. In particular, the predictor, on input x and a subset S, selects y at random, runs the test on inputs x and y, and output $\oplus_{i \in S} y_i$ if $T(x,y) = 1$ and the complement bit otherwise. The following proposition, lower bounds the advantage of the predictor G in terms of the distinguishing gap of the test T.

Computational XOR-Proposition: *Let f and g be arbitrary functions each mapping strings of the same length to strings of the same length. Let T be an algorithm (of the first type). Denote*

$$p \stackrel{\mathrm{def}}{=} \Pr[T(f(r), g(r)) = 1]$$

and

$$q \stackrel{\mathrm{def}}{=} \Pr[T(f(r), y) = 1]$$

where the probability is taken over all possible choices of $r \in \{0,1\}^m$ and $y \in \{0,1\}^{|g(r)|}$ with uniform probability distribution. Let G be an algorithm that on input β and x, selects y uniformly in $\{0,1\}^{|\beta|}$, and outputs $T(x,y) \oplus 1 \oplus (y, \beta)_2$, where $(y,\beta)_2$ is the inner product modulo 2 of y and β. Then,

$$\Pr[G(\beta, f(r)) = (g(r), \beta)_2] \;=\; \frac{1}{2} + \frac{p-q}{2^{|\beta|} - 1}$$

where the probability is taken over all possible choices of $r \in \{0,1\}^m$ and $\beta \in \{0,1\}^{|g(r)|} - 0^{|g(r)|}$ with uniform probability distribution.

A full proof of the proposition is presented in Section 2.

Remarks

- Algorithm G has almost the same complexities as T, with the exception that G must toss few more coins (to select β). Hence, G is randomized even in case T is deterministic.

[1] A function $\mu : \mathbf{N} \mapsto \mathbf{R}$ is non-negligible if there exists a polynomial p such that for all sufficiently large n we have $\mu(n) > 1/p(n)$.

[2] The opposite direction follows by noting that a test of a second type can be easily converted into a test of the first type: just run the predicting algorithm and compare its outcome with the actual xor of the corresponding bits.

- Clearly, there exists a non-zero string β for which $\Pr[G(\beta, f(r)) = (g(r), \beta)_2] = \frac{1}{2} + \frac{p-q}{2^{|\beta|-1}}$, where the probability is taken over all possible choices of $r \in \{0,1\}^m$ with uniform probability distribution. A string β with approximately such a performance can be found by sampling a string β and evaluating the performance of algorithm G with β as its first input. This requires ability to compute the functions f and g on many randomly selected instances (and collect the statistics). One should verify that this added complexity can be afforded. On the other hand, one should note that finding an appropriate β (i.e. on which G has almost the average advantage) may not be required (see remark below).

2.1.2 The Computational XOR-Lemma

As corollary to the Computational XOR-Proposition, we get

Computational XOR-Lemma:*Let $\mathcal{C}$ be a class of randomized (or non-uniform) algorithms, such that $\mathcal{C}$ is closed under sequential application of algorithms and contains an algorithm for computing $|g(r)|$ from $f(r)$. Suppose that every algorithm in the class $\mathcal{C}$, given $f(r)$, can predict the xor of a (given) random subset of the bits of $g(r)$ with (average) success probability bounded above by $\frac{1}{2} + \epsilon$. Then, for every algorithm, T, in the class $\mathcal{C}$*

$$|\Pr[T(f(r), g(r)) = 1] - \Pr[T(f(r), y) = 1]| \ < \ 2^{|g(r)|} \cdot \epsilon$$

where r is selected uniformly in $\{0,1\}^m$, the string y is selected uniformly and independently in $\{0,1\}^{|g(r)|}$.

Remarks

- As motivation to the Computational XOR-Lemma, we point out that it has been used in numerous works (e.g., Vazirani and Vazirani [18], Goldreich and Levin [8]). Another application of the Computational XOR-Lemma is presented in Section 3. In a typical application, the pseudorandomness of a short string is proven by showing that every xor of its bits is unpredictable (and using the Computational XOR-Lemma to argue that this suffices). As it follows that the xor of a (given) random non-empty subset of the bits is unpredictable, the Computational XOR-Lemma can be used directly without finding an appropriate β (as suggested by the previous remark).

- In case there are no computational restrictions on the tests, a stronger statement known as the XOR-Lemma can be proved: the statistical difference from uniform does not exceed $\sqrt{2^{|g(r)|}}$ times the maximum bias of a non-empty subset (see previous chapter).

2.2 Proof of the Computational XOR-Proposition

All that is required is to evaluate the success probability of algorithm G. In the following analysis we denote $\Pr_x[P(x,y)]$ the probability that $P(x,y)$ where x is taken according to a distribution to be understood from the context, and y is fixed. In case the predicate P depends on the test T, the probability will be taken also over the internal coin tosses of T. Hence, the coin tosses of T are implicit in the notation. The additional coin tosses of G, namely the string y, is explicit in the notation.

Hence, we rewrite

$$\begin{aligned} p &\stackrel{\rm def}{=} \mathrm{Pr}_r[T(f(r),g(r))=1] \\ q &\stackrel{\rm def}{=} \mathrm{Pr}_{r,y}[T(f(r),y)=1] \end{aligned}$$

Recall that r is taken uniformly from $\{0,1\}^m$, whereas y is taken uniformly from $\{0,1\}^{|g(r)|}$. In the following analysis β is selected uniformly from $B \stackrel{\rm def}{=} \{0,1\}^{|g(r)|} - 0^{|g(r)|}$. Our aim is to evaluate $\mathrm{Pr}_{r,\beta,y}[G(\beta,f(r))=(g(r),\beta)_2]$. We start by fixing an $r\in\{0,1\}^m$ and evaluating $\mathrm{Pr}_{\beta,y}[G(\beta,f(r))=(g(r),\beta)_2]$. We define $\equiv_\beta$ (resp., $\not\equiv_\beta$) so that $y\equiv_\beta z$ hold iff $(y,\beta)_2=(z,\beta)_2$ (resp., $y\not\equiv_\beta z$ iff $(y,\beta)_2\neq(z,\beta)_2$). We let $n\stackrel{\rm def}{=}|g(r)|$.
By the definition of G (i.e., $G(\beta,f(r)=T(x,y)\oplus 1\oplus(y,\beta)_2$, where $y\in\{0,1\}^{|\beta|}$ is uniformly selected by G) and elementary manipulations, we get

$$\begin{aligned} s_r &\stackrel{\rm def}{=} \mathrm{Pr}_{\beta,y}[G(\beta,f(r))=(g(r),\beta)_2] \\ &= \sum_{\beta\in B}\frac{1}{|B|}\cdot\mathrm{Pr}_y[G(\beta,f(r))=(g(r),\beta)_2] \\ &= \frac{1}{|B|}\cdot\sum_{\beta\in B}\left(\frac{1}{2}\cdot\mathrm{Pr}_y[T(f(r),y)=1|y\equiv_\beta g(r)]+\frac{1}{2}\cdot\mathrm{Pr}_y[T(f(r),y)=0|y\not\equiv_\beta g(r)]\right) \\ &= \frac{1}{2}+\frac{1}{2|B|}\sum_{\beta\in B}\left(\mathrm{Pr}_y[T(f(r),y)=1|y\equiv_\beta g(r)]-\mathrm{Pr}_y[T(f(r),y)=1|y\not\equiv_\beta g(r)]\right) \\ &= \frac{1}{2}+\frac{1}{2|B|}\cdot\frac{1}{2^{n-1}}\cdot\left(\sum_{\beta\in B}\sum_{y\equiv_\beta g(r)}\mathrm{Pr}[T(f(r),y)=1]-\sum_{\beta\in B}\sum_{y\not\equiv_\beta g(r)}\mathrm{Pr}[T(f(r),y)=1]\right) \\ &= \frac{1}{2}+\frac{1}{2^n\cdot|B|}\cdot\left(\sum_{y}\sum_{\beta\in B\ \mathrm{s.t.}\ y\equiv_\beta g(r)}\mathrm{Pr}[T(f(r),y)=1]-\sum_{y}\sum_{\beta\in B\ \mathrm{s.t.}\ y\not\equiv_\beta g(r)}\mathrm{Pr}[T(f(r),y)=1]\right) \end{aligned}$$

Recall $B=\{0,1\}^n-0^n$. If $y\neq g(r)$ then the number of $\beta\in B$ for which $y\not\equiv_\beta g(r)$ is 2^{n-1} (and the number of $\beta\in B$ for which $y\equiv_\beta g(r)$ is $2^{n-1}-1$). If, on the other hand, $y=g(r)$ then all $\beta\in B$ satisfy $y\equiv_\beta g(r)$. Hence, we get

$$\begin{aligned} s_r-\frac{1}{2} &= \frac{1}{2^n|B|}\cdot\sum_{y\neq g(r)}\left((2^{n-1}-1)\cdot\mathrm{Pr}[T(f(r),y)=1]-2^{n-1}\cdot\mathrm{Pr}[T(f(r),y)=1]\right) \\ &\quad+\frac{1}{2^n|B|}\cdot|B|\cdot\mathrm{Pr}[T(f(r),g(r))=1] \\ &= -\frac{1}{2^n|B|}\cdot\sum_{y\neq g(r)}\mathrm{Pr}[T(f(r),y)=1]\ \ +\ \frac{1}{2^n|B|}\cdot|B|\cdot\mathrm{Pr}[T(f(r),g(r))=1] \\ &= -\frac{1}{|B|}\sum_{y}\frac{1}{2^n}\cdot\mathrm{Pr}[T(f(r),y)=1]\ \ +\ \frac{1}{2^n|B|}\cdot(|B|+1)\cdot\mathrm{Pr}[T(f(r),g(r))=1] \\ &= -\frac{1}{|B|}\cdot\mathrm{Pr}_y[T(f(r),y)=1]+\frac{1}{|B|}\cdot\mathrm{Pr}[T(f(r),g(r))=1] \\ &= \frac{1}{|B|}\cdot\left(\mathrm{Pr}[T(f(r),g(r))=1]-\mathrm{Pr}_y[T(f(r),y)=1]\right) \end{aligned}$$

Hence, for every r

$$\Pr_{\beta,y}[G(\beta,f(r))=(g(r),\beta)_2] \;=\; \frac{1}{2}+\frac{\Pr[T(f(r),g(r))=1]-\Pr_y[T(f(r),y)=1]}{|B|}$$

and so we have for uniformly chosen r

$$\Pr_{r,\beta,y}[G(\beta,f(r))=(g(r),\beta)_2] \;=\; \frac{1}{2}+\frac{\Pr_r[T(f(r),g(r))=1]-\Pr_{r,y}[T(f(r),y)=1]}{|B|}$$

and the Proposition follows.

2.3 Application to pseudorandom generators for bounded space

We apply the Computational XOR-Lemma to construct a pseudorandom generator with linear stretching which withstands tests with linearly bounded space. Namely, the generator on input a random string of length n outputs a pseudorandom string of length cn withstanding tests of space en ($e>0$ is a constant depending on the constant $c>1$). An alternative construction is immediate from the techniques presented by Nisan in [13] (hint: use a constant number of hash functions). A third alternative construction was suggested by Noam Nisan (private communication) based on the ideas in [3].

The tests (or predictors) we consider are non-uniform bounded-space machines with one-way access to the input (i.e., the string they consider). Hence, these machines can be represented by finite automata. By an $s(n)$-space bounded machine we mean a finite automata with $2^{s(n)}$ states given an input of length n. For sake of simplicity, we sometimes discuss randomized automata. Clearly, randomness can be eliminated by introducing "more" non-uniformity.

Following is an overview of our construction. We begin by presenting a generator which extends seeds of length n into strings of length cn withstanding tests of space en, for a specific value of $c>1$ (and $e>0$). This generator is based on three observations:

- The unpredictability of the inner-product mod 2 of two vectors with respect to tests with space smaller than the length of these vectors.
- The unpredictability, with respect to such machines, of the exclusive-or of bits resulting from the inner-product mod 2 of one vector and non-cyclic shifts of a second vector. A machine predicting this exclusive-or can be transformed into a machine predicting the inner product [8].
- Using the computational XOR-Lemma to argue that the bits resulting from the various inner-products are indistinguishable from random by space bounded machines.

Next, we use this generator to construct, for every $k>1$, a generator extending seeds of length n into strings of length $c^k n$ withstanding tests of space $(e/3)^k n$.

2.3.1 A construction for a specific expansion constant

The constants $c_1,\epsilon_1,c_0,\epsilon_0$ in the following construction and analysis will be determined in course of the analysis. In particular, $c_0=\frac{1}{4}$, $\epsilon_0=\frac{1}{6}$, $c_1=1+\frac{c_0}{3}$, and $\epsilon_1=\frac{\epsilon_0}{3}$, will do.

Construction 1: Let $p_j(r_1 r_2 \cdots r_{2n}) \overset{\mathrm{def}}{=} r_j r_{j+1} \cdots r_{j+n-1}$ and $b(x,s) \overset{\mathrm{def}}{=} \sum_{i=1}^{n} x_i s_i \bmod 2$. Consider the function $g : \{0,1\}^{3n} \mapsto \{0,1\}^{c_0 n}$ defined by $g(x,r) = b(x, p_1(r)) \cdots b(x, p_{c_0 n}(r))$. We consider the generator

$$g_1(x,r) = (x, r, g(x,r))$$

This generator expands seeds of length $3n$ into strings of length $3n + c_0 n = c_1 \cdot 3n$. Clearly, the function g can be computed by an n-space machine. The robustness of the generator against $\epsilon_0 n$-space machines follows from the following three claims.

Claim 1: Let A be an automaton with q states, and x, y be uniformly and independently selected in $\{0,1\}^n$. Then

$$\mathrm{Pr}_{x,y}(A(x,y) = b(x,y)) < \frac{1}{2} + \sqrt{\frac{2q}{2^n}}$$

proof (adapted from [3]): By Lindsey Lemma (see [6] , p. 88),

$$|\sum_{x \in X, y \in Y} \frac{b(x,y)}{|X| \cdot |Y|} - \frac{1}{2}| \leq \sqrt{\frac{2^n}{|X| \cdot |Y|}}$$

Consider a partition of the set of all possible x's according to the state in which the automaton is after reading x (i.e. the first half of its input), resulting in sets $X_1, X_2, ..., X_q$. Note that for every $x_1, x_2 \in X_j$ and every y, we have $A(x_1, y) = A(x_2, y)$. For each X_i, let Y_i^σ denote the sets of y's for which $A(x,y) = \sigma$ given that $x \in X_i$. It follows that

$$|\mathrm{Pr}_{x,y}(A(x,y) = b(x,y)) - \frac{1}{2}| < \sum_{i=1}^{q} \sum_{\sigma \in \{0,1\}} \mathrm{Pr}_{x,y}(x \in X_i \wedge y \in Y_i^\sigma) \cdot \sqrt{\frac{2^n}{|X_i| \cdot |Y_i^\sigma|}}$$

The claim follows. □

Claim 2: Let $S \subseteq \{1, 2, ..., m\}$, where $m < n$. Suppose that automaton A_S has q states and let

$$p \overset{\mathrm{def}}{=} \mathrm{Pr}_{x,r}(A_S(x,r) = \oplus_{i \in S} b(x, p_i(r)))$$

where the probability is taken over all random choices of $x \in \{0,1\}^n$ and $r \in \{0,1\}^{2n}$. Then, there exists an automaton A with $q \cdot 2^{2m}$ states satisfying

$$\mathrm{Pr}_{x,y}(A(x,y) = b(x,y)) \geq p$$

where the probability is taken over all random choices of $x, y \in \{0,1\}^n$.

proof (adapted from [8]): Following is a construction of a randomized automaton A (randomization can be eliminated via non-uniformity). On input x, y, the predictor A produces a random string $r \in \{0,1\}^{2|y|}$ satisfying $y_i = \sum_{j \in S} r_{i+j-1} \bmod 2$, for every $i \leq n$. This is done by setting the bits of r in increasing order so that r_k is randomly selected if $k \leq t \overset{\mathrm{def}}{=} \max(S)$, r_k is set to $y_{k-t+1} - \sum_{j \in S - \{t\}} r_{k-t+j} \bmod 2$ for $k = t, t+1, ..., t+n-1$, and r_k is randomly selected for $k \geq t+n$. Hence, $\oplus_{j \in S} p_j(r) = y$, where $\oplus_{j \in S} v_j$ denotes the bit-by-bit exclusive or of the vectors v_j ($j \in S$). The predictor A runs $A_S(x,r)$ and obtains a prediction for $\oplus_{j \in S} b(x, p_j(r)) = b(x, \oplus_{j \in S} p_j(r)) = b(x,y)$. The predictor uses at most $2m$ more space than G_S, and the claim follows. □

Claim 3: For every automaton, T, with q states

$$|\Pr(T(x,r,g(x,r))=1)-\Pr(T(x,r,y)=1)| < 2^{|g(r)|}\cdot\sqrt{\frac{2q\cdot 2^{2c_0 n}}{2^n}}$$

where xr is selected uniformly in $\{0,1\}^{3n}$, the string y is selected uniformly in $\{0,1\}^{|g(x,r)|}$.
proof: Immediate by combining Claims 1 and 2, and the Computational XOR-Lemma. □

Setting $c_0 = \frac{1}{4}$ and $\epsilon_1 = \frac{1}{6}$, we conclude that any $\epsilon_1 n$-space bounded machine can distinguish $g_1(x,r)$ ($xr \in \{0,1\}^{3n}$) from a uniformly chosen string of length $(3+c_0)n$ with gap bounded by $2^{-\epsilon_1 n}$. Hence, for constants $c_1 = 1+\frac{1}{12}$ and $e_1 = \frac{1}{18}$, we have a generator extending strings of length n to strings of length $c_1 n$ so that no $\epsilon_1 n$-space bounded machine can distinguish $g_1(x,r)$ ($xr \in \{0,1\}^n$) from a uniformly chosen string of length $c_1 n$ with gap $> 2^{-\epsilon_1 n}$. We say that g_1 has *expansion factor* c_1 and *security constant* e_1.

2.3.2 Construction for any expansion constant

To achieve greater expansion we apply the generator again on small blocks of its output. This idea is taken from [7], but its usage in our context is restricted since in lower level the generator will be applied to shorter strings (and not to strings of the same length as done in [7]). The fact that in lower levels the generator is applied to shorter strings plays a key role in the proof that the resulting generator is indeed pseudorandom with respect to appropriate space-bounded machines.

In the sequel we show how to convert generators with expansion factor c into generators with expansion factor c^2. Larger expansion factors are obtained by repeated application of the construction.

Construction 2: Let g be a generator with expansion factor c and security constant e. We construct a generator g_2 with expansion factor c^2 and security constant $\frac{e^2}{3}$ as follows: $g_2(s) = g(r_1)\cdots g(r_t)$, where $r_1\cdots r_t = g(s)$, $|r_j| = \frac{e}{2}\cdot|s|$ (for all $1\le j\le t$), and $t = 2c/e$.

To prove that the generator g_2 has security $\frac{e^2}{3}$ we consider a *hybrid* distribution H which results by selecting at random a string of length cn, partitioning it into t blocks (each of length $\frac{e}{2}n$), and applying the generator g to each of them. First we show that H is hard to distinguish from random strings of length $c^2 n$. Next, we show that H is hard to distinguish from the strings that g_2 generates on input a random seed of length n.

Claim 4 (indistinguishability of H and random): Suppose that automaton T has q states and let $p_H \overset{\text{def}}{=} \Pr_{s_1\cdots s_t}(T(g(s_1)\cdots g(s_t))=1)$ and $p_R \overset{\text{def}}{=} \Pr_{r_1\cdots r_t}(T(r_1\cdots r_t)=1)$, where the probability is taken over all random choices of $s_1,...,s_t \in \{0,1\}^{\frac{e}{2}n}$ and $r_1,...,r_t \in \{0,1\}^{\frac{ce}{2}n}$. Then, there exists an automaton T' with q states satisfying

$$|\Pr_s(T'(g(s))=1)-\Pr_r(T'(r)=1)| \ge \frac{|p_H - p_R|}{t}$$

where the probability is taken over all random choices of $s \in \{0,1\}^{\frac{e}{2}n}$ and $r \in \{0,1\}^{\frac{ce}{2}n}$. Hence, if $q \le \frac{e^2}{2}n$ then $|p_R - p_H| < t\cdot 2^{-\frac{e^2}{2}n} < \frac{1}{2}2^{-\frac{e^2}{3}n}$.
proof: Define, for every $0\le i\le t$, $p_i \overset{\text{def}}{=} \Pr_{r_1\cdots r_i s_{i+1}\cdots s_t}(T(r_1\cdots r_i g(s_{i+1})\cdots g(s_t))=1)$, where the probability is taken over all random choices of $r_1,...,r_i \in \{0,1\}^{\frac{ce}{2}n}$ and $s_{i+1},...,s_t \in \{0,1\}^{\frac{e}{2}n}$. Namely, p_i is the probability that T outputs 1 on input taken from a hybrid distribution consisting of i "random" blocks and $t-i$ "pseudorandom" blocks. Clearly,

$p_0 = p_H$ whereas $p_t = p_R$, and there exists $0 \leq i \leq t-1$ such that $|p_i - p_{i+1}| \geq \frac{|p_0 - p_t|}{t}$. The test T' is obtained from T as follows. Fix a sequence $r_1, ..., r_i \in \{0,1\}^{\frac{ce}{2}n}$ and $s_{i+2}, ..., s_t \in \{0,1\}^{\frac{e}{2}n}$ maximizing the distinguishing gap between the i^{th} and $i+1^{\text{st}}$ hybrids. The starting state of test T' is the state to which T arrives on input $r_1, ..., r_i$. The accepting states (i.e. states with output 1) of test T' are the state from which T reaches its accepting state when reading the string $s_{i+2}, ..., s_t$. Clearly, T' has at most q states and distinguishes $r \in \{0,1\}^{\frac{ce}{2}n}$ from $g(s)$ (for $s \in \{0,1\}^{\frac{e}{2}n}$) with gap $\geq \frac{|p_H - p_R|}{t}$. Using the security hypothesis for g, the rest of the claim follows. □

Claim 5 (indistinguishability of H and the output of g_2): Suppose that automaton T has q states and let $p_G \stackrel{\text{def}}{=} \Pr_s(T(g_2(s)) = 1)$ and $p_H \stackrel{\text{def}}{=} \Pr_{r_1 \cdots r_t}(T(g(r_1) \cdots g(r_t)) = 1)$, where the probability is taken over all random choices of $s \in \{0,1\}^n$ and $r_1, ..., r_t \in \{0,1\}^{\frac{e}{2}n}$. Then, there exists an automaton T' with $q \cdot 2^{\frac{e}{2}n}$ states satisfying $|\Pr_s(T'(g(s)) = 1) - \Pr_r(T'(r) = 1)| \geq p_G - p_H$, where the probability is taken over all random choices of $s \in \{0,1\}^n$ and $r \in \{0,1\}^{cn}$. Hence, if $q \leq \frac{e}{2}n$ then $|p_G - p_H| < 2^{-en} < \frac{1}{2}2^{-\frac{e^2}{3}n}$.
proof: The test T' is obtained from T as follows. On input $\alpha \in \{0,1\}^{cn}$ (either random or pseudorandom), the test T' breaks α into t blocks, $\alpha_1, ..., \alpha_t$, each of length $\frac{e}{2}n$. Then T' computes $\beta = \beta_1 \cdots \beta_t$ so that $\beta_i = g(\alpha_i)$, and applies T to the string β. (T' accepts α iff T accepts β.) If α is taken from the uniform distribution, then the resulting β is distributed according to H. On the other hand, if α is taken as the output of g on random seed s, then $\beta = g_2(s)$. The test T' distinguishes the above cases with gap $\geq |p_H - p_G|$, and can be implemented using $q \cdot 2^{\frac{e}{2}n}$ states. Using the security hypothesis for g, the rest of the claim follows. □

Note that the test constructed in the proof of Claim 5, evaluates g on strings of length $\frac{e}{2}n$. Combining Claims 4 and 5, we conclude that the generator g_2 has security constant $\frac{e^2}{3}$.

Chapter 3

A Hard-Core Predicate for all One-Way Functions

A theorem of Goldreich and Levin relates two computational tasks. The first task is inverting a function f; namely given y find an x so that $f(x) = y$. The second task is predicting, with non-negligible advatage, the exclusive-or of a subset of the bits of x when only given $f(x)$. More precisely, it has been proved that if f cannot be efficiently inverted then given $f(x)$ and r it is infeasible to predict the inner-product mod 2 of x and r better than obvious.

We present an alternative proof to the original proof as appeared in [8]. The new proof, due to Charlie Rackoff, has two main advantages over the original one: it is simpler to explain and it provides better security (i.e., a more efficient reduction of inverting f to predicting the inner-product). The new proof was inspired by the proof in [1].

3.1 Introduction

The following text has been reproduced from [8].

> One-way functions are fundamental to many aspects of theory of computation. Loosely speaking, one-way are those functions which are easy to evaluate but hard to invert. However, many applications such as pseudorandom generators (see [Blum Micali 82, Yao 82]) and secure probabilistic encryption (see [Goldwasser Micali 82]) require that the function has a "hard-core" predicate b. This $b(x)$ should be easy to evaluate on input x, but hard to guess (with a noticeable correlation) when given only the value of $f(x)$. Intuitively, the hard-core predicate "concentrates" the one-wayness of the function in a strong sense.
> Clearly, only one-way functions may have hard-core predicates. A natural question of practical and theoretical importance is *whether every one-way function has one.* So far only partial answers have been given:
>
> 1. In [Blum Micali 82] it is proved that if the discrete exponentiation function is one-way then it has a hard-core predicate.[1] Analogous results for the RSA and Rabin functions (i.e. raising to a power and squaring modulo an integer, respectively) have been shown in [Alexi Chor Goldreich Schnorr 84].
> 2. In [Yao 82] it is proved that any one-way function f can be used to construct another one-way function f^* which has a hard-core predicate. The function f^* partitions its input into many shorter inputs and applies f to each of them in

[1] This result has been generalized to all Abelian groups in [Kaliski 88].

parallel (i.e. $f^*(x_1 \ldots x_{k^3}) = f(x_1) \ldots f(x_{k^3})$, $\|x_i\| = k$). (For a more refined analysis see [Levin 87].)

The drawback of the first set of results is that they are based on a particular intractability assumption (e.g. the hardness of the discrete logarithm problem). The second result constructs a predicate with security not bounded by a constant power of the security of f.
In this paper we resolve the above question by providing every one-way function with a hard-core predicate. More specifically, for any time limit s (e.g. $s(n) = n$, or $s(n) = 2^{\sqrt{n}}$), the following tasks are equivalent for probabilistic algorithms running in time $s(\|x\|)^{O(1)}$:

1. Given $f(x)$ find x for at least a fraction $s(\|x\|)^{-O(1)}$ of the x's.
2. Given $f(x)$ and p, $\|p\| = \|x\|$, guess the Boolean inner-product $B(x,p)$ of x and p with a correlation (i.e. the difference between the success and failure probabilities) of $s(\|x\|)^{-O(1)}$.

For any polynomial time computable f, b, there is always the smallest (within a polynomial) such s called the *security* of f and b, respectively. The security is a constructible function, can be computed by trying all small guessing algorithms, and is assumed to grow very fast (at least $n^{1/o(1)}$).

3.2 Definition

A *polynomial-time* function f is called one-way if any efficient algorithm can invert it only with negligible success probability. A *polynomial-time* predicate b is called a hard-core of a function f if all efficient algorithm, given $f(x)$, can guess $b(x)$ only with success probability which is negligibly better than half. To simplify our exposition, we associate efficiency with polynomial-time and negligible functions as such decreasing smaller than $1/\text{poly}(n)$. By U_n we denote a random variable uniformly distributed over $\{0,1\}^n$. For simplicity we consider only length preserving functions.

Definition 1 (one-way fiunction): *A* one-way *function, f, is a polynomial-time computable function such that for every probabilistic polynomial-time algorithm A', every polynomial $p(\cdot)$, and all sufficiently large n's*

$$\Pr\left(f(A'(Y_n)) = Y_n\right) < \frac{1}{2} + \frac{1}{p(n)}$$

where $Y_n = f(U_n)$.

Definition 2 (hard-core predicate): *A polynomial-time computable predicate $b : \{0,1\}^* \mapsto \{0,1\}$ is called a* hard-core *of a function f if for every probabilistic polynomial-time algorithm A', every polynomial $p(\cdot)$, and all sufficiently large n's*

$$\Pr\left(A'(f(U_n)) = b(U_n)\right) < \frac{1}{2} + \frac{1}{p(n)}$$

3.3 The main result and its proof

Theorem 3 *Let f be an arbitrary (strong) one-way function, and let g be defined by $g(x,r) \stackrel{\text{def}}{=} (f(x),r)$, where $|x| = |r|$. Let $b(x,r)$ denote the inner-product mod 2 of the binary vectors x and r. Then the predicate b is a hard-core of the function g.*

In other words, the theorem states that if f is strongly one-way then it is infeasible to guess the exclusive-or of a random subset of the bits of x when given $f(x)$ and the subset itself. We point out that g maintains properties of f such as being length-preserving and being one-to-one. Furthermore, an analogous statement holds for collections of one-way functions with/without trapdoor etc.

Proof: The proof uses a "reducibility argument". This time inverting the function f is reduced to predicting $b(x,r)$ from $(f(x),r)$. Hence, we assume (for contradiction) the existence of an efficient algorithm predicting the inner-product with advantage which is not negligible, and derive an algorithm that inverts f with related (i.e. not negligible) success probability. This contradicts the hypothesis that f is a one-way function.

Let G be a (probabilistic polynomial-time) algorithm that on input $f(x)$ and r tries to predict the inner-product (mod 2) of x and r. Denote by $\varepsilon_G(n)$ the (overall) advantage of algorithm G in predicting $b(x,r)$ from $f(x)$ and r, where x and r are uniformly chosen in $\{0,1\}^n$. Namely,

$$\varepsilon_G(n) \stackrel{\text{def}}{=} \Pr\left(G(f(X_n),R_n) = b(X_n,R_n)\right) - \frac{1}{2}$$

where here and in the sequel X_n and R_n denote two independent random variables, each uniformly distributed over $\{0,1\}^n$. Assuming, to the contradiction, that b is not a hard-core of g means that exists an efficient algorithm G, a polynomial $p(\cdot)$ and an infinite set N so that for every $n \in N$ it holds that $\varepsilon_G(n) > \frac{1}{p(n)}$. We restrict our attention to this algorithm G and to n's in this set N. In the sequel we shorthand ε_G by ε.

Our first observation is that, on at least an $\frac{\varepsilon(n)}{2}$ fraction of the x's of length n, algorithm G has an $\frac{\varepsilon(n)}{2}$ advantage in predicting $b(x,R_n)$ from $f(x)$ and R_n. Namely,

Claim 3.1: there exists a set $S_n \subseteq \{0,1\}^n$ of cardinality at least $\frac{\varepsilon(n)}{2} \cdot 2^n$ such that for every $x \in S_n$, it holds that

$$s(x) \stackrel{\text{def}}{=} \Pr(G(f(x),R_n) = b(x,R_n)) \geq \frac{1}{2} + \frac{\varepsilon(n)}{2}$$

This time the probability is taken over all possible values of R_n and all internal coin tosses of algorithm G, whereas x is fixed.

Proof: The observation follows by an averaging argument. Namely, write $\text{Exp}(s(X_n)) = \frac{1}{2} + \varepsilon(n)$, and apply Markov Inequality.$\Box$

In the sequel we restrict our attention to x's in S_n. We will show an efficient algorithm that on every input y, with $y = f(x)$ and $x \in S_n$, finds x with very high probability. Contradiction to the (strong) one-wayness of f will follow by noting that $\Pr(U_n \in S_n) \geq \frac{\varepsilon(n)}{2}$.

A motivating discussion

Consider a fixed $x \in S_n$. By definition $s(x) \geq \frac{1}{2} + \frac{\varepsilon(n)}{2} > \frac{1}{2} + \frac{1}{2p(n)}$. Suppose, for a moment, that $s(x) > \frac{3}{4} + \frac{1}{2p(n)}$. In this case (i.e., of $s(x) > \frac{3}{4} + \frac{1}{\text{poly}(|x|)}$) retrieving x from $f(x)$ is quite easy. To retrieve the i^{th} bit of x, denoted x_i, we randomly select $r \in \{0,1\}^n$, and compute $G(f(x),r)$ and $G(f(x),r\oplus e^i)$, where e^i is an n-dimensional binary vector with 1 in the i^{th} component and 0 in all the others, and $v\oplus u$ denotes the addition mod 2 of the binary vectors v and u. Clearly, if both $G(f(x),r)=b(x,r)$ and $G(f(x),r\oplus e^i)=b(x,r\oplus e^i)$, then

$$\begin{aligned} G(f(x),r)\oplus G(f(x),r\oplus e^i) &= b(x,r)\oplus b(x,r\oplus e^i) \\ &= b(x,e^i) \\ &= x_i \end{aligned}$$

since $b(x,r)\oplus b(x,s) \equiv \sum_{i=1}^n x_i r_i + \sum_{i=1}^n x_i s_i \equiv \sum_{i=1}^n x_i(r_i+s_i) \equiv b(x,r\oplus s) \bmod 2$. The probability that both equalities hold (i.e., both $G(f(x),r)=b(x,r)$ and $G(f(x),r\oplus e^i)=b(x,r\oplus e^i)$) is at least $1-2\cdot(\frac{1}{4}-\frac{1}{\mathrm{poly}(|x|)}) > 1-\frac{1}{\mathrm{poly}(|x|)}$. Hence, repeating the above procedure sufficiently many times and ruling by majority we retrieve x_i with very high probability. Similarly, we can retrieve all the bits of x, and hence invert f on $f(x)$. However, the entire analysis was conducted under (the unjustifiable) assumption that $s(x) > \frac{3}{4}+\frac{1}{2p(|x|)}$, whereas we only know that $s(x) > \frac{1}{2}+\frac{1}{2p(|x|)}$.

The problem with the above procedure is that it doubles the original error probability of algorithm G on inputs of form $(x,\cdot)$. Under the unrealistic assumption, that the G's error on such inputs is significantly smaller than $\frac{1}{4}$, the "error-doubling" phenomenon raises no problems. However, in general (and even in the special case where G's error is exactly $\frac{1}{4}$) the above procedure is unlikely to invert f. Note that the error probability of G can not be decreased by repeating G several times (e.g., G may always answer correctly on three quarters of the inputs, and always err on the remaining quarter). What is required is an *alternative way of using* the algorithm G, a way which does not double the original error probability of G. The key idea is to generate the r's in a way which requires applying algorithm G only once per each r (and x_i), instead of twice. The good news are that the error probability is no longer doubled. since we only need to use G to get an "estimate" of $b(x,r\oplus e^i)$. The bad news are that we still need to know $b(x,r)$, and it is not clear how we can know $b(x,r)$ without applying G. The answer is that we can guess $b(x,r)$ by ourselves. This is fine if we only need to guess $b(x,r)$ for one r (or logarithmically in $|x|$ many r's), but the problem is that we need to know (and hence guess) $b(x,r)$ for polynomially many r's. An obvious way of guessing these $b(x,r)$'s yields an exponentially vanishing success probability. The solution is to generate these polynomially many r's so that, on one hand they are "sufficiently random" whereas on the other hand we can guess all the $b(x,r)$'s with non-negligible success probability. Specifically, generating the r's in a particular *pairwise independent* manner will satisfy both (seemingly contradictory) requirements. We stress that in case we are successful (in our guesses for the $b(x,r)$'s), we can retrieve x with high probability. Hence, we retrieve x with non-negligible probability.

A word about the way in which the pairwise independent r's are generated (and the corresponding $b(x,r)$'s are guessed) is indeed in place. To generate $m=\mathrm{poly}(n)$ many r's, we uniformly (and independently) select $l \stackrel{\text{def}}{=} \log_2(m+1)$ strings in $\{0,1\}^n$. Let us denote these strings by $s^1,...,s^l$. We then guess $b(x,s^1)$ through $b(x,s^l)$. Let use denote these guesses, which are uniformly (and independently) chosen in $\{0,1\}$, by σ^1 through σ^l. Hence, the probability that all our guesses for the $b(x,s^i)$'s are correct is $2^{-l}=\frac{1}{\mathrm{poly}(n)}$. The different r's correspond to the different non-empty subsets of $\{1,2,...,l\}$. We compute $r^J \stackrel{\text{def}}{=} \oplus_{j\in J}s^j$. The reader can easily verify that the r^J's are pairwise independent and each is uniformly distributed in $\{0,1\}^n$. The key observation is that

$$b(x,r^J) = b(x,\oplus_{j\in J}s^j) = \oplus_{j\in J}b(x,s^j)$$

Hence, our guess for the $b(x, r^J)$'s is $\oplus_{j\in J}\sigma^j$, and with non-negligible probability all our guesses are correct.

Back to the formal argument

Following is a formal description of the inverting algorithm, denoted A. We assume, for simplicity that f is length preserving (yet this assumption is not essential). On input y (supposedly in the range of f), algorithm A sets $n \stackrel{\rm def}{=} |y|$, and $l \stackrel{\rm def}{=} \lceil \log_2(2n\cdot p(n)^2+1)\rceil$, where $p(\cdot)$ is the polynomial guaranteed above (i.e., $\epsilon(n) > \frac{1}{p(n)}$ for the infinitely many n's in N). Algorithm A uniformly and independently select $s^1, ..., s^l \in \{0,1\}^n$, and $\sigma^1, ..., \sigma^l \in \{0,1\}$. It then computes, for every non-empty set $J \subseteq \{1, 2, ..., l\}$, a string $r^J \leftarrow \oplus_{j\in J}s^j$ and a bit $\rho^J \leftarrow \oplus_{j\in J}\sigma^j$. For every $i \in \{1, ..., n\}$ and every *non-empty* $J \subseteq \{1, .., l\}$, algorithm A computes $z_i^J \leftarrow \rho^J \oplus G(y, r^J \oplus e^i)$. Finally, algorithm A sets z_i to be the majority of the z_i^J values, and outputs $z = z_1 \cdots z_n$. (Remark: in an alternative implementation of the ideas, the inverting algorithm, denoted A', tries all possible values for $\sigma^1, ..., \sigma^l$, and outputs only one of resulting strings z, with an obvious preference to a string z satisfying $f(z) = y$.)

Following is a detailed analysis of the success probability of algorithm A on inputs of the form $f(x)$, for $x \in S_n$, where $n \in N$. We start by showing that, in case the σ^j's are correct, then the with constant probability, $z_i = x_i$ for all $i \in \{1, ..., n\}$. This is proven by bounding from below the probability that the majority of the z_i^J's equals x_i.

Claim 3.2: For every $x \in S_n$ and every $1 \le i \le n$,

$$\Pr\left(|\{J : b(x, r^J) \oplus G(f(x), r^J \oplus e^i) = x_i\}| > \frac{1}{2}\cdot(2^l - 1)\right) > 1 - \frac{1}{2n}$$

where $r^J \stackrel{\rm def}{=} \oplus_{j\in J}s^j$ and the s^j's are independently and uniformly chosen in $\{0,1\}^n$.

Proof: For every J, define a 0-1 random variable ζ^J, so that ζ^J equals 1 if and only if $b(x, r^J) \oplus G(f(x), r^J \oplus e^i) = x_i$. The reader can easily verify that each r^J is uniformly distributed in $\{0,1\}^n$. It follows that each ζ^J equals 1 with probability $s(x)$, which by $x \in S_n$, is at least $\frac{1}{2} + \frac{1}{2p(n)}$. We show that the ζ^J's are pairwise independent by showing that the r^J's are pairwise independent. For every $J \neq K$ we have, without loss of generality, $j \in J$ and $k \in K - J$. Hence, for every $\alpha, \beta \in \{0,1\}^n$, we have

$$\begin{aligned} \Pr\left(r^K = \beta \mid r^J = \alpha\right) &= \Pr\left(s^k = \beta \mid s^j = \alpha\right) \\ &= \Pr\left(s^k = \beta\right) \\ &= \Pr\left(r^K = \beta\right) \end{aligned}$$

and pairwise independence of the r^J's follows. Let $m \stackrel{\rm def}{=} 2^l - 1$. Using Chebyshev's Inequality, we get

$$\begin{aligned} \Pr\left(\sum_J \zeta^J \le \frac{1}{2}\cdot m\right) &\le \Pr\left(|\sum_J \zeta^J - (\frac{1}{2} + \frac{1}{2p(n)})\cdot m| \ge \frac{1}{2p(n)}\cdot m\right) \\ &< \frac{\mathrm{Var}(\zeta^{\{1\}})}{(\frac{1}{2p(n)})^2\cdot(2n\cdot p(n)^2)} \\ &< \frac{\frac{1}{4}}{(\frac{1}{2p(n)})^2\cdot(2n\cdot p(n)^2)} \\ &= \frac{1}{2n} \end{aligned}$$

The claim now follows. □

Recall that if $\sigma^j = b(x, s^j)$, for all j's, then $\rho^J = b(x, r^J)$ for all non-empty J's. In this case z output by algorithm A equals x, with probability at least half. However, the first event happens with probability $2^{-l} = \frac{1}{2n \cdot p(n)^2}$ independently of the events analyzed in Claim 3.2. Hence, in case $x \in S_n$, algorithm A inverts f on $f(x)$ with probability at least $\frac{1}{4p(|x|)}$ (whereas, the modified algorithm, A', succeeds with probability $\geq \frac{1}{2}$). Recalling that $|S_n| > \frac{1}{2p(n)} \cdot 2^n$, we conclude that, for every $n \in N$, algorithm A inverts f on $f(U_n)$ with probability at least $\frac{1}{8p(n)^2}$. Noting that A is polynomial-time (i.e., it merely invokes G for $2n \cdot p(n)^2 = \text{poly}(n)$ times in addition to making a polynomial amount of other computations), a contradiction, to our hypothesis that f is strongly one-way, follows. ■

Improving the Efficiency of the Inverting Algorithm

In continuation to the proof of Theorem 3, we present guidelines for a more efficient inverting algorithm. In the sequel it will be more convenient to use arithmetic of reals instead of that of Boolean. Hence, we denote $b'(x, r) = (-1)^{b(r,x)}$ and $G'(y, r) = (-1)^{G(y,r)}$.

1. Prove that for every x it holds that $\text{Exp}(b'(x, r) \cdot G'(f(x), r + e^i)) = s'(x) \cdot (-1)^{x_i}$, where $s'(x) \stackrel{\text{def}}{=} 2 \cdot (s(x) - \frac{1}{2})$.

2. Let v be an l-dimensional Boolean vector, and let R be a uniformly chosen l-by-n Boolean matrix. Prove that for every $v \neq u \in \{0,1\}^l$ it holds that vR and uR are pairwise independent and uniformly distributed in $\{0,1\}^n$.

3. Prove that $b'(x, vR) = b'(xR^T, v)$, for every $x \in \{0,1\}^n$ and $v \in \{0,1\}^l$.

4. Prove that, with probability at least $\frac{1}{2}$, there exists $\sigma \in \{0,1\}^l$ so that for every $1 \leq i \leq n$ the sign of $\sum_{v \in \{0,1\}^l} b'(\sigma, v) G'(f(x), vR + e^i))$ equals the sign of $(-1)^{x_i}$. (Hint: $\sigma \stackrel{\text{def}}{=} xR^T$.)

5. Let B be an 2^l-by-2^l matrix with the (σ, v)-entry being $b'(\sigma, v)$, and let $\overline{g}^i$ be an 2^l-dimensional vector with the v^{th} entry equal $G'(f(x), vR + e^i)$. The inverting algorithm computes $\overline{z}_i \leftarrow Bg^i$, for all i's, and forms a matrix Z in which the columns are the $\overline{z}_i$'s. The output is a row that when applying f to it yields $f(x)$. Evaluate the success probability of the algorithm. Using the special structure of matrix B, show that the product Bg^i can be computed in time $l \cdot 2^l$.
   Hint: B is the Sylvester matrix, which can be written recursively as

$$S_k = \begin{pmatrix} S_{k-1} & S_{k-1} \\ S_{k-1} & \overline{S_{k-1}} \end{pmatrix}$$

   where $S_0 = +1$ and $\overline{M}$ means flipping the $+1$ entries of M to -1 and vice versa.

3.4 Hard-Core Functions

We have just seen that every one-way function can be easily modified to have a hard-core predicate. In other words, the result establishes one bit of information about the preimage which is hard to approximate from the value of the function. A stronger result may say that several bits of information about the preimage are hard to approximate. For example, we may want to say that a specific pair of bits is hard to approximate, in the sense that it is infeasible to guess this pair with probability significantly larger than $\frac{1}{4}$. In general, a *polynomial-time* function, h, is called a hard-core of a function f if no efficient algorithm can distinguish $(f(x), h(x))$ from $(f(x), r)$, where r is a random string of length $|h(x)|$. We assume for simplicity that h is length regular (see below).

Definition 4 (hard-core function): *Let $h : \{0,1\}^* \mapsto \{0,1\}^*$ be a polynomial-time computable function, satisfying $|h(x)| = |h(y)|$ for all $|x| = |y|$, and let $l(n) \stackrel{\rm def}{=} |h(1^n)|$. The function $h : \{0,1\}^* \mapsto \{0,1\}^*$ is called a* hard-core *of a function f if for every probabilistic polynomial-time algorithm D', every polynomial $p(\cdot)$, and all sufficiently large n's*

$$|\Pr(D'(f(X_n), h(X_n))=1) - \Pr(D'(f(X_n), R_{l(n)})=1)| < \frac{1}{p(n)}$$

where X_n and $R_{l(n)}$ are two independent random variables the first uniformly distributed over $\{0,1\}^n$, and the second uniformly distributed over $\{0,1\}^{l(n)}$,

Theorem 5 *Let f be an arbitrary strong one-way function, and let g_2 be defined by $g_2(x,s) \stackrel{\rm def}{=} (f(x), s)$, where $|s|=2|x|$. Let $c > 0$ be a constant, and $l(n) \stackrel{\rm def}{=} \lceil c \log_2 n \rceil$. Let $b_i(x,s)$ denote the inner-product mod 2 of the binary vectors x and $(s_{i+1}, ..., s_{i+n})$, where $s = (s_1, ..., s_{2n})$. Then the function $h(x,s) \stackrel{\rm def}{=} b_1(x,s) \cdots b_{l(|x|)}(x,s)$ is a hard-core of the function g_2.*

The proof of the theorem follows by combining a proposition concerning the structure of the specific function h with a general lemma concerning hard-core functions. Loosely speaking, the proposition "reduces" the problem of approximating $b(x,r)$ given $g(x,r)$ to the problem of approximating the exclusive-or of any non-empty set of the bits of $h(x,s)$ given $g_2(x,s)$, where b and g are the hard-core and the one-way function presented in the previous section. Since we know that the predicate $b(x,r)$ cannot be approximated from $g(x,r)$, we conclude that no exclusive-or of the bits of $h(x,s)$ can be approximated from $g_2(x,s)$. The general lemma states that, for every "logarithmically shrinking" function h' (i.e., h' satisfying $|h'(x)| = O(\log|x|)$), the function h' is a hard-core of a function f' if and only if the exclusive-or of any non-empty subset of the bits of h' cannot be approximated from the value of f'.

Proposition 6 *Let f, g_2 and b_i's be as above. Let $I(n) \subseteq \{1, 2, ..., l(n)\}$, $n \in \mathbf{N}$, be an arbitrary sequence of non-empty subsets, and let $b_{I(|x|)}(x,s) \stackrel{\rm def}{=} \oplus_{i \in I(|x|)} b_i(x,s)$. Then, for every probabilistic polynomial-time algorithm A', every polynomial $p(\cdot)$, and all sufficiently large n's*

$$\Pr(A'(g_2(U_{3n})) = b_{I(n)}(U_{3n})) < \frac{1}{2} + \frac{1}{p(n)}$$

Proof: The proof is by a "reducibility" argument. It is shown that the problem of approximating $b(X_n, R_n)$ given $(f(X_n), R_n)$ is reducible to the problem of approximating $b_{I(n)}(X_n, S_{2n})$ given $(f(X_n), S_{2n})$, where X_n, R_n and S_{2n} are independent random variable and the last is uniformly distributed over $\{0,1\}^{2n}$. The underlying observation is that, for every $|s| = 2 \cdot |x|$,

$$b_I(x,s) = \oplus_{i \in I} b_i(x,s) = b(x, \oplus_{i \in I} \mathrm{sub}_i(s)$$

where $\mathrm{sub}_i(s_1, ..., s_{2n}) \stackrel{\rm def}{=} (s_{i+1}, ..., s_{i+n})$. Furthermore, the reader can verify that for every non-empty $I \subseteq \{1, ..., n\}$, the random variable $\oplus_{i \in I} \mathrm{sub}_i(S_{2n})$ is uniformly distributed over $\{0,1\}^n$, and that given a string $r \in \{0,1\}^n$ and such a set I one can efficiently select a string uniformly in the set $\{s : \oplus_{i \in I} \mathrm{sub}_i(s) = r\}$. (Verification of both claims is left as an exercise.)

Now, assume to the contradiction, that there exists an efficient algorithm A', a polynomial $p(\cdot)$, and an infinite sequence of sets (i.e., $I(n)$'s) and n's so that

$$\Pr\left(A'(g_2(U_{3n})) = b_{I(n)}(U_{3n})\right) \geq \frac{1}{2} + \frac{1}{p(n)}$$

We first observe that for n's satisfying the above inequality we can find in probabilistic polynomial time (in n) a set I satisfying

$$\Pr\left(A'(g_2(U_{3n})) = b_I(U_{3n})\right) \geq \frac{1}{2} + \frac{1}{2p(n)}$$

(i.e., by going over all possible I's and experimenting with algorithm A' on each of them). Of course we may be wrong here, but the error probability can be made exponentially small.

We now present an algorithm for approximating $b(x,r)$, from $y \stackrel{\rm def}{=} f(x)$ and r. On input y and r, the algorithm first finds a set I as described above (this stage depends only on $|x|$ which equals $|r|$). Once I is found, the algorithm uniformly select a string s so that $\oplus_{i \in I} \mathrm{sub}_i(s) = r$, and return $A'(y, s)$. Evaluation of the success probability of this algorithm is left as an exercise. ■

Lemma 7 (Computational XOR Lemma): *Let f and h be arbitrary length regular functions, and let $l(n) \stackrel{\rm def}{=} |h(1^n)|$. Let D be an algorithm. Denote*

$$p \stackrel{\rm def}{=} \Pr\left(D(f(X_n), h(X_n)) = 1\right) \quad and \quad q \stackrel{\rm def}{=} \Pr\left(D(f(X_n), R_{l(n)}) = 1\right)$$

where X_n and R_l are as above. Let G be an algorithm that on input y, S (and $l(n)$), selects r uniformly in $\{0,1\}^{l(n)}$, and outputs $D(y,r) \oplus 1 \oplus (\oplus_{i \in S} r_i)$, where $r = r_1 \cdots r_l$ and $r_i \in \{0,1\}$. Then,

$$\Pr\left(G(f(X_n), I_l, l(n)) = \oplus_{i \in I_l}(h_i(X_n))\right) = \frac{1}{2} + \frac{p-q}{2^{l(n)} - 1}$$

where I_l is a randomly chosen non-empty subset of $\{1, ..., l(n)\}$ and $h_i(x)$ denotes the $i^{\rm th}$ bit of $h(x)$.

Proof: see previous chapter. ■

It follows that, for logarithmically shrinking h's, the existence of an efficient algorithm that distinguishes (with a gap which is not negligible in n) the random variables $(f(X_n), h(X_n))$ and $(f(X_n), R_{l(n)})$ implies the existence of an efficient algorithm that approximates the exclusive-or of a random non-empty subset of the bits of $h(X_n)$ from the value of $f(X_n)$ with an advantage that is not negligible.

Bibliography

[1] W. Alexi, B. Chor, O. Goldreich and C.P. Schnorr, "RSA and Rabin Functions: Certain Parts Are As Hard As the Whole", *FOCS* (1984) and *SIAM Journ. on Computing*, Vol. 17, 1988, pp. 194-209.

[2] N. Alon, O. Goldreich, J. Hastad and R. Peralta, "Simple Construction of Almost k-wise Indepedent Random Variables", *Random Structures and Algorithms*, Vol. 3, No. 3, 1992.

[3] Babai, L., N. Nisan, and M. Szegedy, "Multiparty protocols and logspace-hard pseudorandom sequences", *21st STOC*, 1989, pp. 1–11.

[4] Blum, M., and Micali, S., "How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits", *FOCS* (1982); *SIAM Journ. on Computing*, Vol. 13, 1984, pp. 850-864.

[5] B. Chor, J. Friedmann, O. Goldreich, J. Hastad, S. Rudich and R. Smolansky, "The Bit Extraction Problem or t-Resilient Functions", *Proc. of the 26th IEEE Symp. on Foundation Of Computer Science (FOCS)*, 1985, pp. 396-407.

[6] Erdos, P., and J. Spenser, *Probabilistic Methods in Combinatorics*, Academic Press, New York, 1974.

[7] Goldreich, O., S. Goldwasser, and S. Micali, "How to Construct Random Functions", *Journ. of ACM*, Vol. 33, No. 4, 1986, pp. 792-807.

[8] Goldreich O., and L.A. Levin, "Hard-core Predicates for any One-Way Function", *21th STOC*, pp. 25–32, 1989.

[9] Goldwasser, S., and S. Micali, "Probabilistic Encryption", *STOC* (1982); *JCSS*, Vol. 28, No. 2, 1984, pp. 270-299.

[10] B.S. Kaliski, Jr., "Elliptic Curves and Cryptography: A Pseudorandom Bit Generator and Other Tools", Ph.D. Thesis, LCS, MIT, 1988.

[11] L.A. Levin, "One-Way Function and Pseudorandom Generators", *Combinatorica*, Vol. 7, No. 4, 1987, pp. 357-363. A preliminary version in *STOC-85*.

[12] J. Naor and M. Naor, "Small-bias Probability Spaces: Efficient Constructions and Applications", *22nd STOC*, 1990, pp. 213–223.

[13] N. Nisan, "Pseudorandom Generators for Space-Bounded Computations", *22nd STOC*, 1990, pp. 204–212.

[14] M.O. Rabin, "Digitalized Signatures and Public Key Functions as Intractable as Factoring", MIT/LCS/TR-212, 1979.

[15] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", *Comm. ACM*, Vol. 21, Feb. 1978, pp 120-126

[16] U.V. Vazirani, "Randomness, Adversaries and Computation", Ph.D. Thesis, EECS, UC Berkeley, 1986.

[17] U.V. Vazirani, "Efficiency Considerations in Using Semi-random Sources", *Proc. 19th ACM Symp. on Theory of Computing*, 1987, pp. 160-168.

[18] U.V. Vazirani, and V.V. Vazirani, "Efficient and Secure Pseudo-Random Number Generation", *Proc. 25th IEEE Symp. on Foundation of Computer Science*, 1984, pp. 458-463.

[19] Yao, A.C., "Theory and Applications of Trapdoor Functions", *Proc. of the 23rd IEEE Symp. on Foundation of Computer Science*, 1982, pp. 80-91.