A Hard-Core Predicate for all One-Way Functions

Oded Goldreich*
Computer Sci. Dept.
Technion
Haifa, Israel
e-mail: oded@techsel.bitnet

Leonid A. Levin†
Boston University
111 Cummington Str.
Boston, MA 02215
e-mail: Lnd@bu-cs.bu.edu

Abstract

A central tool in constructing pseudorandom generators, secure encryption functions, and in other areas are "hard-core" predicates b of functions (permutations) f, discovered in [Blum Micali 82]. Such $b(x)$ cannot be efficiently guessed (substantially better than 50-50) given only $f(x)$. Both b, f are computable in polynomial time.

[Yao 82] transforms any one-way function f into a more complicated one, f^*, which has a hard-core predicate. The construction applies the original f to many small pieces of the input to f^* just to get one "hard-core" bit. The security of this bit may be smaller than any constant positive power of the security of f. In fact, for inputs (to f^*) of practical size, the pieces effected by f are so small that f can be inverted (and the "hard-core" bit computed) by exhaustive search.

In this paper we show that every one-way function, padded to the form $f(p,x) = (p, g(x))$, $\|p\| = \|x\|$, has by itself a hard-core predicate of the same (within a polynomial) security. Namely, we prove a conjecture of [Levin 87, sec. 5.6.2] that the scalar product of boolean vectors p, x is a hard-core of every one-way function $f(p,x) = (p, g(x))$. The result extends to multiple (up to the logarithm of security) such bits and to any distribution on the x's for which f is hard to invert.

1 Introduction

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 [Blum Micali 82, Yao 82] and secure probabilistic encryption [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

*Part of this research was done while the first author was visiting the International Computer Science Institute (ICSI) in Berkeley, California. The first author was supported by grant No. 86-00301 from the United States - Israel Binational Science Foundation (BSF), Jerusalem, Israel.

†Supported by NSF grant DCR-8607492.

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, permutations with hard-core predicates must be one-way. A natural question of practical and theoretical importance is *which* one-way functions have one. So far only partial answers have been given:

1. [Blum Micali 82] showed that the discrete exponentiation function, if it is one-way, has a hard-core predicate.[1] Analogous results for the RSA and Rabin functions (i.e. raising to a power modulo a composite integer) have been shown in [Alexi Chor Goldreich Schnorr 84].

2. [Yao 82] showed how any one-way permutation f can be used to construct another one-way permutation 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 parallel: $f^*(x_1 \dots x_{k^2}) = f(x_1) \dots f(x_{k^2})$, $\|x_i\| = k$. (For a proof and more refined analysis see [Levin 87].)

The drawback of the first set of results is their dependence on a specific intractability assumption (e.g. the hardness of the discrete logarithm problem). The second result offers a function f^* with security smaller than any constant power of the security of f.

This paper resolves the above question by providing a hard-core predicate for every one-way function. More specifically, for any 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, and can be computed by trying all small guessing algorithms. It is assumed to grow very fast (at least $n^{1/o(1)}$).

2 Conventions

Let S be the set of finite and Ω of infinite strings in the alphabet $\{0,1\}$; let $\|x\|$ be the length of x, $S_n \subset S$ be the set of strings of length n, and $x \circ y$ be the concatenation of x and y. We identify S (in lexicographical order) with $\mathbf{N} = \{0,1,...\}$.

For simplicity, we consider only length preserving functions $f : S \to S$, $\|f(x)\| = \|x\|$. Let F be the set of such functions. The set of functions computable in polynomial time is denoted P. We almost always refer to predicates as having range $\{\pm 1\}$ instead of $\{0,1\}$. The Boolean value σ corresponds to $(-1)^\sigma \in \{\pm 1\}$, and the exclusive-or of Boolean values corresponds to multiplication.

When the probability distribution of x is clear from the context, $\mathrm{E}_x f(x)$ denotes the expected value of f. Thus, a Boolean predicate P holds with probability $\mathrm{E}_x P(x)$.

Time bounds of the (adversarial) algorithms is assumed here to be huge comparative to input lengths. Thus we allow this

[1] Extended to all Abelian groups in [Kaliski 88].

preliminary draft to ignore factors of $\|x\|^{O(1)}$ in the running time. So time bound of $T(x)$ means that the algorithm may make $\|x\|^{O(1)}T(x)$ steps. These factors are easy to figure out from the context. This convention makes the choice of model of computation not so important.

3 The Results

Below, the function d merely generates a probability distribution of instances (and the condition $d \in \mathrm{F}$ is not really essential). A simple case of interest is $d(x) = x$.

Let $I(\omega, y)$ be a probabilistic algorithm which attempts to invert $f \in \mathrm{F}$, i.e. to compute a list containing x from $y = f(x)$, using $\omega \in \Omega$ as the source of internal coin flips. Its performance has two aspects: the running time $T_I(\omega, y)$ and the probability of success:

$$R_{I,f,d}(n) \stackrel{\mathrm{def}}{=} \mathrm{E}_{r,\omega}(x \in I(\omega, f(x))),$$

where $r \in S_n$, $x = d(r)$. We can combine these two measures (i.e. absorb the running time into $R_{I,f,d}$) in the following way. Without loss of generality, we require the mean running time $\mathrm{E}_\omega T_I(\omega, y)$ of all *inverting algorithms* to be $O(1)$. Any algorithm can be modified to satisfy this requirement. For this purpose, I may use its power of flipping coins to set itself random time limits 2^t with probability, say, $2^{-t}/t^2$. This will decrease the probability of success in proportion to I's original running time. Then $R_{I,f,d}$ accounts for running time as well as the probability of success and is called the *inverting rate* of I for f on d. It reflects the reciprocal of the time needed to notice by sampling the instances where I inverts f.

One-way functions are those invertible with a negligible rate only. Namely,

Definition 1 (One-way Functions): A function $f \in \mathrm{F}$ is *one-way* on $d \in F$ with security $s : \mathbb{N} \to \mathbb{N}$ if $R_{I,f,d}(n) = o(1)/s(n)^\epsilon$, for some $\epsilon > 0$ and all probabilistic inverting algorithms I. The security is *strict* if $\epsilon > 1$.

Let $G(\omega, y) \in \{\pm 1, 0\}$ be a probabilistic algorithm which, given $y = f(x)$, attempts to guess a predicate $b(x) \in \{\pm 1\}$, $b \in \mathrm{P}$, using ω as the source of internal coin flips (0 means a refusal to guess). As with inverting algorithms above, without loss of generality, we restrict its average running time: $\mathrm{E}_\omega T_G(\omega, y) = O(1)$. We define the *guessing rate* R of G for b from f to reflect the reciprocal of the time needed to notice by sampling the correlation between b and G, i.e. the number of trials needed to evaluate the expectation of their product. Due to the $O(1)$ restriction on expected running time, this number accounts for both the running-time and the correlation of G with b. The number of trials is determined by the reciprocal of

$$R_{G,f,b,d}(n) \stackrel{\mathrm{def}}{=} \frac{(\mathrm{E}_{\omega,r} b(x) G(\omega, y))^2}{\mathrm{E}_{\omega,r} G(\omega, x)^2},$$

where $r \in S_n$, $x = d(r)$, $y = f(x)$, $\omega \in \Omega$.

Hard-core predicates of a function are those that can be guessed from its output only with a negligible rate. Namely,

Definition 2 (Hard-Core Predicates): A predicate b is called a *hard-core* with security s for a function $f \in \mathrm{F}$ on $d \in F$ if $R_{G,f,b,d}(n) = o(1)/s(n)^\epsilon$, for some $\epsilon > 0$ and all probabilistic guessing algorithms G. The security is *strict* if $\epsilon > 1$.

We call *padded* a function $f \in \mathrm{F}$ of the form $f(x \circ p) = f'(x) \circ p$ for $\|p\| = \|x\|$. *Double-padded* f has $\|p\| = 2\|x\|$. Let $B(x, p) = \pm 1$ depending on the inner product mod 2 of the Boolean vectors x and p, $\|p\| = \|x\|$.

Theorem 1 *Let f and d be arbitrary padded functions and f be one-way on d with security s. Then B is a hard-core predicate for f on d (with the same security s).*

The theorem follows from the following Lemma, which efficiently reduces the task of retrieving x from $f(x,p)$ to the task of approximating $B(x,p)$ given $f(x,p)$. The rate ε in the Lemma is chosen at random with distribution assuring the $O(1)$ average running time. Let $\tau(p,y) = \mathrm{E}_\omega |G(\omega,y,p)|$, $\tau(y) = \mathrm{E}_p \tau(p,y)$ and $\overline{\tau}(y) = \mathrm{E}_p \tau^2(p,y)$. Note that the guessing rate of G does not exceed the average over x of its *local* (i.e. taken over fixed $x, y = f'(x)$) guessing rates $R(x) = (\mathrm{E}_{\omega,p} B(x,p) G(\omega,y,p))^2 / \tau(y)$.

Lemma 1 (Main): *There exists an algorithm I that given a subroutine G, the coin-flip source $\omega \in \Omega$, and inputs $y \in S, \varepsilon \in (0,1]$, outputs a list of $\overline{\tau}/\tau\varepsilon$ strings including all $x \in S_{\|y\|}$ with $R(x) \geq \varepsilon$. $I(\omega,y,\varepsilon)$ may fail for ε fraction of ω. The first of the two stages of I takes $1/\varepsilon$ steps of I and G. The second stage does not call G and takes $(\overline{\tau}/\tau\varepsilon)^2 < \varepsilon^{-2}$ steps.*

The proof is in Section 4. It seems likely that the second stage can also be sped up to $1/\varepsilon$ steps. The computation can be parallelized using $\varepsilon^{-O(1)}$ processors in $-\log \varepsilon$ time.

The theorem extends to $\log s(n)$ secure bits. For $c \in \{0,1\}$, $a_0, a_1 \in S$, define $\mathrm{Pr}_c(a_0,a_1) = a_c$.

Definition 3 (Hard-core function): A function $h\colon S \to S$ in P is called a *hard-core* (with security s) of $f \in \mathrm{F}$ on $d \in \mathrm{F}$ if $b(x,r,c) = (-1)^c$ is a hard-core predicate (with security s) on d for the function $f_h(x,r,c) = f(x) \circ \mathrm{Pr}_c(r, h(x)) \circ 0$, $\|r\| = \|h(x)\|$.

So, given $f(x)$, the output of a hard-core $h(x)$ should be indistinguishable in feasible time from a randomly chosen string r.

A *Toeplitz* matrix is a matrix M such that for all i,j, $M_{i,j} = M_{i+1,j+1}$. Let $k\colon \mathbf{N} \to \mathbf{N}$ and M_p be $k(n) \times n$ Boolean Toeplitz matrix with the first row and column determined by the corresponding bits of p, $\|p\| \geq k(n) + n - 1$. Let Boolean vector $H_k(x,p)$ be the matrix product M_p times x, for $\|p\| = 2\|x\|$.

Corollary 1 *Let f, d be arbitrary double-padded functions and f is one-way on d with security $s \in \mathrm{P}$. Then, for some $\epsilon > 0$, $k(n) = \epsilon \log s(n)$, H_k is a hard-core function for f on d with the same security s.*

The proof is in Section 5. This number of pseudorandom bits cannot be improved without additional assumptions or a major breakthrough in complexity theory. The reason is that a one-way function with security s may act only on $\log s(\|x\|)$ of the bits of x and leave the rest unchanged.[2] A restriction of the "optimality" claim to bits extractable through linear transformations (as in the Corollary) can be easily proven.

4 Proof of Main Lemma

Our inverting algorithm $I(\omega,y,\varepsilon)$ lists all strings x for which G guesses B with a local rate $R(x) = (\mathrm{E}_{p,\omega} B(x,p) G(\omega,y,p))^2 / \tau(y) \geq \varepsilon$. I constructs the list L containing these strings and all their prefixes bit by bit, in $n = \|y\|$ rounds. The k-th round of I generates $L_k = L \cap S_k$. During the next round I

[2] To exclude this possibility one must rule out the existence of one-way functions f with security $2^{\varepsilon n}$. Otherwise $f_s(x'x'') = f(x')x''$, $\epsilon\|x'\| = \log s(\|x'x''\|)$ has security s.

examines all one-bit extensions of strings in L_k, discards some of them and keeps the rest.

Throughout the proof y, k, ε are fixed and used implicitly. So, let g be the matrix with components $g_{r,s} = \mathrm{E}_\omega G(\omega, y, rs)$, and b be the matrix with components $b_{z,r} = B(z,r)$ for $r, z \in S_k$.

Note 1 *Matrix $2^{-k/2}b$ is symmetric and orthonormal, i.e. $b = b^T = 2^k b^{-1}$.*

Proof: Indeed, $\sum_r (B(z,r))^2 = 2^k$. When z_1, z_2 differ by i-th digit, let $z = z_1 \oplus z_2$ and r' be r with i-th digit changed.

Then $\sum_r B(z_1,r)B(r,z_2) = \sum_r B(z,r) = 0$, since $B(z,r) = -B(z,r')$. ∎

We discard prefixes z from L using an upper bound $c(z)$ for $R(x)\tau(y)$, $x = zz'$. It is $c(z) \stackrel{\text{def}}{=} \mathrm{E}_s(\mathrm{E}_{r,\omega} B(zz', rs)G(\omega, y, rs))^2 = \mathrm{E}_s(\mathrm{E}_r B(z', s)B(z,r)g_{r,s})^2$, which then simplifies to $\mathrm{E}_s(\mathrm{E}_r B(z,r)g_{r,s})^2 = \mathrm{E}_s(2^{-k}bg)^2_{z,s}$, since $B(z',s) = \pm 1$.

The number of $z \in S_k$ with $c(z) > \varepsilon\tau(y)$ remains limited:

Lemma 2 $\sum_z c(z) = \mathrm{E}_{r,s} g^2_{r,s} \le 1$.

Proof: Any matrix multiplied by orthonormal matrix $2^{-k/2}b$ preserves its mean square of the elements. So, $\sum_z c(z) = 2^k \mathrm{E}_z c(z) = \mathrm{E}_{z,s}(2^{-k/2}bg)^2_{z,s} = \mathrm{E}_{r,s} g^2_{r,s}$. ∎

All the above does not yield an efficient algorithm, since the straightforward computation of $c(z)$ takes exponential time. However, the exact value is not needed, a good approximation suffices:

Let $\tau_p = \tau(p,y)$ for $\|p\| = n$, $\tau_s = \mathrm{E}_r \tau_{rs}$ for $|rs\| = n$; $\overline{\tau}_s = \mathrm{E}_r \tau^2_{rs}$.

Lemma 3 *There is a probabilistic algorithm $A(\omega, z, y, \delta)$ (using ω for its internal coin flips) that outputs an approximation $\tilde{c}(z)$ to $c(z)$ with accuracy $O(\delta)$ and probability of failure δ^3. A halts within τ/δ steps, including the steps of the subroutine G. Only its last $\overline{\tau}/\delta$ steps depend on z.*

Proof Sketch: The approximation, $\tilde{c}(z)$, is computed as $\sum_i 4^{-i}\mathrm{E}_s(c_s(z) > 2^{-i})$. If we get estimates with standard deviation α, repeating them $l = O(-\log\delta)$ times and taking the median gives an approximation deviating from c_s by 2α with exponentially small, in l, probability. For every i we will need (within log factors) $4^{-i}/\delta$ samples of s and $4^i\tau_s$ samples of (r, ω'), of which $4^i\tau^2_s$ will produce a ± 1 guess. These "productive" samples may be reused for each z. ∎

A speed up may be achieved by computing $\tilde{c}(z)$ simultaneously for many z.

Clearly all steps of passing from L_k to L_{k+1} can be made in parallel. Still each $k \le n$ takes a sequential iteration. A parallel speed-up can be obtained by keeping the lists of candidates for all substrings of a particular length. Let $L_{i,l}$ be the list of candidates for the l-bit long substring of x starting at location i. Then $L_{i,2l}$ is formed by all strings z in the concatenation of $L_{i,l}, L_{i+l,l}$ having $c_i(z) \stackrel{\text{def}}{=} \mathrm{E}_{s',s''}(\mathrm{E}_r B(z,r)g_{s'rs''})^2 > \varepsilon\tau$.

5 Proof of the Corollary

Corollary 1 is a special case of the following Lemma 4, as its conditions are obviously satisfied by the family of Toeplitz matrices. Another simple case is a family of all Boolean matrices. Let $a, k : \mathbf{N} \to \mathbf{N}$ be in P. Let $\{M_p : p \in S_{a(n)}\}$ be a family of $k(n) \times n$ Boolean matrices, $P(u, q, \omega)$ be an algorithm selecting (on inputs u, q and coin tosses ω) an index p such that $uM_p = q$, and $M_{P(u,q,\omega)}$ be computable in polynomial time.

Lemma 4 *Suppose M, P are as above and, for each $u \neq 0$, $\mathrm{E}_{q,z}(p = P(u,q,z)) = 2^{-\|p\|}$. Then the function $H(x,p) = M_p x$ is hard-core for f on d with strict security s, if the predicate $B(x,q)$ is hard-core with strict security $s(n)4^{k(n)}$.*

The proof of Lemma 4 is based on the idea of [Vazirani 87]. It also incorporates the "XOR condition" of [Vazirani Vazirani 84], proving that a function is hard-core iff the exclusive-or of any non-empty subset of its bits is.

Proof: The orthonormal functions $B_u\colon r \mapsto B(r,u)$ form a linear basis in the Euclidean space of all real functions on S_k. So, any function can be expressed as $g(r) = \sum_u c_u B_u(r)$, where $c_u = \mathrm{E}_v B(v,u)g(v)$. A general form for the case $c_0 = \mathrm{E}_r g(r) = 0$ is $g(r) = N\mathrm{E}_{u\neq 0,v} B(v,u)g(v)B(r,u)$, with $N = 2^{\|r\|}-1$.

Let an algorithm $G_{\omega,y,p}(r_c)$ guess $(-1)^c$ with correlation ϵ given $(y,p) = f(x,p) = (f'(x),p)$, the source ω of internal coin flips and $r_c = \mathrm{Pr}_c(M_p x, r)$. Without loss of generality we may assume $\mathrm{E}_{\omega,r} G_{\omega,y,p}(r) = 0$, which may be achieved by modifying G so that with probability $1/2$ it is applied to a random string instead of its argument r and the sign of the output changed. Then $\overline{G}_{y,p}(r) \stackrel{\mathrm{def}}{=} E_\omega G_{\omega,y,p}(r) = N\mathrm{E}_{u\neq 0,v} B(v,u)\overline{G}_{y,p}(v)B(r,u)$.

G's correlation is: $\mathrm{E}(-1)^c\overline{G}_{f'(x),p}(r_c) =$

$$N\mathrm{E}\,(-1)^c E_{u\neq 0,v} B(v,u)\overline{G}_{f'(x),p}(v)B(r_c,u),$$

where $r_c = \mathrm{Pr}_c(M_p x, r)$ and E averages over x, p, r, c. For $c=1$, it is

$$-N\mathrm{E}_{u\neq 0,v,x,p,r} B(v,u)\,\overline{G}_{f'(x),p}(v)\,B(r,u) = 0,$$

since $\mathrm{E}_r B(r,u) = 0$, for each $u \neq 0$.

For the term $c = 0$, we use $B(Mx,u) = E(x,uM)$ and express $(1/N$ of) it as

$$\begin{aligned} & \mathrm{E}_{u\neq 0,v,x,p}\, B(v,u)\,\overline{G}_{f'(x),p}(v)\,B(x,uM_p) \\ = \;& \mathrm{E}_{\omega',x,q} g(\omega', f'(x), q)B(x,q), \end{aligned}$$

where $g(\omega',y,q) \stackrel{\mathrm{def}}{=} B(v,u)G_{\omega,y,P(u,q,\omega)}(v)$, with $u \neq 0, v, w, \omega$ generated with uniform distribution from ω'. So, g guesses $B(x,q)$ with correlation ϵ/N. ∎

6 Acknowledgements

We are grateful to Peter Elias whose comments about list decoding were crucial for discovering an earlier version of Lemma 2. We would like to thank Mike Luby, and Noam Nissan for very useful discussions concerning extensions of the main result. We are also grateful to Manuel Blum, and R. Karp for comments and suggestions.