Complete Classification of Bilinear Hard-Core Functions

Thomas Holenstein, Ueli Maurer, and Johan Sjödin
{thomahol,maurer,sjoedin}@inf.ethz.ch

Department of Computer Science,
Swiss Federal Institute of Technology (ETH),
Zürich, Switzerland

Abstract. Let $f : \{0,1\}^n \to \{0,1\}^l$ be a one-way function. A function $h : \{0,1\}^n \to \{0,1\}^m$ is called a hard-core function for f if, when given $f(x)$ for a (secret) x drawn uniformly from $\{0,1\}^n$, it is computationally infeasible to distinguish $h(x)$ from a uniformly random m-bit string. A (randomized) function $h : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^m$ is a *general* hard-core function if it is hard-core for every one-way function $f : \{0,1\}^n \to \{0,1\}^l$, where the second input to h is a k-bit uniform random string r. Hard-core functions are a crucial tool in cryptography, in particular for the construction of pseudo-random generators and pseudo-random functions from any one-way function.
The first general hard-core predicate, proposed by Goldreich and Levin, and several subsequently proposed hard-core functions, are bilinear functions in the two arguments x and r. In this paper we introduce a parameter of bilinear functions $h : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^m$, called exponential rank loss, and prove that it characterizes exactly whether or not h is a general hard-core function. The security proofs for the previously proposed bilinear hard-core functions follow as simple consequences. Our results are obtained by extending the class of list-decodable codes and by generalizing Hast's list-decoding algorithm from the Reed-Muller code to general codes.

Keywords: List-decoding, hard-core functions, Goldreich-Levin predicate.

1 Introduction

Blum and Micali [BM84] showed a hard-core predicate[1] for the exponentiation function modulo a prime, which is widely conjectured to be one-way (except for special primes). They also showed how to construct a pseudo-random generator based on it. Hard-core predicates are also known for some other specific (conjectured) one-way functions.

In a seminal paper [GL89], Goldreich and Levin proved that for any one-way function $f : \{0,1\}^n \to \{0,1\}^l$, the XOR of a random subset of the n bits

[1] The term predicate is used throughout to denote a function with range $\{0,1\}$.

of the input x constitutes a hard-core predicate. This function is randomized (because of the choice of a random subset), and it is easy to see that any general hard-core function must be randomized. An alternative view is to interpret the randomizing input of the hard-core function as an extra input and output of a modified one-way function $f' : \{0,1\}^{2n} \to \{0,1\}^{l+n}$ defined by

$$f'(x,r) = (f(x), r)$$

which now has a deterministic hard-core function $h(x,r)$.[2] The Goldreich-Levin hard-core function is simply the inner product of x and r, which is a bilinear function $h : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$.

Any such bilinear map h is characterized by a binary $n \times n$ matrix M, where $h(x,r) = x^T \cdot M \cdot r$. For the Goldreich-Levin predicate, M is simply the identity matrix.

One can show (see [Lub96]) that $m = O(\log n)$ independent Goldreich-Levin predicates are jointly hard-core, i.e., they form a hard-core function $h : \{0,1\}^n \times \{0,1\}^{mn} \to \{0,1\}^m$. An important issue is to reduce the required amount of randomness in a hard-core function. A construction presented in [GL89] (see also [Gol01]) requires only $n+m-1$ instead of mn random bits for an m-bit hard-core function. Goldreich, Rubinfeld, and Sudan [GRS00] reduced the number of random bits down to n, as for the Goldreich-Levin function which produces only one (rather than m) bits. While some of the proofs of these results as they appear in the literature are non-trivial, they will all follow as simple consequences of our main theorem.

More generally, one can consider bilinear functions for vector spaces over any finite field $\mathbb{F}$, i.e., functions $h : \mathbb{F}^n \times \mathbb{F}^k \to \mathbb{F}^m$. We are interested in characterizing which of these functions are general hard-core functions. This characterization turns out to be given by a quite simple parameter of such a bilinear function. The characterization is complete in the sense that when the parameter is below a certain threshold, then the function is hard-core, and otherwise there exist one-way functions (under some reasonable complexity-theoretic assumption) such that h is not a hard-core function for f.

Let us discuss this parameter. For any linear function $\ell : \mathbb{F}^m \to \mathbb{F}$, the function $\ell \circ h$ is a bilinear function $\mathbb{F}^n \times \mathbb{F}^k \to \mathbb{F}$ which can be characterized by an $n \times k$ matrix over $\mathbb{F}$. The parameter of interest, which we call exponential rank loss, is defined as the expected value of the exponentially weighted rank of this matrix, when averaged over all non-zero functions ℓ.

The main technical part of [GL89] consists in showing that an error-correcting code has certain list-decoding properties, i.e., that it is possible to find a list of all codewords in a Hamming ball of a certain size. In this paper we show how to list-decode a larger class of codes. The stated characterization of hard-core functions will then follow.

An application of one-way functions and hard-core predicates are pseudorandom generators. It is easy to obtain a pseudorandom generator from any one-way

[2] Yao's method (implicit in [Yao82]) of using several copies of a one-way function and computing the XOR of some of the inputs can also be seen in the same light.

permutation f by iterating f and after each iteration extracting a (the same) hard-core predicate. It is much more complicated and less efficient to use any one-way function (see [HILL99]).

The security of a cryptographic scheme that uses a pseudo-random generator is proven by showing that an algorithm breaking the scheme could distinguish the pseudo-randomness from real randomness. Hast [Has03] showed that in many cryptographic applications, breaking the scheme is actually stronger than just distinguishing the randomness from pseudorandomness with small probability, in the sense that if an algorithm is given a pseudo-random or random input and it breaks the scheme, then it is almost certain that the input was pseudo-random rather than random. Hast then shows that this leads to an improved security analysis for many constructions. The main technical tool is an extension of the list-decoding algorithm to the case where erasures in the codewords are allowed. We use this extension, and furthermore generalize Hast's result by giving list-decoding algorithms that are able to handle erasures for more general codes.

Section 2 introduces the notation and discusses bilinear functions and list-decoding, the main technical tool of the paper. Previous work is also summarized in this section. In Section 3, we analyze a special case of bilinear functions, namely these for which all matrices mentioned above (i.e., for all non-zero linear functions) have full rank. This special case already suffices to prove previous results in the literature. We generalize the algorithm in Section 4 such that it works with *any* bilinear code, where the running time and the produced list will grow linearly with the exponential rank loss of the code. In Section 5 we discuss the application to characterizing hard-core functions.

2 Preliminaries

We use calligraphic letters to denote sets. Capital letters denote random variables over the corresponding sets; and lowercase letters denote specific values of these random variables, i.e., values in the sets.

The notation $f : \mathcal{X} \to \mathcal{Y}$ is used to denote a function f from the domain $\mathcal{X}$ to the range $\mathcal{Y}$. Sometimes, functions take additional randomness (i.e., for every input $x \in \mathcal{X}$ the function only specifies a probability distribution over $\mathcal{Y}$). In this case we write $f : \mathcal{X} \rightsquigarrow \mathcal{Y}$, a notation which also will be used to denote randomized algorithms with domain $\mathcal{X}$ and range $\mathcal{Y}$. If an algorithm has access to a randomized function, we use the term *oracle* for the randomized function.

2.1 Bilinear Functions

Let $\mathbb{F} = \mathrm{GF}(q)$ be the finite field with q elements and let $\mathbb{F}^n$ be the n-dimensional vector space of n-tuples over $\mathbb{F}$. As a special case, we identify $\{0, 1\}$ with $\mathrm{GF}(2)$, and the bitstrings $\{0, 1\}^n$ of length n with the n-dimensional vector space over $\mathrm{GF}(2)$.

A linear function $\ell : \mathbb{F}^n \to \mathbb{F}$ can be specified by a vector $w \in \mathbb{F}^n$ such that $\ell(v) = \langle w, v\rangle := \sum_i v_i w_i$. We use $\mathscr{L}_n$ to denote the set of all linear functions

$\ell : \mathbb{F}^n \to \mathbb{F}$. Furthermore, $\mathbf{0}$ will denote the zero function $\mathbf{0}(v) \equiv 0$ and we use $\mathscr{L}_n^* := \mathscr{L}_n \backslash \{\mathbf{0}\}$ for the set of all linear functions excluding $\mathbf{0}$.

A *bilinear map* $h : \mathbb{F}^n \times \mathbb{F}^k \to \mathbb{F}$ can be specified by a matrix $M \in \mathbb{F}^{n \times k}$ such that $h(v, w) = v^T M w$. The rank of a bilinear map is just the rank of this matrix. A *bilinear function* $h : \mathbb{F}^n \times \mathbb{F}^k \to \mathbb{F}^m$ is a function where every entry in the output vector is specified by a bilinear map. Note that for any function $\ell \in \mathscr{L}_m$ the concatenation $\ell \circ h$ is a bilinear map. If L is a uniformly chosen random linear function from $\mathscr{L}_m^*$, the *exponential rank loss* $\rho(h)$ is defined as

$$\rho(h) := \mathrm{E}[q^{n - \mathrm{rank}(L \circ h)}].$$

We say that a bilinear function is *full-rank*, if $\mathrm{rank}(\ell \circ h) = n$ for every $\ell \in \mathscr{L}_m^*$ (in which case $\rho(h) = 1$).

2.2 List-Decoding

The main tool in the construction of hard-core functions is the notion of a list-decodable code. Such a code has the property that, given a noisy codeword, it is possible to find a list of *all* codewords which have a certain agreement with the noisy codeword.

Consider a code $\mathcal{C}$ given as a function $\mathcal{C} : \mathcal{X} \to \mathcal{Z}^k$. Note that the input to the function (usually the message) is an element of $\mathcal{X}$ while the output (the codeword) is a k-tuple over $\mathcal{Z}$. The Hamming distance of two words of $\mathcal{Z}^k$ is the number of coordinates in which the words differ. List-decoding is the task of finding for a given $z^k \in \mathcal{Z}^k$ all the values x for which $\mathcal{C}(x)$ has a Hamming distance from z^k that is smaller than some predefined bound. This is in contrast to usual error-correcting, where one aims to find the *one* codeword which is closest to the received word. The most ambitious task is to list-decode close to the noise barrier: given any $\varepsilon > 0$ one wants to find all values x for which $\mathcal{C}(x)$ has a Hamming distance of at most $(1 - \frac{1}{|\mathcal{Z}|} - \varepsilon)k$ from a given word. Since a random word has expected distance $(1 - \frac{1}{|\mathcal{Z}|})k$ from any codeword, this is clearly the best one can expect to achieve.

Instead of considering the function $\mathcal{C}(x)$, one can equivalently consider a function $h : \mathcal{X} \times \{1, \ldots, k\} \to \mathcal{Z}$, such that $h(x, i)$ is the value of $\mathcal{C}(x)$ at the i-th position. More generally we consider functions $h : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ for any domain $\mathcal{Y}$. Analogous, we assume that we have *oracle access* to the noisy word to be decoded: instead of reading the complete word it will be convenient to assume that an oracle $\mathcal{O} : \mathcal{Y} \rightsquigarrow \mathcal{Z}$, on input y, returns the symbol at position y. This allows us to list-decode in sublinear time, i.e., without looking at every position of the word, which in turn allows the codewords to be exponentially large. The oracle is stateless, but may be randomized and is not required to return the same symbol if queried twice with the same input. The agreement of an oracle with a codeword is then expressed as $\Pr[h(x, Y) = \mathcal{O}(Y)]$, where the probability is over the choices of Y and the randomness of the oracle.

Additionally, we allow erasures in the word which will be denoted by $\perp$. Thus, the oracle is a randomized function $\mathcal{O} : \mathcal{Y} \rightsquigarrow \mathcal{Z} \cup \{\perp\}$. The *rate* δ of such an oracle is the probability that a symbol in $\mathcal{Z}$ is returned,

$$\delta := \Pr[\mathcal{O}(Y) \neq \perp].$$

For a fixed word x, the *advantage* ε of $\mathcal{O}$ is defined as

$$\varepsilon := \Pr[\mathcal{O}(Y) = h(x, Y) \mid \mathcal{O}(Y) \neq \perp] - \frac{1}{|\mathcal{Z}|}.$$

This motivates the following definition:

Definition 1 (List-decodable code).[3] *The function $h : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ is (δ, ε)-*list-decodable *with κ oracle calls and list size λ if there exists an oracle algorithm with running time $\lambda \cdot \mathrm{poly}(\log(|\mathcal{X}|))$ which, after at most κ oracle calls to an oracle $\mathcal{O} : \mathcal{Y} \rightsquigarrow \mathcal{Z} \cup \{\perp\}$ with rate at least δ, generates a set Λ of size at most λ, such that for every x with $\Pr[\mathcal{O}(Y) = h(x, Y) \mid \mathcal{O}(Y) \neq \perp] \geq \frac{1}{|\mathcal{Z}|} + \varepsilon$ the set satisfies $\Pr[x \in \Lambda] \geq 1/2$.*

2.3 Hard-Core Functions

Informally, a one-way function is a function which is easy to evaluate but hard to invert.

Definition 2 (One-way function). *An efficiently computable function family $f : \{0,1\}^n \to \{0,1\}^{p(n)}$ with $p(n) \in \mathrm{poly}(n)$ is a* one-way function *if for every probabilistic polynomial time (in n) algorithm A the inverting probability $\Pr[f(A(f(X))) = f(X)]$ is negligible.*

A hard-core function $h : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^m$ can intuitively extract bits from the input of a one-way function f such that these bits look random, even given $f(x)$. We can distinguish *(strong) hard-core functions*, where the output is indistinguishable from a random string of length m (which we denote by U^m), and *weak hard-core functions*, where the output of the function is hard to predict.

Definition 3 (Strong hard-core function). *An efficiently computable family $h : \{0,1\}^n \times \{0,1\}^{k(n)} \to \{0,1\}^{m(n)}$ of functions, with $k(n), m(n) \in \mathrm{poly}(n)$ is a* (strong) hard-core function *if, for every one way function $f : \{0,1\}^n \to \{0,1\}^{p(n)}$ and every probabilistic polynomial time algorithm A, the distinguishing advantage given by $\Pr[A(f(X), R, h(X, R)) = 1] - \Pr[A(f(X), R, U^m) = 1]$, is negligible in n.*

Definition 4 (Weak hard-core function). *An efficiently computable family $h : \{0,1\}^n \times \{0,1\}^{k(n)} \to \{0,1\}^{m(n)}$ with $k(n), m(n) \in \mathrm{poly}(n)$ of functions is a* weak hard-core function *if, for every one-way function $f : \{0,1\}^n \to \{0,1\}^{p(n)}$ and every probabilistic polynomial time algorithm A, the advantage of A in guessing $h(x, r)$ on input $f(x)$ and r, defined as $\Pr[A(f(X), R) = h(X, R)] - \frac{1}{2^m}$, is negligible in n.*

[3] We require the list-decoding algorithm to work in time $\lambda \cdot \mathrm{poly}(\log(|\mathcal{X}|))$. Note that in some cases, λ will be superpolynomial in the input size $\log(|\mathcal{X}|)$ and $\log(|\mathcal{Y}|)$.

In general, weak hard-core functions are easier to construct than strong ones. However, we will see that for small outputs the notions are equivalent.

As shown in [Sud00], any list-decodable code $h : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^m$ as defined above yields a weak hard-core function. To prove this, one assumes for the sake of contradiction that an algorithm B is given which on input $f(x)$ and r predicts $h(x,r)$ with probability higher than $\frac{1}{2^m} + \varepsilon$, for some non-negligible[4] ε. After arguing that B needs to have a reasonable success probability for a significant subset of the possible values for x, one then uses B as the oracle in the list-decoding algorithm. The resulting list, which is small, then contains x with non-negligible probability, and one can find a preimage of $f(x)$ by applying f to all values in the list.

In such a reduction, the running time of the resulting algorithm is dominated by the running time of B. Thus, one is interested in the exact number κ of oracle calls, while the exponent in the running time of the (polynomial) algorithm is of minor importance. In this application, the second input (from $\{0,1\}^k$) corresponds to a random string. As randomness is an expensive resource, one wants k to be as small as possible. We show how to achieve $k = n$ for any n.

2.4 Previous Work

The fundamental result on bilinear list-decodable codes implicitly appears in [GL89], stating that the Reed-Muller code of first order, defined as $h : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, $h(x,y) = \langle x,y \rangle = \sum_i x_i y_i$, has an algorithm which efficiently list-decodes it up to an error rate of $1/2 + \varepsilon$, for any $\varepsilon > 0$.

The standard proof used today was found independently by Levin and Rackoff and is given in [Gol01] (see also [Lev87]). In [Has03], Hast introduces the extension of list-decoding algorithms for oracles with erasures. The existence of the resulting algorithm is asserted in the following theorem:

Theorem 5 (Goldreich-Levin, cf. [Has03]). *For any $\varepsilon, \delta > 0$, the function $h : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, $h(x,r) = \langle x,r \rangle$ is (δ,ε)-list-decodable with list size $O(\frac{1}{\delta\varepsilon^2})$ and $\Theta(n\frac{1}{\delta\varepsilon^2})$ oracle calls. The list-decoding algorithm needs $\delta\varepsilon^2$ as input.*

This theorem is slightly stronger than the original version in [Has03], where an additional factor n appears in the number of oracle calls and the list size. The version as stated here can be obtained by applying a trick that appears in [Gol01, Section 2.5.2.4].[5]

It is natural to generalize this theorem to vector spaces over any finite field. For this, the best known result is given in [GRS00].

Theorem 6. *For any $\delta, \varepsilon > 0$, the function $h : \mathbb{F}^n \times \mathbb{F}^n \to \mathbb{F}$, $h(x,r) = \langle x,r \rangle$ is (δ,ε)-list-decodable with list size $\mathrm{poly}(n, \delta^{-1}\varepsilon^{-1})$ and $\mathrm{poly}(n, \delta^{-1}\varepsilon^{-1})$ oracle calls. The list-decoding algorithm needs $\delta\varepsilon$ as input.*

---

[4] We use *non-negligible* to denote a function which is *not* negligible.

[5] Basically, one uses a linear, asymptotically optimal error-correcting code to find x instead of finding the bits one by one.

The algorithm which is used to prove Theorem 6 is similar to the original algorithm given in [GL89]. The exponents in $\mathrm{poly}(n, \delta^{-1}\varepsilon^{-1})$ are rather high, so we refrain from stating them explicitly.

Näslund shows in [Näs95] that for any one-way function $f(x)$, a hard-core predicate can be obtained if one interprets x as a value in $\mathrm{GF}(2^n)$, and outputs any bit of $ax + b$ for randomly chosen a and b; a result which also follows from the characterization in this paper. Furthermore, he proves that for randomly chosen a, b and prime p the least significant bit of $ax + b \bmod p$ is a hard-core predicate. More generally, in [Näs96] he shows that all bits of $ax + b \bmod p$ are hard-core.

In a different line of research, in [STV01] Sudan et al. give very strong list-decodable codes which are not bilinear, based on Reed-Muller codes. These codes can also be used to obtain hard-core functions for any one-way function.

In [AGS03], Akavia et al. show that list-decoding can also be used to prove specific hard-core results. For example, they give a proof based on list-decodable codes that the least significant bit of RSA is hard-core (which was first shown in [ACGS88]).

3 Full-Rank Bilinear Functions

The main technical goal of this paper is to give a list-decoding procedure for any bilinear function $h : \mathbb{F}^n \times \mathbb{F}^k \to \mathbb{F}^m$. In this section, we will first consider a simple, but very general subset of bilinear functions, namely full-rank bilinear functions h (i.e., $\mathrm{rank}(\ell \circ h) = n$ for every $\ell \neq \mathbf{0}$). We show that these functions have very good list-decoding algorithms.

In a second step we will construct full-rank bilinear functions $h : \mathbb{F}^n \times \mathbb{F}^k \to \mathbb{F}^m$ which are optimal in the sense that for fixed n the dimension k is made as small as possible, while for m every value $0 < m \leq k$ is possible. This allows us to give a very large class of strong hard-core functions.

3.1 List-Decoding of Full-Rank Functions

In this section, we give a list-decoding algorithm for every full-rank bilinear function $h : \mathbb{F}^n \times \mathbb{F}^k \to \mathbb{F}^k$. In particular, for the case $\mathbb{F} = \mathrm{GF}(2)$, we will show that there exists a list-decoding algorithm for h which is as strong as the one guaranteed in Theorem 5.

Theorem 7. *Let $h : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^m$ be a full rank bilinear function. For any $\delta, \varepsilon > 0$, the function $h(x, y)$ is (δ, ε)-list-decodable with list size $O(\frac{1}{\delta\varepsilon^2})$ and $\Theta(n\frac{1}{\delta\varepsilon^2})$ oracle calls. The list-decoding algorithm needs $\delta\varepsilon^2$ as input.*

For general finite fields, analogously to Theorem 6, the following holds.

Theorem 8. *Let $h : \mathbb{F}^n \times \mathbb{F}^k \to \mathbb{F}^m$ be a full-rank bilinear function. For any $\delta, \varepsilon > 0$, the function $h(x, y)$ is (δ, ε)-list-decodable with list size* $\mathrm{poly}(n, \delta^{-1}\varepsilon^{-1})$ *and* $\mathrm{poly}(n, \delta^{-1}\varepsilon^{-1})$ *oracle calls. The list-decoding algorithm needs $\delta\varepsilon$ as input.*

To prove Theorems 7 and 8, we describe an algorithm which, on access to an oracle $\mathcal{O}$ with rate δ, outputs a list of all $x \in \mathbb{F}^n$ which satisfy

$$\Pr[\mathcal{O}(Y) = h(x, Y) \mid \mathcal{O}(Y) \neq \perp] \geq \frac{1}{q^m} + \varepsilon. \tag{1}$$

For this purpose we convert $\mathcal{O}$ to an oracle $\mathcal{O}'$ with the same rate and related advantage, but for a different code. Namely, $\mathcal{O}'$ will have advantage $\varepsilon/2$ on $\langle x, r\rangle$ for any x which satisfies (1), i.e., $\Pr[\mathcal{O}'(R) = \langle x, R\rangle \mid \mathcal{O}'(R) \neq \perp] \geq \frac{1}{q} + \frac{\varepsilon}{2}$. Applying Theorems 5 and 6, respectively, then yields the result.

In the following, let L be a uniform random function from $\mathscr{L}_m^*$, i.e., L is a random variable taking as values functions from $\mathscr{L}_m^*$. We show that if a value z returned by the oracle is better than a random guess for $h(x, y)$, then $L(z)$ is better than a random guess for $L(h(x, y))$ as well. To see why this holds, we first compute the probability that $L(a)$ equals $L(b)$ for two distinct values a and b; this probability is close to $1/q$.

Lemma 9. *For any distinct $a, b \in \mathbb{F}^m$,* $\Pr[L(a) = L(b)] = \dfrac{q^{m-1} - 1}{q^m - 1}$.

Proof. First note that $\Pr[L(a) = L(b)] = \Pr[L(a - b) = 0] = \Pr[L(v) = 0]$ for some $v \neq 0$. If L' is chosen uniformly at random from all functions in $\mathscr{L}_m$ (not excluding $\mathbf{0}$), then $\Pr[L'(v) = 0] = \frac{1}{q}$, and since $\mathbf{0}(v) = 0$ for every v, we can write

$$\frac{1}{q} = \Pr[L'(v) = 0] = \underbrace{\frac{1}{q^m}}_{\Pr[L'=\mathbf{0}]} + \underbrace{\frac{q^m - 1}{q^m}}_{\Pr[L'\neq\mathbf{0}]} \Pr[L(v) = 0],$$

which implies the lemma. $\square$

Now we can estimate the probability that $L(Z_1)$ equals $L(Z_2)$ for two random variables Z_1 and Z_2. Later, Z_2 will be $h(x, Y)$ and Z_1 a guess of an oracle for $h(x, Y)$.

Lemma 10. *Let Z_1 be a random variable over $\mathbb{F}^m \cup \{\perp\}$ and Z_2 a random variable over $\mathbb{F}^m$. If, for any $\varepsilon > 0$,*

$$\Pr[Z_1 = Z_2 \mid Z_1 \neq \perp] = \frac{1}{q^m} + \varepsilon,$$

then

$$\Pr[L(Z_1) = L(Z_2) \mid Z_1 \neq \perp] \geq \frac{1}{q} + \frac{\varepsilon}{2}.$$

Proof. Obviously, if $Z_1 = Z_2$ we also have $\ell(Z_1) = \ell(Z_2)$ for every $\ell \in \mathscr{L}_m^*$. Using Lemma 9 we obtain

$$\begin{aligned}\Pr[L(Z_1) = L(Z_2) \mid Z_1 \neq \perp] &= \underbrace{\frac{1}{q^m} + \varepsilon}_{\Pr[Z_1 = Z_2 \mid Z_1 \neq \perp]} + \underbrace{\left(\frac{q^m - 1}{q^m} - \varepsilon\right)}_{\Pr[Z_1 \neq Z_2 \mid Z_1 \neq \perp]} \frac{q^{m-1} - 1}{q^m - 1} \\ &= \frac{1}{q^m} + \varepsilon + \left(\frac{1}{q} - \frac{1}{q^m}\right) - \varepsilon \frac{q^{m-1} - 1}{q^m - 1} \geq \frac{1}{q} + \frac{\varepsilon}{2}. \quad \square\end{aligned}$$

Next, we translate a uniform query r into a uniform pair $(\ell, y) \in \mathscr{L}^* \times \{0,1\}^k$, such that $\langle x, r\rangle = \ell(h(x,y))$. We will be able to use this by giving y to the oracle $\mathcal{O}$ which predicts $h(x,y)$ and then apply ℓ to get a prediction for $\langle x, r\rangle$. Since y is uniform we will know the advantage of the oracle in predicting $h(x,y)$, and since ℓ is uniform, we can apply Lemma 10.

Lemma 11. *Let $h : \mathbb{F}^n \times \mathbb{F}^k \to \mathbb{F}^m$ be a full-rank bilinear function. There exists an efficiently computable random mapping $G_h : \mathbb{F}^n \rightsquigarrow \mathbb{F}^k \times \mathscr{L}^*_m$, which, for a uniformly chosen input r outputs a uniform random pair (ℓ, y) such that $\ell(h(x,y)) = \langle x, r\rangle$ for every x.*

Proof. The algorithm implementing G_h first chooses an $\ell \in \mathscr{L}^*_m$ uniformly at random. For a fixed ℓ, let M be the matrix for which $\ell(h(x,y)) = x^T My$; note that $\mathrm{rank}(M) = n$. As a second step, the algorithm chooses y as a uniform random solution of $My = r$, and returns the pair (ℓ, y). For every fixed ℓ if r is uniformly distributed; the vector y will be uniformly distributed. Furthermore, $\ell(h(x,y)) = x^T My = x^T r = \langle x, r\rangle$. □

The next lemma proves the claimed conversion; i.e., given an oracle which predicts $h(x,y)$ we implement an oracle which predicts $\langle x, r\rangle$. For this, on input r the algorithm first gets a pair (ℓ, y) using Lemma 11. Then, it queries the given oracle $\mathcal{O}$ with y, applies ℓ to the output and returns the result.

Lemma 12. *Let $h : \mathbb{F}^n \times \mathbb{F}^k \to \mathbb{F}^m$ be a full-rank bilinear function. There is an efficient oracle algorithm A such that for any $\varepsilon > 0$, every $x \in \mathbb{F}^n$ and any oracle $\mathcal{O} : \mathbb{F}^k \rightsquigarrow \mathbb{F}^m$ which satisfies*

$$\Pr[\mathcal{O}(Y) = h(x,Y) \mid \mathcal{O}(Y) \neq \perp] \geq \frac{1}{q^m} + \varepsilon$$

algorithm $A^{\mathcal{O}}$ satisfies

$$\Pr[A^{\mathcal{O}}(R) = \langle x, R\rangle \mid A^{\mathcal{O}}(R) \neq \perp] \geq \frac{1}{q} + \frac{\varepsilon}{2}$$

and $\Pr[A^{\mathcal{O}}(R) \neq \perp] = \Pr[\mathcal{O}(Y) \neq \perp]$. *Algorithm A makes one oracle call to $\mathcal{O}$.*

Proof. Given a uniformly chosen r, the algorithm first evaluates the function $G_h(r)$ as guaranteed by Lemma 11, to get a uniform pair (ℓ, y) with $\ell(h(x,y)) = \langle x, r\rangle$. It then queries the oracle with y. In case the answer z is not $\perp$ it returns $\ell(z)$; otherwise it returns $\perp$.

Let x be fixed such that

$$\Pr[\mathcal{O}(Y) = h(x,Y) \mid \mathcal{O}(Y) \neq \perp] \geq \frac{1}{q^m} + \varepsilon.$$

Lemma 10 implies that

$$\Pr[L(\mathcal{O}(Y)) = L(h(x,Y)) \mid \mathcal{O}(Y) \neq \perp] \geq \frac{1}{q} + \frac{\varepsilon}{2}.$$

Since (ℓ, y) is uniformly distributed this together with $\ell(h(x,y)) = \langle x, r\rangle$ concludes the proof. □

Lemma 12 can be seen as a reduction of a code to another one, in the sense that given a noisy codeword of one code we can generate a noisy codeword of a related code such that the Hamming distances to codewords are related in some sense. The proofs of Theorems 7 and 8 are now obvious.

Proof (of Theorems 7 and 8). Use Lemma 12, and apply Theorems 5 and 6, respectively. □

3.2 Construction of Full-Rank Functions

As mentioned before, a list-decodable code can be used to obtain a hard-core function, which means that a family of full-rank bilinear functions can be used as a hard-core function. This is stated in the following proposition (a more exact version will be given in Theorem 25, Section 5).

Proposition 13. *Any efficiently computable family of full-rank bilinear functions* $h : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^m$*, where* $k \in \mathrm{poly}(n)$ *and* $m \in O(\log n)$ *is a strong hard-core function.*

The proposition implies that in order to give a hard-core function it is sufficient to construct a full-rank bilinear function family. In this section, we will present constructions which appear in the literature as hard-core functions, and show that they satisfy $\mathrm{rank}(\ell \circ h) = n$ for every $\ell \neq \mathbf{0}$.

As usual in the context of hard-core functions, we will explain the constructions for vector spaces over $\{0,1\}$. However, all constructions immediately generalize to vector spaces over any finite field.

Recall that any bilinear function $h : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^m$ can be described by a sequence $M_1, \ldots, M_m$ of $n \times k$ matrices over GF(2) as $h(x,r) = \big(x^T M_1 r, \ldots, x^T M_m r\big)$. It follows that for every ℓ there exists a non-empty subset $I \subseteq \{1, \ldots, m\}$ such that the function $\ell \circ h$ can be written as $\ell(h(x,r)) = x^T \big(\sum_{i \in I} M_i\big) r$.

In order to get a full-rank bilinear function it is therefore sufficient to give matrices $M_1, \ldots, M_m$ which satisfy

$$\mathrm{rank}\Big(\sum_{i \in I} M_i\Big) = n \qquad \text{for every } I \neq \emptyset. \tag{2}$$

Example 14. In [Lub96] it is shown that $O(\log n)$ independent inner product bits give a hard-core function. This function $h : \{0,1\}^n \times \{0,1\}^{nm} \to \{0,1\}^m$ is defined by matrices $M_1, \ldots, M_m$ such that M_i consists of all zeros, except that from column $n(i-1)+1$ to ni it contains a $n \times n$ identity matrix. Here it is obvious that (2) is satisfied.

Example 15. In order to keep the dimension k small, one can obtain a full-rank bilinear function $h : \{0,1\}^n \times \{0,1\}^{n+m-1} \to \{0,1\}^m$ with the construction given in [Gol01] and [GL89]. There, M_i is a matrix of size $n \times (n+m-1)$ which contains only zeros with the exception of an $n \times n$ identity matrix starting at column i. Again, it is obvious that (2) holds.

Note that since $\mathrm{rank}(\ell \circ h)$ cannot be larger than k for any ℓ, it is necessary to have $k \geq n$. If m is small enough this is indeed sufficient:

Theorem 16. *Let vector spaces $\{0,1\}^n$, $\{0,1\}^k$ and $\{0,1\}^m$ over $\{0,1\}$ be given. If $n \leq k$ and $m \leq k$, then there exists a full-rank bilinear function $h : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^m$.*

Proof. We first note that it is sufficient to give a full-rank bilinear function $\{0,1\}^k \times \{0,1\}^k \to \{0,1\}^k$ for every k, since one can first obtain a bilinear function $\{0,1\}^k \times \{0,1\}^k \to \{0,1\}^m$ by ignoring some of the output coordinates, and in a second step one can get a full-rank bilinear function $\{0,1\}^n \times \{0,1\}^k \to \{0,1\}^m$ by setting some of the inputs to the first arguments to zero.

To construct a full-rank bilinear function $h : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}^k$ we observe that the finite field $\mathrm{GF}(2^k)$ is a vector space over $\{0,1\}$ of dimension k, and for every $x \in \mathrm{GF}(2^k)$ the map $g_x(r) = x \cdot r$ is linear. Let $z_1, \ldots, z_k$ be a basis of $\mathrm{GF}(2^k)$ and let M_i be the matrix which describes the linear mapping g_{z_i} in this basis. Since for any $I \neq \emptyset$ the matrix $\sum_{i \in I} M_i$ describes the linear mapping g_z for some non-zero $z \in \mathrm{GF}(2^k)$, this map is invertible and thus has rank k. $\square$

The bilinear function used in this proof is strongly related to the hard-core function given at the end of [GRS00], and indeed the function given there also satisfies the rank condition needed for Theorem 8.[6]

4 General Bilinear Functions

In this section we give a list-decoding algorithm for every (possibly non full-rank) bilinear function. Using the same technique as in Section 3.1 we prove the following analogue of Theorem 7 (recall that $\rho(h) = \mathrm{E}[q^{n-\mathrm{rank}(L \circ h)}]$).

Theorem 17. *Let $h : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^m$ be any bilinear function. After a preprocessing phase taking time $2^m \cdot \mathrm{poly}(k, n)$, the function $h(x, y)$ is (δ, ε)-list-decodable with list size $O(\frac{\rho(h)}{\delta\varepsilon^2})$ and an expected number $\Theta(n\frac{1}{\delta\varepsilon^2})$ of oracle calls. The algorithm needs $\delta\varepsilon^2$ as input.*

Note that $\Theta(\frac{n}{\delta\varepsilon^2})$ is the *expected* number of queries. For general finite fields Theorem 18 holds.

Theorem 18. *Let $h : \mathbb{F}^n \times \mathbb{F}^k \to \mathbb{F}^m$ be any bilinear function over $\mathbb{F}$. After a preprocessing phase taking time $q^m \cdot \mathrm{poly}(n, k)$, the function $h(x, y)$ is (δ, ε)-list-decodable with list size $\rho(h) \cdot \mathrm{poly}(n, k, \delta^{-1}\varepsilon^{-1})$ and an expected number of $\mathrm{poly}(n, k, \delta^{-1}\varepsilon^{-1})$ oracle calls. The list-decoding algorithm needs $\delta\varepsilon$ as input.*

[6] The functions are not identical, but if one considers the "cube" given by stacking the matrices for different linear maps ℓ, then the functions are obtained from each other by a rotation of this cube. It is possible to show that for any two cubes which are obtained by rotation from each other, the corresponding function satisfies the full-rank condition if and only if the same holds for the other cube.

As before we prove these theorems by converting a given oracle $\mathcal{O}$ which on input y predicts $h(x,y)$ to an oracle $\mathcal{O}'$ which on input r predicts $\langle x,r\rangle$. We use Lemma 10 again (and thus Lemma 9), but we modify Lemmas 11 and 12.

A problem is that for some r it may be impossible to choose a pair (ℓ,y) with $\ell(h(x,y)) = \langle x,r\rangle$ for every x. This will force our reduction to return $\perp$ on input r, since there is no way to get a reasonable guess for $\langle x,r\rangle$ from $\mathcal{O}$. Furthermore, the pair (ℓ,y) must be uniformly distributed which makes the conversion return $\perp$ more often. We get the following generalization of Lemma 11:

Lemma 19. *Let $h : \mathbb{F}^n \times \mathbb{F}^k \to \mathbb{F}^m$ be a bilinear function. There exists an efficiently computable mapping $G_h : \mathbb{F}^n \rightsquigarrow (\mathbb{F}^k \times \mathscr{L}_m^*) \cup \{\perp\}$ which, on uniformly distributed input r outputs $\perp$ with probability $1 - \frac{1}{\rho(h)}$, and otherwise a uniform random pair (ℓ,y), satisfying $\ell(h(x,y)) = \langle x,r\rangle$ for all x. The algorithm uses a precomputation with time complexity $q^m \cdot \mathrm{poly}(n,k)$.*

Proof. First, as a precomputation, for every $\ell \in \mathscr{L}_m^*$ the algorithm calculates $q^{\mathrm{rank}(\ell\circ h)}$, and stores it in such a way that later it is possible to efficiently draw an element $\ell \in \mathscr{L}_m^*$ with probability $q^{n-\mathrm{rank}(\ell\circ h)} / \hat{\rho}(h)$, where $\hat{\rho}(h) = \sum_{\ell\neq\mathbf{0}} q^{n-\mathrm{rank}(\ell\circ h)} = (q^m - 1)\,\rho(h)$.

After the precomputation, on input r, the algorithm chooses ℓ according to this probability distribution and obtains the matrix M with $\ell(h(x,y)) = x^T M y$. If the system $My = r$ is solvable, it chooses a solution y uniformly at random and returns (ℓ,y); otherwise it returns $\perp$.

Note that the precomputation can obviously be done in time $q^m \cdot \mathrm{poly}(n,k)$ and every returned pair (ℓ,y) satisfies $\ell(h(x,y)) = \langle x,r\rangle$.

For a fixed ℓ and uniformly chosen r, the probability that there exists a y such that $My = r$ is $q^{\mathrm{rank}(M)-n} = q^{\mathrm{rank}(\ell\circ h)-n}$. Furthermore, conditioned on the event that the system above is solvable, every vector y has the same probability. This implies that the probability that a fixed pair (ℓ,y) is returned is

$$\Pr[G_h(R) = (\ell,y)] = \frac{q^{n-\mathrm{rank}(\ell\circ h)}}{\hat{\rho}(h)} \cdot q^{\mathrm{rank}(\ell\circ h)-n} \cdot \frac{1}{q^k} = \frac{1}{q^k\,\hat{\rho}(h)},$$

which is independent of the pair (ℓ,y). Summing over all possible pairs (ℓ,y) we get $\Pr[G_h(R) \neq \perp] = 1/\rho(h)$. □

We point out that the probability of G_h *not* returning $\perp$ cannot be made any higher. To see why, first note that a pair (ℓ,y) can only be the answer for one specific input r. Furthermore, there are $q^k\,\hat{\rho}(h)$ possible pairs (ℓ,y), which can only be output for $y = 0$; implying that every pair can occur with probability at most $q^{-k}\,\hat{\rho}^{-1}(h)$.

Along the same line of reasoning as in Section 3, we can now prove the generalized version of Lemma 12.

Lemma 20. *Let $h : \mathbb{F}^n \times \mathbb{F}^k \to \mathbb{F}^m$ be a bilinear function. There is an efficient oracle algorithm A such that for any $\varepsilon > 0$, every $x \in \mathbb{F}^n$ and any oracle $\mathcal{O} : \mathbb{F}^k \rightsquigarrow \mathbb{F}^m$ which satisfies*

$$\Pr[\mathcal{O}(Y) = h(x,Y) \mid \mathcal{O}(Y) \neq \perp] \geq \frac{1}{q^m} + \varepsilon,$$

algorithm $A^{\mathcal{O}}$ satisfies

$$\Pr[A^{\mathcal{O}}(R) = \langle x, R\rangle \mid A^{\mathcal{O}}(R) \neq \perp] \geq \frac{1}{q} + \frac{\varepsilon}{2}$$

and $\Pr[A^{\mathcal{O}}(R) \neq \perp] = \frac{1}{\rho(h)} \Pr[\mathcal{O}(R) \neq \perp]$. *The algorithm makes one query to $\mathcal{O}$ with probability $\frac{1}{\rho(h)}$. It uses a preprocessing phase with time complexity* $q^m \cdot \mathrm{poly}(n, k)$.

Proof. The preprocessing is the one needed for G_h of Lemma 19. On input r, the algorithm first uses G_h to obtain either a pair (ℓ, y) or $\perp$. In the second case, the algorithm returns $\perp$ and does not make an oracle query; this happens with probability $1 - \frac{1}{\rho(h)}$. If a pair (ℓ, y) is returned, the algorithm makes one query $z = \mathcal{O}(y)$. If $z \neq \perp$ the algorithm returns $\ell(z)$, otherwise it returns $\perp$.

We fix ε and x such that $\Pr[\mathcal{O}(Y) = h(x, Y) \mid \mathcal{O}(Y) \neq \perp] \geq \frac{1}{q^m} + \varepsilon$. Lemma 10 implies that $\Pr[L(\mathcal{O}(Y)) = L(h(x, Y)) \mid \mathcal{O}(Y) \neq \perp] \geq \frac{1}{q} + \frac{\varepsilon}{2}$. Conditioned on the event that A makes a query to $\mathcal{O}$ the pair (ℓ, y) is uniformly distributed and satisfies $\ell(h(x, y)) = \langle x, r\rangle$. Also, when A does not make a query to $\mathcal{O}$ it returns $\perp$. This implies

$$\Pr[L(\mathcal{O}(Y)) = L(h(x, Y)) \mid \mathcal{O}(Y) \neq \perp] = \Pr[A^{\mathcal{O}}(R) = \langle x, R\rangle \mid A^{\mathcal{O}}(R) \neq \perp] .$$

Finally, we see that A does not return $\perp$ if both G_h of Lemma 19 and $\mathcal{O}$ do not return $\perp$, which happens with probability $\frac{1}{\rho(h)} \Pr[\mathcal{O}(Y) \neq \perp]$. □

Using this conversion, the proofs of Theorems 17 and 18 are now straightforward.

Proof (of Theorems 17 and 18). Use Lemma 20 and apply Theorems 5 and 6, respectively.

5 Implications for Hard-Core Functions

The results of the previous sections have implications in cryptography, namely for one-way functions. In particular, under a reasonable complexity-theoretic assumption the results allow us to classify basically every bilinear function family $h : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^m$ according to whether it is a strong hard-core function or not.

We formulate our results in the context of uniform algorithms, but they immediately generalize to a non-uniform context.

5.1 Weak vs. Strong Hard-Core Functions

In general, it is easier to construct weak hard-core functions than to construct strong ones. For example the identity function $h(x) = x$ is a weak hard-core function for any one-way function f (predicting x given $f(x)$ is the same as inverting f), but not a strong hard-core function (given $f(x)$ it is easy to distinguish x from a random value).

For small output values the two notions are equivalent: every weak hard-core function $h : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^m$ for $m \in O(\log n)$ is also a strong one. This follows from the fact that any distinguisher for such a function can be converted to a predictor. More concretely, assume that an oracle $\mathcal{O}$ has advantage ε in distinguishing $h(x,y)$ from a random value. It is well known that one can get a predictor with advantage $2^{-m}\varepsilon$ from $\mathcal{O}$ (see for example [Lub96]). The following lemma improves this fact by following the idea of Hast that, in cryptographic applications, a distinguisher often comes from an algorithm which tries to break a scheme; if it succeeds then it is almost certain that the input was not random. This can be used to obtain a predictor with lower rate but higher advantage. In the following lemma we use this idea since the probability p_0 that a distinguisher answers 1 on random input can be very small. By replacing $\perp$ with a uniform random output one obtains the well-known version mentioned above.

Lemma 21. *There exists a randomized oracle algorithm A such that for any $z \in \{0,1\}^m$, oracle $\mathcal{O}$ with*

$$p_0 := \Pr[\mathcal{O}(U^m) = 1]$$

and ε defined by

$$p_0(1+\varepsilon) = \Pr[\mathcal{O}(z) = 1],$$

algorithm A queries $\mathcal{O}$ once and outputs a value from $\{0,1\}^m \cup \{\perp\}$ such that

$$\Pr[A^{\mathcal{O}} \neq \perp] = p_0$$

and

$$\Pr[A^{\mathcal{O}} = z \mid A^{\mathcal{O}} \neq \perp] = \frac{1}{2^m} + 2^{-m}\varepsilon.$$

Proof. Algorithm A chooses a uniform random value $z' \in \{0,1\}^m$. It then queries $\mathcal{O}(z')$ and outputs z' if the oracle outputs 1. Otherwise, it outputs $\perp$.

The probability that A outputs $\perp$ is $1-p_0$. The probability that A outputs z is $\frac{1}{2^m}(p_0(1+\varepsilon))$ and thus the probability that A outputs z conditioned on the event that it does not output $\perp$ is $\frac{1+\varepsilon}{2^m}$. □

As a corollary we obtain the following result:

Corollary 22. *Let $h : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^m$ be a weak hard-core function and $m \in O(\log n)$. Then, h is a strong hard-core function.*

Proof. Assume that h is not a strong hard-core function. Then, there exists an algorithm A which on input $(f(x), r)$ can distinguish $h(x,r)$ from a uniform random string with non-negligible advantage ε. According to Lemma 21 we can use this algorithm to obtain an algorithm which predicts the same string with success probability at least $\frac{1}{2^m} + \frac{\varepsilon}{2^m}$, and thus h is not a weak hard-core function. □

5.2 List-Decodable Codes and Weak Hard-Core Functions

Every list-decodable code can be used as a weak hard-core function. The idea to prove this is to assume that the function h is *not* a weak hard-core function, and to use the algorithm A which predicts $h(x,r)$ given $f(x)$ and r together with the list-decoding algorithm to find a list which contains x with probability at least $1/2$. Applying f to each element of the list and comparing the input we are guaranteed to find a preimage of $f(x)$ with high probability.

In our case, we would like to use the algorithm guaranteed in Theorem 17. This algorithm requires to know the product $\delta\varepsilon^2$, and works as long as the correct value is *at least as large* as the value given to the algorithm.

Note that the value of x is fixed during a run of algorithm A. Consequently such an algorithm can only be successful if the rate δ_x and advantage ε_x for a fixed x is large enough. However, typically only the rate δ and advantage ε averaged over all x is guaranteed to have a certain value. In order to show that this is sufficient we first prove that $\mathrm{E}[\delta_X\varepsilon_X^2] \geq \delta\varepsilon^2$. In the following lemma, it is useful to think of Z as an indicator variable which is 1 if the predictor guesses correctly; 0 on a wrong guess and $\perp$ if the predictor refuses to produce a guess. The random variable X corresponds to the value of x.

Lemma 23. *Let X be a uniformly distributed random variable over $\mathcal{X}$ and let Z be some random variable in $\{0,1,\perp\}$. Let $\delta := \Pr[Z \neq \perp]$ and $\delta_x := \Pr[Z \neq \perp \mid X = x]$. Fix any constant c and let $\varepsilon := \Pr[Z = 1 \mid Z \neq \perp] - c$ and $\varepsilon_x := \Pr[Z = 1 \mid X = x \wedge Z \neq \perp] - c$. Then,*

$$\mathrm{E}[\delta_X\varepsilon_X^2] \geq \delta\varepsilon^2.$$

Proof. First we observe that $\delta = \frac{1}{|\mathcal{X}|}\sum_{x\in\mathcal{X}}\delta_x$. Furthermore we have

$$\begin{aligned} c+\varepsilon &= \frac{\Pr[Z=1]}{\Pr[Z\neq\perp]} = \frac{\frac{1}{|\mathcal{X}|}\sum_{x\in\mathcal{X}}\Pr[Z=1 \mid X=x]}{\frac{1}{|\mathcal{X}|}\sum_{x\in\mathcal{X}}\delta_x} \\ &= \frac{\sum_{x\in\mathcal{X}}\delta_x(c+\varepsilon_x)}{\sum_{x\in\mathcal{X}}\delta_x} = c + \frac{\sum_{x\in\mathcal{X}}\delta_x\varepsilon_x}{\sum_{x\in\mathcal{X}}\delta_x} \end{aligned}$$

and thus $\varepsilon = \left(\sum_x \delta_x\varepsilon_x\right)/\left(\sum_x\delta_x\right)$. To show that

$$\mathrm{E}[\delta_X\varepsilon_X^2] = \frac{1}{|\mathcal{X}|}\sum_{x\in\mathcal{X}}\delta_x\varepsilon_x^2 \geq \Big(\frac{1}{|\mathcal{X}|}\sum_{x\in\mathcal{X}}\delta_x\Big)\Big(\frac{\sum_{x\in\mathcal{X}}\delta_x\varepsilon_x}{\sum_{x\in\mathcal{X}}\delta_x}\Big)^2 = \delta\varepsilon^2,$$

we note that this is equivalent to

$$\Big(\sum_{x\in\mathcal{X}}\delta_x\Big)\Big(\sum_{x\in\mathcal{X}}\delta_x\varepsilon_x^2\Big) \geq \Big(\sum_{x\in\mathcal{X}}\delta_x\varepsilon_x\Big)^2,$$

which follows directly from the Cauchy-Schwarz inequality. □

We now show how to use the list-decoding algorithm to invert a function f. The following lemma is usually used when f is a one-way function, $m \in O(\log n)$ and $\rho(h) \in \mathrm{poly}(n)$, in which case it states that h is a weak hard-core function.

Lemma 24. *Let $f : \{0,1\}^n \to \{0,1\}^p$ be any efficiently computable function family. Let $h : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^m$ be any efficiently computable bilinear function with $k \in \mathrm{poly}(n)$. There exists an oracle algorithm A such that for any $\mathcal{O} : \{0,1\}^p \times \{0,1\}^k \rightsquigarrow \{0,1\}^m \cup \{\perp\}$ which satisfies $\Pr[\mathcal{O}(f(X),Y) \neq \perp] = \delta$, and $\Pr\big[\mathcal{O}(f(X),Y) = h(X,Y) \mid \mathcal{O}(f(X),Y) \neq \perp\big] = \frac{1}{2^m} + \varepsilon$, algorithm $A^{\mathcal{O}}$ is running in time $\frac{\rho(h)}{\delta\varepsilon^2} \cdot \mathrm{poly}(n) + 2^m \cdot \mathrm{poly}(n)$ and satisfies*

$$\Pr\big[f\big(A^{\mathcal{O}}(f(X))\big) = f(X)\big] \geq \frac{\delta\varepsilon^2}{4},$$

while making an expected number $\Theta(n\frac{1}{\delta\varepsilon^2})$ of oracle calls to $\mathcal{O}$. Algorithm A needs $\delta\varepsilon^2$ as an input.

If h is a full-rank bilinear function, the term $2^m \cdot \mathrm{poly}(n)$ in the running time can be omitted.

Proof. For any fixed $x \in \{0,1\}^n$, let $\delta_x := \Pr[\mathcal{O}(f(x),Y) \neq \perp]$ and $\varepsilon_x := \Pr[\mathcal{O}(f(x),Y) = h(x,Y) \mid \mathcal{O}(f(x),Y) \neq \perp] - \frac{1}{2^m}$. Using Lemma 23 we obtain $\mathrm{E}[\delta_X\varepsilon_X^2] \geq \delta\varepsilon^2$. Since $0 \leq \delta_x\varepsilon_x^2 \leq 1$ for any x, we can apply Markov's inequality to obtain $\Pr[\delta_X\varepsilon_X^2 > \frac{1}{2}\delta\varepsilon^2] > \frac{1}{2}\delta\varepsilon^2$. A run of the algorithm guaranteed in Theorem 17 with input $\delta\varepsilon^2/2$ thus gives a set Λ of size at most $O(\frac{\rho(h)2^n}{\delta\varepsilon^2})$ containing x with probability at least $\frac{1}{4}\delta\varepsilon^2$, while doing an expected number $\Theta(n\frac{1}{\delta\varepsilon^2})$ of oracle calls. Applying f to each $x \in \Lambda$ and testing if it is correct yields the claimed result. □

5.3 Bilinear Hard-Core Functions

Lemma 21 converts a distinguisher to a predictor, while Lemma 24 uses a predictor to invert a function. Combining these two lemmas gives the following theorem:

Theorem 25. *Let $f : \{0,1\}^n \to \{0,1\}^p$ be any efficiently computable function. Let $h : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^m$ be any efficiently computable bilinear function with $k \in \mathrm{poly}(n)$. There exists an oracle algorithm A such that for $\varepsilon, \delta > 0$ and any $\mathcal{O} : \{0,1\}^p \times \{0,1\}^k \rightsquigarrow \{0,1\}$ which satisfies*

$$\Pr\big[\mathcal{O}\big(f(X), R, h(X,R)\big)\big] = \delta, \text{ and } \Pr\big[\mathcal{O}\big(f(X), R, U^m\big)\big] = \delta(1+\varepsilon),$$

algorithm A satisfies

$$\Pr\big[f(A^{\mathcal{O}}(f(X))) = f(X)\big] \geq \frac{\delta\varepsilon^2}{4 \cdot 2^{2m}}$$

and makes an expected number of

$$\kappa = \Theta\Big(n\frac{2^{2m}}{\delta\varepsilon^2}\Big)$$

oracle queries to $\mathcal{O}$. Algorithm A runs in time $\frac{\rho(h)2^{2m}}{\delta\varepsilon^2}\operatorname{poly}(n) + 2^m \operatorname{poly}(n)$ and needs $\delta\varepsilon^2$ as input.

Proof. Combine Lemma 21 with Lemma 24. $\square$

This theorem implies that any bilinear function $h : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^m$ with $m \in O(\log n)$ and $\rho(h) \in \operatorname{poly}(n)$ can be used as a hard-core function.

Corollary 26. *Let $h : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^m$ be a bilinear function with $m \in O(\log n)$ and $\rho(h) \in \operatorname{poly}(n)$. Then h is a strong hard-core function.*

Proof. Assume otherwise and use Theorem 25 to arrive at a contradiction. $\square$

5.4 Bilinear Functions not Suitable as Hard-Core Functions

In this section we also consider bilinear functions $h : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^m$ for which $m \notin O(\log n)$ or $\rho(h) \notin \operatorname{poly}(n)$. One can show that $m \notin O(\log n)$ implies the existence of a function $\widetilde{m} \in \omega(\log n)$ which is infinitely often smaller than m. Analogously, $\rho(h) \notin \operatorname{poly}(n)$ implies the existence of a function $\tilde{\rho}$ which is strictly superpolynomial (i.e., $\log(\tilde{\rho}) \in \omega(\log n)$) and infinitely often smaller than $\rho(h)$. We say that a hard-core function is *regular* if $m \in O(\log n)$ or a polynomial time computable function $\widetilde{m}$ as above exists; and $\rho \in \operatorname{poly}(n)$ or a polynomial time computable $\tilde{\rho}$ as above exists.

We show that any regular bilinear function not satisfying the conditions of Corollary 26 is not a hard-core function if some reasonable complexity-theoretic assumption holds, namely the existence of a one-way permutation with exponential security.

Definition 27 (Very strong one-way permutation).[7] *A family of polynomial time computable functions $f : \{0,1\}^n \to \{0,1\}^n$ is a* very strong one-way permutation *if there exists a constant $c > 0$, such that for every algorithm A with running time at most 2^{cn}, the inverting probability* $\Pr[f(A(f(X))) = f(X)]$ *is at most 2^{-cn} for all but finitely many n.*

Proving that no such functions exist would be a breakthrough in complexity theory. Furthermore, Gennaro and Trevisan show in [GT00] that in relativized worlds such functions exist, and thus our results exclude a relativizing hard-core result for any bilinear function which does not satisfy the conditions of Corollary 26 unconditionally.

As a first step, we show that it is impossible to use a bilinear function to extract $\omega(\log n)$ hard bits from x. Such a lemma was already hinted at in [GL89].

Lemma 28. *Let $h : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^m$ be a regular bilinear function with $m \notin O(\log n)$. If a very strong one-way permutation exists, then h is not a strong hard-core function.*

---

[7] We use permutations for the sake of simplicity. It is easy to see that arbitrary one-way functions with exponential security suffice to prove Theorem 30.

Proof. Since $m \notin O(\log n)$ and h is regular, there exists a polynomial-time computable function $\widetilde{m} \in \omega(\log n)$ with $\widetilde{m}(n) < m(n)$ for infinitely many n.

We define a one-way function $f : \{0,1\}^n \to \{0,1\}^n$ for which it is easy to give a distinguisher for $h(x, r)$. For this purpose, let $g : \{0,1\}^{\widetilde{m}/2} \to \{0,1\}^{\widetilde{m}/2}$ be a very strong one-way permutation. On input $x \in \{0,1\}^n$, split the input x into two parts, $x_1 \in \{0,1\}^{\widetilde{m}/2}$ and $x_2 \in \{0,1\}^{n-\widetilde{m}/2}$. The output of f is then $g(x_1)$ concatenated with x_2. We see that f is a one-way function, since an algorithm A which inverts f in poly(n)-time with non-negligible success probability can be used to invert g in time $2^{o(\widetilde{m}(n))}$ with probability $2^{-o(\widetilde{m}(n))}$ for infinitely many n.

Furthermore, for any n with $\widetilde{m}(n) < m(n)$ it is easy to distinguish $h(x, r)$ from a random string, given $f(x)$ and r. First, we find x_2 from $f(x)$. Since $h(x, r) = x^T M r$ we see that for fixed x_2 and r only a subspace of dimension at most $\widetilde{m}/2$ is possible as output value for $h(x, r)$. Also, it is easy to check whether a given value is within this subspace or not. Since a random value will be in the subspace with probability at most $2^{-\widetilde{m}/2}$, h cannot be a hard-core function. $\square$

Using basically the same technique, we can now show that only functions with nearly full rank can be used as hard-core functions.

Lemma 29. *Let $h : \{0,1\}^n \times \{0,1\}^p \to \{0,1\}^m$ be a regular bilinear function with $m \in O(\log n)$ and $\rho(h) \notin \mathrm{poly}(n)$. If a very strong one-way permutation exists, then h is not a strong hard-core function.*

Proof. Since h is regular and $\rho(h) \notin \mathrm{poly}(n)$, there exists a function $\tilde{\rho}$ such that $\log(\tilde{\rho}) \in \omega(\log n)$ and $\tilde{\rho}(n) < \rho(h)(n)$ for infinitely many n.

As in the proof of Lemma 28, we construct a one-way function $f : \{0,1\}^n \to \{0,1\}^n$ by embedding a preimage of size $\{0,1\}^{\log(\tilde{\rho}(n))/2}$ to a very strong one-way permutation g. Consider an n for which $\tilde{\rho}(n) < \rho(h)(n)$. For such an n it is easy to find a linear map to embed the preimage to g such that for some $\ell \in \mathscr{L}_m^*$ the value of $\ell(h(x, y))$ does not depend on the input to g. As in the proof of Lemma 28 it follows immediately that f is a one-way function, and since $\ell(h(x, y))$ only depends on a part of x which can be found by a linear transformation of the output, h cannot be a hard-core function. $\square$

Together, this implies the following theorem.

Theorem 30. *Let $h : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^m$ be a regular bilinear function, and assume the existence of a very strong one-way permutation. Then h is a strong hard-core function if and only if $\rho(h) \in \mathrm{poly}(n)$ and $m \in O(\log n)$.*

Proof. If $\rho(h) \in \mathrm{poly}(n)$ and $m \in O(\log n)$, then h is a hard-core function according to Corollary 26. If $m \in O(\log n)$ and $\rho(h) \notin \mathrm{poly}(n)$, then h is not a hard-core function according to Lemma 29. If $m \notin O(\log n)$ then h is not a hard-core function according to Lemma 28. $\square$

Acknowledgments

We would like to thank Gustav Hast and Johan Håstad for helpful discussions. This research was supported by the Swiss National Science Foundation, project no. 2000-066716.01/1.

References

[ACGS88] Werner Alexi, Benny Chor, Oded Golreich, and Claus P. Schnorr. RSA and Rabin functions: Certain parts are as hard as the whole. *Siam Journal on Computation*, 17(2):194–209, 1988.

[AGS03] Adi Akavia, Shafi Goldwasser, and Samuel Safra. Proving hard-core predicates using list decoding. In *The 44th Annual Symposium on Foundations of Computer Science*, pages 146–157, 2003.

[BM84] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *Siam Journal on Computation*, 13(4):850–864, 1984.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 25–32, 1989.

[Gol01] Oded Goldreich. *Basic Tools*. Foundations of Cryptography. Cambridge University Press, first edition, 2001. ISBN 0-521-79172-3.

[GRS00] Oded Goldreich, Ronitt Rubinfeld, and Madhu Sudan. Learning polynomials with queries: The highly noisy case. *Siam Journal on Discrete Mathematics*, 13(4):535–570, 2000.

[GT00] Rosario Gennaro and Luca Trevisan. Lower bounds on the efficiency of generic cryptographic constructions. In *The 41st Annual Symposium on Foundations of Computer Science*, pages 305–313, 2000.

[Has03] Gustav Hast. Nearly one-sided tests and the Goldreich-Levin predicate. In Eli Biham, editor, *Advances in Cryptology — EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 195–210, 2003. Extended version to appear in *Journal of Cryptology*.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *Siam Journal on Computation*, 28(4):1364–1396, 1999.

[Lev87] Leonid A. Levin. One-way functions and pseudorandom generators. *Combinatorica*, 7(4):357–363, 1987.

[Lub96] Michael Luby. *Pseudorandomness and Cryptographic Applications*. Princeton University Press, first edition, 1996. ISBN 0-691-02546-0.

[Näs95] Mats Näslund. Universal hash functions & hard core bits. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *Advances in Cryptology — EUROCRYPT '95*, volume 921 of *Lecture Notes in Computer Science*, pages 356–366, 1995.

[Näs96] Mats Näslund. All bits in $ax + b \bmod p$ are hard. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 114–128, 1996. Extended Abstract.

[STV01] Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the XOR lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001.

[Sud00] Madhu Sudan. List decoding: Algorithms and applications. *SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory)*, 31(1):16–27, 2000.

[Yao82] Andrew C. Yao. Theory and applications of trapdoor functions (extended abstract). In *The 23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, 1982.