Multilinear Pseudorandom Functions

Aloni Cohen and Justin Holmgren

MIT

Abstract. We define the new notion of a *multilinear* pseudorandom function (PRF), and give a construction with a proof of security assuming the hardness of the decisional Diffie-Hellman problem. A direct application of our construction yields (non-multilinear) PRFs with aggregate security from the same assumption, resolving an open question in [CGV15]. Additionally, multilinear PRFs give a new way of viewing existing algebraic PRF constructions: our main theorem implies they too satisfy aggregate security.

1 Introduction

Pseudorandom functions (PRFs) are of fundamental importance in modern cryptography. A PRF is efficiently computable and succinctly described, but is indistinguishable from a random function. But a random functions are too unstructured for many applications, and as a result many specialized pseudorandom functions with more structure have emerged. Goldreich, Goldwasser, and Nussboim [GGN03] define a general notion of pseudo-implementing huge random objects with extra structure. In this paper, we define and construct *multilinear* pseudorandom functions assuming the decisional Diffie-Hellman assumption (DDH), and we show applications to prior work. Before presenting an informal definition, we discuss the motivating applications.

The recent work of Cohen, Goldwasser, and Vaikuntanathan [CGV15] introduced the notion of an aggregate pseudorandom function family $\mathcal{F}$ with extra efficiency and security properties. First, the key K for a PRF f enables efficient computation of $\mathsf{Agg}_f(S) = \sum_{x \in S} f(x)$ for some class of succinctly described, but possibly exponentially large, sets S. Second, no efficient algorithm can distinguish oracle access to $f(\cdot)$ and $\mathsf{Agg}_f(\cdot)$ from oracle access to $g(\cdot)$ and $\mathsf{Agg}_g(\cdot)$, where g is a truly random function. The main constructions of [CGV15] were proven secure assuming the subexponential hardness of DDH.

A different line of work studies a notion of algebraic pseudorandom functions by Benabbas, Gennaro, and Vahlis [BGV11]. This notion is incomparable to aggregate PRFs: algebraic PRFs generalize efficient aggregation, but provide no security guarantees when an adversary has an aggregation oracle. Algebraic PRFs have a number of applications in verifiable computation and multiparty computation [BGV11,Haz15]. Because of their restricted security, algebraic PRFs have thus far only been considered for polynomially-sized domains – security over large domains would require subexponential hardness of DDH.

In both works, the reliance on subexponential hardness of DDH (or the small-domain restriction) is unsatisfying. Subexponential hardness is a significantly stronger assumption, and is, for example, false in $\mathbb{Z}_p^*$. Additionally, security reductions from subexponential hardness assumptions necessitate larger security parameters and thus lose efficiency.

We define a multilinear PRF family as a family of functions $\{\mathcal{F} : V_1 \times \cdots \times V_n \to Y\}$ mapping a product of vector spaces V_i to another vector space Y, in which a random function from the family is indistinguishable from a random *multilinear* function with the same domain and codomain. One case of particular interest is when each V_i is $\mathbf{F}_p^2$. Then any multilinear function from $V_1 \times \cdots \times V_n \to W$ is defined by 2^n values, which we think of as inducing a PRF mapping $\{0,1\}^n \to Y$. Multilinearity allows us to efficiently compute specific weighted sums of exponentially many PRF values from the above works. This encompasses "hypercube" aggregation from [CGV15] and the closed-form efficiency requirements of [BGV11].

2 Preliminaries

Notation For a set S, we use $x \leftarrow S$ to mean that x is sampled uniformly at random from S. We denote the finite field of order p by $\mathbf{F}_p$. All vectors $\mathbf{v}$ are column vectors, and $\mathbf{v}^t$ denotes the transpose.

2.1 Linear Maps

Given a vector spaces V and W over a field $\mathbb{F}$, we say that a map $T : V \to W$ is *linear* if $T(c_1\mathbf{v}_1 + c_2\mathbf{v}_2) = c_1T(\mathbf{v}_1) + c_2T(\mathbf{v}_2)$ is a valid identity for all vectors $\mathbf{v}_1$, $\mathbf{v}_2$ in V and scalars c_1, c_2 in $\mathbb{F}$. We say that a map $T : V_1 \times \cdots \times V_n \to W$ is *multilinear* if it is linear in each component.

When V and W have finite dimensions d_V and d_W (which is the only case we consider in this paper), a linear map from V to W can be represented by a matrix in $\mathbb{F}^{d_W \times d_V}$. This representation depends on the choice of bases for V and W. When $\mathbb{F}$ is a finite field (also the only case we consider), a *random* linear map from $V \to W$ can be sampled by picking an arbitrary basis for V and W and sampling a uniformly random matrix from $\mathbb{F}^{d_W \times d_V}$. The set of all linear maps from V to W will be denoted as W^V.

2.2 Tensor Products of Vector Spaces

Given vector spaces V and W of dimensions d_V and d_W over $\mathbb{F}$, the tensor product $V \otimes W$ is defined as a $d_V d_W$-dimensional vector space over $\mathbb{F}$. For any $v \in V$ and $w \in W$, their tensor product $v \otimes w \in V \otimes W$ can be defined by the following laws:

1. $(v_1 + v_2) \otimes w = v_1 \otimes w + v_2 \otimes w$
2. $v \otimes (w_1 + w_2) = v \otimes w_1 + v \otimes w_2$

3. For any $c \in \mathbb{F}$, $(cv) \otimes w = v \otimes (cw) = c(v \otimes w)$.

If V has a basis $v_1, \ldots, v_{d_V}$, and W has a basis $w_1, \ldots, w_{d_W}$, there is a natural basis for $V \otimes W$, namely $\{v_i \otimes w_j\}_{i \in [d_V], j \in [d_W]}$. Expanding $v = \sum_{i \in [d_V]} a_i v_i$ and $w = \sum_{j \in [d_W]} b_j w_j$ in the respective bases of V and W, and applying the above laws yields $v \otimes w = \sum_{i \in [d_V], j \in [d_W]} a_i b_j (v_i \otimes w_j)$. A *simple* tensor is defined as one which can be written as $v \otimes w$.

One can repeat the tensor product operation to obtain a space $V_1 \otimes \cdots \otimes V_n$ for any n vector spaces, with simple tensors of the form $v_1 \otimes \cdots \otimes v_n$. Vectors in such a space are sometimes called tensors, and the vector spaces in which they reside are called tensor spaces.

It is easy to observe that the mapping $\phi : V_1 \times \cdots \times V_n \to V_1 \otimes \cdots \otimes V_n$ given by $\phi(v_1, \ldots, v_n) = v_1 \otimes \cdots \otimes v_n$ is multilinear. In fact, this map is in some sense the most general multilinear map on $V_1 \times \cdots V_n$. Given any other vector space Z and a multilinear map $h : V_1 \times \cdots V_n \to Z$, there exists a unique linear map $f_h : V_1 \otimes \cdots \otimes V_n \to Z$ such that $h = f_h \circ \phi$. As a result, multilinear functions mapping $V_1 \times \cdots \times V_n \to Z$ naturally correspond to linear functions mapping $V_1 \otimes \cdots \otimes V_n \to Z$, and vice versa. This is known as the universal property of a tensor product space. This correspondence will be essential to proving correctness of Algorithm 2 and thereby the main theorem of this work.

Abusing our notation for the set of linear maps, we will write $Y^{V_1 \otimes \cdots \otimes V_n}$ to denote the set of all multilinear functions mapping $V_1 \times \cdots \times V_n$ into Y.

SpanSearch solver for simple tensors We will also need an algorithm which can solve the following problem: Suppose we are given $m + 1$ simple tensors $\mathbf{u}^0, \ldots, \mathbf{u}^m$ in $V_1 \otimes \cdots \otimes V_n$, with each $\mathbf{u}^i$ given in the form $\mathbf{v}_1^i \otimes \cdots \otimes \mathbf{v}_n^i$. Can we find coefficients $c_1, \ldots, c_m \in \mathbb{F}$ such that $\mathbf{u}^0 = \sum_{i=1}^m c_i \mathbf{u}^i$? Or is $\mathbf{u}^0$ independent of $\mathbf{u}^1, \ldots, \mathbf{u}^m$? The standard linear algebra algorithm of Gaussian elimination takes time which is $\mathsf{poly}(\prod_i \dim(V_i))$, which is exponential in the problem description length n due to the simple tensors' succinct representation.

[BW04] gives a deterministic polynomial-time algorithm solving this problem, which we will use in our security proof. For completeness, we reproduce a version of their simpler randomized algorithm in Section A.

2.3 Decisional Diffie-Hellman Assumption

We define an adversary's *advantage* in distinguishing distributions:

Definition 1. *We say that a probabilistic algorithm $\mathcal{A}$ has advantage $|\epsilon|$ in distinguishing distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ if*

$$\Pr\left[\mathcal{A}(x_b) = b | x_0 \leftarrow \mathcal{D}_0, x_1 \leftarrow \mathcal{D}_1, b \leftarrow \{0,1\}\right] = \frac{1}{2} + \epsilon.$$

We now recall the standard DDH assumption. For self-consistency of notation we will denote the group operation additively, even though the DDH assumption

is more commonly presented with a multiplicative group operation[1]. Suppose a group G with generator g and prime order p are fixed, and denote by T_G the time to perform the group operation.

Definition 2. *Define* $DDH_{\mathbf{R}}$ *as the distribution of*

$$(ag, bg, cg)$$

where a*,* b*, and* c *are chosen independently and uniformly at random from* $\mathbf{F}_p$*.*

Definition 3. *Define* $DDH_{\mathbf{PR}}$ *as the distribution of*

$$(ag, bg, abg)$$

where a *and* b *are chosen independently and uniformly at random from* $\mathbf{F}_p$*.*

Assumption 1 ((τ, ϵ)-DDH) *All probabilistic algorithms* $\mathcal{A}$ *running in time at most* τ *have advantage at most* ϵ *in distinguishing* $DDH_{\mathbf{R}}$ *from* $DDH_{\mathbf{PR}}$*.*

The standard DDH assumption postulates an ensemble of groups $\{\mathcal{G}_\lambda\}_{\lambda \in \mathbb{N}}$ such that when $G \leftarrow \mathcal{G}_\lambda$, G satisfies $(\mathsf{poly}(\lambda), \mathsf{negl}(\lambda))$-DDH.

($d \times T$)-Matrix DDH Our proof of security will use the Matrix DDH assumption of Boneh et al. [BHHO08], which is known to follow from the standard DDH assumption.

Definition 4. *Define* $I_{\mathbf{R}}^{d \times T}$ *as the distribution of* $\mathbf{C}g$ *when* $\mathbf{C}$ *is chosen uniformly at random from* $\mathbf{F}_p^{d \times T}$*.*

Definition 5. *Define* $I_{\mathbf{PR}}^{d \times T}$ *as the distribution of* $\mathbf{a}\mathbf{b}^t g$ *where* $\mathbf{a}$ *and* $\mathbf{b}$ *are chosen uniformly at random from* $\mathbf{F}_p^d$ *and* $\mathbf{F}_p^T$ *respectively.*

Boneh et al. prove the following (which in their paper is also Lemma 1).

Lemma 1 ([BHHO08]). *For every* (d, T)*, if there is an adversary* $\mathcal{A}$ *distinguishing* $I_{\mathbf{PR}}^{d \times T}$ *from* $I_{\mathbf{R}}^{d \times T}$ *with advantage* ϵ *in time* τ*, there is a distinguisher* $\mathcal{D}$ *which distinguishes* $DDH_{\mathbf{PR}}$ *from* $DDH_{\mathbf{R}}$ *with advantage* ϵ/d *in time* $\tau + O(T_G \cdot d \cdot T \log p)$*, where* T_G *is the time to perform the group operation in* G*.*

3 Definition

The security definition for a multilinear pseudorandom function family parallels the usual definition of a pseudorandom function family. That is, oracle access to a multilinear pseudorandom function must be indistinguishable from oracle access to a random multilinear function.

[1] This is probably because the first groups suspected to satisfy the Diffie-Hellman assumption were subgroups of $\mathbb{Z}_p^*$.

Definition 6. *Syntactically, a* multilinear pseudorandom function family *consists of a probabilistic polynomial-time algorithm* KeyGen *and a deterministic polynomial-time algorithm* Eval*.*

- $\mathsf{KeyGen}(1^\lambda)$*:* KeyGen *takes a security parameter in unary.* KeyGen *outputs a secret key K, and also outputs as public parameters a field $\mathbb{F}$, input vector spaces $V_1, \ldots, V_n$, and a codomain vector space Y.*
- $\mathsf{Eval}(K, \mathbf{v}_1, \ldots, \mathbf{v}_n)$*:* Eval *takes as input a secret key K and vectors $(\mathbf{v}_1, \ldots, \mathbf{v}_n)$, and outputs a vector $y = F_K(\mathbf{v}_1, \ldots, \mathbf{v}_n)$ in the codomain.*

KeyGen *and* Eval *must satisfy* security*: for all probabilistic polynomial-time algorithms $\mathcal{A}$,*

$$\Pr\left[\mathcal{A}^{F_b}(PP, 1^\lambda) = b \,\middle|\, \begin{array}{l}(K, PP) \leftarrow \mathsf{KeyGen}(1^\lambda), b \leftarrow \{0,1\} \\ F_0 = \mathsf{Eval}(K, \cdot), F_1 \leftarrow Y^{V_1 \otimes \cdots \otimes V_n}\end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

Truthfulness While our definition only requires *indistinguishability* from a random multilinear function, Construction 1 is actually multilinear itself. This fact allows our construction to satisfy the definition of an aggregate PRF, as discussed in Section 5. We adopt the terminology of [GGN03], calling this property "truthfulness".

Remark 1. One can imagine variants on Definition 6. Specifically, we imagine specifying the domain and codomain arbitrarily rather than receiving them as outputs of KeyGen. Our construction achieves this in a limited sense; we can specify n and $\dim(V_1), \ldots, \dim(V_n)$, but Y must always be a DDH-hard group of large prime order, and $\mathbb{F}$ must be $\mathbf{F}_p$. Constructing a multilinear pseudorandom function family over arbitrary finite fields or rings is an intriguing open question. A special case of this question was posed by [GGN03].Paraphrased in our terminology, they asked whether there is a multilinear pseudorandom function family mapping $\mathbf{F}_2^2 \times \cdots \times \mathbf{F}_2^2 \to \mathbf{F}_2$.

One might naively attempt to solve this by composing our construction with a homomorphism from Y to $\mathbf{F}_2$. Unfortunately, in our construction Y must be a DDH-hard group, so no such homomorphism can be efficiently computable.

4 Construction

We now construct a multilinear pseudorandom function family based on DDH-hard groups. Given as public parameters a DDH-hard group G of order p with generator g, and arbitrary dimensions $d_1, \ldots, d_n$, we construct a multilinear pseudorandom function family $\mathcal{F}_{d_1,\ldots,d_n}$ mapping $\mathbf{F}_p^{d_1} \times \cdots \times \mathbf{F}_p^{d_n} \to G$. The security of our construction is determined only by the choice of G, and so we have no explicit security parameter, in contrast to Definition 6. To match the definition, one can easily let KeyGen generate a group in which the (assumed) hardness of the DDH problem corresponds to the given security parameter.

Construction 1 *$\mathcal{F}_{d_1,\ldots,d_n}$ is defined by*

- $\mathsf{KeyGen}()$*:* KeyGen *samples vectors* $\mathbf{w}_1, \ldots, \mathbf{w}_n$*, where* $\mathbf{w}_i \leftarrow \mathbf{F}_p^{d_i}$ *is sampled uniformly at random. It returns the secret key* $K = (\mathbf{w}_1, \ldots, \mathbf{w}_n)$*.*
- $\mathsf{Eval}(K, (\mathbf{v}_1, \ldots, \mathbf{v}_n))$*:* Eval *returns* $\left(\prod_{i=1}^{n} \langle \mathbf{w}_i, \mathbf{v}_i \rangle\right) g$*, where* $\langle \cdot, \cdot \rangle$ *denotes the inner product.*

Remark 2. This construction generalizes the Naor-Reingold PRF[NR04], but we allow richer queries. Specifically, to recover the Naor-Reingold construction, set each $d_i = 2$, and restrict each $\mathbf{v}_i$ to be a basis vector.

Remark 3 (Truthfulness). Every function in $\mathcal{F}_{d_1,\ldots,d_n}$ is truly multilinear (not just indistinguishable from multilinear). This follows from the bilinearity of the inner product and the multilinearity of multiplication (e.g. $(x, y, z) \mapsto xyz$ is multilinear).

4.1 Proof of Security

Our main security proof is the following theorem.

Theorem 1. *When instantiated with a DDH-hard group G, Construction 1 satisfies Definition 6.*

Specifically, if there is an algorithm $\mathcal{A}$ running in time T such that

$$\Pr\left[\mathcal{A}^{F_b}(1^\lambda) = b \middle| \begin{array}{l} K \leftarrow \mathsf{KeyGen}(), F_0 = \mathsf{Eval}(K, \cdot), \\ F_1 \leftarrow G^{\mathbf{F}_p^{d_1} \otimes \cdots \otimes \mathbf{F}_p^{d_n}}, b \leftarrow \{0,1\} \end{array}\right] = \frac{1}{2} + \epsilon$$

then there is a distinguisher $\mathcal{D}$ running in time $\mathsf{poly}(T, T_G, \sum_i d_i)$ which distinguishes $DDH_{\mathbf{PR}}$ from $DDH_{\mathbf{R}}$ with advantage at least $\frac{|\epsilon|}{n \cdot \max_i d_i}$.

Proof Overview In this overview, we outline a proof by induction on n. In our actual proof we "unroll" the induction and prove the theorem directly.

When $n = 1$, our construction is a truly random linear function mapping $V_1 \to G$, given by $\mathbf{v} \mapsto \langle \mathbf{w}, \mathbf{v} \rangle g$ for randomly chosen $\mathbf{w}$ and generator g.

We now show that an oracle implementing our construction is pseudorandom for $n > 1$. By definition, $F_0(\mathbf{v}_1, \ldots, \mathbf{v}_n)$ is equal to $\langle \mathbf{w}_n, \mathbf{v}_n \rangle \prod_{i=1}^{n-1} \langle \mathbf{w}_i, \mathbf{v}_i, \rangle g$. By the inductive hypothesis, oracle access to $\langle \mathbf{w}_n, \mathbf{v}_n \rangle R_{n-1}(\mathbf{v}_1, \ldots, \mathbf{v}_n)$ is indistinguishable, where R_{n-1} is a truly random multilinear function in $G^{V_1 \otimes \cdots \otimes V_{n-1}}$. Although $\dim(V_1 \otimes \cdots \otimes V_{n-1})$ is exponential in n, we are able to efficiently implement an oracle to R_{n-1} in a stateful manner.

It remains to show that oracle access to $\langle \mathbf{w}_n, \mathbf{v}_n \rangle R_{n-1}$ is indistinguishable from a random multilinear function $F_1 = R_n \leftarrow G^{V_1 \otimes \cdots \otimes V_n}$. We show that a distinguisher $\mathcal{A}$ of oracle access to R_n from oracle access to $\langle \mathbf{w}_n, \mathbf{v}_n \rangle R_{n-1}$ violates the Matrix DDH assumption. This indistinguishability relies on two different ways of statefully implementing any R_n, given in Algorithm 1 and Algorithm 2.

While we described the proof as an induction, directly applying these ideas in our main proof does not yield an efficient reduction. Below, we use the standard hybrid argument technique to avoid this pitfall.

Our proof relies on two different algorithms for statefully and efficiently implementing oracle access to a random multilinear function, R_n from $V_1 \times \cdots \times V_n$ to G. We can instead consider R_n as a random linear function from $V = V_1 \otimes \cdots \otimes V_n$ to G, using the correspondence described in the preliminaries. Because we consider linear functions on $V_1 \otimes \cdots \otimes V_n$ only as a tool to describe multilinear functions on $V_1 \times \cdots \times V_n$, we are able to restrict our attention to *simple* tensors in the analysis.

Algorithm 1. We maintain a map M which stores a subset of a mapping $V \to Y$. That is M stores a collection of pairs $(\mathbf{u} \mapsto \mathbf{y})$; we say that $M(\mathbf{u}) = \mathbf{y}$ if such an entry for $\mathbf{u}$ exists in M, and that $M(\mathbf{u}) = \perp$ otherwise. Initially M is the empty set. A query $\mathbf{v}$ is answered by executing the following steps:

1. Check whether $\{\mathbf{v}\} \cup \{\mathbf{u} : M(\mathbf{u}) \neq \perp\}$ is linearly independent. If it is, sample a random vector $\mathbf{y} \leftarrow Y$ and add the mapping $(\mathbf{v} \mapsto \mathbf{y})$ to M.
2. Compute $\mathbf{v} = \sum_j c_j \mathbf{u}_j$ where for each j, $M(\mathbf{u}_j) \neq \perp$.
3. Return $\sum_j c_j M(\mathbf{u}_j)$.

The efficiency of Steps 1 and 2 relies on the $\mathsf{SpanSearch}$ algorithm, which works for simple tensors.

Proposition 1. *Algorithm 1 implements a random linear function mapping $V \to Y$.*

Proof. Suppose the queries up to time t are given by $\mathbf{v}_1, \ldots, \mathbf{v}_t \in V$. Let $(\mathbf{v}_{i_1} \mapsto \mathbf{y}_{i_1}), \ldots, (\mathbf{v}_{i_j} \mapsto \mathbf{y}_{i_j})$ be the first j entries in M. The vectors $\mathbf{v}_{i_1}, \ldots, \mathbf{v}_{i_j}$ are a basis for $\mathsf{span}(\mathbf{v}_1, \ldots, \mathbf{v}_t)$. It is easy to see that Algorithm 1 implements a linear map on $\mathsf{span}(\mathbf{v}_1, \ldots, \mathbf{v}_t)$ which is given by a random matrix. In particular, this matrix has columns $\mathbf{y}_{i_1}, \ldots, \mathbf{y}_{i_j}$.

We now give an alternate algorithm implementing a random linear function mapping $U \otimes W \to Y$ for any vector spaces U, W, and Y. In particular, we will take $U = V_1 \otimes \cdots \otimes V_{j-1}$ and $W = V_j$.

Algorithm 2. Queries are of the form $\mathbf{u} \otimes \mathbf{w} \in U \otimes W$. We maintain a map M which stores a subset of a mapping $U \to Y^W$. That is M stores a collection of pairs $(\mathbf{z} \mapsto f)$, where each f is a linear map from W to Y. We say that $M(\mathbf{z}) = f$ if such an entry for $\mathbf{z}$ exists in M, and that $M(\mathbf{z}) = \perp$ otherwise. Initially M is the empty set. A query $\mathbf{u} \otimes \mathbf{w}$ is answered by executing the following steps:

1. Check whether $\{\mathbf{u}\} \cup \{\mathbf{z} : M(\mathbf{z}) \neq \perp\}$ is linearly independent. If it is, sample a random linear map $f : W \to Y$ and add the mapping $(\mathbf{z} \mapsto f)$ to M.
2. Write $\mathbf{u} = \sum_j c_j \mathbf{z}_j$ where for each j, $M(\mathbf{z}_j) = f_j$.
3. Return $\sum_j c_j f_j(\mathbf{w})$.

Proposition 2. *Algorithm 2 implements a random linear function mapping* $U \otimes W \to Y$.

Proof. A linear function mapping U to the space Y^W of linear functions from W to Y can be equivalently viewed as a bilinear function mapping $U \times W \to Y$. As discussed in the preliminaries, there is a bijective correspondence between such bilinear functions and linear functions mapping $U \otimes W \to Y$. Then Proposition 2 is just a special case of Proposition 1.

The main lemma used in the proof of Theorem 1 is that the following two distributions on linear functions are indistinguishable.

Definition 7. *For* $j > 0$, *let* $\mathcal{RF}_j$ *denote* $G^{\mathbf{F}_p^{d_1} \otimes \cdots \otimes \mathbf{F}_p^{d_j}}$. *Let* $\mathcal{RF}_0 = G$.

Definition 8. *For* $j > 0$, *let* $\mathcal{PRF}_j$ *denote the distribution of multilinear functions defined by*

$$(\mathbf{v}_1 \otimes \cdots \otimes \mathbf{v}_j) \mapsto \langle \mathbf{w}, \mathbf{v}_j \rangle R(\mathbf{v}_1 \otimes \cdots \otimes \mathbf{v}_{j-1})$$

where $\mathbf{w}$ *is sampled from* $\mathbf{F}_p^{d_j}$ *and* R *is sampled from* $\mathcal{RF}_{j-1}$.

Lemma 2. *If there is an oracle algorithm* $\mathcal{A}$ *running in time* T *such that*

$$\Pr\left[\mathcal{A}^{F_b}() = b \middle| F_0 \leftarrow \mathcal{PRF}_j, F_1 \leftarrow \mathcal{RF}_j, b \leftarrow \{0,1\}\right] = \frac{1}{2} + \epsilon$$

then there is a distinguisher $\mathcal{D}$ *running in time* $\mathsf{poly}(T, \sum_{i \le j} d_i)$ *such that* $\mathcal{D}$ *breaks Matrix DDH with the same advantage. That is,*

$$\Pr\left[\mathcal{D}(M_b) = b \middle| M_0 \leftarrow I_{\mathbf{PR}}^{d_j \times T}, M_1 \leftarrow I_{\mathbf{PR}}^{d_j \times T}, b \leftarrow \{0,1\}\right] = \frac{1}{2} + \epsilon.$$

The distinguisher $\mathcal{D}$ is defined to execute the following steps:

1. Take $\tilde{\mathbf{C}}g$ as input. Here $\tilde{\mathbf{C}}$ is either equal to $\mathbf{a}\mathbf{b}^t$ for random $\mathbf{a} \in \mathbf{F}_p^{d_j}$ and $\mathbf{b} \in \mathbf{F}_p^T$ or is sampled uniformly at random $C \leftarrow \mathbf{F}_p^{d_j \times T}$. Denote the k^{th} column of $\tilde{\mathbf{C}}g$ by $\boldsymbol{\gamma}_k$
2. Create an (initially empty) map M to store a subset of $\mathbf{F}_p^{d_1} \times \cdots \times \mathbf{F}_p^{d_{j-1}} \to G^{d_j}$. That is M stores a collection of pairs $(\mathbf{v}_1 \otimes \cdots \otimes \mathbf{v}_{j-1} \mapsto \mathbf{g})$, where each $\mathbf{g} \in G^{d_j}$. We will preserve the invariant that $\{\mathbf{u} : M(\mathbf{u}) \neq \perp\}$ is linearly independent.
3. Run the adversary $\mathcal{A}()$, answering queries as follows:
   On the i^{th} query $\mathbf{v}_1^i \otimes \cdots \otimes \mathbf{v}_{j-1}^i \otimes \mathbf{v}_j^i$, first define $\mathbf{v}_{-j}^i = \mathbf{v}_1^i \otimes \cdots \otimes \mathbf{v}_{j-1}^i$. Use our $\mathsf{SpanSearch}$ solver to check whether $\{\mathbf{v}_{-j}^i\} \cup \{\mathbf{u} : M(\mathbf{u}) \neq \perp\}$ is linearly independent. If it is, add the mapping $(\mathbf{v}_{-j}^i \mapsto \boldsymbol{\gamma}_i)$ to M.
   Otherwise, our $\mathsf{SpanSearch}$ solver tells us how to write $\mathbf{v}_{-j}^i$ as $\sum_k \alpha_k \mathbf{u}_k$, where each $M(\mathbf{u}_k)$ is not $\perp$. $\mathcal{D}$ then answers $\mathcal{A}$'s query with $\sum_k \alpha_k \langle M(\mathbf{u}_k), \mathbf{v}_j^i \rangle$.
4. Finally, $\mathcal{D}$ outputs the same answer that $\mathcal{A}$ outputs.

Lemma 2 follows from the following two claims.

Claim. When $\tilde{\mathbf{C}}$ is uniformly random, then $\mathcal{D}$ answers queries according to the same distribution as $\mathcal{RF}_j$.

Proof. This follows from Proposition 2. Namely, when $\tilde{C}$ is uniformly random, the columns $\boldsymbol{\gamma}_i$ define independent and uniformly random linear maps from $\mathbf{F}_p^{d_j} \to G$. $\mathcal{A}$'s queries are therefore answered according to a random multilinear function, which is the same as $\mathcal{RF}_j$.

Claim. When $\tilde{\mathbf{C}}$ is generated as $\mathbf{a}\mathbf{b}^t$, then $\mathcal{D}$ answers queries according to the same distribution as $\mathcal{PRF}_j$.

Proof. Suppose that $\tilde{\mathbf{C}}$ is $\mathbf{a}\mathbf{b}^t$. Then each $\boldsymbol{\gamma}_i$ is $\mathbf{a}b_i$, where each b_i is sampled independently and uniformly at random from $\mathbf{F}_p$. So $\mathcal{D}$ can equivalently change M to only store $(\mathbf{v}^i_{-j} \mapsto b_i g)$ and now answers queries with $\langle \mathbf{a}, \mathbf{v}^i_j \rangle \left(\sum_k \alpha_k M(\mathbf{u}_k)\right)$. By Proposition 1, this is the same as $\langle \mathbf{a}, \mathbf{v}^i_j \rangle R(\phi(\mathbf{v}^i_{-j}))$ with R sampled from $\mathcal{RF}_j$. By definition, this is the same as answering queries with a randomly sampled R' from $\mathcal{PRF}_j$.

We can now prove Theorem 1.

Proof (of Theorem 1). We define distinguishers $\mathcal{D}_J$ for each $J \in \{0, \ldots, n\}$ that execute the following steps:

1. Prepare a stateful implementation $R \leftarrow \mathcal{RF}_J$ using Algorithm 1 backed by our $\mathsf{SpanSearch}$ solver.
2. Sample $\mathbf{w}_i$ uniformly at random from $\mathbf{F}_p^{d_i}$ for each $i \in \{J+1, \ldots, n\}$.
3. Run $\mathcal{A}$, answering its queries $(\mathbf{v}_1, \ldots, \mathbf{v}_n)$ with $\left(\prod_{i=J+1}^{n} \langle \mathbf{w}_i, \mathbf{v}_i \rangle\right) R(\mathbf{v}_1, \ldots, \mathbf{v}_J)$.
4. Output whatever $\mathcal{A}$ outputs.

First, it is clear that the output of $\mathcal{D}_0$ is the same as the output of $\mathcal{A}^{\mathsf{Eval}(K,\cdot)}()$ where $K \leftarrow \mathsf{KeyGen}()$, and the output of $\mathcal{D}_n$ is the same as the output of $\mathcal{A}^{F_1}()$ where $F_1 \leftarrow \mathcal{RF}_n$. By a standard hybrid argument, there must exist some $j \in \{0, \ldots, n-1\}$ such that $\mathcal{D}_j$ and $\mathcal{D}_{j+1}$ output 1 with probabilities differing by at least $|\epsilon|/n$.

But if we replace $\mathcal{D}_{j+1}$'s (black-box) usage of $F \leftarrow \mathcal{RF}_{j+1}$ by $F \leftarrow \mathcal{PRF}_{j+1}$, then $\mathcal{D}_{j+1}$ is functionally equivalent to $\mathcal{D}_j$. So $\mathcal{D}_{j+1}$ can be used to distinguish oracle access to $\mathcal{RF}_{j+1}$ from oracle access to $\mathcal{PRF}_{j+1}$. Lemma 2 implies that $\mathcal{D}_{j+1}$ can be used to distinguish $I_{\mathbf{R}}^{d_{j+1} \times Q}$ from $I_{\mathbf{PR}}^{d_{j+1} \times Q}$ with advantage at least $|\epsilon|/n$, where Q is any bound on the number of linearly independent queries made by $\mathcal{A}$. In particular $Q \leq T$. Lemma 1 implies that $\mathcal{D}_{j+1}$ can be used to distinguish $DDH_{\mathbf{R}}$ from $DDH_{\mathbf{PR}}$ with advantage at least $\frac{|\epsilon|}{n \cdot d_{j+1}}$. □

5 Applications

In this section, we show how our multilinear PRF simplifies and improves PRF constructions in [BGV11] and [CGV15]. We instantiate the vector spaces $F_p^{d_i}$ of Construction 1 appropriately, and show that oracle access to a multilinear PRF suffices to perfectly simulate oracle access to the functions from those works.

Aggregate PRFs [CGV15] are PRF families with extra efficiency and security properties. First, the key K for a PRF f enables efficient computation of $\mathsf{Agg}_f(S) = \sum_{x \in S} f(x)$ for some class of succinctly described, but possibly exponentially large, sets S. Second, no efficient algorithm can distinguish oracle access to $f(\cdot)$ and $\mathsf{Agg}_f(\cdot)$ from oracle access to $g(\cdot)$ and $\mathsf{Agg}_g(\cdot)$, where g is a truly random function.

One specific setting that [CGV15] addresses is when S can be any "hypercube". A hypercube $H_p \subset \{0,1\}^n$ is described by a pattern $p \in \{\{0\},\{1\},\{0,1\}\}^n$. H_p is defined as $\{x \in \{0,1\}^n : x_i \in p_i\}$. Informally, H_p is the set obtained by fixing the bits of x at particular indices, and allowing all other bits to vary freely. [CGV15] showed a construction with efficent evaluation, but security relied on the subexponential hardness of the DDH problem.

We show that the hypercube construction in [CGV15] is a special case of Construction 1. The correctness of the aggregate queries is implied by the truthfulness of our construction. Thus we prove aggregate security of their construction relying only on the standard DDH assumption.

Corollary 1. *Assuming the (polynomial) hardness of DDH over the group G, [CGV15]'s PRFs for hypercubes*[2] *and decision trees* [3] *are secure aggregate PRFs.*

Proof. As shown in [CGV15], it suffices to prove the case of hypercubes.

Let B denote the 2-dimensional vector space whose basis vectors are $|0\rangle$ and $|1\rangle$. Our construction gives a pseudorandom multilinear function F mapping B^n to G. This function F induces a pseudorandom function $f : \{0,1\}^n \to G$ given by $f(b_1 \dots b_n) = F(|b_1\rangle, \dots, |b_n\rangle)$.

First observe we can compute the sum of $f(x)$ for all x as

$$F\big(|0\rangle{+}|1\rangle \ , \ \dots \ , \ |0\rangle{+}|1\rangle\big).$$

To fix a bit x_i to b – thus aggregating over a smaller hypercube – we replace the i^{th} argument of F above with $|b\rangle$. That is, to compute $\mathsf{Agg}_f(H_p)$ for some hypercube H_p, we evaluate

$$F\left(\sum_{b \in p_1} |b\rangle, \dots, \sum_{b \in p_n} |b\rangle\right)$$

This yields the correct aggregate value by the truthfulness (multilinearity) of our construction. Therefore oracle access to Agg_f can be simulated with even

[2] Section 3.2
[3] Section 3.3

a restricted oracle to F. Namely, we only make queries where each argument is either $|0\rangle$, $|1\rangle$, or $|0\rangle + |1\rangle$. Theorem 1 then implies aggregate PRF security of f. □

We can actually achieve the more generalized aggregation, as required in the work of Benabbas, Gennaro, and Vahlis [BGV11] on algebraic PRFs, while maintaining aggregate security. For example, efficiently evaluating

$$p_f(z) = \sum_{x \in \{0,1\}^n} f(x) z^n$$

has applications in verifiable and multiparty computation [BGV11,Haz15]. We can achieve this functionality with oracle access to our multilinear PRF F, thus keeping aggregate security. Specifically, instantiate Construction 1 as above. One can then compute

$$p_f(z) = F\big(|0\rangle + z^{2^{n-1}} |1\rangle \ , \ \ldots \ , \ |0\rangle + z\,|1\rangle\big).$$

Correctness and security follow directly because F is a pseudorandom multilinear function. This can easily be extended to cover the more general multivariable algebraic PRF considered in [BGV11] (Section 4.2), along with a number of other immediate generalizations.

Each of the above applications uses only the simplest of vector spaces. There are many other ways in which a multilinear PRF can be invoked, but we highlight these two examples as applications which have already appeared in the literature.

6 Extensions

Two other classes of functions which are fundamental in mathematics are *symmetric* and *skew-symmetric* multilinear functions. Informally, a function from $V^n \to Y$ is symmetric if swapping any arguments x_i and x_j does not affect the value, and is skew-symmetric if such a swap negates the value. Pseudorandom implementations of these classes of functions are interesting open problems.

Definition 9. *A function $F : V^n \to Y$ is said to be* symmetric *if for all $i \neq j$,*

$$F(x_1, \ldots, x_n) = F(x'_1, \ldots, x'_n),$$

where

$$x'_k = \begin{cases} x_i & \text{if } k = j \\ x_j & \text{if } k = i \\ x_k & \text{otherwise.} \end{cases}$$

F is said to be skew-symmetric *if $F(x_1, \ldots, x_n) = -F(x'_1, \ldots, x'_n)$.*

Given a group G of order p with generator g, we present a candidate construction of a symmetric multilinear pseudorandom function family,

Construction 2 *$\mathcal{F}_{d,n}$ is defined by*

- $\mathsf{KeyGen}()$*:* KeyGen *samples a vector* $\mathbf{w}$ *uniformly at random from* $\mathbf{F}_p^d$.
- $\mathsf{Eval}(\mathbf{w}, \mathbf{v}_1, \ldots, \mathbf{v}_n)$*:* Eval *returns* $\left(\prod_{i=1}^n \langle \mathbf{w}, \mathbf{v}_i \rangle\right) g$*, where* $\langle \cdot, \cdot \rangle$ *denotes the inner product.*

This is a modification to Construction 1 in which $\mathbf{w}_1 = \cdots = \mathbf{w}_n$, which clearly yields symmetric multilinear functions, but security is less clear.

In case $d = 2$, security reduces to the n-Strong DDH assumption. This assumption states that $(h, xh, \ldots, x^n h)$ is indistinguishable from $n + 1$ random elements of G, when h is a randomly chosen generator of G, and x is a random element of $\mathbf{F}_p$. This is because a symmetric multilinear function on $(\mathbf{F}_p^2)^n$ is defined by $n + 1$ "basis" values, which in the above construction correspond to this tuple.

Conjecture 1. With a suitably chosen group G, Construction 2 defines a symmetric multilinear family of pseudorandom functions.

References

BGV11. Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In *Advances in Cryptology–CRYPTO 2011*, pages 111–131. Springer, 2011.

BHHO08. Dan Boneh, Shai Halevi, Mike Hamburg, and Rafail Ostrovsky. Circular-secure encryption from decision diffie-hellman. In *Advances in Cryptology–CRYPTO 2008*, pages 108–125. Springer, 2008.

BW04. Andrej Bogdanov and Hoeteck Wee. A stateful implementation of a random function supporting parity queries over hypercubes. In *8th International Workshop on Randomization and Computation, RANDOM 2004, Cambridge, MA, USA, August 22-24, 2004, Proceedings*, pages 298–309, 2004.

CGV15. Aloni Cohen, Shafi Goldwasser, and Vinod Vaikuntanathan. Aggregate pseudorandom functions and connections to learning. page To appear. Springer, 2015.

GGN03. Oded Goldreich, Shafi Goldwasser, and Asaf Nussboim. On the implementation of huge random objects. In *Proceedings of the 44th Annual Symposium on Foundations of Computer Science*, pages 68–79, 2003.

Haz15. Carmit Hazay. Oblivious polynomial evaluation and secure set-intersection from algebraic prfs. In *Theory of Cryptography*, page To appear. Springer, 2015.

NR04. Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *Journal of the ACM (JACM)*, 51(2):231–262, 2004.

A Randomized Algorithm for SpanSearch

The algorithm, closely adapted from [BW04], uses a linear code with special properties to construct an efficient randomized algorithm.

Lemma 3. *Let $V_1, \ldots, V_n$ be vector spaces over $\mathbf{F}_q$, and let d denote $\dim(V_1) + \cdots + \dim(V_n)$. Then there is a linear code C with relative distance $1/3$ mapping $V = V_1 \otimes \cdots \otimes V_n$ into $(\mathbf{F}_q)^{q'^d + k}$, where $q' = q^k$ is the smallest power of q that is greater than $3n$. Furthermore, the j^{th} symbol of the codeword for any simple vector v is efficiently computable, and is denoted $C(v)[j]$.*

Proof. We structure the code as the composition of several injective linear maps.

First, we observe that there is an injective linear map from $V_1 \otimes \cdots \otimes V_n$ to the space of homogeneous polynomials of degree n over $\mathbf{F}_q$ as follows. Let d_j denote $\dim(V_j)$, and let $x_{j,1}, \ldots, x_{j,d_j}$ denote a basis of V_j. Given a simple element $v = (\sum_{k=1}^{d_1} c_{1,k} x_{1,k}) \otimes \cdots \otimes (\sum_{k=1}^{d_n} c_{n,k} x_{n,k})$, it's easy to form a polynomial. Just treat each $x_{j,k}$ as a formal variable and replace tensor multiplication by field multiplication. We shall denote this polynomial as p_v. The mapping $v \mapsto p_v$ is injective because $\{x_{1,j_1} \otimes \cdots \otimes x_{n,j_n}\}_{j_k \in [d_k]}$ is a basis for V, and the degree-n monomials $\{\prod_i x_{i,j_i}\}_{j_1 \in [d_1], \ldots, j_n \in [d_n]}$ are linearly independent.

Next, embed $\mathbf{F}_q$ in its field extension $\mathbf{F}_{q'}$. This is an injective linear map, as is the embedding of p_v into the space of polynomials over $\mathbf{F}_{q'}$.

This embedded polynomial (call it p'_v) then defines a Reed-Muller codeword $C'(v) \in \mathbf{F}_{q'}$: we just enumerate the values of p'_v on every input in $\mathbf{F}_{q'}^d$. This is an injective linear map, and furthermore it has a relative distance of $2/3$ by the Schwartz-Zippel lemma.

Finally we encode each symbol (in $\mathbf{F}_{q'}$) of this codeword. First, because $\mathbf{F}_{q'}$ is an extension of $\mathbf{F}_q$, it is isomorphic to $\mathbf{F}_q^k$. The Hadamard encoding thus considers the inner product of that symbol with *every* vector in $\mathbf{F}_q^k$. This encoding incurs a multiplicative loss in relative distance of $(1 - 1/q)$, which could be as little as $1/2$. So this final code has a relative distance of at least $2/3 \cdot 1/2 = 1/3$.

The resulting codeword $C(v)$ is of length $q'^d + k$ and consists of symbols in F_q.

Algorithm 3. Given an instance $(v_0, v_1, \ldots, v_t)$ of SpanSearch, where v_i is given as $v_i = v_{i,1} \otimes \cdots \otimes v_{i,n}$, our algorithm for computing $c_1, \ldots, c_t$ is as follows.

Our algorithm is a randomized solution to a search problem, so we cannot use standard amplification techniques. Instead, we have a correctness parameter λ such that the probability of error is bounded by $2^{-\lambda}$.

1. Let $m = t \cdot \log_{3/2} q + \lambda \cdot \log_{3/2} 2$ and randomly sample $r_1, \ldots, r_m$ each from $[q'^d + k]$.
2. Build the matrix

$$M = \begin{pmatrix} C(v_1)[r_1] & C(v_2)[r_1] & \ldots C(v_t)[r_1] \\ C(v_1)[r_2] & C(v_2)[r_2] & \ldots C(v_t)[r_2] \\ \vdots & \vdots & \ddots \vdots \\ C(v_1)[r_m] & C(v_2)[r_m] & \ldots C(v_t)[r_m] \end{pmatrix}$$

and solve the matrix equation

$$M \cdot \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_t \end{pmatrix} = \begin{pmatrix} C(v_0)[r_1] \\ C(v_0)[r_2] \\ \vdots \\ C(v_0)[r_m] \end{pmatrix} \tag{1}$$

for $c_1, \ldots, c_t$ over $\mathbf{F}_q$. If no solution exists, return $\perp$. Otherwise return $(c_1, \ldots, c_t)$.

Correctness In order to show correctness, we must show that $v_0 = c_1 v_1 + \ldots + c_t v_t$ if and only if Equation 1 is satisfied. The forward implication is clear by the linearity of our code: If $v_0 = c_1 v_1 + \ldots + c_t v_t$, then $C(v_0) = c_1 C(v_1) + \cdots + c_t C(v_t)$. We show that the converse holds with high probability by applying a union bound over every choice of $(c_1, \ldots, c_t)$.

There are q^t linear combinations of $C(v_1), \ldots, C(v_t)$. Each linear combination which differs from $C(v_0)$ in fact differs from $C(v_0)$ on at least $\frac{1}{3}$ of the indices by Lemma 3. Because $r_1, \ldots, r_m$ are chosen randomly, the probability that this combination satisfies Equation 1 is at most $(\frac{2}{3})^m$. By a union bound, the probability that *any* linear combination of $v_1, \ldots, v_t$ which differs from v_0 satisfies Equation 1 is at most

$$\begin{aligned} q^t \left(\frac{2}{3}\right)^m &= q^t \left(\frac{2}{3}\right)^{t \cdot \log_{3/2} q + \lambda \cdot \log_{3/2} 2} \\ &\leq 2^{-\lambda} s \end{aligned}$$

so the "if and only if" holds with very high probability.

Efficiency This algorithm runs in time $\mathsf{poly}(n, d, \log q, t, \lambda)$ where n is the number of vector spaces (over a field of size q) whose tensor product we work with, and d is their maximum dimension. t is the number of vectors given, and λ is the correctness parameter.