Efficient Pseudorandom Functions From the Decisional Linear Assumption and Weaker Variants

Allison B. Lewko [*]
University of Texas at Austin

Brent Waters [†]
University of Texas at Austin

Abstract

In this paper, we generalize Naor and Reingold's construction of pseudorandom functions under the DDH Assumption to yield a construction of pseudorandom functions under the decisional k-Linear Assumption, for each $k \geq 1$. The decisional Linear Assumption was first introduced by Boneh, Boyen, and Shacham as an alternative assumption for settings where the DDH problem is easy, such as bilinear groups. Shacham and Hofheinz and Kiltz independently introduced the generalized decisional k-Linear Assumptions and showed that the decisional $(k+1)$-Linear problem is hard for generic groups even when the decisional k-Linear problem is easy. It is thus desirable to have constructions of cryptographic primitives based on the decisional k-Linear Assumption instead of DDH. Not surprisingly, one must pay a small price for added security: as k increases, our constructed functions become slightly less efficient to compute and the key size increases (quadratically in k).

1 Introduction

Pseudorandom functions were first defined by Goldreich, Goldwasser, and Micali [14]. Informally, a pseudorandom function ensemble is a collection of functions that can be efficiently sampled and computed, but cannot be distinguished from random functions by a polynomial time adversary with only black-box access. We will give a formal definition in the next section. These cryptographic primitives have many applications (e.g. [3, 7, 12, 13, 19, 24, 27]). For example, a pseudorandom function is often substituted for a truly random function in an application where true randomness would be unacceptably inefficient. In private-key cryptography, a relatively short description of a pseudorandom function can be used as a private key, allowing parties who share this private key to send encrypted messages or verify

[*]Supported by National Defense Science and Engineering Graduate Fellowship.

[†]Supported by NSF CNS-0716199, Air Force Office of Scientific Research (AFO SR) under the MURI award for "Collaborative policies and assured information sharing" (Project PRESIDIO).

each other's knowledge of the shared secret without needing to reveal the secret itself. For applications of pseudorandom functions in private-key cryptography, see e.g. [7, 13, 19]. Pseudorandom functions are also used in public-key cryptography (e.g. [3, 12]), learning theory (e.g. [27]), and complexity theory (e.g. [24]).

Motivation These numerous applications make it desirable to have constructions of pseudorandom functions which are both efficient enough to be implemented in practice and based on well-established assumptions. Goldreich, Goldwasser, and Micali gave the first construction of a pseudorandom function ensemble, known as the GGM-Construction [14]. This construction relied only on pseudorandom generators, which can be built from any one-way function (as shown in [15]). In [22], Noar and Reingold gave two constructions of pseudorandom functions which are much more efficient to compute than the GGM functions. One construction was based on the Decisional Diffie-Hellman Assumption (DDH) and the other was based on the assumption that factoring Blum integers is hard. (See section 2 for the definition of the DDH Assumption.)

The DDH Assumption is a commonly used assumption with several attractive qualities. Its applications include the Diffie-Hellman key-exchange protocol (the context in which DDH was introduced) [10], ElGamal encryption [11], Cramer-Shoup CCA-secure public key encryption [8], undeniable signatures [6], verifiable secret sharing [26], and many others. Naor and Reingold gave a reduction between the worst-case and the average-case DDH problem, showing that it is either hard on average or easy even in the worst case.

Though such a reduction gives credence to the belief that the DDH Assumption holds in groups where it is not known to be easy on average, there are groups where the DDH Assumption fails. Most notably, the DDH problem is easy in bilinear groups. In [20], Menezes, Okamoto, and Vanstone showed that there are subexponential attacks on the discrete log problem in certain elliptic curve groups (supersingular curves) which had been previously been suggested for use in cryptographic systems. This example shows that even well-established assumptions can be found to have surprising weaknesses in certain implementations. Developing cryptographic primitives and systems based on progressively weaker assumptions is therefore advantageous because it provides some protection against future advances in cryptanalysis.

There is evidence that groups exist in which the DDH problem is easy but the CDH problem (Computational Diffie-Hellman) is hard [17]. In such groups, we might still rely on the difficulty of computing discrete logarithms, but we must avoid the DDH Assumption. One alternative is the decisional Linear Assumption, introduced by Boneh, Boyen, and Shacham [5]. This assumption was generalized by Shacham [25] and Hofheinz and Kiltz [16] to yield the decisional k-Linear Assumptions, which we will refer to simply as the k-Linear Assumptions for brevity. The 1-Linear Assumption is DDH, and the 2-Linear Assumption is the Linear Assumption from [5]. Both [16, 25] note that the $(k+1)$-Linear Assumption holds in a generic group even when the k-Linear problem is easy and give constructions of chosen-ciphertext secure cryptosystems based on the k-Linear Assumption for any positive integer k.

Our Contribution We generalize the construction of Naor and Reingold to yield pseudorandom function ensembles based on the Linear and k-Linear Assumptions. We thus achieve added security, and we do so with relatively little cost in efficiency. It is not too difficult to see that a change to the Naor-Reingold construction is necessary if the DDH problem is easy. The Naor-Reingold construction maps an n-bit string $x = (x_1, \ldots, x_n)$ into a cyclic group G of prime order p generated by g. A pseudorandom function f is specified by the group G, g, p, and $n+1$ values in $\mathbb{Z}_p$ denoted by $a_0, a_1, \ldots, a_n$. Naor and Reingold define:

$$f_{G,p,g,a_0,\ldots,a_n}(x) = (g^{a_0})^{\prod_{x_i=1} a_i}. \tag{1}$$

If we have access to an algorithm A that solves the DDH problem with non-negligible advantage, then we can distinguish such an f from a truly random function with non-negligible advantage as follows. We query f on four inputs and obtain responses:

$$f(1,1,\ldots,1,1) = B_1, \quad f(1,1,\ldots,1,0) = B_2,$$
$$f(0,0,\ldots,0,1) = B_3, \quad f(0,0,\ldots,0,0) = B_4.$$

If f is a Naor-Reingold pseudorandom function with key $\{G, p, g, a_0, \ldots, a_n\}$, then $B_1 = g^{a_0 a_1 \ldots a_n}$, $B_2 = g^{a_0 a_1 \ldots a_{n-1}}$, $B_3 = g^{a_0 a_n}$, and $B_4 = g^{a_0}$. We set $\tilde{g} = B_4, \tilde{g}^a = B_3, \tilde{g}^b = B_2$, and $T = B_1$. If f is pseudorandom, then $\tilde{g} = g^{a_0}$, $a = a_n$, $b = a_1 \ldots a_{n-1}$, and $T = \tilde{g}^{ab}$. If f is truly random, then T is uniformly random. Hence, when $\tilde{g}$, $\tilde{g}^a$, $\tilde{g}^b$, T are given to A as input, the output of A can be used to distinguish whether f is pseudorandom or truly random with non-negligible advantage.

Our generalized construction and the proof of its security differ from the Naor-Reingold version in two primary ways. First, the additional complexity required to accommodate the weaker assumptions means that our functions can no longer be described by closed-form formulas like (1). Nonetheless, the additional cost in computational efficiency is rather small. Second, the Linear Assumption cannot be embedded into our construction as directly as the DDH Assumption is embedded in Naor-Reingold, so we must use two separate instantiations of the hybrid technique to prove the pseudorandomness of our construction instead of just one. More specifically, the Naor-Reingold construction relies on a pseudorandom generator that doubles its input and arises very naturally from the DDH Assumption. Obtaining a suitable pseudorandom generator that doubles its input from the Linear Assumption is more difficult, and requires use of the hybrid technique. We discuss this issue in more detail in section 3.

Other Related Work In practice, ad hoc designs of cryptographic primitives are often substituted for constructions which are proven to be secure under standard assumptions. This may yield greater efficiency, but it has been shown to be very dangerous. The potential for compromising vulnerabilities in ad hoc designs further motivates our search for efficient constructions of cryptographic primitives with accompanying proofs of security under weak assumptions. Collision attacks against commonly used hash functions like MD5, SHA-0, and SHA-1 have recently been demonstrated [4, 28]. There is also an ad hoc design of pseudorandom functions, known as TLS, that uses both SHA-1 and MD5 [9]. This design is intended to be secure if both SHA-1 and MD5 are secure, but this is not rigorously proven and it may be that both of these hash functions are vulnerable. Bellare, Canetti, and Krawczyk give two constructions, NMAC and HMAC, which are proven to be secure under the assumption that the underlying hash function is suitably secure

[2], but it may still be the case that an ad hoc hash function chosen for a practical implementation has previously unknown weaknesses.

To implement our construction in practice, one must balance efficiency with security in the choice of the group where the k-Linear Assumption will be relied upon. One usually chooses the security parameters based on the best known attacks. In particular, if one chooses two large primes p and q such that p divides $q-1$, we can work in a subgroup of order p in $\mathbb{Z}_q^*$. The known attacks on the discrete logarithm problem in this case include Shanks' baby-step giant-step algorithm [18] and Pollard's rho method [23], both of which take $O(\sqrt{p})$ time. There is also the index calculus algorithm [1], which runs in time $O\left(2^{O(\sqrt{\log q \log\log q})}\right)$. This is a subexponential attack, but still not polynomial time. These known attacks mean that if we want roughly 80 bits of security in this group, we need to set the size of p to at least 160 bits and the size of q to at least 1024 bits.

Naor and Reingold additionally show that the algebraic simplicity of their construction implies that interactive zero-knowledge proofs can be given for statements of the form "$y = f_s(x)$" and "$y \neq f_s(x)$" once the party computing the pseudorandom function f_s has committed to the key s [22]. Micali, Rabin, and Vadhan define and construct verifiable random functions [21], which have an even stronger property: the proofs of statements "$y = f_s(x)$" do not require interaction or a trusted shared random string. We will not be concerned with such properties for our construction, as our primary goals are computational efficiency and heightened security.

Organization In the next section, we formally define pseudorandom functions and the k-Linear Assumptions. We also establish a basic property shared by these assumptions that will be useful to us. In section 3, we give our construction based on the Linear Assumption (the case $k = 2$) and prove it is pseudorandom. In section 4, we generalize our construction to hold under the k-Linear Assumption for each positive integer k and summarize the generalized proof. In section 5, we summarize the important properties of our pseudorandom functions and analyze their performance.

2 Background

2.1 Formal Definition of Pseudorandom Functions

We give the definition that is provided in [22]:

Definition 1 *(efficiently computable pseudorandom function ensemble) Let $\{A_n, B_n\}$ be a sequence of domains and ranges and let $F = \{F_n\}_{n\in\mathbb{N}}$ be a function ensemble where each F_n is a random variable taking values in the set of functions from A_n to B_n. Then F is an efficiently computable pseudorandom function ensemble if it satisfies the following two conditions:*

1. for every probabilistic polynomial time oracle machine $\mathcal{M}$, every constant $c > 0$, and all but finitely many n's,

$$\left|Pr[\mathcal{M}^{F_n}(1^n) = 1] - Pr[\mathcal{M}^{R_n}(1^n) = 1]\right| < \frac{1}{n^c},$$

where R_n is uniformly distributed over the set of all functions from A_n to B_n, *2. there exist probabilistic polynomial time algorithms $\mathcal{I}$ and $\mathcal{V}$ and a mapping ϕ from strings to functions such that $\phi(\mathcal{I}(1^n))$ and F_n are identically distributed (so F_n can be efficiently sampled) and $\mathcal{V}(i,x) = (\phi(i))(x)$ (the sampled function can be efficiently computed).*

We note that oracle machine $\mathcal{M}$ in requirement 1 can know the description of the pseudorandom function ensemble, just not the key of the particular function it is querying.

2.2 The Decisional k-Linear Assumptions

We define the Linear problem in a cyclic group G of prime order p: given $g_0, g_1, g_2, g_1^{r_1}, g_2^{r_2}, g_0^{r_0} \in G$, distinguish whether $r_0 = r_1 + r_2$ or is random. The assumption is that no polynomial time algorithm can distinguish between these two cases of r_0 with non-negligible advantage. More formally,

Definition 2 *Linear problem in G: given $g_0, g_1, g_2, g_1^{r_1}, g_2^{r_2}, g_0^{r_0} \in G$, output "yes" if $r_0 = r_1 + r_2$ and "no" otherwise.*

The advantage of an algorithm A in deciding the Linear problem is defined to be:

$$Adv_A = \left| \begin{array}{l} Pr[A(g_0, g_1, g_2, g_1^{r_1}, g_2^{r_2}, g_0^{r_1+r_2}) = yes : g_0, g_1, g_2 \xleftarrow{R} G, r_1, r_2 \xleftarrow{R} \mathbb{Z}_p] \\ -Pr[A(g_0, g_1, g_2, g_1^{r_1}, g_2^{r_2}, g_0^{r_0}) = yes : g_0, g_1, g_2 \xleftarrow{R} G, r_1, r_2, r_0 \xleftarrow{R} \mathbb{Z}_p] \end{array} \right|.$$

(We use the notation $g_0, g_1, g_2 \xleftarrow{R} G$ to convey that these elements are chosen from G uniformly randomly.)

Definition 3 *Linear Assumption in G: no polynomial time algorithm can achieve non-negligible advantage in deciding the Linear problem in G.*

We now define a generalization of the Linear problem. For $k \geq 1$, the k-Linear problem in G is:

Definition 4 *k-Linear problem in G: given $g_1, g_2, \ldots, g_k, g_0, g_1^{r_1}, g_2^{r_2}, \ldots, g_k^{r_k}, g_0^{r_0} \in G$, output "yes" if $r_0 = \sum_{i=1}^{k} r_i$ and "no" otherwise.*

Definition 5 *k-Linear Assumption in G: no polynomial time algorithm can achieve non-negligible advantage in deciding the k-Linear problem in G.*

(The advantage of an algorithm A in deciding the k-Linear problem is defined analogously to the Linear case above.) We note that the 1-Linear problem is the Decisional Diffie-Hellman (DDH) problem and the 2-Linear problem is the Linear problem. The DDH problem is usually defined with different (though equivalent) notation:

Definition 6 *DDH problem: given g, g^a, g^b, T, output "yes" if $T = g^{ab}$ and "no" otherwise.*

These generalized assumptions are useful because they get weaker as k increases, in generic groups at least (this was proved in [16, 25]). In particular, some k-Linear Assumption might hold in a bilinear group G where the DDH Problem is easy. The k-Linear problem also has a helpful property that we will use later: given a single instance of the problem, one can randomly generate new instances. (This was proved for the DDH problem in [22].)

Lemma 7 *Given $g_1, \ldots, g_k, g_0, g_1^{r_1}, \ldots, g_k^{r_k}, g_0^{r_0} \in G$, one can generate $g_1^{r_1'}, \ldots, g_k^{r_k'}, g_0^{r_0'}$ such that $r_1', \ldots, r_k'$ are uniformly random in $\mathbb{Z}_p$ and $r_0' = \sum_{i=1}^k r_i'$ if $r_0 = \sum_{i=1}^k r_i$ and is uniformly random otherwise.*

Proof. We pick $e_0, e_1, \ldots, e_k \in \mathbb{Z}_p$ uniformly randomly. We then define $r_i' = e_0 r_i + e_i$ for i from 1 to k and $r_0' = e_0 r_0 + e_1 + \cdots + e_k$. We can also write this as:

$$\begin{pmatrix} r_1 & 1 & 0 & 0 & \ldots & 0 \\ r_2 & 0 & 1 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ r_k & 0 & 0 & \vdots & 0 & 1 \\ r_0 & 1 & 1 & \vdots & 1 & 1 \end{pmatrix} \begin{pmatrix} e_0 \\ e_1 \\ \vdots \\ e_{k-1} \\ e_k \end{pmatrix} = \begin{pmatrix} r_1' \\ r_2' \\ \vdots \\ r_k' \\ r_0' \end{pmatrix}.$$

If $r_0 = \sum_{i=1}^k r_i$, then $r_0' = \sum_{i=1}^k r_i'$ and $r_1', \ldots, r_k'$ are uniformly random. If $r_0 \neq \sum_{i=1}^k r_i$, then the square matrix on the left of the above expression has a non-zero determinant and is invertible over $\mathbb{Z}_p$. (One can easily show by induction on k that the determinant of this matrix is $(-1)^k(r_0 - r_1 - \cdots - r_k)$.) Thus, any $(k+1)$-tuple of values $r_1', \ldots, r_k', r_0' \in \mathbb{Z}_p$ can be uniquely formed by one setting of the values of $e_0, e_1, \ldots, e_k$. So $r_1', \ldots, r_k', r_0'$ are uniformly random. $\square$

Iterative application of this lemma allows us to generate any number of random instances of the k-Linear problem from a single instance. As was noted for the DDH case in [22], we can use this lemma and standard amplification techniques to show that the k-Linear problem is either hard on average or easy even in the worst case.

3 Our Core Construction

Intuition The key observation behind the Naor-Reingold construction is that the DDH Assumption implies the existence of a pseudorandom generator that doubles its input. We use the phrase pseudorandom generator somewhat loosely here, since their generator is not efficiently computable without knowledge of secret parameters. Specifically, a group element g^a is fixed, and this gives a generator $\tilde{G}_{g,g^a}$ which takes in one random group element, g^b, and outputs two pseudorandom group elements: $\tilde{G}_{g^a}(g^b) = (g^b, g^{ab})$. (To compute this pseudorandom generator efficiently, one needs to know either a or b, but this does not pose a problem for their construction.) This generator is then used along with the GGM construction (which nests n copies of the generator for an n-bit input) to yield the Naor-Reingold construction.

More formally, the GGM construction takes a pseudorandom generator G that doubles its input and writes its output $G(x)$ as $(G^0(x), G^1(x))$. A pseudorandom function $f_s : \{0,1\}^n \to \{0,1\}^n$ is then defined from a key s (which is a uniformly chosen n-bit string) by:

$$f_s(x) = G^{x_n}(\cdots(G^{x_2}(G^{x_1}(s)))\cdots).$$

Naor and Reingold modify this slightly by using a different g^a for the generator at each step:

$$f_{G,p,g,a_0,\ldots,a_n}(x) = \tilde{G}^{x_n}_{g^{a_n}}(\cdots(\tilde{G}^{x_1}_{g^{a_1}}(g^{a_0}))\cdots) = (g^{a_0})^{\prod_{x_i=1} a_i}.$$

We might hope to obtain a pseudorandom generator G from the Linear Assumption that doubles its input by defining $G_{g_0,g_1,g_2,g_1^{r_1}}(g_2^{r_2}) = (g_2^{r_2}, g_0^{r_1+r_2})$, but this does not work because fixing r_1 makes this generator fail if DDH fails. For example, suppose we receive four output pairs from this generator: $(A,A'), (B,B'), (C,C'), (D,D')$. We can set $g = A/B$, $g^b = C/D$, $g^a = A'/B'$, and $T = C'/D'$. If these pairs were uniformly random in G^2, then T would be uniformly random. Since these pairs come from our generator, we must have $A = g_2^{r_2^1}, A' = g_0^{r_1+r_2^1}, B = g_2^{r_2^2}, B' = g_0^{r_1+r_2^2}, C = g_2^{r_2^3}, C' = g_0^{r_1+r_2^3}, D = g_2^{r_2^4}, D' = g_0^{r_1+r_2^4}$ for some values $r_1, r_2^1, \ldots, r_2^4$. In this case, $g = g_2^{r_2^1-r_2^2}$, $b = \frac{r_2^3-r_2^4}{r_2^1-r_2^2}$, a satisfies $g_2^a = g_0$, and $T = g^{ab}$. This is a DDH tuple, so this generator is no more secure than DDH.

Instead, we can get a generator G' from the Linear Assumption which takes in two random group elements, $g_1^{r_1}, g_2^{r_2}$, and outputs 3 pseudorandom group elements:

$$G'_{g_0,g_1,g_2}(g_1^{r_1}, g_2^{r_2}) = (g_1^{r_1}, g_2^{r_2}, g_0^{r_1+r_2}).$$

Pseudorandomness for this generator under the Linear Assumption follows from Lemma 7. Since this generator does not double its input, we cannot simply plug it into the GGM construction in the way that the Naor-Reingold construction is obtained. There is a standard technique for taking a pseudorandom generator that only slightly stretches its input size and obtaining a new pseudorandom generator that (e.g.) doubles its input size, but the proof that this generic approach maintains pseudorandomness assumes that the pseudorandom generator is efficiently computable, which ours is not. (One needs to know r_1 and r_2 in order to compute it, and if one knows r_1 and r_2, it is no longer pseudorandom.)

To overcome this difficulty, we proceed as follows. We can rename $g_1^{r_1}$ as A, $g_2^{r_2}$ as B, and note that there exist $c, d \in \mathbb{Z}_p$ such that $g_1^c = g_0$ and $g_2^d = g_0$. We rename g_1 as g_c and g_2 as g_d to reflect this relationship (i.e. g_c is defined to be $g_0^{c^{-1}}$). In this notation, our pseudorandom generator G', can be written as:

$$G'_{g_0,g_c,g_d}(A,B) = (A, B, A^c B^d).$$

To modify this so that it doubles its input, we can simply fix additional values $e, f \in \mathbb{Z}_p$ and define:

$$\overline{G}_{g_0,g_c,g_d,g_e,g_f}(A,B) = (A, B, A^c B^d, A^e B^f).$$

One needs to know c, d, e, f in order to compute this generator, but this will not pose a problem for our construction. Perhaps more worrisome is that the Linear Assumption is no longer directly embedded in the generator. Nonetheless, we can use a hybrid argument to show that samples of outputs from this generator $\overline{G}$ are indistinguishable from random under the Linear Assumption. (This is accomplished by Lemma 10 in the proof of security for our pseudorandom function ensemble below.) We will use this pseudorandom generator along with the GGM construction (modifying it like Naor-Reingold) to obtain our construction.

3.1 Construction

We now construct our function ensemble $F = \{F_n\}_{n \in \mathbb{N}}$ and prove it is pseudorandom under the Linear Assumption. Each F_n is a set of functions from n-bit strings to a group G, where G is a cyclic group of prime order p generated by g. A function $f \in F_n$ is associated with a unique key consisting of G, g, p and $4n+2$ elements of $\mathbb{Z}_p$. More formally, we give a Setup algorithm that constructs one of our pseudorandom functions and an Evaluation algorithm that computes its value on a specified input string.

Setup$(\lambda) \rightarrow SK$ Our Setup algorithm takes in a security parameter λ and chooses a group G of prime order p which is large enough with respect to λ. It then chooses a generator g of G and $4n+2$ uniformly random elements of $\mathbb{Z}_p$, denoted by $y_0, z_0, y_1, z_1, w_1, v_1, \ldots, y_n, z_n, w_n, v_n$. It outputs:

$$SK = \{G, p, g, y_0, z_0, y_1, z_1, w_1, v_1, \ldots, y_n, z_n, w_n, v_n\}.$$

We describe the Evaluation algorithm for the function f associated with SK as follows. We let $x = x_1 x_2 \ldots x_n$ denote the input bit string.

Evaluation$(x, SK) \rightarrow f(x)$

```
Initialize variables $a$ and $b$ in $\mathbb{Z}_p$ as $a = y_0, b = z_0$.
For $i$ from 1 to $n$ do:
    If $x_i = 0$, set $a = a$ and $b = b$.
    If $x_i = 1$, set $a = ay_i + bz_i$ and $b = aw_i + bv_i$.
Output $f(x) = g^a$.
```

For the step when $x_i = 1$, the new values of a and b are both defined in terms of the old values of a and b, i.e. we do not first update a and then use the updated value of a in updating b. Technically, our pseudorandom functions deviate slightly from definition 1, since different functions in F_n will have different groups as their ranges. As Naor and Reingold note, this does not pose a problem in many applications. Alternatively, one can get all of the functions in F_n to have the same range by using suitable families of hash functions [22].

We note that our Evaluation algorithm is very efficient: it performs at most $4n$ multiplications in $\mathbb{Z}_p$, $2n$ additions in $\mathbb{Z}_p$, and one exponentiation in G. This construction is a generalization of the construction in [22], which was proven to be pseudorandom under the DDH Assumption. In the next section, we will further generalize the construction to be pseudorandom under the k-Linear Assumption for each $k \geq 1$. But first, we give a proof for this special case of $k = 2$.

3.2 Security

Theorem 8 *Under the Linear Assumption, this function ensemble is pseudorandom.*

Proof. We first note that f can be equivalently defined by the following (less efficient) algorithm:

Inefficient Evaluation$(x, SK) \to f(x)$

```
Initialize variables $A$ and $B$ in $G$ as $A = g^{y_0}, B = g^{z_0}$.
For $i$ from 1 to $n$ do:
    If $x_i = 0$, set $A = A$ and $B = B$.
    If $x_i = 1$, set $A = A^{y_i} B^{z_i}$ and $B = A^{w_i} B^{v_i}$.
Output $f(x) = A$.
```

We note that one would not actually compute f this way in practice, but using this algorithm yields the same function values as the more efficient algorithm we gave above. (To see this, note that A has replaced g^a and B has replaced g^b. We are simply performing exponentiations now as we go along instead of waiting until the end.) We describe f in this alternative way because it is more convenient for our proof, and it also reveals the relationship between our construction and the GGM construction.

We recall that we defined our pseudorandom generator $\overline{G}$ as:

$$\overline{G}_{g_0, g_c, g_d, g_e, g_f}(A, B) = (A, B, A^c B^d, A^e B^f).$$

We can now see that our construction is formed by using $\overline{G}$ in the GGM construction, where at each level we use a new c, d, e, f:

$$f(x) = \overline{G}^{x_n}_{g, g_{y_n}, g_{z_n}, g_{w_n}, g_{v_n}}(\cdots(\overline{G}^{x_1}_{g, g_{y_1}, g_{z_1}, g_{w_1}, g_{v_n}}(g^{y_0}, g^{z_0}))\cdots).$$

We prove this is a pseudorandom function family using the hybrid technique. We begin by defining a sequence of games: Game 0, Game 1, ..., Game n. Each game consists of a challenger and an attacker. The attacker can query the challenger for values of a function on inputs $x = x_1 \ldots x_n$ that it chooses.

Game j The challenger fixes random values

$$y_{j+1}, z_{j+1}, w_{j+1}, v_{j+1}, \ldots, y_n, z_n, w_n, v_n \in \mathbb{Z}_p.$$

The challenger answers queries for input x's by setting A and B to be random functions of the first j bits of the input and then following the iterative procedure above for i from $j+1$ to n. It then outputs the final value of A as the answer to the query. The attacker must output either 0 or 1.

We note that in Game 0, the challenger is implementing a function from our function family and in Game n, the challenger is implementing a truly random function. Thus, our function family is pseudorandom if and only if Game 0 cannot be distinguished from Game n (with non-negligible advantage) by any polynomial time attacker.

So if our functions are not pseudorandom, then there exists a probabilistic algorithm D which can distinguish Game 0 from Game n. In other words, we suppose

$$|Pr[D = 1|\text{Game } n] - Pr[D = 1|\text{Game } 0]| = \epsilon,$$

where $\epsilon > 0$ is non-negligible. We then observe:

$$\epsilon = \left| \sum_{j=0}^{n-1} Pr[D = 1|\text{Game } j+1] - Pr[D = 1|\text{Game } j] \right|.$$

By the triangle inequality, this implies that there exists j such that:

$$|Pr[D = 1|\text{Game } j+1] - Pr[D = 1|\text{Game } j]| \geq \frac{\epsilon}{n}.$$

Since ϵ is non-negligible, $\frac{\epsilon}{n}$ is non-negligible. So we can assume that D also distinguishes some pair of adjacent games j and $j+1$.

The heart of our construction is the pseudorandom generator which takes input A, B and expands it to 4 elements: $A, B, A^{y_i}B^{z_i}, A^{w_i}B^{v_i}$. We will first show that if our construction is not pseudorandom, then this generator is not pseudorandom either. More precisely, we will show that if there is a probabilistic algorithm D able to distinguish Game j from Game $j+1$ for some j, then there is another probabilistic algorithm M which can distinguish samples of the form $(A, B, A^{y_i}B^{z_i}, A^{w_i}B^{v_i})$ from uniformly random 4-tuples. To state this formally, we define two additional games. We call them Game 1 and Game 3 because they will later appear as parts of a three game hybrid.

Game 1 An attacker queries the challenger for 4-tuples in G. Each time the challenger responds by sending a new 4-tuple (A, B, C, D) uniformly chosen in G^4.

Game 3 An attacker queries the challenger for 4-tuples in G. The challenger chooses elements y, z, w, v uniformly from $\mathbb{Z}_p$ and keeps these fixed. For each query, the challenger chooses elements A, B uniformly from G and responds with (A, B, A^yB^z, A^wB^v).

Lemma 9 *Suppose there is an algorithm D such that*

$$|Pr[D = 1|\textit{Game } j+1] - Pr[D = 1|\textit{Game } j]| \geq \epsilon.$$

Then there exists an algorithm M such that

$$|Pr[M = 1|\textit{Game } 1] - Pr[M = 1|\textit{Game } 3]| \geq \epsilon.$$

Proof. Suppose that M receives t 4-tuples $(A_1, B_1, C_1, D_1), \ldots, (A_t, B_t, C_t, D_t)$, where t is an upper bound on the number of queries that D makes. These 4-tuples are either all uniformly random in $G \times G \times G \times G$, or A_i, B_i are uniformly random and $C_i = A_i^yB_i^z, D_i = A_i^wB_i^v$ for all i. (Note that y, z, w, v are fixed and do not change with i.) It is M's task to distinguish between these two cases. M will accomplish this task by simulating the challenger in Game j or Game $j+1$ and calling on D.

M starts the simulation by choosing $y_{j+2}, z_{j+2}, w_{j+2}, v_{j+2}, \ldots, y_n, z_n, w_n, v_n \in \mathbb{Z}_p$ randomly. When D queries M with input $x = x_1 \ldots x_n$, M defines $\ell(x) : \{0,1\}^j \rightarrow [t]$ to be an injective function of the first j bits of x. If $x_{j+1} = 0$, M sets $A = A_{\ell(x)}$ and $B = B_{\ell(x)}$ (these values are taken from the 4-tuples M has been given). If $x_{j+1} = 1$, M sets $A = C_{\ell(x)}$ and $B = D_{\ell(x)}$. M then follows the iterative procedure from our construction for steps $j+2$ to n and outputs the final value of A.

We note that if the 4-tuples M has received are uniformly random, then M has simulated game $j+1$. If instead $C_i = A_i^yB_i^z, D_i = A_i^wB_i^v$ for all i, then M has simulated game j with $y_{j+1} = y$, $z_{j+1} = z$, $w_{j+1} = w$, and $v_{j+1} = v$. So if M outputs 1 when D outputs 1, we have:

$$|Pr[M = 1|\text{Game } 1] - Pr[M = 1|\text{Game } 3]|$$

$$= |Pr[D = 1|\text{Game } j+1] - Pr[D = 1|\text{Game } j]| = \epsilon.$$

□

To complete our proof of Theorem 8, we show that the existence of such an M violates the Linear Assumption. To do this, we once again use the hybrid technique. This time, we only need 3 games. Games 1 and 3 are defined as before, but we include them below for completeness.

Game 1 M is given samples of the form (A, B, C, D) which are uniformly random in G^4.

Game 2 M is given samples of the form $(A, B, A^y B^z, D)$, where A, B, D are uniformly random in G and y, z are fixed.

Game 3 M is given samples of the form $(A, B, A^y B^z, A^w B^v)$ where A, B are uniformly random in G and y, z, w, v are fixed.

We suppose that

$$|Pr[M = 1|\text{Game 1}] - Pr[M = 1|\text{Game 3}]| = \epsilon.$$

This means that at least one of the following must hold:

$$(1) \qquad |Pr[M = 1|\text{Game 1}] - Pr[M = 1|\text{Game 2}]| \geq \frac{\epsilon}{2},$$

$$(2) \qquad |Pr[M = 1|\text{Game 2}] - Pr[M = 1|\text{Game 3}]| \geq \frac{\epsilon}{2}.$$

Lemma 10 *If either (1) or (2) holds, then there exists an algorithm N with $Adv_N \geq \frac{\epsilon}{2}$ in deciding the Linear problem.*

Proof. We suppose that (1) holds (i.e. M can distinguish between Game 1 and Game 2). We will show how to define the algorithm N to break the Linear Assumption. The Linear challenger gives N an instance $g_0, g_1, g_2, g_1^{r_1}, g_2^{r_2}, g_0^{r_0} \in G$ of the Linear problem. N calls M. Each time M requests a 4-tuple, N uses Lemma 1 to generate a new instance of its Linear problem. From this instance, $g_0, g_1, g_2, g_1^{r'_1}, g_2^{r'_2}, g_0^{r'_0} \in G$, N creates a 4-tuple (A, B, C, D) by setting $A = g_1^{r'_1}$, $B = g_2^{r'_2}$, $C = g_0^{r'_0}$, and setting D randomly. If the original r_0 is uniformly random, then the Linear attacker has simulated Game 1. If $r_0 = r_1 + r_2$, then the Linear attacker has simulated Game 2 with y and z such that $g_1^y = g_0$ and $g_2^z = g_0$. (Note that these values of y and z are uniformly random in $\mathbb{Z}_p$ because g_0, g_1, g_2 were chosen uniformly randomly from G.) Hence, if N outputs "yes" when M outputs 1, we will have:

$$\left| Pr[N = \text{“yes”}|r_0 \xleftarrow{R} \mathbb{Z}_p] - Pr[N = \text{“yes”}|r_0 = r_1 + r_2] \right|$$

$$= |Pr[M = 1|\text{Game 1}] - Pr[M = 1|\text{Game 2}]| \geq \frac{\epsilon}{2}.$$

Similarly, if (2) holds (i.e. M can distinguish between Game 2 and Game 3), then N sets y and z randomly and generates 4-tuples (A, B, C, D) by setting $A =$

$g_1^{r'_1}, B = g_2^{r'_2}, C = A^y B^z, D = g_0^{r'_0}$. If r_0 is random, this is Game 2. If $r_0 = r_1 + r_2$, this is Game 3. In both cases, we have shown that we obtain an N such that

$$\left| Pr[N = \text{“yes”} | r_0 \xleftarrow{R} \mathbb{Z}_p] - Pr[N = \text{“yes”} | r_0 = r_1 + r_2] \right| \geq \frac{\epsilon}{2}.$$

□

Putting it all together, we have shown that if our function ensemble is not pseudorandom, then there exists a probabilistic algorithm N which has non-negligible advantage in deciding the Linear problem. Hence, if the Linear Assumption holds, our function ensemble is pseudorandom. This completes our proof of Theorem 8. □

4 Our Generalized Construction

4.1 Construction

We now generalize the construction of the previous section to create a function ensemble which is pseudorandom under the k-Linear Assumption, for each $k \geq 1$. We note that for $k = 1$, this is precisely the construction of [22] which was proven under the DDH Assumption (a.k.a. the 1-Linear Assumption). We will denote our function ensemble by $F_k = \{F_n\}$. Each function in F_n is a function from $\{0,1\}^n$ to a group G of prime order p generated by g. The key specifying a function $f \in F_n$ consists of G, p, g and $k^2 n + k$ elements of $\mathbb{Z}_p$:

Setup$(\lambda) \rightarrow SK$ Our Setup algorithm takes in a security parameter λ and chooses a group G of prime order p which is large with respect to λ. It then chooses a generator g of G and $k^2 n + k$ uniformly random elements of $\mathbb{Z}_p$, denoted by $c_m, b^i_{m,\ell}$ where $1 \leq m, \ell \leq k$, and $1 \leq i \leq n$. It outputs:

$$SK = \{G, p, g, c_m, b^i_{m,\ell} : i \in [n], (m, \ell) \in [k] \times [k]\}.$$

Here, $[n]$ denotes the set $\{1, 2, \ldots, n\}$ and $[k]$ denotes the set $\{1, 2, \ldots, k\}$. (The i's are superscripts, and do not denote exponentiations.) To define and compute $f(x)$ for the function f corresponding to SK, we use the following Evaluation algorithm:

Evaluation$(x, SK) \rightarrow f(x)$

```
Initialize variables $a_1 = c_1, \ldots, a_k = c_k$ in $\mathbb{Z}_p$.
For $i$ from 1 to $n$ do:
    If $x_i = 0$, set $a_m = a_m$ for each $m \in [k]$.
    If $x_i = 1$, set $a_m = \sum_{\ell=1}^{k} a_\ell b^i_{m,\ell}$ for each $m \in [k]$.
Output $f(x) = g^{a_1}$.
```

We note that the output is always one group element while the key size grows quadratically in k. To compute the a value $f(x)$, we only need to do one exponentiation in G, perform $\leq k^2 n$ multiplications in $\mathbb{Z}_p$, and $\leq (k-1)kn$ additions in $\mathbb{Z}_p$.

4.2 Security

Theorem 11 *For each $k \geq 1$, if the k-Linear Assumption holds, then F_k is a pseudorandom function ensemble.*

Proof. The proof is essentially the same as the proof of Theorem 8. Again, we first give an equivalent definition of f using a less efficient algorithm. This would not be used in practice, but is more convenient for use in the proof.

Inefficient Evaluation$(x, SK) \to f(x)$

```
Initialize variables $A_1 = g^{c_1}, \ldots, A_k = g^{c_k}$ in $G$.
For $i$ from 1 to $n$ do:
    If $x_i = 0$, set $A_m = A_m$ for each $m \in [k]$.
    If $x_i = 1$, set $A_m = \prod_{\ell=1}^{k} A_\ell^{b^i_{m,\ell}}$ for each $m \in [k]$.
Output $f(x) = A_1$.
```

We define a sequence of games, Game 0 through Game n. Each game has a challenger and an attacker who makes function queries.

Game j The challenger fixes values $b^i_{m,\ell} \in \mathbb{Z}_p$ for $(m,\ell) \in [k] \times [k]$ and i from $j+1$ to n. To respond to a query, the challenger sets $A_1, \ldots, A_k$ as a random function of the first j bits of the input and then follows the iterative procedure above for bits $j+1$ to n.

If our function family F_n is not pseudorandom, this implies there exists an attacker D who can distinguish between two consecutive games j and $j+1$ with non-negligible advantage.

Using the same argument as in the proof of Theorem 8, we note that D can be used to define an attacker M who receives $2k$-tuples of the form

$$(A_1, \ldots, A_k, B_1, \ldots, B_k),$$

where either these tuples are uniformly chosen from G^{2k} or $(A_1, \ldots, A_k)$ is uniformly chosen from G^k and each $B_m = \prod_{\ell=1}^{k} A_\ell^{b_{m,\ell}}$ for fixed values $b_{m,\ell}$. M will distinguish between these two cases with non-negligible advantage.

We will show that such an M can be used to break the k-Linear Assumption using a hybrid argument. We define new Games 0 through k.

Game j M is given input tuples of the form

$$(A_1, \ldots, A_k, \tilde{B}_1, \ldots, \tilde{B}_j, B_{j+1}, \ldots, B_k),$$

where A_m's and B_m's are chosen uniformly randomly from G and $\tilde{B}_m$ is equal to $\prod_{\ell=1}^{k} A_\ell^{b_{m,\ell}}$ for some fixed values $b_{m,\ell}$.

Since M distinguishes between Game 0 and Game k, it must distinguish between some Game j and Game $j+1$ with non-negligible advantage (assuming k is polynomial in the security parameter).

A k-Linear attacker can then use M as follows. First, upon receiving one instance of the k-Linear Problem from the k-Linear challenger,

$$g_0, g_1, \ldots, g_k, g_1^{r_1}, \ldots, g_k^{r_k}, g_0^{r_0},$$

the k-Linear attacker generates t instances with the same $g_0, \ldots, g_k$ but different $r'_0, \ldots, r'_k$ values using Lemma 2 repeatedly. The i^{th} instance is then used to generate the i^{th} tuple to send to M. In the tuple, $A_m = g_m^{r'_m}$ for m from 1 to k. Then $b_{m,\ell}$ are chosen randomly from $\mathbb{Z}_p$ for ℓ from 1 to k and m from 1 to j. For m from 1 to j, $\tilde{B}_m = \prod_{\ell=1}^{k} A_\ell^{b_{m,\ell}}$. The $j+1$ element of the tuple is set to $g_0^{r'_0}$. Elements $B_{j+2}, \ldots, B_k$ are then chosen randomly from G.

If $r_0 = r_1 + \cdots + r_k$, then the $j+1$ element of the tuple is $\tilde{B}_{j+1} = \prod_{\ell=1}^{k} A_\ell^{b_{j+1,\ell}}$, where $b_{j+1,\ell}$ satisfies $g_\ell^{b_{j+1,\ell}} = g_0$. If r_0 is random, then the $j+1$ element of the tuple is a random element B_{j+1}. Hence, when M correctly distinguishes between Game j and Game $j+1$, it will allow the k-Linear attacker to distinguish $r_0 = r_1 + \cdots + r_k$ from random. Thus, our function ensemble is pseudorandom under the k-Linear Assumption. $\square$

5 Discussion and Performance

The primary advantage of our construction is an increase in security with only a small cost in efficiency. We summarize the relevant properties of our construction in the following table (with domain $\{0,1\}^n$ under the k-Linear Assumption):

Computation time	Private key storage	Computational storage
1 exponentiation in G + k^2n multiplications in $\mathbb{Z}_p$	G, g, p + $k^2n + k$ elements of $\mathbb{Z}_p$	$2k$ elements of $\mathbb{Z}_p$ + $\log p$ elements of $\mathbb{Z}_q$

Table 1: Properties of our construction

The $2k$ elements of $\mathbb{Z}_p$ in the computational storage are used to retain all of the old values of the a_i's in our efficient algorithm while we are computing the updated ones. The $\log p$ elements of $\mathbb{Z}_q$ come from preprocessing: we compute and store the values of $g^{2^i} \bmod q$ for i from 0 to $\log p$ and use these to speed up the final exponentiation. We do not include the private key in our computational storage because we have listed it separately.

We implemented our construction in a subgroup $< g >$ of order p in $\mathbb{Z}_q^*$, where q is a 1024-bit prime and p is a 160-bit prime dividing $q-1$. We used 160-bit inputs (i.e. $n = 160$). We chose a random key and computed our function on randomly chosen inputs. The following table shows our running times as a function of the parameter k. If we approximate these times with a quadratic polynomial using a least squares fit, we get the polynomial $441.4 + 1.57k + 39.57k^2$. Our table also demonstrates how closely our times mimic this function:

Our implementation was programmed in C using the GNU Multiple-Precision Library (GMP). It was run on an Intel Core 2 6600 2.40GHz PC running the Ubuntu (Linux-based) operating system and compiled with gcc 4.2.4. The efficient Evaluation algorithm given above is essentially pseudocode for our implementation

k	microseconds	$441.4 + 1.57k + 39.57k^2$
1	482	483
2	605	603
3	799	802
4	1083	1081
5	1438	1439

Table 2: Running times for $n = 160$ and 1 function evaluation

(we also pre-computed the values of g^{2^i} for i from 0 to $\log p$ to speed up the final exponentiation). These actual times are less important than the general behavior they demonstrate: for small values of k, the quadratic growth in the computation time is rather muted by the constant factor, so the increase in running time caused by increasing k is very mild. In particular, one can increase k from 1 to 2 to rely on the Linear Assumption instead of DDH, and the running time is only increased by a factor of roughly 1.255.

We note that the output of our construction (based on the k-Linear Assumption) could be expanded to k elements, $A_1 = g^{a_1}, \ldots, A_k = g^{a_k}$. Pseudorandomness still holds for this larger output by the same proof. There is a cost in efficiency: computing the larger output takes k exponentiations in G instead of just one. However, if one needs to generate more pseudorandom output elements, then it is more efficient to use this version with k outputs instead of computing the version with one output on k different inputs. This is because the computations in $\mathbb{Z}_p$ are then done only once instead of k times.

6 Conclusion

We have constructed relatively efficient pseudorandom functions and proven their security under the progressively weaker k-Linear Assumptions. Our proof relies on a novel application of two hybrid arguments to accomodate the weaker assumptions. An increase in the value of k leads to an increase in security (for generic groups at least) and only a mild quadratic increase in running time and private key size. Thus, we have made progress towards the important goal of providing provably secure alternatives to ad hoc constructions without sacrificing too much efficiency.

7 Acknowledgements

We thank Drake Dowsett for help with programming our implementation.

References

[1] L. Adleman. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. In *Proceedings of the 20th IEEE Foundations of Computer Science Symposium*, volume 2656, 1979.

[2] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *Advances in Cryptology - CRYPTO '96*, volume 1109 of *LNCS*, pages 1–16. Springer, 1996.

[3] M. Bellare and S. Goldwasser. New paradigms for digital signatures and message authentication based on non-interactive zero knowledge proofs. In *Advances in Cryptology - CRYPTO '89*, volume 435 of *LNCS*, pages 194–211. Springer, 1990.

[4] E. Biham, R. Chen, A. Joux, P. Carribault, C. Lemuet, and W. Jalby. Collisions of sha-0 and reduced sha-1. In *Advances in Cryptology - EUROCRYPT 2005*, LNCS.

[5] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *Advances in Cryptology - CRYPTO 2004*, volume 3152 of *LNCS*, pages 41–55. Springer, 2004.

[6] S. Brands. An efficient off-line electronic cash system based on the representation problem. 1993.

[7] G. Brassard. Modern cryptology. volume 325 of *LNCS*. Springer, 1988.

[8] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Advances in Cryptology - CRYPTO '98*, volume 1462 of *LNCS*, pages 13–25. Springer, 1998.

[9] T. Dierks and C. Allen. The tls protocol version 1.0. rfc 2246. January 1999.

[10] W. Diffie and M. Hellman. New directions in cryptography. In *IEEE Transactions in Information Theory*, volume 22, pages 644–654, 1976.

[11] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology - CRYPTO ' 84*, volume 196 of *LNCS*, pages 10–18. Springer, 1985.

[12] O. Goldreich. Two remarks concerning the goldwasser-micali-rivest signature scheme. In *Advances in Cryptology - CRYPTO '84*, volume 263 of *LNCS*, pages 104–110. Springer, 1987.

[13] O. Goldreich, S. Goldwasser, and S. Micali. On the cryptographic applications of random functions. In *Advances in Cryptology - CRYPTO '84*, volume 196 of *LNCS*, pages 276–288. Springer, 1985.

[14] O. Goldriech, S. Goldwasser, and S. Micali. How to construct random functions. In *Journal of the ACM*, volume 33, pages 792–807, 1986.

[15] J. Hastad, R. Impagliazzo, L. A. Levin, and M. Luby. Construction of a pseudo-random generator from any one-way function. In *SIAM Journal on Computing*, volume 28, pages 1364–1396, 1999.

[16] D. Hofheinz and E. Kiltz. Secure hybrid encryption from weakened key encapsulation. In *Advances in Cryptology - CRYPTO 2007*, volume 4622 of *LNCS*, pages 553–571. Springer, 2007.

[17] A. Joux and K. Nguyen. Separating decision diffie-hellman from computational diffie-hellman in cryptographic groups. In *Journal of Cryptology*, volume 16, pages 239–247, September 2003.

[18] D. E. Knuth. In *The Art of Computer Programming*, volume 3, pages 575–576, 1973.

[19] M. Luby. In *Pseudo-randomness and applications.* Princeton University Press, 1996.

[20] A. Menezes, T. Okamoto, and S. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. In *IEEE Transactions on Information Theory*, volume 39, pages 1639–1646, 1993.

[21] S. Micali, M. Rabin, and S. Vadhan. Verifiable random functions. In *Proceedings of 40th Annual Symposium on Foundations of Computer Science*, pages 120–130, 1999.

[22] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th Annual Symposium on Foundations of Computer Science*, pages 458–467, 1997.

[23] J. Pollard. Monte carlo methods for index computations $(\bmod p)$. In *Mathematics of Computation*, volume 32, pages 918–924, 1978.

[24] A. Razborov and S. Rudich. Natural proofs. In *Journal of Computer and System Sciences*, volume 55, pages 24–35, 1997.

[25] H. Shacham. A cramer-shoup encryption scheme from the linear assumption and from progressively weaker linear variants. 2007.

[26] M. Stadler. Publicly verifiable secret sharing. In *Advances in Cryptology - EUROCRYPT '96*, volume 1070 of *LNCS*, pages 190–199. Springer, 1996.

[27] L. Valiant. A theory of the learnable. In *Communications of the ACM*, volume 27, pages 1134–1142, 1984.

[28] X. Wang and H. Yu. How to break md5 and other hash functions. In *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 19–35. Springer, 2005.