Algebraic frameworks for pseudorandom functions

Alain Passelègue

THÈSE DE DOCTORAT

de l'Université de recherche Paris Sciences et Lettres
PSL Research University

Préparée à l'École normale supérieure

Algebraic Frameworks for Pseudorandom Functions

École doctorale n°386
Sciences Mathématiques de Paris Centre

Spécialité Informatique

COMPOSITION DU JURY

M. FERREIRA ABDALLA Michel
École normale supérieure
Directeur de thèse

M. CASH David
Rutgers University
Rapporteur

M. HOFHEINZ Dennis
Karlsruher Institut für Technologie
Rapporteur

Mme CHEVALIER Céline
Université Panthéon-Assas
Membre du jury

M. FOUQUE Pierre-Alain
Université de Rennes 1 et Institut Universitaire de France
Président du jury

M. ISHAI Yuval
Technion et UCLA
Membre du jury

M. PATERSON Kenneth G.
Royal Holloway, University of London
Membre du jury

M. GÉRARD Benoît
DGA et IRISA
Invité

Soutenue par
Alain PASSELÈGUE
le 9 décembre 2016

Dirigée par
Michel FERREIRA ABDALLA

Algebraic Frameworks for Pseudorandom Functions

Alain Passelègue

Supervisor: Michel Ferreira Abdalla

Abstract

In this thesis, we study the algebraic structure underlying number-theoretic pseudorandom functions. Specifically, we define an algebraic framework that translates the pseudorandomness of a particular form of functions into a simple algebraic property. The resulting generic framework encompasses most of existing constructions and naturally extends to related primitives, such as related-key secure, aggregate, and multilinear pseudorandom functions.

This framework holds under a family of MDDH assumptions, that contains especially different classical assumptions, such as DDH, DLin, or k-Lin. Therefore, setting accordingly the parameters in the constructions, our framework can be used to construct secure pseudorandom functions (or related primitives) whose security holds under these assumptions.

Finally, we also study more specifically the case of related-key security. On the one hand, we propose a fix and some extensions to the Bellare-Cash framework and then build pseudorandom functions secure against larger classes of attacks. On the other hand, we construct the first pseudorandom function provably secure against XOR attacks. The latter construction relies on the existence of a weak form of multilinear maps.

Contents

Chapter 1

Introduction

Since the early ages of "cryptography", the most central and basic problem has been to provide solutions for secret communication over insecure communication channels. The problem can easily be defined as follows: two parties, for instance a source and a journalist, want to communicate with each other while keeping a third party, called the adversary, as ignorant as possible about their discussion. Intuitively, an *encryption scheme* is a protocol designed towards solving this problem, and consists in two procedures, termed *encryption* and *decryption*. Assuming the source wants to send a message to the journalist, it first applies the encryption algorithm to the message in order to compute an encrypted message, called *ciphertext*, that it next sends to the journalist over a (possibly public) communication channel. In particular, it is reasonable to assume that the adversary also knows the ciphertext. Finally, the journalist applies the decryption algorithm to the ciphertext to recover the original message.

As the ciphertext could be known by the adversary, it is obvious that the journalist must know something that the adversary does not. Otherwise, the adversary could just proceed in the same manner as the journalist and get the message. We call this extra knowledge the *secret key*, and assume without loss of generality that the secret key is an argument that is fed, along with the ciphertext, to the decryption procedure, which can then be considered to be a public procedure. We further assume that this secret key is shared by the source and the journalist. This corresponds to the case of *private-key* encryption (a.k.a. *symmetric* encryption).

Now that is it clear that there must exist some secret key, the main question remains:

Which type of secrecy can we hope for?

The best we can hope is that the ciphertext *does not reveal any information* about the message. In this sense, the ciphertext must not contain any information about the message. This security notion is called *information-theoretic* or *perfect* security. It might seem surprising that this perfect security can be achieved, and is actually achieved by an elementary scheme, called the *one-time-pad* encryption. Intuitively, the scheme follows this basic idea: assume the source wants to send secretly to the journalist a time to meet. Then, they share a secret key that is just a time taken uniformly at random, for instance 17:42. When the source wants to send the time to meet to the journalist, for instance 08:30, it simply sends the sum of the two times, which is 02:12. The journalist can then recover 08:30 by subtracting 17:42 to 02:12.

The key point here is that for any ciphertext, there exists one and only one pair of message/secret key that produces this ciphertext. For instance, if the message had been 14:53 and not 08:30, then 02:12 would still have been a valid encryption with the secret key 11:19. Hence, as the secret key was chosen uniformly at random and could have been any time, the ciphertext does not reveal any information about the message, which could also have been any time. In other words, to encrypt a message, one masks it with a message of the same form, such that the mask hides any information about the message, and such that one can remove the mask if one knows it.

The main drawback of this encryption scheme is that the size of the secret key (mask) is the same as the total size of the messages being exchanged. This is actually a necessary condition:

Perfect security requires a key at least as long as the total length of the messages sent.

The above statement immediately implies drastic limitations on the practicality of such encryption: one needs a huge key if one wants to send a long message, and reversely, the number of messages is directly limited by the size of the secret key. Finally, as the key has to be perfectly random, it might be hard to generate a long key.

Thus, it seems that cryptography falls short to both guarantee security while achieving practicality. To put the above statements in a nutshell, either you get perfect security at the cost of impracticality, or you have a practical encryption scheme, but do not achieve perfect security. As we aim for practicality, the natural question is then the following:

Do we need perfect security?

While by definition one cannot hope for a stronger security model, there is one point that could be weakened in the model without impacting much the security. Indeed, perfect security asks that a *ciphertext does not contain any information about the message*. This is ideal, but in the real world, adversaries have limited resources (time, memory, power, . . .). Therefore, the question is not really whether a ciphertext does or does not contain any information about the message, but rather if *information about the message can be efficiently extracted from the message*. In other words, the question is not whether it is *possible* to extract information from the message but rather if it is *feasible*.

For example, let us consider the case in which a source wants to send 10 different times to meet up with a journalist. As explained above, in order to achieve perfect security, they have to share 10 masks, so 10 uniformly random times, which are then used to mask the 10 real times. However, assume that the journalist and the source share a small secret and know a procedure that allows both of them to privately and efficiently generate 10 masks. Then, if it is *infeasible* for an adversary to distinguish the 10 times generated using their small shared secret from 10 uniformly random times, the adversary cannot learn anything about the real times. Such values that one cannot distinguish from truly random values are called *pseudorandom* values and are a central notion of our work, but before giving further details, we need to address the following question:

What is or is not feasible by an adversary?

Modern cryptography relies on the belief that there are problems that can be solved efficiently, while others cannot. In particular, one always assumes $\mathbf{P} \neq \mathbf{NP}$ (and even

$\mathbf{NP} \not\subseteq \mathbf{BPP}$), which states intuitively that the set of problems whose solutions can be efficiently verified is not a subset of the set of problems that can be solved efficiently. That is, one requires that there exist problems such that it is hard to find a solution, but for which, given a solution, it is easy to verify its correctness. Loosely speaking, in terms of encryption, we want the decryption of a ciphertext given the secret key to be an efficient procedure, while decrypting the message without the secret key being a hard problem. Whether $\mathbf{P} \neq \mathbf{NP}$ (or $\mathbf{NP} \not\subseteq \mathbf{BPP}$) is still an open question, and probably the most important open question in theoretical computer science, an overwhelming proportion of the community believes it is the case. Therefore, as cryptographers, we would like to construct encryption schemes proven secure assuming this assumption. That is, we would like to design cryptosystems and prove that if an adversary can break the scheme, then it can solve hard problems (which is unlikely).

Unfortunately, it appears that this is not a sufficient assumption for building encryption. Indeed, this assumption only states that there exist easy-to-verify but hard-to-solve problems. However, a problem is hard to solve as soon as there exist some instances that are hard to solve. Thus, a problem being hard does not provide any guarantee that most instances nor average instances of the problem are hard. Intuitively, basing the security of an encryption scheme on the hardness of such problems could lead to an encryption scheme for which some ciphertexts are hard to decrypt without the knowledge of the secret key, but for which it could still be easy to decrypt most ciphertexts. Hence, we actually need the existence of problems that are hard on the average, and not only in the worst case.

The existence of such problems is not known to be implied by $\mathbf{P} \neq \mathbf{NP}$ (nor by $\mathbf{NP} \not\subseteq \mathbf{BPP}$), and actually, this is still not good enough for having good encryption schemes. Indeed, assuming only the existence of hard-on-average problems, it might be easy to build an encryption scheme for which it is hard to decrypt ciphertexts, but even for a legitimate user (that knows the secret key), which would thus have no advantage in comparison to adversaries. Specifically, a problem being hard on average does not mean that it is easy to construct hard instances (ciphertexts that are hard to decrypt) together with some auxiliary information (the secret key) that allows to efficiently solve this hard instance (decrypt the ciphertexts). However, the latter is precisely what we would like for building an encryption scheme.

This assumption is actually translated by the existence of one-way functions. A one-way function is simply a function that is easy to compute, but hard (on the average) to invert. In regard of the above, intuitively, we are interested in one-way functions as a solution to create hard instances of a problem (the output of the one-way function) and auxiliary information (the input of the one-way function), such that given the auxiliary information, the instance is easy to solve, but given only the instance, one cannot recover the auxiliary information (and thus not solve the instance either). For now, it is not known whether the existence of hard-on-average problems (nor whether $\mathbf{P} \neq \mathbf{NP}$ or $\mathbf{NP} \not\subseteq \mathbf{BPP}$) implies the existence of one-way functions, but once again, the majority of the community believes that one-way functions exist.

It appears that basing cryptography on the existence of one-way functions offers the possibility of designing secure symmetric encryption schemes (and much more) even if the key is much shorter than the total length of the messages sent, and in particular, the size of the key does not limit the number of messages that can be sent. In particular, going back to the example of agreeing on different times, if one-way functions exist, there exist procedures that allow to generate many *pseudorandom* values from a small shared secret. These procedures, termed *pseudorandom functions*, are one of the most central primitives in modern cryptography, and are the main focus of this thesis. In particular, pseudorandom functions have a fundamental role in both theory and practice. Let us now jump into further details by defining more properly pseudorandom functions.

1.1 Pseudorandom Functions

A pseudorandom function is a family of *deterministic* functions indexed by *short keys*. For each key is associated a function that is *efficiently computable* on *many inputs* and such that, without the knowledge of the key defining the function, it is computationally hard to distinguish the outputs of the function from truly random values from the same range, even if the adversary chooses the inputs on which the function is evaluated.

More formally, let $F\colon \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ be a family of functions, indexed by the key space $\mathcal{K}$, so for each key $K \in \mathcal{K}$ is associated a function $F(K, \cdot)\colon \mathcal{D} \to \mathcal{R}$. We say that F is a pseudorandom function if for any polynomial-time adversary, its advantage in the following game is negligible. The game starts by picking a random challenge bit b, a random target key $K \in \mathcal{K}$ and a random function $G\colon \mathcal{D} \to \mathcal{R}$. The adversary can repeatedly query an oracle that, given an input $x \in \mathcal{D}$, returns either $F(K, x)$, if $b = 1$, or $G(x)$, if $b = 0$. Finally, the adversary outputs a bit b', and its advantage is defined by $2\Pr\left[\, b = b' \,\right] - 1$.

Pseudorandom functions play a central role in cryptography, both in theory and in practice. Specifically, on the theoretical side, combining results by Goldreich and Levin [GL89] and by Goldreich, Goldwasser, and Micali [GGM84], we obtain the following result:

Pseudorandom functions exist if and only if one-way functions exist.

On the practical side, due to their simplicity and security properties, pseudorandom functions have been used in numerous applications, including symmetric encryption, authentication, and key exchange. In particular, pseudorandom functions yield simple constructions of symmetric encryption schemes, and this observation is widely used in practice (often implicitly). Furthermore, since pseudorandom functions can be used to model real-world block-ciphers, such as AES [01], they are also extremely useful for the security analysis of protocols that rely on these primitives.

Therefore, the study of pseudorandom functions is at the core of modern cryptography. We now introduce in more details the scope of our work, by first explaining the above statement, and then detailing our case of study.

1.1.1 From One-Way Functions to Pseudorandom Functions

Before the notion of pseudorandom functions was introduced by Goldreich, Goldwasser, and Micali in [GGM84], cryptographers studied the notion of pseudorandom generator. Broadly speaking, a pseudorandom generator with ℓ-bit stretch is a function $G : \{0,1\}^\kappa \to \{0,1\}^{\kappa+\ell}$, with $\ell \geq 1$, that takes as input a bitstring $s \in \{0,1\}^\kappa$, called the seed, and outputs a longer bitstring $G(s) \in \{0,1\}^{\kappa+\ell}$. Furthermore, we require the following property: assuming s is taken uniformly at random, it is computationally hard to distinguish $G(s)$ from a uniformly random string taken from $\{0,1\}^{\kappa+\ell}$ without the knowledge of s. Thus, a pseudorandom generator is simply a function that takes as input a small random bitstring and outputs an ℓ-bit longer pseudorandom bitstring.

Fundamental results in this area were shown by Goldreich and Levin in [GL89]. Specifically, they prove the following two results:

- *Pseudorandom generators with* 1*-bit stretch exist if and only if one-way functions exist.*
- *For any* $\ell = \text{poly}(\kappa)$*, pseudorandom generators with* ℓ*-bit stretch exist if and only if pseudorandom generators with* 1*-bit stretch exist.*

Then, putting these results together, we obtain that pseudorandom generators with polynomial stretch exist if and only if one-way functions exist. In particular, considering length-doubling pseudorandom generators, so with domain $\{0,1\}^\kappa$ and range $\{0,1\}^{2\kappa}$, we immediately have:

Length-doubling pseudorandom generators exist if and only if one-way functions exist.

The above statements are not detailed further in this manuscript, as we focus on constructions of pseudorandom functions, but let us explain how length-doubling pseudorandom generators can then be used to build pseudorandom functions, as shown by Goldreich, Goldwasser, and Micali in their seminal work [GGM84]. Specifically, let us briefly sketch a proof of the following statement:

Pseudorandom functions exist if and only if length-doubling pseudorandom generators exist.

Assume the existence of a length-doubling pseudorandom generator $G\colon \{0,1\}^\kappa \to \{0,1\}^{2\kappa}$, for $\kappa \geq 1$. Then, one can build a pseudorandom function $F\colon \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^\kappa$ as follows, for any $n = \text{poly}(\kappa)$.

For a key $K \in \{0,1\}^\kappa$ and a bitstring $x \in \{0,1\}^\ell$, $0 \leq \ell \leq n$, we recursively define K_x as:

- $K_\varepsilon = K$, where ε denotes the empty string;
- $K_{x \,\|\, 0} \,\|\, K_{x \,\|\, 1} = G(K_x)$, with $|K_{x \,\|\, 0}| = |K_{x \,\|\, 1}| = \kappa$.

Then, the GGM pseudorandom function is defined via $F(K,x) = K_x$.

Intuitively, this corresponds to following a path in a binary tree of depth n whose root is labeled by K and such that the two siblings of any internal node labeled by a string s are respectively labeled by the first and second half of $G(s)$. Computing $F(K,x)$ then simply corresponds in outputting the label of the x-th leaf of this tree. Please refer to Figure 1.1 for further details about this computation.

The security of this pseudorandom function follows from the security of the pseudorandom generator. Let us assume that the adversary makes only one query $x = 0^n$. That is, to compute $F(K,x)$, one first evaluates $G(K)$ and evaluate G again on the first half of $G(K)$, and so on. Then, assuming the security of G, if K is chosen uniformly at random, one can change $G(K) = K_0 \,\|\, K_1$ to a uniformly random string of length 2κ. This change is computationally indistinguishable due to the security of G. Then, one can change $G(K_0)$ to a uniformly random string of length 2κ under the security of G, as K_0 is now a uniformly random bitstring. By repeating n times this process, one change the output $F(K, 0^n)$ to a uniformly random bitstring of length κ, and this change is computationally indistinguishable assuming the security of G. In the case of multiple queries, one just needs to reiterate this process on each path of the tree that is used to answer the (polynomial number of) queries made by the adversary.

The other direction is immediate, as, given a pseudorandom function $F\colon \{0,1\}^\kappa \times \mathcal{D} \to \{0,1\}^\ell$, one can just construct a length-doubling pseudorandom generator by evaluating F on fixed inputs with the seed as a key, and concatenate (or truncate) the outputs to obtain a bitstring of length 2κ.

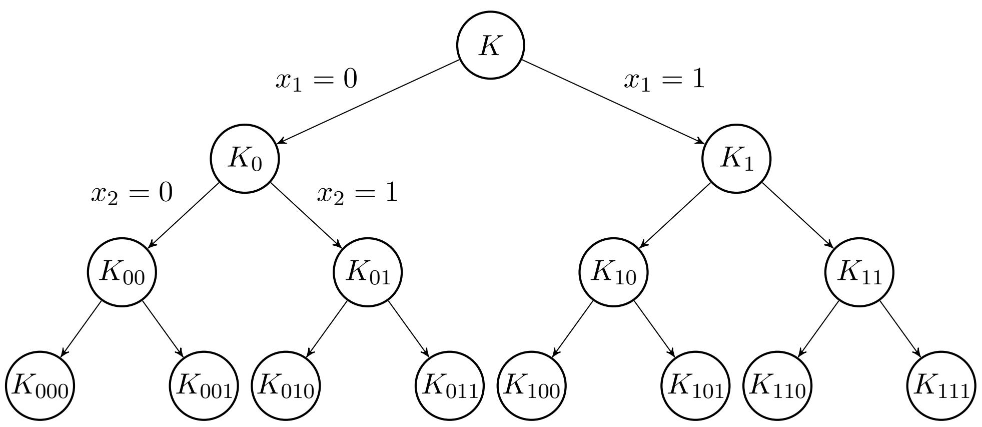

Figure 1.1: The GGM construction for F: $\{0,1\}^\kappa \times \{0,1\}^3 \to \{0,1\}^\kappa$

1.1.2 Number-Theoretic Constructions

Despite its elegance, the original construction of pseudorandom functions by Goldreich, Goldwasser, and Micali based on pseudorandom generators is not very efficient. In order to improve its efficiency while still being able to prove its security under reasonable complexity assumptions, Naor and Reingold [NR97] proposed a new construction based on the Decisional Diffie-Hellman assumption (DDH). Let $\vec{a} = (a_0, \ldots, a_n) \in (\mathbb{Z}_p^*)^{n+1}$ be the key and $x = x_1 \,\|\, \ldots \,\|\, x_n \in \{0,1\}^n$ be the input of the pseudorandom function. Let g be a fixed public generator of a group $\mathbb{G}$ of prime order p. The Naor-Reingold pseudorandom function is then defined as

$$\mathsf{NR}(\vec{a}, x) = \left[a_0 \prod_{i=1}^{n} a_i^{x_i} \right] \quad ,$$

where for any $a \in \mathbb{Z}_p$, $[a]$ stands for g^a, as defined in [EHK+13].

As mentioned in [BMR10], the algebraic nature of the Naor-Reingold pseudorandom function has led to many applications, such as verifiable random functions [ACF09; HW10], distributed pseudorandom functions [NR97], and related-key secure pseudorandom functions [BC10b], which are hard to obtain from generic pseudorandom functions. Hence, due to its importance, several other extensions of the Naor-Reingold pseudorandom function have been proposed [LW09; BMR10] based on different assumptions, such as the Decision Linear assumption (DLin) [BBS04] and the d-DDHI assumption [BMR10; GOR11].

An important part of this work is to study the algebraic structure of such pseudorandom functions, as well as extensions of these constructions, such as related-key secure, aggregate, and multilinear pseudorandom functions. Let us then define more formally these primitives.

1.1.3 Extensions of Pseudorandom Functions

Pseudorandom functions have been extended in different directions, either to add functionalities, for instance with the notions of aggregate or verifiable pseudorandom functions, or to strengthen their security. In particular, due to their central role in constructing block-ciphers, building pseudorandom functions that resist to real-world attacks has been an important area of research. An aspect of this area corresponds to related-key security. In this thesis, we focus on aggregate, multilinear, and related-key secure pseudorandom functions and briefly introduce these notions below.

1.1.3.1 Related-Key Security

As already explained, a common approach to prove the security of a cryptographic scheme, known as provable security, is to relate its security to one of its underlying primitives or to an accepted hard computational problem. While this approach is now standard and widely accepted, there is still a significant gap between the existing models used in security proofs and the actual environment in which these cryptosystems are deployed. For example, most of the existing security models assume that the adversary has no information about the user's secret key. However, it is well known that this is not always true in practice: the adversary may be able to learn partial information about the secrets using different types of side-channel attacks, such as the study of energy consumption, fault injection, or timing analysis. In the particular case of fault injection, for instance, an adversary can learn not only partial information about the secret key, but it may also be able to force a cryptosystem to work with different but related secret keys. Then, if it can observe the outcome of this cryptosystem, it may be able to break it. This is what is known in the literature as a related-key attack (RKA).

Most primitives are designed without taking related-key attacks into consideration so their security proofs do not provide any guarantee against such attacks. Hence, a cryptographic scheme that is perfectly safe in theory may be completely vulnerable in practice. Indeed, many such attacks were found during the last decade, especially against practical block-ciphers [BDK05; BDK08; BDK+10; BK09; BKN09; KHP07]. Inspired by this cryptanalytic work, some years ago, theoreticians started to develop appropriate security models and search for cryptographic primitives which can be proven related-key secure.

Though related-key attacks were first introduced by Biham and Knudsen [Bih94; Knu93] in the early 1990's, it was only in 2003 that Bellare and Kohno [BK03] began the formalization of the theoretical foundations for related-key security. We recall their security definition of related-key security for pseudorandom functions here. Let $F\colon\ \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ be a family of functions for a security parameter κ, and let $\Phi = \{\phi\colon\ \mathcal{K} \to \mathcal{K}\}$ be a set of functions on the key space $\mathcal{K}$, called a related-key deriving (RKD) function set. We say that F is a Φ-related-key secure pseudorandom function if for any polynomial-time adversary, its advantage in the following game is negligible. The game starts by picking a random challenge bit b, a random target key $K \in \mathcal{K}$ and a random function $G\colon\ \mathcal{K} \times \mathcal{D} \to \mathcal{R}$. The adversary can repeatedly query an oracle that, given a pair $(\phi, x) \in \Phi \times \mathcal{D}$, returns either $F(\phi(K), x)$, if $b = 1$, or $G(\phi(K), x)$, if $b = 0$. Finally, the adversary outputs a bit b', and its advantage is defined by $2\Pr[\, b = b'\,] - 1$. Note that if the class Φ of RKD functions contains only the identity function, then this notion matches standard PRF security.

In this work, Bellare and Kohno also proved that there are inherent limitations to security against related-key attacks. In particular, there exist simple classes of related-key deriving functions for which related-key security is impossible. For instance, it is impossible to achieve related-key security against any class of functions that contains a constant function, as it implies that the adversary is able to change the key to a fixed constant value. Hence, it is important to understand which classes of RKD functions can or cannot be handled.

It appears that pseudorandom functions are a central notion in related-key security. Indeed, it has been shown by Bellare, Cash, and Miller in [BCM11], that having related-key secure pseudorandom functions for a class of RKD functions is sufficient to transform several standard cryptographic primitives (including signatures, symmetric encryption, public-key encryption, identity-based encryption, ...) into related-key secure ones for the same class of functions. Therefore, studying pseudorandom functions, in the related-key setting, is of prime interest.

Yet, building related-key secure pseudorandom functions under standard assumptions has been a wide-open problem for years, until the seminal work by Bellare and Cash [BC10b], and achieving related-key security under standard assumptions for real-world classes, such as XOR relations, is still a major open problem.

1.1.3.2 Aggregate Pseudorandom Functions

Aggregate pseudorandom functions were introduced by Cohen, Goldwasser, and Vaikunthanathan in [CGV15] The main interest of an aggregate pseudorandom function is to provide the user with the possibility of aggregating the values of the function over *super-polynomially* many outputs with only a *polynomial-time* computation, without enabling a polynomial-time adversary to distinguish the function from a truly random function. For instance, one such example of an aggregate query could be to compute the product of all the output values of the pseudorandom function corresponding to a given exponentially-sized interval of the input domain.

In addition to proposing the notion of aggregate pseudorandom functions, Cohen, Goldwasser, and Vaikunthanathan [CGV15] also proposed first constructions for several different classes of aggregate queries, such as decision trees, hypercubes, and read-once boolean formulas, achieving different levels of expressiveness. Unfortunately, for most of the constructions proposed in [CGV15], the proofs of security suffer from an exponential (in the input length) overhead in their running time and have to rely on the sub-exponential hardness of the Decisional Diffie-Hellman (DDH) problem.

Indeed, to prove the security of their constructions, the authors use a generic result which is simply saying the following: given an adversary $\mathscr{A}$ against the aggregate pseudorandom function security of a pseudorandom function F, one can build an adversary $\mathscr{B}$ against the standard pseudorandom function security of F. $\mathscr{B}$ just queries all the values required to compute the aggregate or output values, and computes the aggregate values itself before sending them to $\mathscr{A}$.

Clearly, this reduction proves that any secure pseudorandom function is actually also a secure aggregate pseudorandom function. However, this reduction is *not efficient*, since to answer to only one aggregate query, the adversary $\mathscr{B}$ may have to query an exponential number of values to its oracle. Hence, as soon as we can aggregate in one query a super-polynomial number of PRF values, this generic reduction does not run in polynomial time.

1.1.3.3 Multilinear Pseudorandom Functions

In order to overcome the shortcomings of the work of Cohen, Goldwasser, and Vaikuntanathan [CGV15], Cohen and Holmgren introduced the concept of multilinear pseudorandom functions in [CH15]. Informally speaking, a multilinear pseudorandom function is a variant of the standard notion of pseudorandom function, which works with vector spaces and which guarantees indistinguishability from random multilinear functions with the same domain and range. As shown in [CH15], multilinear pseudorandom functions can be used to prove the aggregate pseudorandom function security of the Naor-Reingold (NR) pseudorandom function [NR97] with a polynomial-time reduction for the case of hypercubes and decision trees aggregations. Unfortunately, their technique does not extend to the more general case of read-once formulas aggregation, which is the most expressive form of aggregation in [CGV15]

1.2 Our Contributions

Our main focus in this thesis is the study of pseudorandom functions and above extensions, and our main contributions in these areas are the following.

1.2.1 Extension and Correction of the Bellare-Cash Framework

As already mentioned, Bellare and Cash [BC10b] designed the first related-key secure pseudorandom functions based on standard assumptions. These foundational results are obtained by applying a single, elegant, general framework to the Naor-Reingold pseudorandom function and handle certain multiplicative and additive RKD function classes. Unfortunately, it was discovered later that the framework of [BC10b] had a minor bug that prevented its proof to go through. They corrected this bug by strengthening the requirements of their framework, but their new requirements did no longer allow the framework to be applied to additive classes.

In this thesis, our first contribution is to correct and extend this framework. Specifically, we provide a new proof of their framework that goes through assuming only the original requirements. Therefore, it allows us to apply our framework to the additive case, as originally done in [BC10b]. Moreover, we provide new tools that let us handle larger classes of relations and extend the framework. For instance, we are able to construct a related-key secure pseudorandom function for the class of affine relations.

1.2.2 Algebraic Framework for Pseudorandom Functions

The second contribution of this thesis is an algebraic framework, termed "polynomial linear pseudorandomness security", that can be used to construct and prove the security of a wide variety of pseudorandom functions. Intuitively, we prove the following theorem: let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order p and let us denote by $[a]$ the group element g^a, for any $a \in \mathbb{Z}_p$. Let $a_1, \ldots, a_n$ be n uniformly random secret scalars from $\mathbb{Z}_p$. Then, for any positive integer q and any multivariate polynomials $P_1, \ldots, P_q$ defined over $\mathbb{Z}_p$ and with indeterminates $T_1, \ldots, T_n$, the group elements:

$$[P_1(a_1, \ldots, a_n)], \ldots, [P_q(a_1, \ldots, a_n)]$$

are computationally indistinguishable, under some standard assumptions, from the group elements:

$$[U(P_1)], \ldots, [U(P_q)] \quad ,$$

where U is a random linear map from the set of multivariate polynomials to $\mathbb{Z}_p$.

This theorem is straightforward in the generic group model, but we prove that it is also true assuming only the hardness of some standard assumptions (discussed below). Furthermore, this framework is general enough to encompass standard and related-key securities for pseudorandom functions as well as multilinear and aggregate pseudorandom functions, and allows us to solve various open questions in these areas. In particular, it let us define a fully algebraic framework for related-key security that only requires some basic algebraic properties to be applied, or to provide a polynomial-time reduction that proves the aggregate pseudorandom function security of the Naor-Reingold construction for read-once formulas aggregation.

1.2.3 Assumptions

Our algebraic framework is proven under different assumptions. In particular, depending on certain parameters, such as the degree of polynomials $P_1, \ldots, P_q$, we are able to prove our theorem assuming only the DDH assumption or a stronger assumption, such as the d-DDHI assumption (with d being a bound on the degree of polynomials).

We actually prove our framework under a new family of computational assumptions, termed $\mathcal{E}_{k,d}$-MDDH assumptions (where d is a bound on the degree of polynomials and k is some parameter precised later), that encompasses in particular classical assumptions such as $\mathsf{DDH}, d\text{-}\mathsf{DDHI}, k\text{-}\mathsf{Lin}, \ldots$ and a side contribution of this thesis is to prove the security of all our non-classical assumptions in the generic group model.

1.2.4 Related-Key Secure Pseudorandom Function for XOR Relations

The last contribution of this thesis is a construction of related-key secure pseudorandom functions for XOR relations, based on the existence of a weak form of multilinear maps. Indeed, even if the results discussed above in related-key security allow to handle different and large classes of RKD functions, the functions have to be algebraic transformations (such as multivariate polynomials), and thus do not really correspond to real-world attacks. Here, our construction is secure even assuming that the adversary has the capacity of flipping any bit of the key at each query, which is closer to the actual capacities of a real-world adversary.

However, our result should be seen as a proof of concept, as the existence of multilinear maps is still a very strong assumption, as revealed by the recent devastating attacks. Yet, despite the numerous attacks against current multilinear maps, our construction circumvents all the known attacks, and we only rely on the existence of a weak form of multilinear map. In particular, we hope that such weak form of multilinear maps might be instantiated in the future under standard assumptions, such as lattice-based assumptions.

1.3 Other Contributions

Independently from the work presented in this manuscript, we also worked on the following unrelated subjects.

1.3.1 Order-Revealing Encryption

In a collaboration resulting from an internship at Technicolor Research with Marc Joye, we study order-revealing encryption. An order-revealing encryption scheme is a private-key encryption scheme that provides anyone with the capacity of comparing plaintexts given only the ciphertexts, without revealing anything more about the plaintexts. The first construction of order-revealing encryption was proposed by Boneh *et al.* in [BLR+15] and relied on the existence of multilinear maps. In our work, we propose a first construction of order-revealing encryption assuming one-way functions exist, but only for polynomial-size domains. We also propose a construction that reveals the order but also a bit more, for exponential-sized domains, assuming the hardness of the DLin problem.

1.3.2 Private Circuits

In a joint work with Sonia Belaïd, Fabrice Benhamouda, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud, we study the construction of multiplication circuits in the probing model. This model has been introduced by Ishai, Sahai, and Wagner in [ISW03] to capture the security of implementations against side-channel attacks, such as attacks using the power consumption or the electromagnetic radiation of the device running the cryptosystem. When devices are not protected against such attacks, it is often possible to exploit these physical measures to retrieve sensitive data.

Specifically, this model assumes that the attacker can obtain a bounded number of intermediate values of the computation (called probes), and we call a private circuit a circuit that is secure in this model. The seminal work by Ishai, Sahai, and Wagner, shows how to transform any circuit into a private circuit, at the cost of extra randomness. The main tool for this transformation is the construction of the first private circuit for bit-multiplication.

In our work, we focus on studying the bit-multiplication and try to optimize the number of random bits used by the private circuit computing this operation. Letting d denote the bound on the number of probes that the attacker can make, we obtain the following results: on the theoretical side, we prove a linear lower bound (in d) on the number of additional random bits, and a quasi-linear upper bound ($O(d \cdot \log d)$). These bounds result from an algebraic characterization of the security of a private circuit, which relies on basic linear algebra. On the practical side, we construct a private circuit for bit-multiplication that halves the randomness complexity of the previous circuit by Ishai, Sahai, and Wagner, and also construct private circuits for small (real-world) cases ($d \leq 4$) whose randomness complexity match our linear lower bound. Finally, we also implement a tool, based on our algebraic characterization of security, that allows to find attacks against multiplication circuits which is magnitude faster that previous known tools (such as [BBD+15]), but is not perfectly sound. That is, our tool might not always find attacks against a non-secure scheme so cannot serve for proving security of a scheme, but allows to discard quickly bad candidates.

1.3.3 Functional Encryption and Obfuscation

In a work with Nir Bitansky, Ryo Nishimaki, and Daniel Wichs, resulting from an internship at Northeastern University, we study the relation between private-key functional encryption and indistinguishability obfuscation. By designing different intermediate tools, we are able to construct a compact public-key functional encryption scheme from any unbounded-collusion private-key functional encryption scheme and any plain public-key encryption scheme. Putting this result together with the transform by Bitansky and Vaikunthanathan [BV15] or by Ananth and Jain [AJ15], we then obtain that private-key functional encryption is sufficient, combined with any plain public-key encryption scheme, to build indistinguishability obfuscation. In particular, this proves that private-key functional encryption is (almost) as powerful as public-key functional encryption.

1.4 Organization

The rest of this manuscript is organized as follows: Chapter 2 first introduces basic mathematical, computational, and cryptographic notions. Next, Chapter 3 introduces related-key security and corrects and extends the Bellare-Cash framework. Chapter 4 defines our new family of assumptions and either relates them to classical assumptions or proves their security in a generic-group model. Chapter 5 then uses the new family of assumptions to prove the security of our main algebraic framework. Next, Chapter 6 applies the new algebraic framework to pseudorandom functions, related-key secure pseudorandom functions, aggregate pseudorandom functions, and multilinear pseudorandom functions. Finally, Chapter 7 describes our construction of related-key secure pseudorandom function for XOR relations and Chapter 8 concludes this thesis.

Chapter 2

Preliminaries

In this chapter, we introduce the notation and basic notions used throughout this manuscript. We start by recalling some standard mathematical and computational concepts, and by briefly introducing provable security. We also remind some standard number-theoretic assumptions, and define most of the cryptographic primitives involved in this work, including standard, aggregate, and multilinear pseudorandom functions. Please note that the definition of related-key secure pseudorandom functions is postponed to Chapter 3. We conclude this chapter by introducing multilinear maps and the multilinear map model.

Multilinear maps and the multilinear map model are central in Chapter 7 and Section 4.4, but these two parts are rather independent from the rest of this thesis, so we recommend to inspect Section 2.4 only before reading either of these parts. Most of the other notions introduced in this chapter are rather usual, thus it can be easily skimmed through. However, we encourage the reader to inspect Section 2.3.2 that is central for the sequel.

A list of notation and abbreviations is provided at the end of this manuscript on pages 139 and 141.

Contents

2.1 Notation and Preliminaries

2.1.1 Mathematical Notions

Integers, Sets, and Modular Arithmetic. We denote by $\mathbb{Z}$ the set of integers and by $\mathbb{N}$ the set of non-negative integers. If $\mathscr{S}$ is a set, then $|\mathscr{S}|$ denotes its size.

For two sets $\mathcal{D}$ and $\mathcal{R}$, we denote by $\mathsf{Fun}(\mathcal{D}, \mathcal{R})$ the set of all functions $f\colon \mathcal{D} \to \mathcal{R}$ with domain $\mathcal{D}$ and range $\mathcal{R}$.

For a positive integer N, we denote by $(\mathbb{Z}_N, +)$ the additive group of integers modulo N and by $(\mathbb{Z}_N, +, \cdot)$ the ring of integers modulo N. We often abuse this notation by using $\mathbb{Z}_N$. Furthermore, we denote by $(\mathbb{Z}_N^*, \cdot)$ or simply $\mathbb{Z}_N^*$ the multiplicative subgroup of $(\mathbb{Z}_N, +, \cdot)$ that contains the invertible elements of the ring $\mathbb{Z}_N$. For an integer $x \in \mathbb{Z}$, we denote by x mod N the remainder of the Euclidean division of x by N, which can be considered both as an integer in $\{0, \ldots, N-1\}$ or as an element of $\mathbb{Z}_N$. We then identify elements from $\mathbb{Z}_N$ with integers of the set $\{0, \ldots, N-1\}$. We recall that performing additions and multiplications over $\mathbb{Z}_N$ is efficient (polynomial-time in the size of N).

In this work, N is often a prime integer, thus denoted p. Please note that, for any prime number p, the ring $(\mathbb{Z}_p, +, \cdot)$ is also a finite field, and then $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\}$.

Cyclic Groups. We make extensive use of cyclic groups in this thesis. We recall that a cyclic group is a finite group generated by a single element. In particular, a cyclic group is commutative (or Abelian). Throughout this manuscript, we denote by a tuple $(p, \mathbb{G}, g)$ a multiplicative cyclic group of prime order p generated by an element g, so that $\mathbb{G} = \langle g \rangle = \{1, g, \ldots, g^{p-1}\}$, where 1 denote the identity element (corresponding to g^0). We often abuse this notation by using the notation $\mathbb{G}$ when the order and the generator are clear from the context. We denote by $[x]_g$, or simply $[x]$ when the generator is clear from the context, the element g^x, for any $x \in \mathbb{Z}_p$, so in particular $g = [1]_g$.

We also assume that the multiplication over $\mathbb{G}$ is an efficient operation, so given g and $x \in \mathbb{Z}_p$, one can efficiently compute $[x]_g$.

Linear Algebra. For two vector spaces $\mathcal{D}$ and $\mathcal{R}$, we denote by $\mathsf{L}(\mathcal{D}, \mathcal{R})$ the vector space of linear functions from $\mathcal{D}$ to $\mathcal{R}$. Similarly, if $\mathcal{D}_1, \ldots, \mathcal{D}_n$ denote n vector spaces, then $\mathsf{L}(\mathcal{D}_1 \otimes \cdots \otimes \mathcal{D}_n, \mathcal{R})$ is the vector space of n-linear functions from $\mathcal{D}_1 \times \cdots \times \mathcal{D}_n$ to $\mathcal{R}$. In this thesis, vector spaces are implicitly supposed to be $\mathbb{Z}_p$-vector spaces.

Vectors and Matrices. Vectors are denoted with an arrow (e.g., $\vec{v}$) and matrices by bold capital letters (e.g., $\boldsymbol{A}$). For a vector $\vec{v}$, $|\vec{v}|$ denotes its length and v_i denotes its i-th component, so $\vec{v} = (v_i)_{i=1,\ldots,|\vec{v}|}$. For a vector $\vec{v} \in \mathcal{D}^n$ and a function $f\colon \mathcal{D} \to \mathcal{R}$, we denote by $f(\vec{v})$ the vector $(f(v_i))_{i=1,\ldots,n} \in \mathcal{R}^n$. Reversely, for a vector of functions $\vec{f} = (f_i)_{i=1,\ldots,n}$, we denote by $\vec{f}(x)$ the vector $(f_1(x), \ldots, f_n(x))$. For two vectors $\vec{u}, \vec{v}$ of same length n over a ring $\mathcal{R}$, we denote by $\langle \vec{u}, \vec{v} \rangle = \sum_{i=1}^n u_i \cdot v_i$ their inner product. For a matrix $\boldsymbol{A}$, we denote by $a_{i,j}$ its entry in the i-th row and j-th column. We sometimes combine both notations, by defining vectors of matrices (e.g., $\vec{\boldsymbol{A}} = (\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{|\vec{\boldsymbol{A}}|})$).

For a cyclic group $(p, \mathbb{G}, g)$ and a vector $\vec{a} \in \mathbb{Z}_p^n$, we denote by $[\vec{a}]_g$ the vector of group elements $([a_1]_g, \ldots, [a_n]_g) \in \mathbb{G}^n$. Similarly, for a matrix $\boldsymbol{A} \in \mathbb{Z}_p^{n \times m}$, we denote by $[\boldsymbol{A}]_g$ the matrix of group elements $([a_{i,j}]_g)_{\substack{i=1,\ldots,n \\ j=1,\ldots,m}} \in \mathbb{G}^{n \times m}$. As already stated above, we often abuse these notations by not specifying the generator g when it is clear from the context.

Polynomials. We denote by $\mathbb{Z}_p[T_1, \ldots, T_n]$ the vector space (and ring) of multivariate polynomials in indeterminates $T_1, \ldots, T_n$ over $\mathbb{Z}_p$, and by $\mathbb{Z}_p[T_1, \ldots, T_n]_{\leq d}$ its subspace containing only polynomials whose degree in each indeterminate is at most d. We sometimes denote by $P(\vec{T})$ the polynomial $P(T_1, \ldots, T_n) \in \mathbb{Z}_p[T_1, \ldots, T_n]$, and by $P(\vec{a})$ its evaluation by setting $\vec{T}$ to $\vec{a}$, meaning that we set $T_1 = a_1, \ldots, T_n = a_n$.

Assignation. If $\mathscr{S}$ is a set, $x \xleftarrow{\$} \mathscr{S}$ indicates that x is taken uniformly at random from the set $\mathscr{S}$ (independently of everything else). We also write $x, y \xleftarrow{\$} \mathscr{S}$ to indicate that x and y are chosen independently and uniformly at random from $\mathscr{S}$. We often write that an element is "picked at random" to mean "picked independently and uniformly at random".

2.1.2 Algorithmic Concepts

Bitstrings. Binary strings are denoted with lowercase letters (e.g., x). We denote by $\{0,1\}^*$ the set of all bitstrings and by $\{0,1\}^n$ the set of all bitstrings of length n. For a binary string x, we denote its length by $|x|$, so $x \in \{0,1\}^{|x|}$, and x_i its i-th bit, so $x = x_1 \parallel \ldots \parallel x_{|x|}$. The *exclusive or* (XOR) of two bitstrings x and y of same length is denoted by $x \oplus y$.

Algorithms and Efficiency. For simplicity, we consider our algorithms as probabilistic Turing machines. That is, we implicitly assume that they can use an additional tape containing random bits (also referred to as random coins). In the rest of this thesis, the term *PPT* algorithm stands for *Probabilistic Polynomial-Time* algorithm and we say that an algorithm is *efficient* if it is a PPT algorithm.

For an algorithm A, we denote by $y \xleftarrow{\$} A(x)$ for the fact of running algorithm A on input x and with fresh random coins and letting y denote its output. If A is deterministic, we simply note $y \leftarrow A(x)$.

2.1.3 Provable Security

Negligibility. Let ϵ be a function from $\mathbb{N}$ to $[0,1]$. We say that ϵ is *negligible* or $1 - \epsilon$ is *overwhelming*, if for any constant $c \in \mathbb{N}$, there exists $\eta \in \mathbb{N}$ such that for any $\kappa \geq \eta$, $\epsilon \leq \frac{1}{\kappa^c}$.

Security Parameter. As most of the cryptosystems, our constructions only achieve a computational notion of security and can actually be broken with a powerful enough computer. However, it is widely accepted that if 2^{128} elementary operations are *required* to break a cryptosystem with high probability, then this cryptosystem can be considered as secure. We say that such a cryptosystem provides 128 bits of security.

Of course, with the computers being more and more powerful, it might happen in the future that making 2^{128} elementary operations is reasonable. We then use the notion of *security parameter* to formalize the security of our constructions. Informally speaking, the security parameter is an integer $\kappa \in \mathbb{N}$ that is (sometimes implicitly) fed (in unary) to all the algorithms of a cryptosystem and such that algorithms run in polynomial time in this security parameter and such that a specific instantiation of the cryptosystem with security parameter κ provides κ bits of security.

Adversaries. *Adversaries* are probabilistic Turing machines and are denoted by calligraphic letters (e.g., $\mathscr{A}, \mathscr{D}$). Adversaries implicitly takes as input a unary representation of the security parameter. We consider two types of adversaries: on the one hand PPT adversaries, which run in polynomial time, so in particular in polynomial time in the security parameter, and on the other hand unbounded adversaries.

Experiments, Games, Oracles. We often define our security notions or assumptions using experiments, parametrized by the security parameter κ, and during which an adversary is called one or several times with various inputs. The adversary may also be provided access to oracles, which are Turing machines, however, the running time of an adversary does not depend on the running time of the oracle and a query to an oracle only counts for one operation.

Then, an *experiment* can be seen as a *game* between an adversary $\mathscr{A}$ and an implicit challenger which provides its input to the adversary as well as some oracle access. This game has an **Initialize** procedure, procedures to respond to adversary oracle queries, and a **Finalize** procedure. In the case where the **Finalize** procedure is not explicitly defined, it is implicitly defined as the procedure that simply outputs its input. To execute a game $\mathbf{G}$ with an adversary $\mathscr{A}$, we proceed as follows. First, **Initialize** is executed and its outputs become the input of $\mathscr{A}$. When $\mathscr{A}$ executes, its oracle queries are answered by the corresponding procedures of $\mathbf{G}$. When $\mathscr{A}$ terminates, its outputs become the input of **Finalize**. The output of the latter, denoted $\mathbf{G}^{\mathscr{A}}$ is called the output of the game, and we let "$\mathbf{G}^{\mathscr{A}} \Rightarrow 1$" denote the event that this game output takes the value 1. The running time of an adversary by convention is the worst case time for the execution of the adversary with any of the games defining its security, so that the time of the called game procedures is included.

Advantage. The *advantage* of an adversary $\mathscr{A}$ in an experiment Exp is the probability that this adversary outputs 1 in this experiment:

$$\mathsf{Adv}^{\mathsf{Exp}}(\mathscr{A}, \kappa) := \Pr\left[\,\mathsf{Exp}(\mathscr{A}, \kappa) = 1\,\right] \ .$$

Similarly, we define the *advantage* of an adversary $\mathscr{D}$ in distinguishing two experiments Exp^0 and Exp^1 as:

$$\mathsf{Adv}^{\mathsf{Exp}}(\mathscr{D}, \kappa) := \Pr\left[\mathsf{Exp}^1(\mathscr{D}, \kappa) = 1\right] - \Pr\left[\mathsf{Exp}^0(\mathscr{D}, \kappa) = 1\right] \ .$$

The advantage depends on the security parameter κ. For simplicity, κ is often implicit.

Assumptions, Security, and Indistinguishability. To define an assumption or a security notion, we then define a problem as an experiment or as distinguishing two experiments. We say that the problem is *hard* If no PPT algorithm can solve it.

We say that an assumption (or security) *computationally* holds if the above advantage is negligible for any PPT adversary. We further say that it *statistically* holds if the above advantage is negligible for any (even unbounded) adversary. We finally say that it *perfectly* holds if the above advantage is 0 for any (even unbounded) adversary.

Similarly, we say that two games are *computationally indistinguishable* if the advantage of any PPT adversary in distinguishing these two games is negligible. We say that two experiments are *statistically indistinguishable* (resp. *perfectly indistinguishable* or *equivalent*) if the advantage of any (unbounded) adversary in distinguishing these two games is negligible (resp. 0).

Hybrid Arguments. Most of our security proofs are proofs by games (also called hybrid arguments) as defined by Shoup in [Sho01; KR01; BR06]: to bound an advantage in some game experiment corresponding to some security notion, we construct of sequence of games.

The first game is $\mathbf{G}_0$ is the experiment itself, while the last game corresponds to some security notion or is such that the adversary just cannot win. Furthermore, we prove that two consecutive games are indistinguishable either perfectly, statistically, or computationally. In other words, we bound the difference of advantages by a negligible quantity.

Similarly, to bound an advantage of an adversary in distinguishing two experiments, we construct a sequence of indistinguishable games starting with the first experiment and ending with the second experiment.

2.2 Classical Computational Assumptions

2.2.1 The Discrete Logarithm Problem

For cryptographic purpose, we require our cyclic groups $(p, \mathbb{G}, g)$ to be such that the reverse operation of exponentiation is hard. This operation, called the *discrete logarithm* operation, consists, given an element $g \in \mathbb{G}$ and an element $h = g^x \in \mathbb{G}$, in computing scalar $x \in \mathbb{Z}_p$ such that $h = g^x$. The scalar x is called the *discrete logarithm* of h in the basis g.

2.2.2 Classical Discrete-Logarithm-Based Assumptions

2.2.2.1 Search Assumptions

We define two assumptions, termed the *discrete logarithm* and the *strong discrete logarithm* assumptions, that respectively correspond to the hardness of the following search problems.

Discrete Logarithm. We define the advantage of an adversary $\mathscr{A}$ against the discrete logarithm (DL) problem in $\mathbb{G}$ as:

$$\mathsf{Adv}_{\mathbb{G}}^{\mathsf{dl}}(\mathscr{A}) := \Pr\left[\mathsf{DL}_{\mathbb{G}}^{\mathscr{A}} \Rightarrow 1\right]$$

where the probability is over the choices of $a \in \mathbb{Z}_p$, $g \in \mathbb{G}$, and the random coins used by the adversary, and where $\mathsf{DL}_{\mathbb{G}}$ is described in Figure 2.1.

Strong Discrete Logarithm. For $d \geq 2$, we define the advantage of an adversary $\mathscr{A}$ against the d-strong discrete logarithm (d-SDL) problem in $\mathbb{G}$ as:

$$\mathsf{Adv}_{\mathbb{G}}^{d\text{-}\mathsf{sdl}}(\mathscr{A}) := \Pr\left[d\text{-}\mathsf{SDL}_{\mathbb{G}}^{\mathscr{A}} \Rightarrow 1\right]$$

where the probability is over the choices of $a \in \mathbb{Z}_p$, $g \in \mathbb{G}$, and the random coins used by the adversary, and where d-$\mathsf{SDL}_{\mathbb{G}}$ is described in Figure 2.1.

$\mathsf{DL}_{\mathbb{G}}$	d-$\mathsf{SDL}_{\mathbb{G}}$
proc Initialize	**proc Initialize**
$a \xleftarrow{\$} \mathbb{Z}_p^*$	$a \xleftarrow{\$} \mathbb{Z}_p^*$
Return $([1], [a])$	Return $([1], [a], \ldots, \left[a^d\right])$
proc Finalize(a')	**proc Finalize**(a')
Return $([a] = [a'])$	Return $([a] = [a'])$

Figure 2.1: Games defining the DL and d-SDL problems in $\mathbb{G}$

2.2.2.2 Decisional Assumptions

We first define two assumptions, termed the *Decisional Diffie-Hellman* and the *d-Decisional Diffie-Hellman Inversion* assumptions, that respectively correspond to the hardness of the following decisional problems.

DDH. The advantage of an adversary $\mathscr{D}$ against the DDH problem in $\mathbb{G}$ is defined to be:

$$\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\mathscr{D}) := \Pr\left[\mathsf{DDHReal}^{\mathscr{D}}_{\mathbb{G}} \Rightarrow 1\right] - \Pr\left[\mathsf{DDHRand}^{\mathscr{D}}_{\mathbb{G}} \Rightarrow 1\right]$$

where the probabilities are over the choices of $a, z \in \mathbb{Z}_p$, $g \in \mathbb{G}$, and the random coins used by the adversary, and where $\mathsf{DDHReal}_{\mathbb{G}}$ and $\mathsf{DDHRand}_{\mathbb{G}}$ are described in Figure 2.2.

DDHI. For $d \geq 1$, the advantage of an adversary $\mathscr{D}$ against the d-DDHI problem in $\mathbb{G}$ is defined as:

$$\mathsf{Adv}^{d\text{-}\mathsf{ddhi}}_{\mathbb{G}}(\mathscr{D}) := \Pr\left[d\text{-}\mathsf{DDHIReal}^{\mathscr{D}}_{\mathbb{G}} \Rightarrow 1\right] - \Pr\left[d\text{-}\mathsf{DDHIRand}^{\mathscr{D}}_{\mathbb{G}} \Rightarrow 1\right]$$

where the probabilities are over the choices of $a, z \in \mathbb{Z}_p$, $g \in \mathbb{G}$, and the random coins used by the adversary, and where d-$\mathsf{DDHIReal}_{\mathbb{G}}$ and d-$\mathsf{DDHIRand}_{\mathbb{G}}$ are described in Figure 2.2.

$\mathsf{DDHReal}_{\mathbb{G}}$	$\mathsf{DDHRand}_{\mathbb{G}}$
proc Initialize $a, b \xleftarrow{\$} \mathbb{Z}_p$ Return $([1], [a], [b], [ab])$ **proc Finalize**(b) Return b	**proc Initialize** $a, b \xleftarrow{\$} \mathbb{Z}_p$; $z \xleftarrow{\$} \mathbb{Z}_p$ Return $([1], [a], [b], [z])$ **proc Finalize**(b) Return b
d-$\mathsf{DDHIReal}_{\mathbb{G}}$	d-$\mathsf{DDHIRand}_{\mathbb{G}}$
proc Initialize $a \xleftarrow{\$} \mathbb{Z}_p^*$ Return $([1], [a], \ldots, \left[a^d\right], [1/a])$ **proc Finalize**(b) Return b	**proc Initialize** $a \xleftarrow{\$} \mathbb{Z}_p^*$; $z \xleftarrow{\$} \mathbb{Z}_p^*$ Return $([1], [a], \ldots, \left[a^d\right], [z])$ **proc Finalize**(b) Return b

Figure 2.2: Games defining the DDH and d-DDHI problems in $\mathbb{G}$

We also define the *k-Linear* assumption as the hardness of the following decisional problem. The particular case with $k = 2$ is usually referred to as the Decisional Linear (DLin) assumption.

k-Lin. For $k \geq 2$, the advantage of an adversary $\mathscr{D}$ against the k-Lin problem in $\mathbb{G}$ is defined as:

$$\mathsf{Adv}^{k\text{-}\mathsf{lin}}_{\mathbb{G}}(\mathscr{D}) := \Pr\left[k\text{-}\mathsf{LinReal}^{\mathscr{D}}_{\mathbb{G}} \Rightarrow 1\right] - \Pr\left[k\text{-}\mathsf{LinRand}^{\mathscr{D}}_{\mathbb{G}} \Rightarrow 1\right]$$

where the probabilities are over the choices of $a, z \in \mathbb{Z}_p$, $g \in \mathbb{G}$, and the random coins used by the adversary, and where k-$\mathsf{LinReal}_{\mathbb{G}}$ and k-$\mathsf{LinRand}_{\mathbb{G}}$ are described in Figure 2.3.

```
$k$-LinReal$_{\mathbb{G}}$
proc Initialize
$a_1, \dots, a_k \xleftarrow{\$} \mathbb{Z}_p$
$w_1, \dots, w_k \xleftarrow{\$} \mathbb{Z}_p$
$z \leftarrow w_1 + \dots + w_k$
Return $([1], [a_1], \dots, [a_k],$
        $[a_1 w_1], \dots, [a_k w_k], [z])$
proc Finalize($b$)
Return $b$
```

```
$k$-LinRand$_{\mathbb{G}}$
proc Initialize
$a_1, \dots, a_k \xleftarrow{\$} \mathbb{Z}_p$
$w_1, \dots, w_k \xleftarrow{\$} \mathbb{Z}_p$
$z \xleftarrow{\$} \mathbb{Z}_p$
Return $([1], [a_1], \dots, [a_k],$
        $[a_1 w_1], \dots, [a_k w_k], [z])$
proc Finalize($b$)
Return $b$
```

Figure 2.3: Games defining the k-Lin problem in $\mathbb{G}$

2.2.3 Building DDH-Hard Groups

It is widely accepted that one can build groups in which the DDH assumption holds (and thus, also the DL assumption) from elliptic curves [Mil86; Kob87; Ber06].

Specifically, for a security parameter κ, it is believed that in "reasonably well-chosen" elliptic curves of prime order p, the only way to compute the discrete logarithm is by using generic algorithm (such as Shank's baby-step giant-step algorithm [Sha71], which runs in time $O(\sqrt{p})$). Therefore, with p being a 2κ-bit prime number, one can find an elliptic curve of order p that provides κ bits of security for the discrete logarithm problem. Please note that, using point compression, elements from such a group can be represented using about $2\kappa + 1$ bits.

In the sequel, we do not extend further on how groups in which our assumptions hold can be generated. We only assume that it can be done efficiently (in polynomial time) and that there exist small (polynomial-size) representations for the group and its elements. This is sufficient for our purpose.

2.3 Cryptographic Primitives

2.3.1 Collision-Resistant Hash Functions

Definition 2.3.1 (Collision-Resistant Hash Function)**.** *A collision-resistant hash function is defined by a tuple of two polynomial-time algorithms $\mathcal{H} = (\mathsf{H.Setup}, \mathsf{H.Eval})$ such that:*

- $\mathsf{H.Setup}(1^\kappa)$ *outputs some public parameters* pp*;*
- $\mathsf{H.Eval}_{\mathsf{pp}}(m)$ *is a function that depends on* pp*, takes as input a bitstring $m \in \{0,1\}^*$, and deterministically outputs a bitstring $y \in \{0,1\}^{n(\kappa)}$ called the hash value of m. The size of the output is determined by some polynomial $n(\cdot)$.*

Furthermore, we additionally require that any PPT adversary cannot find two inputs $m_0 \neq m_1$ that have the same hash value, unless with negligible property. Formally, we define the advantage of an adversary $\mathscr{A}$ in attacking the collision-resistance security of $\mathcal{H}$ as:

$$\mathsf{Adv}^{\mathsf{cr}}_{\mathcal{H}}(\mathscr{A}) := \Pr\left[\, \mathsf{Exp}^{\mathsf{cr}}(\mathscr{A}, \kappa) = 1 \,\right] ,$$

where $\mathsf{Exp}^{\mathsf{cr}}$ *is defined in Figure 2.4. We say that* $\mathcal{H}$ *is a collision-resistant hash function if the advantage of any PPT adversary in attacking the collision-resistance security of* $\mathcal{H}$ *is negligible.*

In order to ease the reading, we often abuse notation by simply writing that $\mathcal{H}$ is a collision resistant hash function and letting $\mathcal{H}(\cdot)$ stands for $\mathsf{H.Eval}_{\mathsf{pp}}(\cdot)$, where public parameters pp is implicitly sampled in advance as $\mathsf{pp} \overset{\$}{\leftarrow} \mathsf{H.Setup}(1^\kappa)$ for the given security parameter.

2.3.2 Pseudorandom Functions

2.3.2.1 Definition

Definition 2.3.2 (Pseudorandom Function)**.** *A* pseudorandom function *is defined by a tuple of two polynomial-time algorithms* $F = (\mathsf{F.Setup}, \mathsf{F.Eval})$ *such that:*

- $\mathsf{F.Setup}(1^\kappa)$ *outputs some public parameters* pp*;*
- $\mathsf{F.Eval}_{\mathsf{pp}}(\cdot,\cdot)$ *is a function that depends on* pp *and takes as input a secret key* $K \in \mathcal{K}$ *and an input* $x \in \mathcal{D}$ *and deterministically outputs a value* $y = \mathsf{F.Eval}_{\mathsf{pp}}(K, x) \in \mathcal{R}$*.*

Furthermore, we define the advantage of an adversary $\mathscr{D}$ *in attacking the pseudorandom function security of* F *as:*

$$\mathsf{Adv}_F^{\mathsf{prf}}(\mathscr{D}) := \Pr\left[\mathsf{PRFReal}_F^{\mathscr{D}} \Rightarrow 1\right] - \Pr\left[\mathsf{PRFRand}_F^{\mathscr{D}} \Rightarrow 1\right] ,$$

where $\mathsf{PRFReal}_F$ *and* $\mathsf{PRFRand}_F$ *are described in Figure 2.4. We say that* F *is a pseudorandom function if the advantage of any PPT adversary in attacking the pseudorandom function security of* F *is negligible.*

$\mathsf{Exp}^{\mathsf{cr}}$	$\mathsf{PRFReal}_F$	$\mathsf{PRFRand}_F$
proc Initialize	**proc Initialize**	**proc Initialize**
$\mathsf{pp} \overset{\$}{\leftarrow} \mathsf{H.Setup}(1^\kappa)$	$\mathsf{pp} \overset{\$}{\leftarrow} \mathsf{F.Setup}(1^\kappa)$	$\mathsf{pp} \overset{\$}{\leftarrow} \mathsf{F.Setup}(1^\kappa)$
Return pp	$K \overset{\$}{\leftarrow} \mathcal{K}$	$f \overset{\$}{\leftarrow} \mathsf{Fun}(\mathcal{D}, \mathcal{R})$
proc Finalize(m_0, m_1)	Return pp	Return pp
Return $\mathsf{H.Eval}_{\mathsf{pp}}(m_0) = \mathsf{H.Eval}_{\mathsf{pp}}(m_1)$	**proc Fn**(x)	**proc Fn**(x)
	Return $\mathsf{F.Eval}_{\mathsf{pp}}(K, x)$	Return $f(x)$
	proc Finalize(b)	**proc Finalize**(b)
	Return b	Return b

Figure 2.4: Security games for collision-resistant hash functions and pseudorandom functions

Remark 2.3.3. Please note that in the **Initialize** procedure of game $\mathsf{PRFRand}_F$, a function f is picked uniformly at random from $\mathsf{Fun}(\mathcal{D}, \mathcal{R})$. Defining such a function could be an exponential time procedure if $\mathcal{D}$ has an exponential size. However, as the adversary is allowed to do only a polynomial number of queries, this game can be simulated polynomially by generating f lazily. That is, when the adversary asks for the evaluation of f on an input x, we either pick the output $f(x)$ uniformly at random in $\mathcal{R}$ and store $(x, f(x))$ in a table (if

x has never been queried before), or returns the value y such that (x, y) is in this table (if x has already been queried). This is a polynomial-time procedure (and it also requires a polynomial-size memory).

In order to ease the reading, we often abuse notation by simply writing that $F\colon \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ is a pseudorandom function and letting $F(\cdot,\cdot)$ stands for $\mathsf{F.Eval}_{\mathsf{pp}}(\cdot,\cdot)$, where public parameters pp are implicitly sampled in advance as $\mathsf{pp} \overset{\$}{\leftarrow} \mathsf{F.Setup}(1^\kappa)$ for the given security parameter.

2.3.2.2 Naor-Reingold Construction

The construction by Naor and Reingold [NR97] is the first pseudorandom function based on number-theoretic assumptions. It has the major interest to be very efficient compared to the Goldreich-Goldwasser-Micali construction. We denote $\mathsf{NR} = (\mathsf{NR.Setup}, \mathsf{NR.Eval})$ this construction, defined as follows:

- $\mathsf{NR.Setup}(1^\kappa)$ generates public parameters $\mathsf{pp} = (p, \mathbb{G}, g)$ where $(p, \mathbb{G}, g)$ describes a cyclic group of prime order p generated by g;

- $\mathsf{NR.Eval}_{\mathsf{pp}}(\cdot,\cdot)$ takes as input a key $\vec{a} = (a_0, a_1, \ldots, a_n) \in \mathbb{Z}_p^{n+1}$ and an input $x = x_1 \,\|\, \ldots \,\|\, x_n \in \{0,1\}^n$ and outputs $[a_0 \cdot \prod_{i=1}^n a_i^{x_i}]_g$.

As mentioned above, we often use the notation $\mathsf{NR}(\cdot,\cdot)$ to denote $\mathsf{NR.Eval}_{\mathsf{pp}}(\cdot,\cdot)$.

Theorem 2.3.4. *Assuming the* DDH *assumption holds in* $\mathbb{G}$*,* NR *is a pseudorandom function.*

2.3.2.3 Boneh-Montgomery-Raghunathan Construction

We recall the construction by Boneh, Montgomery, and Raghunathan [BMR10]. We denote $\mathsf{BMR} = (\mathsf{BMR.Setup}, \mathsf{BMR.Eval})$ this construction, defined as follows:

- $\mathsf{BMR.Setup}(1^\kappa)$ generates public parameters $\mathsf{pp} = (p, \mathbb{G}, g)$ where $(p, \mathbb{G}, g)$ describes a cyclic group of prime order p generated by g;

- $\mathsf{BMR.Eval}_{\mathsf{pp}}(\cdot,\cdot)$ takes as input a key $\vec{a} = (a_1, \ldots, a_n) \in \mathbb{Z}_p^n$ as well as an input $x = x_1 \,\|\, \ldots \,\|\, x_n \in \{1, \ldots, d\}^n$ and outputs $\left[\prod_{i=1}^n \frac{1}{a_i + x_i}\right]_g$.

Once again, we often use the notation $\mathsf{BMR}(\cdot,\cdot)$ to denote $\mathsf{BMR.Eval}_{\mathsf{pp}}(\cdot,\cdot)$.

Theorem 2.3.5. *Assuming the* d-DDHI *assumption holds in* $\mathbb{G}$*,* BMR *is a pseudorandom function.*

We also introduce two extensions of pseudorandom functions below. Namely, we define aggregate and multilinear pseudorandom functions. These two notions are later studied in Chapter 6. Please note that the definition of related-key secure pseudorandom functions is given only in Chapter 3.

2.3.3 Aggregate Pseudorandom Functions

2.3.3.1 Definition

Aggregate pseudorandom function are an extension of pseudorandom functions that allows to aggregate multiple(possibly exponentially many) outputs of the pseudorandom functions efficiently without impacting the security. The notion has been originally defined by Cohen, Goldwasser, and Vaikunthanathan in [CGV15]. We first define an aggregation function and then recall the definition of an aggregate pseudorandom function.

Aggregation Function. Let $f\colon \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ be a function. We define an aggregation function by describing two objects:

- a collection $\mathscr{S}$ of subsets S of the domain $\mathcal{D}$;
- an aggregation function $\Gamma\colon \mathcal{R}^* \to \mathcal{V}$ that takes as input a tuple of values from the range $\mathcal{R}$ of f and aggregates them to produce a value in an output set $\mathcal{V}$.

In addition, we require the set ensemble $\mathscr{S}$ to be *efficiently recognizable*, meaning that for any $S \in \mathscr{S}$, there exists a polynomial-time procedure to check if $x \in S$, for any $x \in \mathcal{D}$. Also, we require the aggregation function Γ to be polynomial-time and the output of the function not to depend on the order of the elements provided as inputs. Finally, we require all sets S to have a representation of polynomial size in the security parameter κ.

Given an aggregation function $(\mathscr{S}, \Gamma)$, we define the aggregate function $\mathrm{AGG} = \mathrm{AGG}_{f,\mathscr{S},\Gamma}$ as the function that takes as input a set $S \in \mathscr{S}$ and outputs the aggregation of all values $f(x)$ for all $x \in S$. That is, $\mathrm{AGG}(S)$ outputs $\Gamma(f(x_1), \ldots, f(x_{|S|}))$, where $S = \{x_1, \ldots, x_{|S|}\}$. We will require the computation of AGG to be polynomial-time (even if the input set S is exponentially large) if the function f provided is the evaluation function $\mathsf{F.Eval}_{\mathsf{pp}}(K, \cdot)$ of a pseudorandom function $F = (\mathsf{F.Setup}, \mathsf{F.Eval})$ we consider, for some fixed key K.

Definition 2.3.6 (Aggregate Pseudorandom Function)**.** *Let* $F = (\mathsf{F.Setup}, \mathsf{F.Eval})$ *denote a pseudorandom function whose key space, domain, and range are respectively* $\mathcal{K}, \mathcal{D}$*, and* $\mathcal{R}$ *and let* $(\mathscr{S}, \Gamma)$ *be an associated aggregation function. We say that* F *is an* $(\mathscr{S}, \Gamma)$-aggregate pseudorandom function *if the advantage of any PPT adversary in attacking the aggregate pseudorandom function security of* F *is negligible, where the advantage of an adversary* $\mathscr{D}$ *is defined via*

$$\mathsf{Adv}^{\mathsf{agg\text{-}prf}}_{F,\mathscr{S},\Gamma}(\mathscr{D}) := \Pr\left[\mathsf{AGGPRFReal}^{\mathscr{D}}_F \Rightarrow 1\right] - \Pr\left[\mathsf{AGGPRFRand}^{\mathscr{D}}_F \Rightarrow 1\right] \ ,$$

where games $\mathsf{AGGPRFReal}_F$ *and* $\mathsf{AGGPRFRand}_F$ *are depicted in Figure 2.5.*

Remark 2.3.7. Again, game $\mathsf{AGGPRFRand}_F$ may not be polynomial-time, as $\mathrm{AGG}_{f,\mathscr{S},\Gamma}$ may require to compute an exponential number of values $f(x)$. However, for all the aggregate pseudorandom functions that we consider in this manuscript, this game is statistically indistinguishable from a polynomial-time game, by doing lazy simulation. Further details are given in the corresponding sections.

2.3.3.2 Cohen-Goldwasser-Vaikunthanathan Construction

To illustrate this definition, we give one of the constructions proposed in [CGV15], for hypercube aggregation.

$\mathsf{AGGPRFReal}_F$	$\mathsf{AGGPRFRand}_F$
proc Initialize	**proc Initialize**
$\mathsf{pp} \xleftarrow{\$} \mathsf{F.Setup}(1^\kappa)$	$\mathsf{pp} \xleftarrow{\$} \mathsf{F.Setup}(1^\kappa)$
$K \xleftarrow{\$} \mathcal{K}$	$f \xleftarrow{\$} \mathsf{Fun}(\mathcal{D}, \mathcal{R})$
Return pp	Return pp
proc Fn(x)	**proc Fn**(x)
Return $\mathsf{F.Eval}_{\mathsf{pp}}(K, x)$	Return $f(x)$
proc AGG(S)	**proc AGG**(S)
Return $\mathrm{AGG}_{\mathsf{F.Eval}_{\mathsf{pp}}(K,\cdot),\mathscr{S},\Gamma}(S)$	Return $\mathrm{AGG}_{f,\mathscr{S},\Gamma}(S)$

Figure 2.5: Security games for aggregate pseudorandom functions

We consider the Naor-Reingold pseudorandom function, whose key space, domain, and range are respectively $\mathbb{Z}_p^{n+1}$, $\{0,1\}^n$, and $\mathbb{G} = \langle g \rangle$ a cyclic group of prime order p. Then, the aggregation function for hypercubes is defined as:

- $\mathscr{S} = \{S_z \mid z \in \{0,1,\star\}^n\}$, where for any $z \in \{0,1,\star\}^n$, S_z is the subset of $\{0,1\}^n$ containing all bitstrings x such that $x_i = z_i$ if $z_i \in \{0,1\}$ and $x_i \in \{0,1\}$ if $z_i = \star$, for $i = 1, \ldots, n$. For instance, $S_{\star^n} = \{0,1\}^n$ and $S_{0\star} = \{00, 01\}$;
- Γ: $\mathbb{G}^* \to \mathbb{G}$ takes as input a tuple of elements $(g_1, \ldots, g_\ell) \in \mathbb{G}^\ell$ and outputs their product $\prod_{i=1}^{\ell} g_i$.

Then, it is clear that $\mathrm{AGG}_{\mathsf{NR.Eval}_{\mathsf{pp}}(\vec{a},\cdot),\mathscr{S},\Gamma}(S_z) = \left[a_0 \cdot \prod_{z_i=1} a_i \cdot \prod_{z_i=\star}(a_i + 1)\right]$, and security immediately follows from the security of NR (please refer to the introduction for further explanation).

Theorem 2.3.8. *Assuming the sub-exponential hardness of the* DDH *problem in* $\mathbb{G}$*,* NR *is an* $(\mathscr{S}, \Gamma)$*-aggregate pseudorandom function.*

It was later shown by Cohen and Holmgren in [CH15] that the above construction is actually secure assuming only the polynomial hardness of the DDH problem in $\mathbb{G}$, introducing multilinear pseudorandom functions.

2.3.4 Multilinear Pseudorandom Functions

2.3.4.1 Definition

Multilinear pseudorandom functions are a variant of the standard notion of pseudorandom functions, which works with vector spaces. It has been originally defined in [CH15] as follows.

Definition 2.3.9 (Multilinear Pseudorandom Function)**.** *A* multilinear pseudorandom function *is defined by a tuple of two polynomial-time algorithms* $F = (\mathsf{F.Setup}, \mathsf{F.Eval})$*, such that:*

- $\mathsf{F.Setup}(1^\kappa)$ *outputs some public parameters* pp*;*

- $\mathsf{F.Eval}_{\mathsf{pp}}(\cdot,\cdot)$ *is a function that depends on* pp *and takes as input a secret key* $K \in \mathcal{K}$ *and an input* $x \in \mathcal{D}$ *and deterministically outputs a value* $y = \mathsf{F.Eval}_{\mathsf{pp}}(K, x) \in \mathcal{R}$.

Moreover, its domain $\mathcal{D} = \mathcal{D}_1 \times \cdots \times \mathcal{D}_n$ *is a cartesian product of* n *vector spaces* $\mathcal{D}_1, \ldots, \mathcal{D}_n$*, for some integer* n*, and its range* $\mathcal{R}$ *is a vector space. We say that* F *is a multilinear pseudorandom function if the advantage of any PPT adversary in attacking the multilinear pseudorandom function security of* F *is negligible, where the advantage of an adversary* $\mathscr{D}$ *is defined via*

$$\mathsf{Adv}_F^{\mathsf{mprf}}(\mathscr{D}) := \Pr\left[\mathsf{MPRFReal}_F^{\mathscr{D}} \Rightarrow 1\right] - \Pr\left[\mathsf{MPRFRand}_F^{\mathscr{D}} \Rightarrow 1\right] ,$$

where games $\mathsf{MPRFReal}_F$ *and* $\mathsf{MPRFRand}_F$ *are depicted in Figure 2.6.*

$\mathsf{MPRFReal}_F$	$\mathsf{MPRFRand}_F$
proc Initialize	**proc Initialize**
$\mathsf{pp} \xleftarrow{\$} \mathsf{F.Setup}(1^\kappa)$	$\mathsf{pp} \xleftarrow{\$} \mathsf{F.Setup}(1^\kappa)$
$K \xleftarrow{\$} \mathcal{K}$	$f \xleftarrow{\$} \mathsf{L}(\mathcal{D}_1 \otimes \cdots \otimes \mathcal{D}_n, \mathcal{R})$
Return pp	Return pp
proc Fn$(\vec{x})$	**proc Fn**$(\vec{x})$
Return $\mathsf{F.Eval}_{\mathsf{pp}}(K, \vec{x})$	Return $f(\vec{x})$

Figure 2.6: Security games for multilinear pseudorandom function

Remark 2.3.10. Once again, game $\mathsf{MPRFRand}_F$ can be implemented in polynomial time using lazy simulation, for instance using a deterministic algorithm to check linearity of simple tensors. Such a procedure is given in [BW04].

2.3.4.2 Cohen-Holmgren Construction

We also recall the first construction given by Cohen and Holmgren in [CH15]. We denote $\mathsf{CH} = (\mathsf{CH.Setup}, \mathsf{CH.Eval})$ this construction, defined as follows:

- $\mathsf{CH.Setup}(1^\kappa)$ generates public parameters $\mathsf{pp} = (p, \mathbb{G}, g, \ell_1, \ldots, \ell_n)$ where $(p, \mathbb{G}, g)$ describes a cyclic group of prime order p generated by g, and $\ell_1, \ldots, \ell_n$ describe the dimensions of vector spaces such that $\mathcal{D} := \mathcal{K} := \mathbb{Z}_p^{\ell_1} \times \cdots \times \mathbb{Z}_p^{\ell_n}$;

- $\mathsf{CH.Eval}_{\mathsf{pp}}(\cdot,\cdot)$ takes as input a key $(\vec{a_1}, \ldots, \vec{a_n}) \in \mathbb{Z}_p^{\ell_1} \times \cdots \times \mathbb{Z}_p^{\ell_n}$ as well as an input $(\vec{x_1}, \ldots, \vec{x_n}) \in \mathbb{Z}_p^{\ell_1} \times \cdots \times \mathbb{Z}_p^{\ell_n}$ and outputs $\left[\prod_{i=1}^n \langle \vec{a_i}, \vec{x_i} \rangle\right]_g$.

Once again, we often use the notation $\mathsf{CH}(\cdot,\cdot)$ to denote $\mathsf{CH.Eval}_{\mathsf{pp}}(\cdot,\cdot)$.

Theorem 2.3.11. *Assuming the* DDH *assumption holds in* $\mathbb{G}$*,* CH *is a multilinear pseudorandom function.*

2.4 Multilinear Maps and the Generic Multilinear Map Model

In this section, we introduce multilinear maps and the generic multilinear map model, that are central in Chapter 7. The generic multilinear map model is also used in Section 4.4 under the term multilinear group model but in a quite informal manner, and we provide a simple intuition of this model in the corresponding section. Thus, though we encourage the reader to get used to this model before reading Section 4.4, this section is crucial only for reading Chapter 7.

2.4.1 Multilinear Maps

Multilinear maps, also known as graded multilinear maps or graded encodings, are a generalization of bilinear maps. We recall that a bilinear map is an application $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ that satisfies $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$, where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are cyclic groups of same order N, respectively generated by $g_1, g_2, e(g_1, g_2)$. From a high-level view, a multilinear map allows to transform scalars x and y into encodings, denoted $\hat{x}$ and $\hat{y}$, at any level, of a given hierarchy. Moreover, it allows to perform arithmetic operations: specifically, given encodings $\hat{x}, \hat{y}$ at a same level $\mathcal{S}$, one can compute an encoding of $x + y$ at level $\mathcal{S}$; given encodings $\hat{x}, \hat{y}$ at levels $\mathcal{S}_1, \mathcal{S}_2$ respectively, one can compute an encoding of xy at a level $\mathcal{S}_1 \star \mathcal{S}_2$, where $\star$ is an operation over the levels to be precised. In the case of a symmetric bilinear map $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, one can let the levels to be 0 for scalars, 1 for elements in $\mathbb{G}$ and 2 for elements in $\mathbb{G}_T$, so that given two encodings at a same level i, one can create an encoding of their sum (by summing scalars or multiplying group elements), and given two encodings at levels i and j, one can create an encoding of their product at level $i + j$ if $i + j \leq 2$ (by multiplying scalars, raising a group element to some scalar, or applying the pairing to two group elements), so $\star$ is simply the addition in this case. As in the case of symmetric bilinear maps, for multilinear maps, these levels in the hierarchy limit the operations that one can perform: one can always perform additions at a fixed level and one always "increases" the level when performing a multiplication.

In the case of an asymmetric bilinear map $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, we usually identify levels with the subset lattice $\emptyset \subseteq \{A\}, \{B\} \subseteq \{A, B\}$, with $\emptyset$ corresponding to scalars, $\{A\}$ to $\mathbb{G}_1$, $\{B\}$ to $\mathbb{G}_2$ and $\{A, B\}$ to $\mathbb{G}_T$. With these notations, the operation $\star$ simply becomes the union of the levels, as set union. This notion naturally extends to the case of asymmetric multilinear maps: we identify levels to sets of formal symbols. In this thesis, we follow this notion, and formal definitions are detailed below and follow the notations of [Zim15].

Definition 2.4.1 (Formal Symbol). *A* formal symbol *is a bitstring in* $\{0, 1\}^*$. *Distinct variables denote distinct bitstrings, and we call a* fresh *formal symbol any bitstring in* $\{0, 1\}^*$ *that has not already been assigned to a formal symbol.*

Definition 2.4.2 (Index Sets). *An* index set *is a set of formal symbols called* indices.

Definition 2.4.3 (Multilinear Map). *A multilinear map is defined by a tuple of six algorithms* (MM.Setup, MM.Encode, MM.Add, MM.Mult, MM.ZeroTest, MM.Extract) *with the following properties:*

- MM.Setup *takes as inputs the security parameter κ in unary and an index set $\mathscr{U}$, termed the* top-level index set*, and generates public parameters* mm.pp*, secret parameters* mm.sp*, and a prime number p;*

- $\mathsf{MM.Encode}$ *takes as inputs secret parameters* $\mathsf{mm.sp}$*, a scalar* $x \in \mathbb{Z}_p$*, and an index set* $\mathcal{S} \subseteq \mathscr{U}$ *and outputs:*

$$\mathsf{MM.Encode}(\mathsf{mm.sp}, x, \mathcal{S}) \to [x]_{\mathcal{S}} \quad ;$$

Please note that for the index set $\mathcal{S} = \emptyset$*,* $[x]_{\emptyset}$ *is simply the scalar* $x \in \mathbb{Z}_p$*.*

- $\mathsf{MM.Add}$ *takes as inputs public parameters* $\mathsf{mm.pp}$ *and two encodings with same index set* $\mathcal{S} \subseteq \mathscr{U}$ *and outputs:*

$$\mathsf{MM.Add}(\mathsf{mm.pp}, [x]_{\mathcal{S}}, [y]_{\mathcal{S}}) \to [x+y]_{\mathcal{S}} \quad ;$$

- $\mathsf{MM.Mult}$ *takes as inputs public parameters* $\mathsf{mm.pp}$ *and two encodings with index sets* $\mathcal{S}_1, \mathcal{S}_2 \subseteq \mathscr{U}$ *respectively and outputs:*

$$\mathsf{MM.Mult}(\mathsf{mm.pp}, [x]_{\mathcal{S}_1}, [y]_{\mathcal{S}_2}) \to \begin{cases} [xy]_{\mathcal{S}_1 \cup \mathcal{S}_2} & \textit{if } \mathcal{S}_1 \cup \mathcal{S}_2 \subseteq \mathscr{U} \\ \bot & \textit{otherwise} \end{cases} \quad ;$$

- $\mathsf{MM.ZeroTest}$ *takes as inputs public parameters* $\mathsf{mm.pp}$ *and a* top-level *encoding (with index set* $\mathscr{U}$*) and outputs:*

$$\mathsf{MM.ZeroTest}(\mathsf{mm.pp}, [x]_{\mathcal{S}}) \to \begin{cases} \textit{“zero”} & \textit{if } \mathcal{S} = \mathscr{U} \textit{ and } x = 0 \\ \textit{“non-zero”} & \textit{otherwise} \end{cases} \quad ;$$

- $\mathsf{MM.Extract}$ *takes public parameters* $\mathsf{mm.pp}$ *and a top-level encoding* $[x]_{\mathscr{U}}$ *as inputs and outputs a canonical and random representation of* $[x]_{\mathscr{U}}$*.*

Remark 2.4.4. The $\mathsf{MM.Extract}$ algorithm is needed for our pseudorandom function, constructed in Chapter 7, to be deterministic with all currently known instantiations of multilinear maps [GGH13; CLT13; GGH15; CLT15]. Indeed, in these instantiations, the same group element has many different representations, and the extraction procedure enables to extract a unique representation from any top-level group element (i.e., of index $\mathscr{U}$).

This extraction is necessary for our proof under non-interactive assumptions in Section 7.4.2 to work. For our proofs in the generic multilinear map model, this is not required. For this reason, our generic multilinear map model does not support extraction for the sake of simplicity. Actually, this only strengthens the result, as before extraction, the adversary still has to possibility to add top-level group elements while extracted values are not necessarily homomorphic.

2.4.2 Generic Multilinear Map Model

The generic multilinear map model is a generalization of the generic group model [Sho97]. Roughly speaking, the adversary has only the capability to apply operations (add, multiply, and zero-test) of the multilinear map to encodings. A scheme is secure in the generic multilinear map model if for any adversary breaking the real scheme, there is a generic adversary that breaks a modified scheme in which encodings are replaced by fresh nonces, called *handles*, that it can supply to a stateful oracle $\mathscr{M}$, defined as follows:

Definition 2.4.5 (Generic Multilinear Map Oracle). *A* generic multilinear map oracle *is a stateful oracle $\mathscr{M}$ that responds to queries as follows:*

- *On a query* $\mathsf{MM.Setup}(1^\kappa, \mathscr{U})$*, $\mathscr{M}$ generates a prime number p and parameters* $\mathsf{mm.pp}$, $\mathsf{mm.sp}$ *as fresh nonces chosen uniformly at random from $\{0,1\}^\kappa$. It also initializes an internal table $T \leftarrow []$ that it uses to store queries and handles. It finally returns* $(\mathsf{mm.pp}, \mathsf{mm.sp}, p)$ *and set internal state so that subsequent* $\mathsf{MM.Setup}$ *queries fail.*
- *On a query* $\mathsf{MM.Encode}(z, x, \mathcal{S})$*, with $z \in \{0,1\}^\kappa$ and $x \in \mathbb{Z}_p$, it checks that $z =$* $\mathsf{mm.sp}$ *and $\mathcal{S} \subseteq \mathscr{U}$ and outputs $\perp$ if the check fails, otherwise it generates a fresh handle $h \xleftarrow{\$} \{0,1\}^\kappa$, adds $h \mapsto (x, \mathcal{S})$ to T, and returns h.*
- *On a query* $\mathsf{MM.Add}(z, h_1, h_2)$*, with $z, h_1, h_2 \in \{0,1\}^\kappa$, it checks that $z =$* $\mathsf{mm.pp}$*, that h_1 and h_2 are handles in T which are mapped to values $(x_1, \mathcal{S}_1)$ and $(x_2, \mathcal{S}_2)$ such that $\mathcal{S}_1 = \mathcal{S}_2 = \mathcal{S} \subseteq \mathscr{U}$, and returns $\perp$ if the check fails. If it passes, it generates a fresh handle $h \xleftarrow{\$} \{0,1\}^\kappa$, adds $h \mapsto (x_1 + x_2, \mathcal{S})$ to T, and returns h.*
- *On a query* $\mathsf{MM.Mult}(z, h_1, h_2)$*, with $z, h_1, h_2 \in \{0,1\}^\kappa$, it checks $z =$* $\mathsf{mm.pp}$*, that h_1 and h_2 are handles in T which are mapped to values $(x_1, \mathcal{S}_1)$ and $(x_2, \mathcal{S}_2)$ such that $\mathcal{S}_1 \cup \mathcal{S}_2 \subseteq \mathscr{U}$, and returns $\perp$ if the check fails. If it passes, it generates a fresh handle $h \xleftarrow{\$} \{0,1\}^\kappa$, adds $h \mapsto (x_1 x_2, \mathcal{S}_1 \cup \mathcal{S}_2)$ to T, and returns h.*
- *On a query* $\mathsf{MM.ZeroTest}(z, h)$*, with $z, h \in \{0,1\}^\kappa$, it checks $z =$* $\mathsf{mm.pp}$*, that h is a handle in T such that it is mapped to a value $(x, \mathscr{U})$, and returns $\perp$ if the check fails. If it passes, it returns "zero" if $x = 0$ and "non-zero" otherwise.*

Chapter 3

Introduction to Related-Key Security

In this chapter, we provide an introduction to related-key security. We start by defining the security model for pseudorandom functions and by recalling first results by Bellare and Kohno [BK03]. In particular, we recall some impossibility results. Next, we describe how related-key secure pseudorandom functions play a central role in related-key security. Indeed, many related-key secure cryptographic primitives can be built directly from related-key pseudorandom functions, as shown by Bellare, Cash, and Miller [BCM11]. Finally, we recall, correct, and extend the Bellare-Cash framework [BC10b] and provide some simple applications of the resulting frameworks.

This framework was originally introduced in 2010 as a solution to construct the first related-key secure pseudorandom functions under standard assumptions. Unfortunately, its proof suffered from a minor bug and the framework had to be slightly changed, weakening the results of the paper. Here, we provide a different approach for proving its security, which allows us to recover all the results from the original paper, and more. This justifies our claim of repairing the Bellare-Cash framework.

Contents

3.1 Definition and Security Model

As already detailed in the introduction of this thesis, the classical security models usually assume that an adversary has only black-box access to the cryptosystem. For instance, in the case of pseudorandom functions, the classical security model (see Definition 2.3.2) only allows the adversary to query the pseudorandom function on inputs of its choice. In particular, the key is perfectly hidden and the adversary has no power regarding the choice of the key. However, in the real world, it has been shown that an adversary might be more powerful, and might be able to modify the key.

The related-key security model provides the adversary with such capacities. Specifically, in this model, an adversary is associated to a set Φ of functions, termed related-key deriving functions, and is given the capacity to force the cryptosystem to work on different but related keys, obtained by applying functions of Φ to the original key. In particular, if Φ only contains the identity function, we obtain the classical security notion. This model has been formalized by Bellare and Kohno in [BK03] as follows.

Definition 3.1.1. *[Related-Key Secure Pseudorandom Function] Let $F = (\mathsf{F.Setup}, \mathsf{F.Eval})$ denote a pseudorandom function whose key space, domain, and range are respectively denoted $\mathcal{K}, \mathcal{D}$, and $\mathcal{R}$. We define the advantage of an adversary $\mathscr{D}$ in attacking the* related-key security *of F as:*

$$\mathsf{Adv}^{\mathsf{rka\text{-}prf}}_{F,\Phi}(\mathscr{D}) := \Pr\left[\mathsf{RKPRFReal}^{\mathscr{D}}_{F} \Rightarrow 1\right] - \Pr\left[\mathsf{RKPRFRand}^{\mathscr{D}}_{F} \Rightarrow 1\right] \ ,$$

where $\mathsf{RKPRFReal}_F$ and $\mathsf{RKPRFRand}_F$ are described in Figure 3.1. Furthermore, for a set of functions $\Phi \subseteq \mathsf{Fun}(\mathcal{K}, \mathcal{K})$, termed a class of related-key deriving (RKD) functions*, we say that an adversary $\mathscr{D}$ is Φ-restricted, if it is restricted to make only queries (ϕ, x) with $\phi \in \Phi$. Then, we say that F is a* Φ-related-key secure pseudorandom function *if the advantage of any Φ-restricted PPT adversary in attacking the related-key security of F is negligible.*

$\mathsf{RKPRFReal}_F$	$\mathsf{RKPRFRand}_F$
proc Initialize	**proc Initialize**
$\mathsf{pp} \xleftarrow{\$} \mathsf{F.Setup}(1^\kappa)$	$\mathsf{pp} \xleftarrow{\$} \mathsf{F.Setup}(1^\kappa)$
$K \xleftarrow{\$} \mathcal{K}$	$f \xleftarrow{\$} \mathsf{Fun}(\mathcal{K} \times \mathcal{D}, \mathcal{R})$
Return pp	$K \xleftarrow{\$} \mathcal{K}$
proc RKFn(ϕ, x)	Return pp
Return $\mathsf{F.Eval}_{\mathsf{pp}}(\phi(K), x)$	**proc RKFn**(ϕ, x)
proc Finalize(b)	Return $f(\phi(K), x)$
Return b	**proc Finalize**(b)
	Return b

Figure 3.1: Security games for related-key pseudorandom functions

Remark 3.1.2. Similarly to the classical pseudorandom function security model, the **Initialize** procedure of game $\mathsf{RKPRFRand}$ could not be a polynomial-time procedure, according to our definition. This issue can easily be circumvent (see Remark 2.3.3). Please also note that the adversary cannot modify the public parameters pp. In particular, assuming the public parameters contain the description of a group $(p, \mathbb{G}, g)$, the adversary does not have any power regarding these values.

As described in the following section, there exist classes Φ of RKD functions such that one cannot construct Φ-related-key secure pseudorandom functions. Therefore, one of the goals in related-key security is to understand which classes of RKD functions can be handled and if one can protect cryptosystems against meaningful (real-world) classes.

3.2 First Impossibility and Feasibility Results

Beyond formalizing the security model for related-key security, Bellare and Kohno also proposed first impossibility and feasibility results in this setting. Here we describe succinctly and informally these results. The main point is to emphasize that depending on the classes of related-key deriving functions, it might be impossible to achieve related-key security. For further details, please refer to the original paper [BK03].

3.2.1 Impossibility Results

While related-key security is something we need in the real world, Bellare and Kohno proved that it is not achievable for certain classes of functions. The first example is obvious: assuming a class Φ of related-key deriving functions contains a constant function f equals to c, then achieving Φ-related-key security is impossible. This impossibility result is immediate, as the adversary has the power, applying f to the key, to make the cryptosystem run with the known key c.

They also proposed more elaborate impossibility results. For instance, assume the key space is $\{0,1\}^\kappa$ and assume that Φ contains the identity function and the permutations ϕ_i, for $i = 1, \ldots, k$, where $\phi_i(K) = K$ if $K_i = 0$ and $\phi_i(K) = K \oplus 1^{i-1} \,\|\, 0 \,\|\, 1^{\kappa-i}$ otherwise. Then, one can just pick any input x in the domain of the pseudorandom function and query $(\mathsf{id}, x), (\phi_1, x), \ldots, (\phi_\kappa, x)$. For every $i = 1, \ldots, \kappa$, if the values obtained with queries (id, x) and (ϕ_i, x) match, then the i-th bit of K is 0 with overwhelming probability (assuming the range is big, otherwise one can just do this comparison with multiple inputs). Thus, it is obvious that one cannot achieve related-key security against such a class.

Similarly, assume that Φ contains XOR relations with bitstrings $0^{i-1} \,\|\, 1 \,\|\, 0^{\kappa-i}$ and additions with 2^{i-1} modulo 2^κ (seeing the key as the bit-representation of an integer modulo 2^κ), for $i = 1, \ldots, \kappa$, then a similar attack can be mounted from the fact that $K \oplus 0^{i-1} \,\|\, 1 \,\|\, 0^{\kappa-i}$ and $K + 2^{\kappa-i-1} \bmod 2^\kappa$ match if and only if $K_i = 0$.

Ton conclude, it is clear that there is no hope in proving related-key security for certain classes, and it is of prime interest to understand which types of classes can be handled (and if natural classes, such as XOR relations, can be handled).

3.2.2 Feasibility Results

Though it is impossible to protect against certain classes of functions, related-key security for real-world classes remains one of the main goals of modern blockciphers, such as AES. Hence, Bellare and Kohno also studied the related-key security of blockciphers and proposed first feasibility results. Specifically, they prove that an ideal blockcipher is Φ-related-key secure as soon as the class of functions Φ satisfies two properties, termed collision-resistance and output-unpredictability. Informally, consider a random key K, a small subset S of Φ, and a small subset X of the key space $\mathcal{K}$, then collision-resistance asks that the probability that $\phi_1(K) = \phi_2(K)$ for two distinct functions $\phi_1, \phi_2 \in S$, is small, while output-unpredictability asks that the probability that there exists a function $\phi \in S$ such that $\phi(K) \in X$ is small. Intuitively, collision-resistance guarantees that every related-key is distinct, while output-unpredictability guarantees that every related-key is random, and thus, related-key security simply follows from standard security, as running the related-key security game is not different from running multiple instances of the standard security game with different, random, and independent keys.

In particular, if Φ contains only permutations, collision-resistance is immediate and an ideal blockcipher is Φ-related-key secure as soon as Φ satisfies output-unpredictability. Thus, as the class of XOR relations contains only permutations and satisfies output-unpredictability, this feasibility proves that ideal blockciphers are secure against XOR relations.

3.3 The Central Role of Pseudorandom Functions

Almost ten years after this first theoretical treatment of related-key security, building related-key secure cryptosystems in the standard model remained a wide-open problem. Bellare and Cash proposed the first construction of related-key secure pseudorandom function under standard assumptions only in 2010. This description is described in the next section, but before giving details about this construction, let us discuss about other primitives and why pseudorandom functions are actually fundamental in related-key security.

Related-key security solves practical issues, such as certain types of side-channel attacks, so it might seem natural to focus on primitives that are widely used in practice, such as symmetric encryption or signatures. However, it appears that building related-key secure pseudorandom functions is actually sufficient to build most of the primitive we might be interested in. Indeed, in [BCM11], Bellare, Cash, and Miller proved that, given a Φ-related-key secure pseudorandom function, one can transform any symmetric encryption (resp. signature, public-key encryption, identity-based encryption) scheme that is secure in the classical security model into a Φ-related-key secure one.

Moreover, the transform is rather simple and practical: informally, given a related-key secure pseudorandom function, one can build a related-key secure pseudorandom generator. Then, the transform mainly consists in fixing the secret key to be some random K and in switching the randomness used in the key generation algorithm to the output of the related-key secure pseudorandom generator evaluated on K. As the pseudorandom generator is related-key secure, running multiple instances with different related keys simply corresponds to running multiple instances with independent and uniformly random keys, and security follows.

Therefore, pseudorandom functions should be the main focus in related-key security, as the allows to build most of the interesting primitives for real-world applications. We do not further discuss this result and encourage the reader to read [BCM11] for details.

3.4 The Bellare-Cash Framework, Revisited

In this section, we describe the Bellare-Cash framework [BC10b]. This framework, introduced in 2010, allowed to build the first related-key secure pseudorandom functions under standard assumptions. Unfortunately, the proof of this framework presented a minor bug, and the authors modified the construction, by strengthening the requirements of their framework, in order to correct this issue. This modification weakens the framework and prevents it from being applied to certain classes of functions. Here, we provide a new proof that slightly deviates from the original (and incorrect) proof by Bellare and Cash. Our approach allows us to correct the bug from the original framework and to recover all the original results. In particular, our requirements are exactly the same as those of the original framework. Here, we directly present a simple extension of this framework that let us handle larger classes of RKD functions. The original Bellare-Cash framework consists in a particular case of our framework.

Until the end of this chapter, we denote by $F = (\mathsf{F.Setup}, \mathsf{F.Eval})$ a pseudorandom function whose key space, domain, and range are respectively denoted by $\mathcal{K}$, $\mathcal{D}$, and $\mathcal{R}$. For simplicity, we often denote $\mathsf{F.Eval}_{\mathsf{pp}}(K, x)$ by $F(K, x)$ throughout this chapter, and when doing so, setup and public parameters are implicit. In particular, we adopt these notations for the proof of the framework to ease the reading. We first introduce some additional material that is later used for the framework and its proof.

3.4.1 Additional Notions

We define the notions of *strong key fingerprint, unique-input adversary, key-malleability*, and *compatible hash function* introduced in [BC10b] and used in the framework.

Strong Key Fingerprint. A strong key fingerprint is a tool used to detect whether a key arises more than once in a simulation, even if we do not have any information about the key itself. Let $\vec{\omega}$ be a vector over $\mathcal{D}$ and let n its length. We say that $\vec{\omega}$ is a *strong key fingerprint* for the pseudorandom function $F = (\mathsf{F.Setup}, \mathsf{F.Eval})$ if

$$(\mathsf{F.Eval}_{\mathsf{pp}}(K, \omega_1), \ldots, \mathsf{F.Eval}_{\mathsf{pp}}(K, \omega_n)) \neq (\mathsf{F.Eval}_{\mathsf{pp}}(K', \omega_1), \ldots, \mathsf{F.Eval}_{\mathsf{pp}}(K', \omega_n)) \ ,$$

for all distinct $K, K' \in \mathcal{K}$ and any public parameters pp.

Unique-Input Adversary. We say that an adversary against the related-key security of a pseudorandom function is *unique-input* if its oracle queries $(\phi_1, x_1), \ldots, (\phi_q, x_q)$ are such that $x_1, \ldots, x_q$ are always distinct.

Key-Malleability. Let Φ be a class of RKD functions. Suppose KT is a deterministic algorithm that, given an oracle $f\colon \mathcal{D} \to \mathcal{R}$ and inputs $(\phi, x) \in \Phi \times \mathcal{D}$, returns a point $\mathsf{KT}^f(\phi, x) \in \mathcal{R}$. We say that KT is a *key-transformer* for (F, Φ) if it satisfies the following *correctness* and *uniformity* conditions. *Correctness* requires that, for any public parameters pp, $\mathsf{KT}^{\mathsf{F.Eval}_{\mathsf{pp}}(K,\cdot)}(\phi, x) = \mathsf{F.Eval}_{\mathsf{pp}}(\phi(K), x)$ for every $(\phi, K, x) \in \Phi \times \mathcal{K} \times \mathcal{D}$. *Uniformity* requires that for any (even inefficient) Φ-restricted unique-input adversary $\mathscr{U}$,

$$\Pr\left[\mathsf{KTReal}_{\mathsf{KT}}^{\mathscr{U}} \Rightarrow 1\right] = \Pr\left[\mathsf{KTRand}_{\mathsf{KT}}^{\mathscr{U}} \Rightarrow 1\right] \ ,$$

where games $\mathsf{KTReal}_{\mathsf{KT}}$ and $\mathsf{KTRand}_{\mathsf{KT}}$ are described in Figure 3.2. Finally, we say that F is Φ-*key-malleable* if such a key-transformer for (F, Φ) exists.

```
$\mathsf{KTReal}_{\mathsf{KT}}$
proc Initialize
    $f \xleftarrow{\$} \mathsf{Fun}(\mathcal{D},\mathcal{R})$
proc KTFn($\phi, x$)
    Return $\mathsf{KT}^f(\phi(K), x)$
proc Finalize($b$)
    Return $b$
```

```
$\mathsf{KTRand}_{\mathsf{KT}}$
proc KTFn($\phi, x$)
    $y \xleftarrow{\$} \mathcal{R}$
    Return $y$
proc Finalize($b$)
    Return $b$
```

Figure 3.2: Games defining the uniformity of a key-transformer

Compatible Hash Function. Let us assume that F is Φ-key malleable and denote by KT a key-transformer for (F,Φ). Let $\vec{\omega} \in \mathcal{D}^m$ and let $\overline{\mathcal{D}} = \mathcal{D} \times \mathcal{R}^m$. We denote by $\mathsf{Qrs}(\mathsf{KT}, F, \Phi, \vec{\omega})$ the set of all $w \in \mathcal{D}$ such that there exists $(f, \phi, i) \in \mathsf{Fun}(\mathcal{D},\mathcal{R}) \times \Phi \times \{1,\ldots,m\}$ such that the computation of $\mathsf{KT}^f(\phi, \omega_i)$ makes oracle query w. Then, we say that a hash function $H\colon\ \overline{\mathcal{D}} \to \mathcal{S}$ is *compatible* with $(\mathsf{KT}, F, \Phi, \vec{\omega})$, if $\mathcal{S} \subseteq \mathcal{D} \setminus \mathsf{Qrs}(\mathsf{KT}, F, \Phi, \vec{\omega})$.

Remark 3.4.1. A first attempt to fix the minor bug of the Bellare-Cash framework was proposed by the authors in [BC10a]. To circumvent the issue in the proof, they used a stronger notion of compatible hash function. While providing a correct proof, this change weakens the framework and prevents it from being applied to certain classes. Here, we stick to the original (and weaker) notion of compatible hash function, which let us recover all the original results. We simply fix the proof by using a slightly different strategy.

3.4.2 Dealing with Key-Collisions

Before describing the framework, we introduce two new notions that allow us to slightly generalize the framework. Indeed, the original framework by Bellare and Cash only allowed to handle classes Φ of RKD functions that are *claw-free*, which means that for any $K \in \mathcal{K}$ and any distinct $\phi, \phi' \in \Phi$, $\phi(K) \neq \phi'(K)$. In this section, we give a method to deal with classes of RKD functions that are not claw-free, such as affine classes. We introduce two new security notions: the first one is called Φ-*Key-Collision Security* and captures the likelihood that an adversary finds two distinct RKD functions which lead to the same derived key given oracle access to F, while the second one, called Φ-*Statistical-Key-Collision Security*, is similar but replaces the oracle access to the pseudorandom function with an oracle access to a truly random function.

Φ-Key-Collision (Φ-kc) Security. We define the advantage of an adversary $\mathscr{A}$ against the Φ-*key-collision security* of F, denoted by $\mathsf{Adv}^{\mathsf{kc}}_{\Phi,F}(\mathscr{A})$, to be the probability of success in the game on the left side of Figure 3.3, where the functions ϕ appearing in $\mathscr{A}$'s queries are restricted to lie in Φ. In particular, if Φ is claw-free, this advantage is 0.

Φ-Statistical-Key-Collision (Φ-skc) Security. We define the advantage of an adversary $\mathscr{A}$ against the Φ-*statistical-key-collision security* for $\mathsf{Fun}(\mathcal{K} \times \mathcal{D}, \mathcal{R})$, denoted by $\mathsf{Adv}^{\mathsf{skc}}_{\Phi}(\mathscr{A})$, to be the probability of success in the game on the right side of Figure 3.3. Here the functions ϕ appearing in $\mathscr{A}$'s queries are again restricted to lie in Φ. In particular, if Φ is claw-free, this advantage is 0.

Φ-kc security	Φ-skc security
proc Initialize $\mathsf{pp} \xleftarrow{\$} \mathsf{F.Setup}(1^\kappa)$ $K \xleftarrow{\$} \mathcal{K}$ **proc RKFn**(ϕ, x) $y \leftarrow \mathsf{F.Eval}_{\mathsf{pp}}(\phi(K), x)$ Return y **proc Finalize**(ϕ_1, ϕ_2) Return $(\phi_1 \neq \phi_2$ and $\phi_1(K) = \phi_2(K))$	**proc Initialize** $K \xleftarrow{\$} \mathcal{K}$; $D \leftarrow \emptyset$; $E \leftarrow \emptyset$ $f \xleftarrow{\$} \mathsf{Fun}(\mathcal{K} \times \mathcal{D}, \mathcal{R})$; $b' \leftarrow 0$ **proc RKFn**(ϕ, x) If $\phi(K) \in E$ and $\phi \notin D$ then $b' \leftarrow 1$ $D \leftarrow D \cup \{\phi\}$; $E \leftarrow E \cup \{\phi(K)\}$ $y \leftarrow f(\phi(K), x)$ Return y **proc Finalize** Return $(b' = 1)$

Figure 3.3: Security games for Φ-key-collision of a pseudorandom function F (left) and for Φ-statistical-key-collision of $\mathsf{Fun}(\mathcal{K} \times \mathcal{D}, \mathcal{R})$ (right)

3.4.3 The (Extended) Framework

We now describe our framework, which both repairs and extends the main result of [BC10b]. Intuitively, this framework allows to build a Φ-related-key secure pseudorandom function from any pseudorandom function that is Φ-key-malleable, as soon as there exists a strong key fingerprint $\vec{\omega}$ for F and a collision-resistant hash function that is compatible with $(\mathsf{KT}, F, \Phi, \vec{\omega})$. The construction is detailed below. Assuming the class Φ is claw-free, the advantages against key-collision and statistical-key-collisions securities are 0 and the framework below matches exactly the original Bellare-Cash framework.

Theorem 3.4.2 (Bellare-Cash Framework, Extended)**.** *Let $M = (\mathsf{M.Setup}, \mathsf{M.Eval})$ be a pseudorandom function whose key space, domain, and range are denoted $\mathcal{K}, \mathcal{D}, \mathcal{R}$, respectively. Let Φ be a class of related-key deriving functions that contains the identity function id. Let KT be a key-transformer for (M, Φ), and let $\vec{\omega} \in \mathcal{D}^m$ be a strong key fingerprint for M. Let $\overline{\mathcal{D}} = \mathcal{D} \times \mathcal{R}^m$ and let $H\colon \overline{\mathcal{D}} \to \mathcal{S}$ be a hash function that is compatible with $(\mathsf{KT}, M, \Phi, \vec{\omega})$. Define $F = (\mathsf{F.Setup}, \mathsf{F.Eval})$ as $\mathsf{F.Setup} = \mathsf{M.Setup}$ and:*

$$\mathsf{F.Eval}_{\mathsf{pp}}(K, x) := \mathsf{M.Eval}_{\mathsf{pp}}(K, H(x, \mathsf{M.Eval}_{\mathsf{pp}}(K, \vec{\omega}))) \ ,$$

for all $K \in \mathcal{K}$ and $x \in \mathcal{D}$.

Then, assuming M is a pseudorandom function and is Φ-key-collision secure, H is a collision-resistant hash function, and $\mathsf{Fun}(\mathcal{K} \times \mathcal{D}, \mathcal{R})$ is Φ-statistical-key-collision secure, F is a Φ-related key secure pseudorandom function.

Furthermore, the running time of the reduction is polynomial in the running time of the key-transformer and of the adversary against the Φ-related-key security of F.

Proof Overview. The proof of the above theorem is detailed below and relies on the sequence of 11 games (games $\mathbf{G}_0 - \mathbf{G}_{10}$) described in Figure 3.4. Here we provide a brief overview. Since the RKD functions that we consider in our case may have claws, we start by dealing with possible collisions on the related-keys in the $\mathsf{RKPRFReal}$ case, using the key-collision notion (games $\mathbf{G}_0 - \mathbf{G}_2$). Then, in games $\mathbf{G}_3 - \mathbf{G}_4$, we deal with possible collisions on hash values in order to ensure that the hash values h used to compute the output y are pairwise distinct so the attacker is unique-input. Then, using the properties of the key-transformer and the compatibility condition, we show that it is hard to distinguish the output from a uniformly random output (games $\mathbf{G}_5 - \mathbf{G}_7$) based on the standard pseudorandom function security of M. Finally, we use the statistical-key-collision security notion to deal with possible key collisions in the $\mathsf{RKPRFRand}$ case (games $\mathbf{G}_8 - \mathbf{G}_{10}$) so that $\mathbf{G}_{10}$ matches the description of the $\mathsf{RKPRFRand}$ game.

Remark 3.4.3. It is worth noting that we deviate from the original proof of [BC10b] in games $\mathbf{G}_5 - \mathbf{G}_7$, filling the gap in their original proof, but under the same technical conditions on compatibility. Unlike in their proof, we are able to show that the output of F is already indistinguishable from a uniformly random output as soon as one replaces the underlying pseudorandom function M with a random function f due to the uniformity condition of the transformer. In order to build a unique-input adversary against the uniformity condition, the main trick is to precompute the values of $f(w)$ for all $w \in \mathsf{Qrs}(\mathsf{KT}, M, \Phi, \vec{\omega})$ and use these values to compute $\mathsf{KT}^f(\phi, \omega_i)$, for $i = 1, \ldots, |\vec{\omega}|$ and $\phi \in \Phi$, whenever needed. This avoids the need to query the oracle in the uniformity game twice on the same input when computing the fingerprint.

Proof of Theorem 3.4.2. The proof is based on the sequence of games in Figure 3.4. Much of the proof is similar to that of the general framework of Bellare and Cash from [BC10b]. However, we have additional games to deal with non-claw-freeness (games $\mathbf{G}_1$, $\mathbf{G}_2$, $\mathbf{G}_9$ and $\mathbf{G}_{10}$), and some games (games $\mathbf{G}_6$ and $\mathbf{G}_7$) are modified to deal with the gap in the proof of the corresponding theorem in [BC10b]. Let $\textsc{Succ}_i$ denote the event that game $\mathbf{G}_i$ output takes the value 1. Let $\mathscr{A}$ be an adversary against the Φ-related-key security of F, and let us assume (without loss of generality) that it never repeats a query.

$\mathbf{G}_0$ matches the description of the $\mathsf{RKPRFReal}_F$ game, so:

$$\Pr[\textsc{Succ}_0] = \Pr\left[\mathsf{RKPRFReal}_F^{\mathscr{A}} \Rightarrow 1\right] .$$

Game $\mathbf{G}_1$ introduces storage of used RKD functions and values of $\vec{\vec{\omega}}$ in sets D and E respectively and sets flag_1 to true if the same value of $\vec{\vec{\omega}}$ arises for two different RKD functions. Since this storage does not affect the values returned by **RKFn**, we have

$$\Pr[\textsc{Succ}_1] = \Pr[\textsc{Succ}_0] .$$

Game $\mathbf{G}_2$ adds the boxed code which changes how the repetition of an $\vec{\vec{\omega}}$ value is handled, by picking instead a random value from $\mathcal{R}^m \setminus E$ that will not repeat any previous one. Games $\mathbf{G}_1$ and $\mathbf{G}_2$ are identical until flag_1 is set to true, hence we have

$$\Pr[\textsc{Succ}_1] \le \Pr[\textsc{Succ}_2] + \Pr[E_1] ,$$

where E_1 denotes the event that the execution of $\mathscr{A}$ with game $\mathbf{G}_1$ sets flag_1 to true. We design an adversary $\mathscr{D}$ attacking the Φ-key-collision security of M such that

$$\Pr[E_1] \le \mathsf{Adv}^{\mathsf{kc}}_{\Phi,M}(\mathscr{D}) .$$

Adversary $\mathscr{D}$ runs $\mathscr{A}$. When the latter makes an **RKFn**-query (ϕ, x), adversary $\mathscr{D}$ queries (ϕ, ω_i), for $i = 1, \ldots, |\vec{\omega}|$, to its oracle, then computes $\vec{\vec{\omega}}$ and then $h = H(x, \vec{\vec{\omega}})$ and finally queries (ϕ, h) to its oracle and sends it to $\mathscr{A}$. When $\mathscr{A}$ stops, $\mathscr{D}$ searches for two different RKD functions ϕ queried by $\mathscr{A}$ that lead to the same value $\vec{\varpi}$ and returns these two functions if found. Since $\vec{\omega}$ is a strong key fingerprint, two such functions lead to the same key, so $\mathscr{D}$ wins if it finds such two functions. (Of course, if the class of RKD functions is claw-free, the advantage of the attacker is 0.)

Game $\mathbf{G}_3$ introduces storage of hash values in a set G and sets flag_2 to true if the same hash output arises twice. Since this storage does not affect the values returned by **RKFn**

$$\Pr[\,\textsc{Succ}_3\,] = \Pr[\,\textsc{Succ}_2\,] \ .$$

Game $\mathbf{G}_4$ adds the boxed code which changes how repetition of hash values is handled, by picking instead a random value h from $\mathcal{S} \setminus G$ that will not repeat any previously used hash value. Games $\mathbf{G}_3$ and $\mathbf{G}_4$ are identical until flag_2 is set to true, hence we have

$$\Pr[\,\textsc{Succ}_3\,] \ \leq \ \Pr[\,\textsc{Succ}_4\,] + \Pr[\,E_2\,] \ ,$$

where E_2 denotes the event that the execution of $\mathscr{A}$ with game $\mathbf{G}_3$ sets flag_2 to true. We design an adversary $\mathscr{C}$ attacking the cr security of H such that

$$\Pr[\,E_2\,] \ \leq \ \mathsf{Adv}^{\mathsf{cr}}_{H}(\mathscr{C}) \ .$$

Adversary $\mathscr{C}$ starts by picking $K \xleftarrow{\$} \mathcal{K}$ and initializes $j \leftarrow 0$. It runs $\mathscr{A}$. When the latter makes an **RKFn**-query (ϕ, x), adversary $\mathscr{C}$ responds via:

```
For $i = 0, \ldots, |\vec{\omega}|$ do: $\varpi_i \leftarrow M(\phi(K), \omega_i)$
$j \leftarrow j + 1$ ; $\phi_j \leftarrow \phi$ ; $x_j \leftarrow x$
If $\vec{\varpi} \in E$ and $\phi \notin D$ then $\vec{\varpi} \xleftarrow{\$} \mathcal{S} \setminus E$          $(*)$
Else $D \leftarrow D \cup \{\phi\}$
$E \leftarrow E \cup \{\vec{\varpi}\}$
$w_j \leftarrow \vec{\varpi}$
$h \leftarrow H(x, \vec{\varpi})$
$h_j \leftarrow h$
$y \leftarrow M(\phi(K), h)$
Return $y$.
```

When $\mathscr{A}$ halts, $\mathscr{C}$ searches for a, b satisfying $1 \leq a < b \leq j$ such that $h_a = h_b$ and, if it finds them, outputs $(x_a, w_a), (x_b, w_b)$ and halts. The pairs (x_a, w_a) and (x_b, w_b) are distinct. Indeed, consider two cases: first, if $\phi_a = \phi_b$ then since $\mathscr{A}$ never repeats an oracle query, $x_a \neq x_b$ hence $(x_a, w_a) \neq (x_b, w_b)$. Second, if $\phi_a \neq \phi_b$, then condition $(*)$ ensures that $w_a \neq w_b$. Hence once again, $(x_a, w_a) \neq (x_b, w_b)$, and then

$$\Pr[\,\textsc{Succ}_3\,] \ \leq \ \Pr[\,\textsc{Succ}_4\,] + \mathsf{Adv}^{\mathsf{cr}}_{H}(\mathscr{C}) \ .$$

In game $\mathbf{G}_5$, we use the key-transformer KT to compute $M(\phi(K), \cdot)$ via oracle calls to $M(K, \cdot)$. The correctness property of the key-transformer implies

$$\Pr[\,\textsc{Succ}_4\,] = \Pr[\,\textsc{Succ}_5\,] \ .$$

In game $\mathbf{G}_6$, we replace the oracle $M(K, \cdot)$ given to the key-transformer KT by a random function f. We design an adversary $\mathscr{B}$ attacking the pseudorandom function security of M such that

$$\Pr[\,\textsc{Succ}_5\,] \ \leq \ \Pr[\,\textsc{Succ}_6\,] + \mathsf{Adv}^{\mathsf{prf}}_{M}(\mathscr{B}) \ .$$

Adversary $\mathscr{B}$ runs $\mathscr{A}$. When the latter makes an RKFn-query (ϕ, x), adversary $\mathscr{B}$ responds via

```
For $i = 0, \ldots, |\vec{\omega}|$ do $\overline{\omega}_i \leftarrow \mathsf{KT}^{\mathbf{Fn}}(\phi, \omega_i)$
If $\vec{\overline{\omega}} \in E$ and $\phi \notin D$ then $\vec{\overline{\omega}} \xleftarrow{\$} \mathcal{S} \setminus E$
Else $D \leftarrow D \cup \{\phi\}$
$E \leftarrow E \cup \{\vec{\overline{\omega}}\}$
$h \leftarrow H(x, \vec{\overline{\omega}})$
$y \leftarrow \mathsf{KT}^{\mathbf{Fn}}(\phi, h)$
Return $y$
```

where **Fn** is $\mathscr{B}$'s own oracle. When $\mathscr{A}$ halts, $\mathscr{B}$ halts with the same output. Then

$$\Pr\left[\mathsf{PRFReal}_M^{\mathscr{B}} \Rightarrow 1\right] = \Pr[\textsc{Succ}_5] \quad \text{and} \quad \Pr\left[\mathsf{PRFRand}_M^{\mathscr{B}} \Rightarrow 1\right] = \Pr[\textsc{Succ}_6] \ .$$

In game $\mathbf{G}_7$, instead of computing the output y using the key-transformer, we set the value y to a uniformly random value. To show that games $\mathbf{G}_6$ and $\mathbf{G}_7$ are perfectly indistinguishable, we use the uniformity condition of the Key-Transformer KT. Let us recall that, as formally defined in [BC10b, Section 3.1], the uniformity condition states that for any (even inefficient) Φ-restricted, unique-input adversary $\mathscr{U}$,

$$\Pr\left[\mathsf{KTReal}_{\mathsf{KT}}^{\mathscr{U}} \Rightarrow 1\right] = \Pr\left[\mathsf{KTRand}_{\mathsf{KT}}^{\mathscr{U}} \Rightarrow 1\right] \ ,$$

where game $\mathsf{KTReal}_{\mathsf{KT}}$ picks $f \xleftarrow{\$} \mathsf{Fun}(\mathcal{D}, \mathcal{R})$ during the initialization and responds to oracle query $\mathbf{KTFn}(\phi, x)$ via $\mathsf{KT}^f(\phi, x)$, while game $\mathsf{KTRand}_{\mathsf{KT}}$ has no initialization and responds to oracle query $\mathbf{KTFn}(\phi, x)$ by returning a value $y \xleftarrow{\$} \mathcal{R}$ chosen uniformly at random in $\mathcal{R}$. We show that if an adversary $\mathscr{A}$ can distinguish games $\mathbf{G}_6$ and $\mathbf{G}_7$, then we can construct a unique-input adversary $\mathscr{U}$ that can distinguish games $\mathsf{KTReal}_{\mathsf{KT}}$ and $\mathsf{KTRand}_{\mathsf{KT}}$; since KT is a key-transformer, these two games are perfectly indistinguishable for a unique-input adversary by the uniformity condition. Hence, so are $\mathbf{G}_6$ and $\mathbf{G}_7$.

Adversary $\mathscr{U}$ starts by initializing sets $D \leftarrow \emptyset$, $E \leftarrow \emptyset$, $G \leftarrow \emptyset$, then makes the queries (id, w) to its oracle, for every $w \in \mathsf{Qrs}(\mathsf{KT}, M, \Phi, \vec{\omega})$ and stores these values. This is possible under our assumption that $\mathsf{id} \in \Phi$. We let f_w denote the value that $\mathscr{D}$ gets from its oracle in response to the query (id, w). Depending on $\mathscr{U}$'s oracle, the value of f_w for $w \in \mathsf{Qrs}(\mathsf{KT}, M, \Phi, \vec{\omega})$ is either $\mathsf{KT}^f(\mathsf{id}, w) = f(w)$ ($\mathsf{KTReal}_{\mathsf{KT}}$), with f the random function defined in the **Initialize** procedure of $\mathsf{KTReal}_{\mathsf{KT}}$, or a uniformly random value from $\mathcal{R}$ ($\mathsf{KTRand}_{\mathsf{KT}}$). All these values will be used by $\mathscr{U}$ to compute the value $\vec{\overline{\omega}}$ in its simulation. Now, $\mathscr{U}$ runs $\mathscr{A}$. When $\mathscr{A}$ makes an oracle query (ϕ, x), $\mathscr{U}$ starts by computing the values $\overline{\omega}_i$, for $i = 1, \ldots, |\vec{\omega}|$, using the values f_w it has stored, the function ϕ it gets from $\mathscr{A}$, and the key-transformer KT. Note that, because $\mathscr{U}$ already queried (id, w) to its oracle for every $w \in \mathsf{Qrs}(\mathsf{KT}, M, \Phi, \vec{\omega})$, $\mathscr{U}$ is able to compute by itself the values $\overline{\omega}_i$, for $i = 1, \ldots, |\vec{\omega}|$. This is because, in making these queries, $\mathscr{U}$ already sets a value f_w for every $w \in \mathsf{Qrs}(\mathsf{KT}, M, \Phi, \vec{\omega})$, and this is the set of all values that might be needed in computing $\mathsf{KT}^f(\phi, \omega_i)$, for $i = 1, \ldots, |\vec{\omega}|$ and $\phi \in \Phi$. Notice that, in making these queries all at once at the beginning, $\mathscr{U}$ remains a unique-input adversary. After computing $\vec{\overline{\omega}}$, $\mathscr{U}$ checks if $\vec{\overline{\omega}} \in E$ and $\phi \notin D$. If these conditions hold, $\mathscr{U}$ picks $\vec{\overline{\omega}} \xleftarrow{\$} \mathcal{R}^m \setminus E$ at random, otherwise $\mathscr{U}$ sets $D \leftarrow D \cup \{\phi\}$. It then sets $E \leftarrow E \cup \{\vec{\overline{\omega}}\}$. Next, $\mathscr{U}$ computes $h \leftarrow H(x, \vec{\overline{\omega}})$ and checks if $h \in G$. If this holds, $\mathscr{U}$ picks $h \xleftarrow{\$} \mathcal{S} \setminus G$ at random. Notice that this step guarantees that all values h are in $\mathcal{S}$ and are all distinct as long as $\mathscr{A}$ makes at most $|\mathcal{S}|$ queries. Finally, $\mathscr{U}$ sets $G \leftarrow G \cup \{h\}$, makes the query (ϕ, h) to its oracle, and returns the value it gets, which is either $\mathsf{KT}^f(\phi, h)$ or a uniformly random value, to $\mathscr{A}$. When $\mathscr{A}$ halts, $\mathscr{U}$ halts with the same output. The compatibility condition ensures that $\mathcal{S}$ does not contain any w with $w \in \mathsf{Qrs}(\mathsf{KT}, M, \Phi, \vec{\omega})$. It follows from these observations that $\mathscr{U}$ is a unique-input adversary. Finally, it is clear that if $\mathscr{U}$'s oracle is $\mathsf{KTReal}_{\mathsf{KT}}$, then it simulates exactly game $\mathbf{G}_6$ with f being the function f chosen at random in the **Initialize** procedure of game $\mathsf{KTReal}_{\mathsf{KT}}$. If $\mathscr{U}$'s oracle is $\mathsf{KTRand}_{\mathsf{KT}}$, then it simulates exactly game $\mathbf{G}_7$ since the values given to $\mathscr{A}$ are uniformly random values. Then, we have

$$\Pr\left[\mathsf{KTReal}_{\mathsf{KT}}^{\mathscr{U}} \Rightarrow 1\right] = \Pr\left[\,\textsc{Succ}_6\,\right] \quad \text{and} \quad \Pr\left[\mathsf{KTRand}_{\mathsf{KT}}^{\mathscr{U}} \Rightarrow 1\right] = \Pr\left[\,\textsc{Succ}_7\,\right] \ ,$$

and then, since $\Pr\left[\mathsf{KTReal}_{\mathsf{KT}}^{\mathscr{U}} \Rightarrow 1\right] = \Pr\left[\mathsf{KTRand}_{\mathsf{KT}}^{\mathscr{U}} \Rightarrow 1\right]$ for any unique-input adversary $\mathscr{U}$, by the uniformity condition, we finally have

$$\Pr\left[\,\textsc{Succ}_6\,\right] = \Pr\left[\,\textsc{Succ}_7\,\right] \ .$$

Games $\mathbf{G}_7$ and $\mathbf{G}_8$ are identical since even if two different queries lead to the same key, the "If" test ensures that the returned value is still uniformly random over $\mathcal{R}$. Hence,

$$\Pr\left[\,\textsc{Succ}_7\,\right] = \Pr\left[\,\textsc{Succ}_8\,\right] \ .$$

Games $\mathbf{G}_8$ and $\mathbf{G}_9$ are identical until flag_3 is set to true, hence we have

$$\Pr\left[\,\textsc{Succ}_8\,\right] \ \leq \ \Pr\left[\,\textsc{Succ}_9\,\right] + \Pr\left[\,E_3\,\right] \ ,$$

where E_3 denotes the event that the execution of $\mathscr{A}$ with game $\mathbf{G}_9$ sets flag_3 to true. We design an adversary $\mathscr{E}$ breaking Φ-statistical-key-collision security for $\mathsf{Fun}(\mathcal{K}, \mathcal{D}, \mathcal{R})$ such that:

$$\Pr\left[\,E_3\,\right] \ \leq \ \mathsf{Adv}_{\Phi}^{\mathsf{skc}}(\mathscr{E}) \ .$$

Adversary $\mathscr{E}$ runs $\mathscr{A}$. When the latter makes an **RKFn**-query (ϕ, x), so does $\mathscr{E}$ and $\mathscr{E}$ returns the value it receives to $\mathscr{A}$. When $\mathscr{A}$ stops, if $\mathscr{A}$ has queried two different functions ϕ_1 and ϕ_2 such that $\phi_1(K) = \phi_2(K)$ then b' was set to 1 when the second of these two functions was queried by $\mathscr{E}$, and then $\mathscr{E}$ wins. (Of course, if the class of RKD functions is claw-free, this probability is 0.)

Games $\mathbf{G}_9$ and $\mathbf{G}_{10}$ are identical, so

$$\Pr\left[\,\textsc{Succ}_9\,\right] = \Pr\left[\,\textsc{Succ}_{10}\,\right] \ .$$

Finally, $\mathbf{G}_{10}$ matches the description of the $\mathsf{RKPRFRand}_F$ game, so:

$$\Pr\left[\,\textsc{Succ}_{10}\,\right] = \Pr\left[\mathsf{RKPRFRand}_F^{\mathscr{A}} \Rightarrow 1\right] \ .$$

Theorem 3.4.2 now follows by combining the bounds arising in the different game hops. $\square$

```
proc Initialize // $\mathbf{G}_0$
$K \xleftarrow{\$} \mathcal{K}$

proc RKFn($\phi, x$) // $\mathbf{G}_0$
For $i = 1, \ldots, |\vec{\omega}|$ do
    $\overline{\omega}_i \leftarrow M(\phi(K), \omega_i)$
$h \leftarrow H(x, \vec{\overline{\omega}})$
$y \leftarrow M(\phi(K), h)$
Return $y$

proc Finalize($b$) // All Games
Return $b$
```

```
proc Initialize // $\mathbf{G}_1$, $\boxed{\mathbf{G}_2}$
$K \xleftarrow{\$} \mathcal{K}$ ; $D \leftarrow \emptyset$ ; $E \leftarrow \emptyset$

proc RKFn($\phi, x$) // $\mathbf{G}_1$, $\boxed{\mathbf{G}_2}$
For $i = 1, \ldots, |\vec{\omega}|$ do
    $\overline{\omega}_i \leftarrow M(\phi(K), \omega_i)$
If $\vec{\overline{\omega}} \in E$ and $\phi \notin D$ then
    $\mathsf{flag}_1 \leftarrow \mathsf{true}$ ; $\boxed{\vec{\overline{\omega}} \xleftarrow{\$} \mathcal{R}^m \setminus E}$
$\boxed{\text{Else}}$ $D \leftarrow D \cup \{\phi\}$
$E \leftarrow E \cup \{\vec{\overline{\omega}}\}$
$h \leftarrow H(x, \vec{\overline{\omega}})$
$y \leftarrow M(\phi(K), h)$
Return $y$
```

```
proc Initialize // $\mathbf{G}_3$, $\boxed{\mathbf{G}_4}$
$K \xleftarrow{\$} \mathcal{K}$ ; $D \leftarrow \emptyset$ ; $E \leftarrow \emptyset$ ; $G \leftarrow \emptyset$

proc RKFn($\phi, x$) // $\mathbf{G}_3$, $\boxed{\mathbf{G}_4}$
For $i = 1, \ldots, |\vec{\omega}|$ do
    $\overline{\omega}_i \leftarrow M(\phi(K), \omega_i)$
If $\vec{\overline{\omega}} \in E$ and $\phi \notin D$ then
    $\vec{\overline{\omega}} \xleftarrow{\$} \mathcal{R}^m \setminus E$
Else $D \leftarrow D \cup \{\phi\}$
$E \leftarrow E \cup \{\vec{\overline{\omega}}\}$
$h \leftarrow H(x, \vec{\overline{\omega}})$
If $h \in G$ then $\mathsf{flag}_2 \leftarrow \mathsf{true}$
    $\boxed{h \xleftarrow{\$} \mathcal{S} \setminus G}$
$G \leftarrow G \cup \{r\}$
$y \leftarrow M(\phi(K), h)$
Return $y$
```

```
proc Initialize // $\mathbf{G}_5$
$K \xleftarrow{\$} \mathcal{K}$ ; $D \leftarrow \emptyset$ ; $E \leftarrow \emptyset$ ; $G \leftarrow \emptyset$

proc RKFn($\phi, x$) // $\mathbf{G}_5$
For $i = 1, \ldots, |\vec{\omega}|$ do
    $\overline{\omega}_i \leftarrow \mathsf{KT}^{M(K,\cdot)}(\phi, \omega_i)$
If $\vec{\overline{\omega}} \in E$ and $\phi \notin D$ then
    $\vec{\overline{\omega}} \xleftarrow{\$} \mathcal{R}^m \setminus E$
Else $D \leftarrow D \cup \{\phi\}$
$E \leftarrow E \cup \{\vec{\overline{\omega}}\}$
$h \leftarrow H(x, \vec{\overline{\omega}})$
If $h \in G$ then $h \xleftarrow{\$} \mathcal{S} \setminus G$
$G \leftarrow G \cup \{r\}$
$y \leftarrow \mathsf{KT}^{M(K,\cdot)}(\phi, h)$
Return $y$
```

```
proc Initialize // $\mathbf{G}_6$
$K \xleftarrow{\$} \mathcal{K}$ ; $D \leftarrow \emptyset$ ; $E \leftarrow \emptyset$ ; $G \leftarrow \emptyset$
$f \xleftarrow{\$} \mathsf{Fun}(\mathcal{D}, \mathcal{R})$

proc RKFn($\phi, x$) // $\mathbf{G}_6$
For $i = 1, \ldots, |\vec{\omega}|$ do
    $\overline{\omega}_i \leftarrow \mathsf{KT}^f(\phi, \omega_i)$
If $\vec{\overline{\omega}} \in E$ and $\phi \notin D$ then
    $\vec{\overline{\omega}} \xleftarrow{\$} \mathcal{R}^m \setminus E$
Else $D \leftarrow D \cup \{\phi\}$
$E \leftarrow E \cup \{\vec{\overline{\omega}}\}$
$h \leftarrow H(x, \vec{\overline{\omega}})$
If $h \in G$ then $h \xleftarrow{\$} \mathcal{S} \setminus G$
$G \leftarrow G \cup \{r\}$
$y \leftarrow \mathsf{KT}^f(\phi, h)$
Return $y$
```

```
proc Initialize // $\boxed{\mathbf{G}_8}$, $\mathbf{G}_9$
$K \xleftarrow{\$} \mathcal{K}$ ; $D \leftarrow \emptyset$ ; $E \leftarrow \emptyset$
$G \xleftarrow{\$} \mathsf{Fun}(\mathcal{K}, \mathcal{D}, \mathcal{R})$

proc RKFn($\phi, x$) // $\boxed{\mathbf{G}_8}$, $\mathbf{G}_9$
$y \leftarrow G(\phi(K), x)$
If $\phi(K) \in E$ and $\phi \notin D$ then
    $\mathsf{flag}_3 \leftarrow \mathsf{true}$ ; $\boxed{y \xleftarrow{\$} \mathcal{R}}$
$D \leftarrow D \cup \{\phi\}$ ; $E \leftarrow E \cup \{\phi(K)\}$
Return $y$
```

```
proc Initialize // $\mathbf{G}_7$
$K \xleftarrow{\$} \mathcal{K}$ ; $D \leftarrow \emptyset$ ; $E \leftarrow \emptyset$
$G \xleftarrow{\$} \mathsf{Fun}(\mathcal{K}, \mathcal{D}, \mathcal{R})$

proc RKFn($\phi, x$) // $\mathbf{G}_7$
$y \xleftarrow{\$} \mathcal{R}$
Return $y$
```

```
proc Initialize // $\mathbf{G}_{10}$
$K \xleftarrow{\$} \mathcal{K}$ ; $G \xleftarrow{\$} \mathsf{Fun}(\mathcal{K}, \mathcal{D}, \mathcal{R})$

proc RKFn($\phi, x$) // $\mathbf{G}_{10}$
$y \leftarrow G(\phi(K), x)$
Return $y$
```

Figure 3.4: Games for the proof of Theorem 3.4.2

3.5 Application: Related-Key Security for Affine Relations

In this section, we apply the above framework to the variant NR^* of the Naor-Reingold pseudorandom function to build a related-key secure pseudorandom function for affine relations. Recall that NR^* was defined in [BC10b] as:

- $\mathsf{NR}^*.\mathsf{Setup}(1^\kappa)$ generates public parameters $\mathsf{pp} = (p, \mathbb{G}, g)$ where $(p, \mathbb{G}, g)$ describes a cyclic group of prime order p generated by g;
- $\mathsf{NR}^*.\mathsf{Eval}_{\mathsf{pp}}(\cdot,\cdot)$ takes as input a key $\vec{a} = (a_1, \dots, a_n) \in \mathbb{Z}_p^n$ and an input $x = x_1 \parallel \dots \parallel x_n \in \{0,1\}^n \setminus \{0^n\}$ and outputs $[\prod_{i=1}^n a_i^{x_i}]_g$.

We first recall the following lemma from [BC10b].

Lemma 3.5.1. *Assuming the* DDH *assumption holds in* $\mathbb{G}$*,* NR^* *is a pseudorandom function.*

We define the class Φ_{aff} of affine relations as the set of all functions $\phi : \mathbb{Z}_p^n \to \mathbb{Z}_p^n$ such that there exist $\alpha_i \in \mathbb{Z}_p^*, \beta_i \in \mathbb{Z}_p$, for $i = 1, \dots, n$, such that

$$\phi(\vec{a}) = (\alpha_1 \cdot a_1 + \beta_1, \dots, \alpha_n \cdot a_n + \beta_n) \in \mathbb{Z}_p^n \ .$$

Then, using the above framework, we prove that NR^* can be used to build a Φ_{aff}-related-key secure pseudorandom function under the DDH assumption and therefore recover and strengthen the withdrawn result from [BC10b]. In what follows, we prove the properties needed to apply Theorem 3.4.2 to NR^*.

3.5.1 Ingredients

Strong Key Fingerprint. Let $\omega_i = 0^{i-1} \parallel 1 \parallel 0^{n-i}$, for $i = 1, \dots, n$. Then $\vec{\omega}$ is a strong key fingerprint for NR^*. Indeed, we have $(\mathsf{NR}^*(\vec{a}, \omega_1), \dots, \mathsf{NR}^*(\vec{a}, \omega_n)) = ([a_1], \dots, [a_n])$, so if $\vec{a} \neq \vec{a}'$ are two distinct keys in $\mathcal{K} = \mathbb{Z}_p^n$, then there exists $i \in \{1, \dots, n\}$ such that $a_i \neq a_i'$, so $[a_i] \neq [\vec{a}_i']$.

Compatible Hash Function. We have $\mathsf{Qrs}(\mathsf{KT}_{\Phi_{\mathsf{aff}}}, \mathsf{NR}^*, \Phi_{\mathsf{aff}}, \vec{\omega}) = \{\omega_1, \dots, \omega_n\}$, so let $\mathcal{D} = \{0,1\}^n \times \mathbb{G}^n$ and let h: $\mathcal{D} \to \{0,1\}^{n-2}$ be a collision resistant hash function. Then the hash function defined by $H(x, \vec{z}) = 11 \parallel h(x, \vec{z})$ is a collision resistant hash function that is compatible with $(\mathsf{KT}_{\Phi_{\mathsf{aff}}}, \mathsf{NR}^*, \Phi_{\mathsf{aff}}, \vec{\omega})$ since every element of $\mathsf{Qrs}(\mathsf{KT}_{\Phi_{\mathsf{aff}}}, \mathsf{NR}^*, \Phi_{\mathsf{aff}}, \vec{\omega})$ has at most one 1 bit and every output of H has at least two 1 bits. Note that in particular the output of H is never 0^n, so it is always in the domain of NR^*.

Key-Collision Security. The Φ_{aff}-key-collision security of NR^* immediately follows from the hardness of the discrete logarithm problem in $\mathbb{G}$, as stated in the lemma below.

Lemma 3.5.2. *Assuming the hardness of the discrete logarithm in* $\mathbb{G}$*,* NR^* *is* Φ_{aff}*-key-collision secure.*

Since the hardness of DDH implies the hardness of DL, the above lemma does not introduce any additional hardness assumptions beyond DDH.

Proof of Lemma 3.5.2. Let $\mathscr{A}$ be an adversary against the Φ_{aff}-key-collision security of NR^* that makes $Q_{\mathscr{A}}$ oracle queries. Then we construct an adversary $\mathscr{B}$ against the DL problem in $\mathbb{G}$ as follows. Adversary $\mathscr{B}$ receives as input a DL tuple $([1],[a])$. Adversary $\mathscr{B}$ then picks $j \xleftarrow{\$} \{1,\ldots,n\}$ at random; this is a guess of a coordinate where the two vectors of affine functions $\vec{\phi}^{(1)}$ and $\vec{\phi}^{(2)}$ that $\mathscr{A}$ will use as inputs in the **Finalize** procedure are different. Then $\mathscr{B}$ picks $a_i \xleftarrow{\$} \mathbb{Z}_p$ for $i = 1,\ldots,n, i \neq j$ at random. Adversary $\mathscr{B}$ implicitly sets $a_j = a$.

When $\mathscr{A}$ makes a query (ϕ, x), $\mathscr{B}$ computes $y = ([\phi_j(a_j)^{x_j}])^{\prod_{i=1, i\neq j}^{n} \phi_i(a_i)^{x_i}} = [\prod_{i=1}^{n} \phi_i(a_i)^{x_i}] = \mathsf{NR}^*(\vec{a}, x)$, where $\vec{a} = (a_1,\ldots,a_n)$. Here, $\mathscr{B}$ uses its input $[a]$ to compute an "affine function in the exponent" for $[\phi_j(a_j)]$. At the end, $\mathscr{A}$ sends $(\vec{\phi}^{(1)}, \vec{\phi}^{(2)})$ to $\mathscr{B}$ and $\mathscr{A}$ wins if $\vec{\phi}^{(1)} \neq \vec{\phi}^{(2)}$ and $\vec{\phi}^{(1)}(\vec{a}) = \vec{\phi}^{(2)}(\vec{a})$, where $\vec{\phi}^{(i)} = (\phi_1^{(i)},\ldots,\phi_n^{(i)}), i \in \{1,2\}$. Since j was chosen uniformly at random and $\vec{\phi}^{(1)} \neq \vec{\phi}^{(2)}$, with probability at least $\frac{1}{n}$, we have $\phi_j^{(1)} \neq \phi_j^{(2)}$ but $\phi_j^{(1)}(a_j) = \phi_j^{(2)}(a_j)$. In this case, $a_j = a$ is the root of the non-zero affine function $\psi = \phi_j^{(1)} - \phi_j^{(2)}$, that can be easily computed. Hence, we have

$$\mathsf{Adv}^{\mathsf{kc}}_{\Phi_{\mathsf{aff}},\mathsf{NR}^*}(\mathscr{A}) \;\leq\; n \cdot \mathsf{Adv}^{\mathsf{dl}}_{\mathbb{G}}(\mathscr{B}) \ ,$$

and the claim follows. □

Key-Transformer. We define a key-transformer for $(\mathsf{NR}^*, \Phi_{\mathsf{aff}})$ as:

$$\mathsf{KT}^{f}_{\Phi_{\mathsf{aff}}}(\phi, x) = \left[\prod_{i\in S(x)} \beta_i\right] \cdot \prod_{y \preceq x, y \neq 0^n} f(y)^{\prod_{j\in S(y)} \alpha_j \prod_{k \in S(x)\setminus S(y)} \beta_k} \ ,$$

where $\phi(\vec{a}) = (\alpha_1 \cdot a_1 + \beta_1, \ldots, \alpha_n \cdot a_n + \beta_n) \in \mathbb{Z}_p^n$. Then, we have the following lemma:

Lemma 3.5.3. $\mathsf{KT}_{\Phi_{\mathsf{aff}}}$ *is a key-transformer for* $(\mathsf{NR}^*, \Phi_{\mathsf{aff}})$ *and its worst-case running time is the time required to compute* $O(2^n)$ *exponentiations in* $\mathbb{G}$.

Proof of Lemma 3.5.3. Let us first check the correctness condition.

$$\begin{aligned}
\mathsf{KT}^{\mathsf{NR}^*(\vec{a},\cdot)}_{\Phi_{\mathsf{aff}}}(\vec{\phi}, x) &= \left[\prod_{i\in S(x)} \beta_i\right] \cdot \prod_{y \preceq x, y\neq 0^n} \left[\prod_{l \in S(y)} a_l\right]^{\prod_{j\in S(y)} \alpha_j \prod_{k\in S(x)\setminus S(y)} \beta_k} \\
&= \prod_{R\subseteq S(x)} \left[\prod_{i\in R} (\alpha_i \cdot a_i) \prod_{j \in S(x)\setminus R} \beta_j\right] = \left[\sum_{R\subseteq S(x)} \prod_{i\in R} (\alpha_i \cdot a_i) \prod_{j\in S(x)\setminus R} \beta_j\right] \\
&= \left[\prod_{i\in S(x)} (\alpha_i \cdot a_i + \beta_i)\right] = \left[\prod_{i=1}^{n} (\alpha_i \cdot a_i + \beta_i)^{x_i}\right] = \mathsf{NR}^*(\vec{\phi}(\vec{a}), x) \ .
\end{aligned}$$

Then, we have verified the correctness condition, and it is clear that the worst-case running time is the time to compute 2^n exponentiations in $\mathbb{G}$, when $x = 11 \| \ldots \| 1$ and none of the exponents is 0. Hence, only the uniformity condition remains to prove. We use the sequence of games in Figure 3.5. Let us recall that the adversary is supposed to be unique-input, meaning that for any sequence of queries $(\vec{\phi}_1, x_1), \ldots, (\vec{\phi}_q, x_q)$, the entries x_i, for $i = 1,\ldots,q$ are all distinct. We denote by $\mathsf{hw}(\mathsf{x})$ the Hamming weight of a bitstring x. Let $\textsc{Succ}_i$ denote the event that game $\mathbf{G}_i$ output takes the value 1.

In game $\mathbf{G}_0$, the "If" statement will always pass since $\mathsf{hw}(x) \leq n$ for any bitstring of length n. Hence, we have

$$\Pr[\,\textsc{Succ}_0\,] = \Pr\left[\,\mathsf{KTReal}^{\mathscr{A}}_{\mathsf{KT}} \Rightarrow 1\,\right] \ .$$

```
proc Initialize  // $\mathbf{G}_i$, $i = 0, \ldots, n$
$f \xleftarrow{\$} \mathsf{Fun}(\{0,1\}^n \setminus \{0^n\}, \mathbb{G})$
proc Finalize($b$)
Return $b$

proc RKFn($\phi, x$)  // $\mathbf{G}_i$, $i = 0, \ldots, n$
If $\mathsf{hw}(x) \le n - i$ then
    $y \leftarrow \mathsf{KT}^f_{\Phi_{\mathsf{aff}}}(\phi, x)$
Else $y \xleftarrow{\$} \mathbb{G}$
Return $y$
```

Figure 3.5: Games for the proof of Lemma 3.5.3

We claim that for all $0 \le i \le n-1$,

$$\Pr[\,\text{SUCC}_i\,] = \Pr[\,\text{SUCC}_{i+1}\,] \ .$$

The only difference between games $\mathbf{G}_i$ and $\mathbf{G}_{i+1}$ is in the way that bitstrings x of Hamming weight $n-i$ are handled. Indeed, such a string is fed to $\mathsf{KT}^f_{\Phi_{\mathsf{aff}}}(\vec{\phi}, x)$ in $\mathbf{G}_i$, which computes

$$\mathsf{KT}^f_{\Phi_{\mathsf{aff}}}(\vec{\phi}, x) = \left[\prod_{i \in S(x)} \beta_i\right] \cdot \prod_{y \preceq x, y \neq 0^n} f(y)^{\prod_{j \in S(y)} \alpha_j \prod_{k \in S(x) \setminus S(y)} \beta_k} \ ,$$

where $\vec{\phi} = (\phi_1, \ldots, \phi_n) \in \Phi_{\mathsf{aff}}$, with ϕ_i: $\vec{T} \mapsto \alpha_i T_i + \beta_i$, $\alpha_i \neq 0$, for $i = 1, \ldots, n$. Now, since we need only deal with unique-input adversaries, this is the only time that $\mathbf{G}_i$ will query f at input x (all other queries to f will be at other points with the same Hamming weight or at points with strictly smaller Hamming weight). Hence, the entire value computed above can equivalently be set to a value chosen uniformly at random. (This relies on the exponent for $f(x)$ used in the computation being non-zero; this is guaranteed by the requirement that $\alpha_i \neq 0$, for $i = 1, \ldots, n$ and the fact that when $y = x$, the product $\prod_{k \in S(x) \setminus S(y)} \beta_k$ is empty.) Setting the entire value to a uniformly random value is exactly what is done in $\mathbf{G}_{i+1}$, and the claim follows.

Finally, in $\mathbf{G}_n$, the "If" statement will never pass since $\mathsf{hw}(x) > 0$ for any $x \in \{0,1\}^n \setminus \{0\}^n$, so we have

$$\Pr[\,\text{SUCC}_n\,] = \Pr\left[\mathsf{KTRand}^{\mathscr{A}}_{\mathsf{KT}} \Rightarrow 1\right] \ .$$

The uniformity condition follows. □

Statistical-Key-Collision Security. The Φ_{aff}-statistical-key-collision security for $\mathsf{Fun}(\mathbb{Z}_p^n \times \{0,1\}^n \setminus \{0^n\}, \mathbb{G})$ holds statistically, as stated below.

Lemma 3.5.4. *Let $\mathscr{A}$ be an adversary against the Φ_{aff}-statistical-key-collision security for $\mathsf{Fun}(\mathbb{Z}_p^n, \{0,1\}^n, \mathbb{G})$ making $Q_{\mathscr{A}}$ queries. Then we have:*

$$\mathsf{Adv}^{\mathsf{skc}}_{\Phi_{\mathsf{aff}}}(\mathscr{A}) \ \le \ \frac{Q_{\mathscr{A}}^2}{2p} \ .$$

Proof of Lemma 3.5.4. Let $\mathscr{A}$ be an adversary against the Φ_{aff}-statistical-key-collision security for $\mathsf{Fun}(\mathbb{Z}_p^n, \{0,1\}^n, \mathbb{G})$ that makes $Q_{\mathscr{A}}$ queries. Since the function F defined in the **Initialize** procedure is a random function, $\mathscr{A}$ does not learn any information on the key $\vec{a}$ until $b' \leftarrow 1$, so $\mathsf{Adv}^{\mathsf{skc}}_{\Phi_{\mathsf{aff}}}(\mathscr{A})$ is bounded by the probability that $\mathscr{A}$ makes use in its queries of two different RKD functions that lead to the same key. We claim that

$$\mathsf{Adv}^{\mathsf{skc}}_{\Phi_{\mathsf{aff}}}(\mathscr{A}) \ \le \ \frac{Q_A^2}{2p} \ .$$

This follows easily on noting that, if two different RKD functions lead to the same key, then those two functions must differ in some coordinate k. This means that the difference in those components is a non-constant affine function ψ_k such that $\psi_k(a_k) = 0$, where a_k is the k-th component of key $\vec{a}$ that was taken uniformly at random in the **Initialize** procedure. Since ψ_k is a non-constant affine function and a_k is uniformly random in $\mathbb{Z}_p$, the probability that $\psi_k(a_k) = 0$ is bounded by $\frac{1}{p}$. To obtain the final result, one simply applies a union bound over the (at most) $\binom{Q_{\mathscr{A}}}{2}$ pairs of choices of different RKD functions accessed by $\mathscr{A}$. $\square$

We now have everything we need to apply Theorem 3.4.2 to NR^*.

3.5.2 Putting Everything Together

Combining Theorem 3.4.2, Lemmata 3.5.1–3.5.4 and the above properties, we obtain the following theorem.

Theorem 3.5.5. *Assuming* DDH *holds in $\mathbb{G}$, the construction obtained with the above ingredients by applying Theorem 3.4.2 is a Φ_{aff}-related-key secure pseudorandom function.*

Moreover, the running time of the reduction is polynomial in $2^n \times T$, where T is the running time of the adversary against the Φ_{aff}-related-key pseudorandom function security.

3.6 Further Generalization of the Bellare-Cash Framework

3.6.1 Relaxing the Requirements of the Framework

In this section, we introduce a new type of pseudorandom functions called $(\mathcal{S}, \Phi)$-*unique-input-related-key pseudorandom functions*. This notion allows us to design a different framework that can be applied to *non*-key-malleable pseudorandom functions. We later show that any key-malleable pseudorandom function is also a unique-input-related-key pseudorandom function as soon as the key-transformer satisfies an adequate property.

Definition 3.6.1 (Unique-Input-Related-Key Pseudorandom Function)**.** *Let $F = (\mathsf{F.Setup}, \mathsf{F.Eval})$ be a function whose key space, domain, and range are denoted by $\mathcal{K}, \mathcal{D}$, and $\mathcal{R}$, respectively. Let $\mathcal{S}$ be a subset of $\mathcal{D}$ and Φ be a class of RKD functions. We consider the class of Φ-restricted adversaries $\mathscr{D}$ such that all queries (ϕ, x) with $x \in \mathcal{S}$ made by $\mathscr{D}$ to its oracle are for distinct values of x. We define of such an adversary $\mathscr{D}$ in attacking the $(\mathcal{S}, \Phi)$-unique-input-related-key security of F as:*

$$\mathbf{Adv}^{\mathsf{ui\text{-}rka\text{-}prf}}_{F,\mathcal{S},\Phi}(\mathscr{D}) := \Pr\left[\mathsf{UIRKPRFReal}^{\mathscr{D}}_{F,\mathcal{S}} \Rightarrow 1\right] - \Pr\left[\mathsf{UIRKPRFRand}^{\mathscr{D}}_{F,\mathcal{S}} \Rightarrow 1\right] ,$$

where $\mathsf{UIRKPRFReal}_{F,\mathcal{S}}$ and $\mathsf{UIRKPRFRand}_{F,\mathcal{S}}$ are described in Figure 3.6. We say that F is an $(\mathcal{S}, \Phi)$-unique-input-related-key secure pseudorandom function if for any such PPT adversary $\mathscr{D}$, this advantage is negligible.

The following theorem is an analogue of Theorem 3.4.2 in which the roles of key-malleability and hash function compatibility are replaced by our new security notion.

Theorem 3.6.2. *Let $M = (\mathsf{M.Setup}, \mathsf{M.Eval})$ be a pseudorandom function whose key space, domain, and range are denoted $\mathcal{K}, \mathcal{D}, \mathcal{R}$, respectively. Let Φ be a class of* RKD *functions. Let*

$\mathsf{UIRKPRFReal}_{F,\mathcal{S}}$	$\mathsf{UIRKPRFRand}_{F,\mathcal{S}}$
proc Initialize $\mathsf{pp} \xleftarrow{\$} \mathsf{F.Setup}(1^\kappa)$ $K \xleftarrow{\$} \mathcal{K}$ Return pp **proc RKFn**(ϕ, x) Return $\mathsf{F.Eval}_{\mathsf{pp}}(\phi(K), x)$ **proc Finalize**(b) Return b	**proc Initialize** $\mathsf{pp} \xleftarrow{\$} \mathsf{F.Setup}(1^\kappa)$ $K \xleftarrow{\$} \mathcal{K}$ Return pp **proc RKFn**(ϕ, x) If $x \in \mathcal{S}$ then $y \xleftarrow{\$} \mathcal{R}$ Else $y \leftarrow \mathsf{F.Eval}_{\mathsf{pp}}(\phi(K), x)$ Return y **proc Finalize**(b) Return b

Figure 3.6: Security games for unique-input-related-key pseudorandom functions

$\overline{\mathcal{D}} = \mathcal{D} \times \mathcal{R}^m$ and let $H \colon \overline{\mathcal{D}} \to \mathcal{S}$ be a hash function, where $\mathcal{S} \subseteq \mathcal{D} \setminus \{\omega_1, \ldots, \omega_m\}$. Define $F = (\mathsf{F.Setup}, \mathsf{F.Eval})$ as $\mathsf{F.Setup} = \mathsf{M.Setup}$ and:

$$\mathsf{F.Eval}_{\mathsf{pp}}(K, x) := \mathsf{M.Eval}_{\mathsf{pp}}(K, H(x, \mathsf{M.Eval}_{\mathsf{pp}}(K, \vec{\omega}))) \ ,$$

for all $K \in \mathcal{K}$ and $x \in \mathcal{D}$.

Then, assuming M is an $(\mathcal{S}, \Phi)$-unique-input-related-key secure pseudorandom function and is Φ-key-collision secure, H is a collision-resistant hash function, and $\mathsf{Fun}(\mathcal{K} \times \mathcal{D}, \mathcal{R})$ is Φ-statistical-key-collision secure, then F is a Φ-related key secure pseudorandom function.

Moreover, the running time of the reduction is polynomial in the running time of the adversary and of the key-transformer.

Proof Overview. The proof of the above theorem is detailed below and is very similar to the proof of Theorem 3.4.2. The difference lies essentially in the fact that we use the unique-input-related-key security and the compatibility condition to show that it is hard to distinguish the output of F from a uniformly random output, in games $\mathbf{G}_5, \mathbf{G}_6$, instead of using the key-malleability in the previous proof.

Proof of Theorem 3.6.2. The proof is based on the sequence of games in Figure 3.7. Much of the proof is similar to the proof of Theorem 3.4.2 (which itself is based on the proof of the general framework of Bellare and Cash from [BC10b]). The current proof, however, is somewhat simpler and has fewer games since it relies on a stronger security property, namely the $(\mathcal{S}, \Phi)$-unique-input-related-key pseudorandom function security of M. Let $\textsc{Succ}_i$ denote the event that game $\mathbf{G}_i$ output takes the value 1. Let $\mathscr{A}$ be an adversary against the Φ-related-key security of F, and let us assume (without loss of generality) that it never repeats a query.

$\mathbf{G}_0$ matches the description of the $\mathsf{RKPRFReal}_F$ game, so:

$$\Pr[\,\textsc{Succ}_0\,] = \Pr\left[\mathsf{RKPRFReal}_F^{\mathscr{A}} \Rightarrow 1\right] \ .$$

```
proc Initialize  // $\mathbf{G}_0$
$K \overset{\$}{\leftarrow} \mathcal{K}$

proc RKFn($\phi, x$)  // $\mathbf{G}_0$
For $i = 1, \ldots, |\vec{\omega}|$ do
    $\overline{\omega}_i \leftarrow M(\phi(K), \omega_i)$
$h \leftarrow H(x, \overrightarrow{\overline{\omega}})$
$y \leftarrow M(\phi(K), h)$
Return $y$

proc Finalize($b'$)  // All Games
Return $b'$
```

```
proc Initialize  // $\mathbf{G}_1$, $\boxed{\mathbf{G}_2}$
$K \overset{\$}{\leftarrow} \mathcal{K}$ ; $D \leftarrow \emptyset$ ; $E \leftarrow \emptyset$

proc RKFn($\phi, x$)  // $\mathbf{G}_1$, $\boxed{\mathbf{G}_2}$
For $i = 1, \ldots, |\vec{\omega}|$ do
    $\overline{\omega}_i \leftarrow M(\phi(K), \omega_i)$
If $\overrightarrow{\overline{\omega}} \in E$ and $\phi \notin D$ then
    $\mathsf{flag}_1 \leftarrow \mathsf{true}$ ; $\boxed{\overrightarrow{\overline{\omega}} \overset{\$}{\leftarrow} \mathcal{R}^m \setminus E}$
$\boxed{\text{Else}}$ $D \leftarrow D \cup \{\phi\}$
$E \leftarrow E \cup \{\overrightarrow{\overline{\omega}}\}$
$h \leftarrow H(x, \overrightarrow{\overline{\omega}})$
$y \leftarrow M(\phi(K), h)$
Return $y$
```

```
proc Initialize  // $\mathbf{G}_3$, $\boxed{\mathbf{G}_4}$
$K \overset{\$}{\leftarrow} \mathcal{K}$ ; $D \leftarrow \emptyset$ ; $E \leftarrow \emptyset$ ; $G \leftarrow \emptyset$

proc RKFn($\phi, x$)  // $\mathbf{G}_3$, $\boxed{\mathbf{G}_4}$
For $i = 1, \ldots, |\vec{\omega}|$ do
    $\overline{\omega}_i \leftarrow M(\phi(K), \omega_i)$
If $\overrightarrow{\overline{\omega}} \in E$ and $\phi \notin D$ then
    $\overrightarrow{\overline{\omega}} \overset{\$}{\leftarrow} \mathcal{R}^m \setminus E$
Else $D \leftarrow D \cup \{\phi\}$
$E \leftarrow E \cup \{\overrightarrow{\overline{\omega}}\}$
$h \leftarrow H(x, \overrightarrow{\overline{\omega}})$
If $h \in G$ then $\mathsf{flag}_2 \leftarrow \mathsf{true}$
    $\boxed{h \overset{\$}{\leftarrow} \mathcal{S} \setminus G}$
$G \leftarrow G \cup \{r\}$
$y \leftarrow M(\phi(K), h)$
Return $y$
```

```
proc Initialize  // $\mathbf{G}_5$
$K \overset{\$}{\leftarrow} \mathcal{K}$ ; $D \leftarrow \emptyset$ ; $E \leftarrow \emptyset$ ; $G \leftarrow \emptyset$

proc RKFn($\phi, x$)  // $\mathbf{G}_5$
For $i = 1, \ldots, |\vec{\omega}|$ do
    $\overline{\omega}_i \leftarrow M(\phi(K), \omega_i)$
If $\overrightarrow{\overline{\omega}} \in E$ and $\phi \notin D$ then
    $\overrightarrow{\overline{\omega}} \overset{\$}{\leftarrow} \mathcal{R}^m \setminus E$
Else $D \leftarrow D \cup \{\phi\}$
$E \leftarrow E \cup \{\overrightarrow{\overline{\omega}}\}$
$h \leftarrow H(x, \overrightarrow{\overline{\omega}})$
If $h \in G$ then $h \overset{\$}{\leftarrow} \mathcal{S} \setminus G$
$G \leftarrow G \cup \{r\}$
$y \overset{\$}{\leftarrow} \mathcal{R}$
Return $y$
```

```
proc Initialize  // $\mathbf{G}_6$
$K \overset{\$}{\leftarrow} \mathcal{K}$ ; $D \leftarrow \emptyset$ ; $E \leftarrow \emptyset$
$G \overset{\$}{\leftarrow} \mathsf{Fun}(\mathcal{K}, \mathcal{D}, \mathcal{R})$

proc RKFn($\phi, x$)  // $\mathbf{G}_6$
$y \overset{\$}{\leftarrow} \mathcal{R}$
Return $y$
```

```
proc Initialize  // $\boxed{\mathbf{G}_7}$, $\mathbf{G}_8$
$K \overset{\$}{\leftarrow} \mathcal{K}$ ; $D \leftarrow \emptyset$ ; $E \leftarrow \emptyset$
$G \overset{\$}{\leftarrow} \mathsf{Fun}(\mathcal{K}, \mathcal{D}, \mathcal{R})$

proc RKFn($\phi, x$)  // $\boxed{\mathbf{G}_7}$, $\mathbf{G}_8$
$y \leftarrow G(\phi(K), x)$
If $\phi(K) \in E$ and $\phi \notin D$ then
    $\mathsf{flag}_3 \leftarrow \mathsf{true}$ ; $\boxed{y \overset{\$}{\leftarrow} \mathcal{R}}$
$D \leftarrow D \cup \{\phi\}$ ; $E \leftarrow E \cup \{\phi(K)\}$
Return $y$
```

```
proc Initialize  // $\mathbf{G}_9$
$K \overset{\$}{\leftarrow} \mathcal{K}$ ; $G \overset{\$}{\leftarrow} \mathsf{Fun}(\mathcal{K}, \mathcal{D}, \mathcal{R})$

proc RKFn($\phi, x$)  // $\mathbf{G}_9$
$y \leftarrow G(\phi(K), x)$
Return $y$
```

Figure 3.7: Games for the proof of Theorem 3.6.2

Game $\mathbf{G}_1$ introduces storage of used RKD functions and values of $\overrightarrow{\overline{\omega}}$ in sets D and E respectively and sets flag_1 to true if the same value of $\overrightarrow{\overline{\omega}}$ arises for two different RKD functions. Since this storage does not affect the values returned by **RKFn**

$$\Pr[\,\textsc{Succ}_1\,] = \Pr[\,\textsc{Succ}_0\,] \ .$$

Game $\mathbf{G}_2$ adds the boxed code which changes how the repetition of an $\vec{\overline{\omega}}$ value is handled, by picking instead a random value from $\mathcal{R}^m \setminus E$ that will not repeat any previous one. Games $\mathbf{G}_1$ and $\mathbf{G}_2$ are identical until flag_1 is set to true, hence we have

$$\Pr[\,\textsc{Succ}_1\,] \;\leq\; \Pr[\,\textsc{Succ}_2\,] + \Pr[\,E_1\,]\ ,$$

where E_1 denotes the event that the execution of $\mathscr{A}$ with game $\mathbf{G}_1$ sets flag_1 to true. We design an adversary $\mathscr{D}$ attacking the Φ-key-collision security of M such that

$$\Pr[\,E_1\,] \;\leq\; \mathsf{Adv}^{\mathsf{kc}}_{\Phi,M}(\mathscr{D})\ .$$

Adversary $\mathscr{D}$ runs $\mathscr{A}$. When the latter makes an **RKFn**-query (ϕ, x), adversary $\mathscr{D}$ queries (ϕ, ω_i), for $i = 1, \ldots, |\vec{\omega}|$, to its oracle, then computes $\vec{\overline{\omega}}$ and then $h = H(x, \vec{\overline{\omega}})$ and finally queries (ϕ, h) to its oracle and sends it to $\mathscr{A}$. When $\mathscr{A}$ stops, $\mathscr{D}$ searches for two different RKD functions ϕ queried by $\mathscr{A}$ that lead to the same value $\vec{\overline{\omega}}$ and returns these two functions if found. Since $\vec{\omega}$ is a strong key fingerprint, two such functions lead to the same key, so $\mathscr{D}$ wins if it finds such two functions. (Of course, if the class of RKD functions is claw-free, the advantage of the attacker is 0.)

Game $\mathbf{G}_3$ introduces the storage of hash values in a set G and sets flag_2 to true if the same hash output arises twice. Since this storage does not affect the values returned by **RKFn**, we have

$$\Pr[\,\textsc{Succ}_3\,] = \Pr[\,\textsc{Succ}_2\,]\ .$$

Game $\mathbf{G}_4$ adds the boxed code which changes how repetition of hash values is handled, by picking instead a random value h from $\mathcal{S} \setminus G$ that will not repeat any previously used hash value. Games $\mathbf{G}_3$ and $\mathbf{G}_4$ are identical until flag_2 is set to true, hence we have

$$\Pr[\,\textsc{Succ}_3\,] \;\leq\; \Pr[\,\textsc{Succ}_4\,] + \Pr[\,E_2\,]\ ,$$

where E_2 denotes the event that the execution of $\mathscr{A}$ with game $\mathbf{G}_3$ sets flag_2 to true. We design an adversary $\mathscr{C}$ attacking the cr-security of H such that

$$\Pr[\,E_2\,] \;\leq\; \mathsf{Adv}^{\mathsf{cr}}_{H}(\mathscr{C})\ .$$

Adversary $\mathscr{C}$ starts by picking $K \xleftarrow{\$} \mathcal{K}$ and initializes $j \leftarrow 0$. It runs $\mathscr{A}$. When the latter makes an **RKFn**-query (ϕ, x), adversary $\mathscr{C}$ responds via

```
For $i = 0, \ldots, |\vec{\omega}|$ do: $\overline{\omega}_i \leftarrow M(\phi(K), \omega_i)$
$j \leftarrow j + 1$ ; $\phi_j \leftarrow \phi$ ; $x_j \leftarrow x$
If $\vec{\overline{\omega}} \in E$ and $\phi \notin D$ then $\vec{\overline{\omega}} \xleftarrow{\$} \mathcal{S} \setminus E$          $(*)$
Else $D \leftarrow D \cup \{\phi\}$
$E \leftarrow E \cup \{\vec{\overline{\omega}}\}$
$w_j \leftarrow \vec{\overline{\omega}}$
$h \leftarrow H(x, \vec{\overline{\omega}})$
$h_j \leftarrow h$
$y \leftarrow M(\phi(K), h)$
Return $y$.
```

When $\mathscr{A}$ halts, $\mathscr{C}$ searches for a, b satisfying $1 \leq a < b \leq j$ such that $h_a = h_b$ and, if it finds them, outputs $(x_a, w_a), (x_b, w_b)$ and halts. The pairs (x_a, w_a) and (x_b, w_b) are distinct. Indeed, consider two cases: first, if $\phi_a = \phi_b$ then since $\mathscr{A}$ never repeats an oracle query, $x_a \neq x_b$ hence $(x_a, w_a) \neq (x_b, w_b)$. Second, if $\phi_a \neq \phi_b$, then condition $(*)$ ensures that $w_a \neq w_b$. Hence once again, $(x_a, w_a) \neq (x_b, w_b)$, and then

$$\Pr[\,\textsc{Succ}_3\,] \;\leq\; \Pr[\,\textsc{Succ}_4\,] + \mathsf{Adv}^{\mathsf{cr}}_{H}(\mathscr{C}) \;\; .$$

In game $\mathbf{G}_5$, instead of returning the value $M(\phi(K), h)$, we always return a random value. To show that games $\mathbf{G}_4$ and $\mathbf{G}_5$ are indistinguishable, we design an adversary $\mathscr{B}$ against the $(\mathcal{S}, \Phi)$-unique-input-related-key pseudorandom function security of M such that

$$\Pr[\,\textsc{Succ}_4\,] \;\leq\; \Pr[\,\textsc{Succ}_5\,] + \mathbf{Adv}^{\mathsf{ui\text{-}rka\text{-}prf}}_{M,\mathcal{S},\Phi}(\mathscr{B}) \;\; .$$

Adversary $\mathscr{B}$ starts by initializing sets $D \leftarrow \emptyset$, $E \leftarrow \emptyset$, $G \leftarrow \emptyset$. Then $\mathscr{B}$ runs $\mathscr{A}$. When the latter makes an **RKFn**-query (ϕ, x), $\mathscr{B}$ responds as follows. For $i = 1, \ldots, |\vec{\omega}|$, it asks (ϕ, ω_i) to its oracle and sets $\overline{\omega}_i$ to this value. Since for all $i = 1, \ldots, |\vec{\omega}|$, $\omega_i \notin \mathcal{S}$ by assumption, the value it gets is $M(\phi(K), \omega_i)$, whatever its oracle is. Then, $\mathscr{B}$ checks if $\vec{\overline{\omega}} \in E$ and $\phi \notin D$. If they do, $\mathscr{B}$ picks $\vec{\overline{\omega}} \xleftarrow{\$} \mathcal{R}^m \setminus E$ at random, otherwise $\mathscr{B}$ sets $D \leftarrow D \cup \{\phi\}$. $\mathscr{B}$ then sets $E \leftarrow E \cup \{\vec{\overline{\omega}}\}$. Next, $\mathscr{B}$ computes $h \leftarrow H(x, \vec{\overline{\omega}})$ and checks if $h \in G$. If it does, $\mathscr{B}$ picks $h \xleftarrow{\$} \mathcal{S} \setminus G$ at random. Notice that this step guarantees that all values h are in $\mathcal{S}$ and are all distinct as long as $\mathscr{A}$ makes at most $|\mathcal{S}|$ queries. Finally, $\mathscr{B}$ sets $G \leftarrow G \cup \{h\}$, makes the query (ϕ, h) to its oracle, and returns the value it gets, which is either $M(\phi(K), h)$ or a uniformly random value, to $\mathscr{A}$. When $\mathscr{A}$ halts, $\mathscr{B}$ halts with the same output. The definition of $\mathcal{S}$ ensures that it does not contain any ω_i for $i = 1, \ldots, |\vec{\omega}|$. It follows from these observations that $\mathscr{B}$ is a unique-input adversary for queries in $\mathcal{S}$. Finally, it is clear that if $\mathscr{B}$'s oracle gives real outputs of M for queries in $\mathcal{S}$, then it simulates exactly game $\mathbf{G}_4$ and if $\mathscr{B}$'s oracle gives uniformly random values for queries in $\mathcal{S}$, then it simulates exactly game $\mathbf{G}_5$.

In game $\mathbf{G}_6$, we simply set the value y to a uniformly random value. Clearly, $\mathbf{G}_5$ and $\mathbf{G}_6$ are identical since the value returned is a uniformly random value for any query. Then, we have

$$\Pr[\,\textsc{Succ}_5\,] = \Pr[\,\textsc{Succ}_6\,] \;\; .$$

In game $\mathbf{G}_7$, we check if two different queries can lead to a key collision. Since the "If" test ensures that the returned value is still uniformly random over $\mathcal{R}$ even when two different queries result in the same key, games $\mathbf{G}_6$ and $\mathbf{G}_7$ are identical. Hence,

$$\Pr[\,\textsc{Succ}_6\,] = \Pr[\,\textsc{Succ}_7\,] \;\; .$$

In game $\mathbf{G}_8$, we compute the output of **RKFn** using a random function G in $\mathsf{Fun}(\mathcal{K}, \mathcal{D}, \mathcal{R})$. Since games $\mathbf{G}_7$ and $\mathbf{G}_8$ are identical until flag_3 is set to true, we have

$$\Pr[\,\textsc{Succ}_7\,] \;\leq\; \Pr[\,\textsc{Succ}_8\,] + \Pr[\,E_3\,] \;\; ,$$

where E_3 denotes the event that the execution of $\mathscr{A}$ with game $\mathbf{G}_8$ sets flag_3 to true. To bound the probability of event E_3, we design an adversary $\mathscr{E}$ attacking Φ-statistical-key-collision security for $\mathsf{Fun}(\mathcal{K}, \mathcal{D}, \mathcal{R})$ such that

$$\Pr[\,E_3\,] \;\leq\; \mathsf{Adv}^{\mathsf{skc}}_{\Phi}(\mathscr{E}) \;\; .$$

Adversary $\mathscr{E}$ runs $\mathscr{A}$. When the latter makes an **RKFn**-query (ϕ, x), so does $\mathscr{E}$ and $\mathscr{E}$ returns the value it receives to $\mathscr{A}$. When $\mathscr{A}$ stops, if $\mathscr{A}$ has queried two different functions ϕ_1 and ϕ_2 such that $\phi_1(K) = \phi_2(K)$ then b' was set to 1 when the second of these two functions was queried by $\mathscr{E}$, and then $\mathscr{E}$ wins. (Of course, if the class of RKD functions is claw-free, this probability is 0.)

Since $\mathscr{A}$ does not repeat oracle queries and since key collisions are dealt with in a similar way, it follows that games $\mathbf{G}_8$ and $\mathbf{G}_9$ are identical. Thus,

$$\Pr[\,\textsc{Succ}_8\,] = \Pr[\,\textsc{Succ}_9\,] \ .$$

Finally, $\mathbf{G}_{10}$ matches the description of the $\mathsf{RKPRFRand}_F$ game, so:

$$\Pr[\,\textsc{Succ}_{10}\,] = \Pr\left[\mathsf{RKPRFRand}_F^{\mathscr{A}} \Rightarrow 1\right] \ .$$

Theorem 3.6.2 now follows by combining the bounds arising in the different game hops. □

3.6.2 From Malleability to Unique-Input-Related-Key Security

In this section, we explore the relationship between key-malleability and unique-input-related-key security. Specifically, we show that a Φ-key malleable pseudorandom function is also an $(\mathcal{S}, \Phi)$-unique-input-related-key secure pseudorandom function under the standard pseudorandom function security as soon as the key-transformer KT associated with M satisfies a new notion of uniformity that we call $\mathcal{S}$-uniformity and which is defined below.

$\mathcal{S}$-Uniform Key-Transformer. Let M be a pseudorandom function whose key space, domain, and range are respectively denoted $\mathcal{K}, \mathcal{D}$, and $\mathcal{R}$, and let Φ be a class of RKD functions such that there exists a key-transformer KT for (M, Φ). We generalize the uniformity property of a key-transformer by allowing the uniformity condition to be restricted to a subset $\mathcal{S}$ of $\mathcal{D}$. Indeed, let $\mathcal{S}$ be a subset of $\mathcal{D}$; then we say that KT is an *$\mathcal{S}$-uniform key-transformer* if the games $\mathcal{S}$-$\mathsf{KTReal}_{\mathsf{KT}}$ and $\mathcal{S}$-$\mathsf{KTRand}_{\mathsf{KT}}$ defined in Figure 3.8 are *perfectly indistinguishable* for any Φ-restricted adversary $\mathscr{A}$, where $\mathscr{A}$ belongs to the class of adversaries such that all queries (ϕ, x) with $x \in \mathcal{S}$ made by $\mathscr{A}$ to its oracle are for distinct values of x. That is,

$$\Pr\left[\mathcal{S}\text{-}\mathsf{KTReal}_{\mathsf{KT}}^{\mathscr{A}} \Rightarrow 1\right] = \Pr\left[\mathcal{S}\text{-}\mathsf{KTRand}_{\mathsf{KT}}^{\mathscr{A}} \Rightarrow 1\right] \ .$$

Please note that the standard uniformity condition given on page 33 corresponds to the KT being a $\mathcal{D}$-uniform key-transformer. Whether $\mathcal{S}$-uniformity is implied by (regular) uniformity is an open question.

Theorem 3.6.3. *Let M be a pseudorandom function and Φ be a class of* RKD *functions. Assuming there exists an $\mathcal{S}$-uniform key-transformer* KT *for (M, Φ), then M is an $(\mathcal{S}, \Phi)$-unique-input-related-key secure pseudorandom function under the pseudorandom function security of M.*

Moreover, the running time of the reduction is polynomial in the running time of the adversary and of the key-transformer.

Proof of Theorem 3.6.3. The proof is based on the sequence of games in Figure 3.9. Let $\textsc{Succ}_i$ denote the event that game $\mathbf{G}_i$ output takes the value 1. Let $\mathscr{A}$ be an adversary against the $(\mathcal{S}, \Phi)$-unique-input-related-key security of M, and let us assume (without loss of generality) that it never repeats a query.

```
$\mathcal{S}$-$\mathsf{KTReal}_{\mathsf{KT}}$
proc Initialize
$f \xleftarrow{\$} \mathsf{Fun}(\mathcal{D}, \mathcal{R})$
proc KTFn($\phi, x$)
Return $\mathsf{KT}^f(\phi(K), x)$
proc Finalize($b$)
Return $b$
```

```
$\mathcal{S}$-$\mathsf{KTRand}_{\mathsf{KT}}$
proc Initialize
$f \xleftarrow{\$} \mathsf{Fun}(\mathcal{D}, \mathcal{R})$
proc KTFn($\phi, x$)
If $x \in \mathcal{S}$ then
    $y \xleftarrow{\$} \mathcal{R}$
Else
    $y \leftarrow \mathsf{KT}^f(\phi(K), x)$
Return $y$
proc Finalize($b$)
Return $b$
```

Figure 3.8: Games defining the $\mathcal{S}$-uniformity of a key-transformer KT

```
proc Initialize  // $\mathbf{G}_0$
$K \xleftarrow{\$} \mathcal{K}$
proc RKFn($\phi, x$)  // $\mathbf{G}_0$
$y \leftarrow M(\phi(K), x)$
Return $y$
proc Finalize($b$)  // All Games
Return $b$
```

```
proc Initialize  // $\mathbf{G}_1$
$K \xleftarrow{\$} \mathcal{K}$
proc RKFn($\phi, x$)  // $\mathbf{G}_1$
$y \leftarrow \mathsf{KT}^M(\phi, x)$
Return $y$
```

```
proc Initialize  // $\mathbf{G}_2$
$K \xleftarrow{\$} \mathcal{K}$
$f \xleftarrow{\$} \mathsf{Fun}(\mathcal{K}, \mathcal{D}, \mathcal{R})$
proc RKFn($\phi, x$)  // $\mathbf{G}_2$
$y \leftarrow \mathsf{KT}^f(\phi, x)$
Return $y$
```

```
proc Initialize  // $\mathbf{G}_3$
$K \xleftarrow{\$} \mathcal{K}$
$f \xleftarrow{\$} \mathsf{Fun}(\mathcal{K}, \mathcal{D}, \mathcal{R})$
proc RKFn($\phi, x$)  // $\mathbf{G}_3$
If $x \in \mathcal{S}$ then $y \xleftarrow{\$} \mathcal{R}$
Else $y \leftarrow \mathsf{KT}^f(\phi, x)$
Return $y$
```

```
proc Initialize  // $\mathbf{G}_4$
$K \xleftarrow{\$} \mathcal{K}$
proc RKFn($\phi, x$)  // $\mathbf{G}_4$
If $x \in \mathcal{S}$ then $y \xleftarrow{\$} \mathcal{R}$
Else $y \leftarrow \mathsf{KT}^M(\phi, x)$
Return $y$
```

```
proc Initialize  // $\mathbf{G}_5$
$K \xleftarrow{\$} \mathcal{K}$
proc RKFn($\phi, x$)  // $\mathbf{G}_5$
If $x \in \mathcal{S}$ then $y \xleftarrow{\$} \mathcal{R}$
Else $y \leftarrow M(\phi(K), x)$
Return $y$
```

Figure 3.9: Games for the proof of Theorem 3.6.3

$\mathbf{G}_0$ matches the description of the $\mathsf{UIRKPRFReal}_{M,\mathcal{S}}$ game, so:

$$\Pr[\,\textsc{Succ}_0\,] = \Pr\left[\,\mathsf{UIRKPRFReal}^{\mathscr{A}}_{M,\mathcal{S}} \Rightarrow 1\,\right] \ .$$

In game $\mathbf{G}_1$, we use the key-transformer KT to compute $M(\phi(K), \cdot)$ via oracle calls to $M(K, \cdot)$. The correctness property of the key-transformer implies

$$\Pr[\,\textsc{Succ}_0\,] = \Pr[\,\textsc{Succ}_1\,] \ .$$

In game $\mathbf{G}_2$, we replace the oracle $M(K, \cdot)$ given to the key-transformer KT by a random function f. We design an adversary $\mathscr{B}$ attacking the standard pseudorandom function security of M such that

$$\Pr[\,\textsc{Succ}_1\,] \leq \Pr[\,\textsc{Succ}_2\,] + \mathsf{Adv}^{\mathsf{prf}}_{M}(\mathscr{B}) \ .$$

Adversary $\mathscr{B}$ runs $\mathscr{A}$. When the latter makes an RKFn-query (ϕ, x), adversary $\mathscr{B}$ responds via

$y \leftarrow \mathsf{KT}^{\mathbf{Fn}}(\phi, x)$
Return y

where **Fn** is $\mathscr{B}$'s own oracle. When $\mathscr{A}$ halts, $\mathscr{B}$ halts with the same output. Then

$$\Pr\left[\mathsf{PRFReal}_M^{\mathscr{B}} \Rightarrow 1\right] = \Pr\left[\,\textsc{Succ}_1\,\right] \quad\text{and}\quad \Pr\left[\mathsf{PRFRand}_M^{\mathscr{B}} \Rightarrow 1\right] = \Pr\left[\,\textsc{Succ}_2\,\right]\ .$$

Games $\mathbf{G}_2$ and $\mathbf{G}_3$ differ only in the way that queries (ϕ, x) with $x \in \mathcal{S}$ are handled. Indeed, for such queries, in $\mathbf{G}_3$, the output is just taken uniformly at random instead of computed using the key-transformer. Since the adversary $\mathscr{A}$ belongs to the class of Φ-restricted adversaries such that all queries (ϕ, x) with $x \in \mathcal{S}$ made by $\mathscr{A}$ to its oracle are for distinct values of x, games $\mathbf{G}_2$ and $\mathbf{G}_3$ match exactly games $\mathcal{S}$-$\mathsf{KTReal}_{\mathsf{KT}}$ and $\mathcal{S}$-$\mathsf{KTRand}_{\mathsf{KT}}$, respectively. Since we assume that KT is an $\mathcal{S}$-uniform key-transformer KT, these two games are perfectly indistinguishable. So

$$\Pr\left[\,\textsc{Succ}_2\,\right] = \Pr\left[\,\textsc{Succ}_3\,\right]\ .$$

In game $\mathbf{G}_4$, we replace the random function f given to the key-transformer KT by the oracle $M(K, \cdot)$. We design an adversary $\mathscr{B}$ attacking the pseudorandom function security of M such that

$$\Pr\left[\,\textsc{Succ}_3\,\right] \le \Pr\left[\,\textsc{Succ}_4\,\right] + \mathsf{Adv}_M^{\mathsf{prf}}(\mathscr{B})\ .$$

Adversary $\mathscr{B}$ runs $\mathscr{A}$. When the latter makes an RKFn-query (ϕ, x), adversary $\mathscr{B}$ responds via

If $x \in \mathcal{S}$ then $y \xleftarrow{\$} \mathcal{R}$
Else $y \leftarrow \mathsf{KT}^{\mathbf{Fn}}(\phi, x)$
Return y

where **Fn** is $\mathscr{B}$'s own oracle. When $\mathscr{A}$ halts with output b, $\mathscr{B}$ halts with output $1 - b$. Then

$$\Pr\left[\mathsf{PRFReal}_M^{\mathscr{B}} \Rightarrow 1\right] = \Pr\left[\,\textsc{Succ}_3\,\right] \quad\text{and}\quad \Pr\left[\mathsf{PRFRand}_M^{B} \Rightarrow 1\right] = \Pr\left[\,\textsc{Succ}_4\,\right]\ .$$

Finally, in game $\mathbf{G}_5$, instead of using the key-transformer KT to compute $M(\phi(K), \cdot)$ via oracle calls to $M(K, \cdot)$, we use M directly. The correctness property of the key-transformer implies that

$$\Pr\left[\,\textsc{Succ}_4\,\right] = \Pr\left[\,\textsc{Succ}_5\,\right]\ .$$

Finally, $\mathbf{G}_5$ matches the description of the $\mathsf{UIRKPRFRand}_{M,\mathcal{S}}$ game, so:

$$\Pr\left[\,\textsc{Succ}_5\,\right] = \Pr\left[\mathsf{UIRKPRFRand}_{M,\mathcal{S}}^{\mathscr{A}} \Rightarrow 1\right]\ .$$

Theorem 3.6.3 now follows by combining the bounds arising in the different game hops. $\square$

3.7 Application: Related-Key Security for Affine Relations.

We can apply Theorem 3.6.2 and Theorem 3.6.3 to NR^* to prove that the construction given in Section 3.5 is a Φ_{aff}-related-key secure pseudorandom function. Note that this gives an alternative and simpler proof of Theorem 3.5.5. As the result has already been proven in a previous section, we do not detail every step of this proof.

Let $\mathcal{S} = \{0,1\}^n \setminus (\{\omega_1, \ldots, \omega_n\} \cup \{0^n\})$. The only point that remains to prove is that there exists an $\mathcal{S}$-uniform key-transformer $\mathsf{KT}_{\Phi_{\mathsf{aff}}}$ for $(\mathsf{NR}^*, \Phi_{\mathsf{aff}})$. This result is actually implied by Lemma 3.5.3. Indeed, the same key-transformer is an $\mathcal{S}$-uniform key-transformer for $(\mathsf{NR}^*, \Phi_{\mathsf{aff}})$. This statement is implied by the fact that games $\mathbf{G}_0$ and $\mathbf{G}_{n-1}$, defined in the proof of Lemma 3.5.3, are indistinguishable, even when the adversary is not a unique-input adversary with respect to the set $\overline{\mathcal{S}}$ (where $\overline{\mathcal{S}}$ denotes the complement of $\mathcal{S}$).

To see why, note that the argument used to prove that Games $\mathbf{G}_j$ and $\mathbf{G}_{j+1}$ are indistinguishable in that proof remains valid because all points in $\overline{\mathcal{S}}$ have a strictly smaller Hamming weight than those in $\mathcal{S}$. Hence, there exists an $\mathcal{S}$-uniform key-transformer $\mathsf{KT}_{\Phi_{\mathsf{aff}}}$ for $(\mathsf{NR}^*, \Phi_{\mathsf{aff}})$, and the $(\mathcal{S}, \Phi_{\mathsf{aff}})$-unique-input-related-key pseudorandom function security of NR^* follows. Finally, by applying Theorem 3.6.2, we can prove a similar statement to the one in Theorem 3.5.5.

Remark 3.7.1. While this result is immediate from the results of Section 3.5, it is worth noting that it is not clear, in general, that the usual uniformity condition of a key-transformer directly implies that the same key-transformer is $\mathcal{S}$-uniform for $\mathcal{S} \subset \mathcal{D}$ without making additional assumptions about the key-transformer.

Chapter 4

A New Family of Assumptions

In this chapter, we study particular forms Matrix-Decisional-Diffie-Hellman (MDDH) assumptions. MDDH assumptions were introduced in 2013 by Escala *et al.* [EHK+13]. The authors defined an elegant algebraic framework for describing a wide variety of assumptions, that encompasses in particular most of classical assumptions, such as DDH, DLin, or k-Lin.

Here, we first give a simple introduction to the generic multilinear group model and briefly describe the framework and main results from [EHK+13]. We later introduce our new family of assumptions, that fits partially in their framework, and that we use to prove our main result from Chapter 5. We then argue about the security of these assumptions, by first directly relating a subset of our assumptions to classical assumptions, and then by providing a proof that every of our new assumption holds in the generic multilinear group model.

The latter proof of security, provided in Section 4.4, is rather technical but is not necessary towards understanding the rest of this manuscript and thus can easily be omitted. However, these assumptions will be used throughout this manuscript and we encourage the reader to study carefully Sections 4.2 and 4.3.1. In particular, the relations between our new assumptions and classical assumptions, summarized in Table 4.1 on page 56, should be kept in mind as our statements are often written using directly the classical assumptions.

Contents

4.1 The Matrix-Decisional-Diffie-Hellman Assumptions

Escala *et al.* defined the MDDH assumptions as the hardness of decisional problems. Specifically, given $\mathbb{G} = \langle g \rangle$ a cyclic group of order p, an MDDH assumption is parametrized by a distribution $\mathcal{D}$ over $\mathbb{Z}_p^{\ell \times k}$, and states that, for any matrix from $\mathbf{\Gamma}$ chosen at random from $\mathcal{D}$, it is hard to distinguish, given $[1]$ and $[\mathbf{\Gamma}]$, a group element $[\mathbf{\Gamma} \cdot \vec{r}]$, with $\vec{r} \in \mathbb{Z}_p^{\ell}$ a uniformly random vector, from $[\vec{u}]$, with $\vec{u} \in \mathbb{Z}_p^{\ell}$ a uniformly random vector. This notion is formally defined as follows:

Definition 4.1.1 (MDDH-Assumption)**.** *For $\mathcal{D}_{\ell,k}$ an efficiently samplable distribution over $\mathbb{Z}_p^{\ell \times k}$, with $\ell > k$, the advantage of an adversary $\mathscr{D}$ against $\mathcal{D}_{\ell,k}$-MDDH problem $\mathbb{G}$ is defined as:*

$$\mathbf{Adv}_{\mathbb{G}}^{\mathcal{D}_{\ell,k}\text{-mddh}}(\mathscr{D}) := \Pr\left[\mathcal{D}_{\ell,k}\text{-MDDHReal}_{\mathbb{G}}^{\mathscr{D}} \Rightarrow 1\right] - \Pr\left[\mathcal{D}_{\ell,k}\text{-MDDHRand}_{\mathbb{G}}^{\mathscr{D}} \Rightarrow 1\right] \ ,$$

where the probabilities are over the choice of $\mathbf{\Gamma} \overset{\$}{\leftarrow} \mathcal{D}_{\ell,k}$, $\boldsymbol{W} \overset{\$}{\leftarrow} \mathbb{Z}_p^{k \times 1}$, $\boldsymbol{U} \overset{\$}{\leftarrow} \mathbb{Z}_p^{\ell \times 1}$, and the random coins used by the adversary, and where $\mathcal{D}_{\ell,k}$-MDDHReal$_{\mathbb{G}}$ and $\mathcal{D}_{\ell,k}$-MDDHRand$_{\mathbb{G}}$ are described in Figure 4.1.

$\mathcal{D}_{\ell,k}$-MDDHReal$_{\mathbb{G}}$	$\mathcal{D}_{\ell,k}$-MDDHRand$_{\mathbb{G}}$
proc Initialize	**proc Initialize**
$\mathbf{\Gamma} \overset{\$}{\leftarrow} \mathcal{D}_{\ell,k}$	$\mathbf{\Gamma} \overset{\$}{\leftarrow} \mathcal{D}_{\ell,k}$
$\boldsymbol{W} \overset{\$}{\leftarrow} \mathbb{Z}_p^{k \times 1}$	$\boldsymbol{U} \overset{\$}{\leftarrow} \mathbb{Z}_p^{\ell \times 1}$
Return $([1], [\mathbf{\Gamma}], [\mathbf{\Gamma} \cdot \boldsymbol{W}])$	Return $([1], [\mathbf{\Gamma}], [\boldsymbol{U}])$
proc Finalize(b)	**proc Finalize**(b)
Return b	Return b

Figure 4.1: Games defining the $\mathcal{D}_{\ell,k}$-MDDH problem in $\mathbb{G}$

Specifying the distribution $\mathcal{D}_{\ell,k}$, the MDDH assumption can catch most of classical assumptions, such as DDH, DLin, k-Lin, ... For instance, setting $\mathcal{D}_{\ell,k}$ to be the distribution of matrices of the form $\begin{pmatrix} a_1 & 0 \\ 0 & a_2 \\ 1 & 1 \end{pmatrix}$ with $a_1, a_2 \overset{\$}{\leftarrow} \mathbb{Z}_p$, we obtain exactly the DLin assumption, and extending this to matrices of size $(k+1) \times k$, the k-Lin assumption.

Moreover, the distribution over $\mathbb{Z}_p^{\ell \times k}$ that results in the weakest MDDH assumption is simply the uniform distribution, denoted $\mathcal{U}_{\ell,k}$. In particular, we denote by $\mathcal{U}_k$ the uniform distribution over $\mathbb{Z}_p^{k+1,k}$, so $\mathcal{U}_k = \mathcal{U}_{k+1,k}$, and the $\mathcal{U}_k$-MDDH assumption is then a weaker assumption than k-Lin.

Another nice feature of the MDDH assumption lies in their random self-reducibility. Namely, let $(\mathcal{D}_{\ell,k}, Q)$-MDDH denote the Q-fold $\mathcal{D}_{\ell,k}$-MDDH assumption, which is similar to the $\mathcal{D}_{\ell,k}$-MDDH assumption, except that $\boldsymbol{W} \overset{\$}{\leftarrow} \mathbb{Z}_p^{k \times Q}$, $\boldsymbol{U} \overset{\$}{\leftarrow} \mathbb{Z}_p^{\ell \times Q}$, so the advantage of an adversary $\mathscr{D}$ against the $(\mathcal{D}_{\ell,k}, Q)$-MDDH problem in $\mathbb{G}$ is defined as:

$$\mathbf{Adv}_{\mathbb{G}}^{(\mathcal{D}_{\ell,k},Q)\text{-mddh}}(\mathscr{D}) := \Pr\left[(\mathcal{D}_{\ell,k}, Q)\text{-MDDHReal}_{\mathbb{G}}^{\mathscr{D}} \Rightarrow 1\right] - \Pr\left[(\mathcal{D}_{\ell,k}, Q)\text{-MDDHRand}_{\mathbb{G}}^{\mathscr{D}} \Rightarrow 1\right]$$

where the probabilities are over the choice of $\boldsymbol{\Gamma} \overset{\$}{\leftarrow} \mathcal{D}_{\ell,k}$, $\boldsymbol{W} \overset{\$}{\leftarrow} \mathbb{Z}_p^{k \times Q}$, $\boldsymbol{U} \overset{\$}{\leftarrow} \mathbb{Z}_p^{\ell \times Q}$, and the random coins used by the adversary, and where experiments $(\mathcal{D}_{\ell,k}, Q)$-$\mathsf{MDDHReal}_{\mathbb{G}}$ and $(\mathcal{D}_{\ell,k}, Q)$-$\mathsf{MDDHRand}_{\mathbb{G}}$ are described in Figure 4.2. Then, this assumption is actually implied by the latter $\mathcal{D}_{\ell,k}$-MDDH, as stated in the above lemma, for any distribution $\mathcal{D}_{\ell,k}$.

$(\mathcal{D}_{\ell,k}, Q)$-$\mathsf{MDDHReal}_{\mathbb{G}}$	$(\mathcal{D}_{\ell,k}, Q)$-$\mathsf{MDDHRand}_{\mathbb{G}}$
proc Initialize	**proc Initialize**
$\boldsymbol{\Gamma} \overset{\$}{\leftarrow} \mathcal{D}_{\ell,k}$	$\boldsymbol{\Gamma} \overset{\$}{\leftarrow} \mathcal{D}_{\ell,k}$
$\boldsymbol{W} \leftarrow \mathbb{Z}_p^{k \times Q}$	$\boldsymbol{U} \overset{\$}{\leftarrow} \mathbb{Z}_p^{\ell \times Q}$
Return $([1], [\boldsymbol{\Gamma}], [\boldsymbol{\Gamma} \cdot \boldsymbol{W}])$	Return $([1], [\boldsymbol{\Gamma}], [\boldsymbol{U}])$
proc Finalize(b)	**proc Finalize**(b)
Return b	Return b

Figure 4.2: Games defining the $(\mathcal{D}_{\ell,k}, Q)$-$\mathsf{MDDH}$ problem in $\mathbb{G}$

Lemma 4.1.2 ([EHK+13, Lemma 1]). *Assuming the $\mathcal{D}_{\ell,k}$-MDDH assumption holds in $\mathbb{G}$, then so does the $(\mathcal{D}_{\ell,k}, Q)$-$\mathsf{MDDH}$ assumption.*

Finally, Escala *et al.* also proposed some conditions on the distribution $\mathcal{D}_{\ell,k}$ to guarantee the hardness of the $\mathcal{D}_{\ell,k}$-MDDH problem in the generic multilinear group model. As our distribution of matrices does not satisfy these conditions in general, we do not explain these conditions. Please refer to [EHK+13] for details. This is the reason we provide a proof of security of our assumptions in the generic multilinear group model, in the general case, though most of our assumptions are related to classical assumptions. Let us now define formally the distribution of matrices we are interested in.

4.2 A New Family of Matrix-Diffie-Hellman Assumptions

We now introduce a new family of MDDH assumptions, termed the $\mathcal{E}_{k,d}$-MDDH assumptions. Our main result, a generic and algebraic framework for pseudorandom functions and associated primitives, proves that any function that achieves certain simple algebraic properties is a secure pseudorandom function under the $\mathcal{E}_{k,d}$-MDDH assumption. Our family of MDDH assumptions is simply an MDDH for a particular distribution of matrices over $\mathbb{Z}_p$, denoted $\mathcal{E}_{k,d}$, defined below.

Definition 4.2.1 ($\mathcal{E}_{k,d}$-MDDH Assumption). *Let $k, d \geq 1$. We define the matrix distribution $\mathcal{E}_{k,d}$ as the set of all matrices $\boldsymbol{\Gamma}$ of the form:*

$$\boldsymbol{\Gamma} = \begin{pmatrix} \boldsymbol{A}^0 \cdot \boldsymbol{B} \\ \boldsymbol{A}^1 \cdot \boldsymbol{B} \\ \vdots \\ \boldsymbol{A}^d \cdot \boldsymbol{B} \end{pmatrix} \in \mathbb{Z}_p^{k(d+1) \times k} \quad \text{with } \boldsymbol{A}, \boldsymbol{B} \overset{\$}{\leftarrow} \mathbb{Z}_p^{k \times k} \ .$$

We then define the $\mathcal{E}_{k,d}$-MDDH assumption as the hardness of the MDDH problem parametrized by the distribution $\mathcal{E}_{k,d}$.

4.3 Connexion with Standard Assumptions

4.3.1 Summary of Relations

Our family of assumptions encompasses a wide-variety of assumptions. In particular, for $d = 1$, it is immediate that the $\mathcal{E}_{k,d}$-MDDH assumption is implied by the DDH assumption when $k = 1$ and by the $\mathcal{U}_k$-MDDH for $k \geq 2$. Also, for $k = 1$ and $d \geq 2$, the $\mathcal{E}_{k,d}$-MDDH assumption is implied by the d-DDHI assumption, as detailed below. Finally, we show in Section 4.4, that for the remaining cases ($k \geq 2, d \geq 2$), the $\mathcal{E}_{k,d}$-MDDH assumption holds in the generic k-linear group model. These security results are summarized in Table 4.1.

Table 4.1: Security of $\mathcal{E}_{k,d}$-MDDH

	$k = 1$	$k = 2$	$k \geq 3$
$d = 1$	$= \mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}$	$\lesssim 2 \cdot \mathbf{Adv}_{\mathbb{G}}^{\mathcal{U}_2\text{-}\mathsf{mddh}}$	$\lesssim k \cdot \mathbf{Adv}_{\mathbb{G}}^{\mathcal{U}_k\text{-}\mathsf{mddh}}$
$d \geq 2$	$\lesssim d \cdot \mathsf{Adv}_{\mathbb{G}}^{d\text{-}\mathsf{ddhi}}$	generic bilinear group	generic k-linear group

4.3.2 Relation with the DDHI Assumption

In this section, we show that the $\mathcal{E}_{1,d}$-MDDH assumption is implied by the d-DDHI assumption. Towards this goal, we first introduce a new intermediate assumption, termed the $\mathcal{E}_{1,(d,\ell)}$-MDDH assumption. Here, we abuse notation as this assumption is not exactly an MDDH assumption in the sense of the above definition. For $1 \leq \ell \leq d$, we define the advantage of an adversary against the $\mathcal{E}_{1,(d,\ell)}$-MDDH problem as:

$$\mathbf{Adv}_{\mathbb{G}}^{\mathcal{E}_{1,(d,\ell)}\text{-}\mathsf{mddh}}(\mathscr{D}) := \Pr\left[\mathcal{E}_{1,(d,\ell)}\text{-}\mathsf{MDDHReal}_{\mathbb{G}}^{\mathscr{D}} \Rightarrow 1\right] - \Pr\left[\mathcal{E}_{1,(d,\ell)}\text{-}\mathsf{MDDHRand}_{\mathbb{G}}^{\mathscr{A}} \Rightarrow 1\right] ,$$

where the probabilities are over the choice of $\mathbf{\Gamma} \xleftarrow{\$} \mathcal{E}_{1,d}$, $w, c \xleftarrow{\$} \mathbb{Z}_p$, and the random coins used by the adversary, and where $\mathcal{E}_{1,(d,\ell)}$-$\mathsf{MDDHReal}_{\mathbb{G}}$ and $\mathcal{E}_{1,(d,\ell)}$-$\mathsf{MDDHRand}_{\mathbb{G}}$ are described in Figure 4.3.

$\mathcal{E}_{1,(d,\ell)}$-$\mathsf{MDDHReal}_{\mathbb{G}}$	$\mathcal{E}_{1,(d,\ell)}$-$\mathsf{MDDHRand}_{\mathbb{G}}$
proc Initialize	**proc Initialize**
$a, b, w \xleftarrow{\$} \mathbb{Z}_p$	$a, b, w, c \xleftarrow{\$} \mathbb{Z}_p$
$\mathbf{\Gamma} \leftarrow (b, ab, a^2b, \ldots, a^db)$	$\mathbf{\Gamma} \leftarrow (b, ab, a^2b, \ldots, a^db)$
$\boldsymbol{Z} \leftarrow (bw, abw, a^2bw, \ldots, a^{\ell-1}bw, a^{\ell}bw)$	$\boldsymbol{Z} \leftarrow (bw, abw, a^2bw, \ldots, a^{\ell-1}bw, c)$
Return $([1], [\mathbf{\Gamma}], [\boldsymbol{Z}])$	Return $([1], [\mathbf{\Gamma}], [\boldsymbol{Z}])$
proc Finalize(b)	**proc Finalize**(b)
Return b	Return b

Figure 4.3: Games defining the $\mathcal{E}_{1,(d,\ell)}$-MDDH problem in $\mathbb{G}$

Hence, this assumption simply states that, given $[\mathbf{\Gamma}]$ with $\mathbf{\Gamma} \xleftarrow{\$} \mathcal{E}_{1,d}$-$\mathsf{MDDH}$ and the first ℓ coordinates of $[\mathbf{\Gamma} \cdot w]$, with $w \xleftarrow{\$} \mathbb{Z}_p$, it is hard to distinguish the $\ell + 1$ coordinate of $[\mathbf{\Gamma} \cdot w]$ from a uniformly random group element.

The following lemma follows immediately from a standard hybrid argument.

Lemma 4.3.1. *Assuming $\mathcal{E}_{1,(d,\ell)}$-MDDH holds in $\mathbb{G}$, then so does $\mathcal{E}_{1,d}$-MDDH.*

Proof of Lemma 4.3.1. The proof follows a standard hybrid argument. We define games H_ℓ for $\ell = 0, \ldots, d$ as in Figure 4.4.

```
proc Initialize // $\mathrm{H}_\ell$, $\ell = 0, \ldots, d$
$a, b, w \xleftarrow{\$} \mathbb{Z}_p$, $u_k \xleftarrow{\$} \mathbb{Z}_p$, for $k = \ell+1, \ldots, d$
$\mathbf{\Gamma} \leftarrow (b, ab, a^2b, \ldots, a^d b)$
$\boldsymbol{Z} \leftarrow (bw, abw, a^2bw, \ldots, a^\ell bw, u_{\ell+1}, \ldots, u_d)$
Return $([1], [\mathbf{\Gamma}], [\boldsymbol{Z}])$

proc Finalize($b$)
Return $b$
```

Figure 4.4: Games for the proof of Lemma 4.3.1

Clearly, we have $\mathrm{H}_0 \equiv \mathcal{E}_{1,d}$-MDDHRand and $\mathrm{H}_d \equiv \mathcal{E}_{1,d}$-MDDHReal. Moreover, let $\mathscr{D}$ be an adversary against the $\mathcal{E}_{1,d}$-MDDH problem in $\mathbb{G}$. It is straightforward to construct an adversary $\mathscr{A}_\ell$ such that $\Pr[\mathrm{H}_\ell \Rightarrow 1] - \Pr[\mathrm{H}_{\ell-1} \Rightarrow 1] \leq \mathbf{Adv}_{\mathbb{G}}^{\mathcal{E}_{1,(d,\ell)}\text{-mddh}}(\mathscr{A}_l)$, for $\ell = 1, \ldots, d$. Adversary $\mathscr{A}_\ell$ simply samples $d - \ell$ random group elements to complete its matrix and the simulation is perfect.

Finally, it is clear that $\mathbf{Adv}_{\mathbb{G}}^{\mathcal{E}_{1,(d,\ell)}\text{-mddh}}(\mathscr{A}_l) \leq \mathbf{Adv}_{\mathbb{G}}^{\mathcal{E}_{1,(d,d)}\text{-mddh}}(\mathscr{A}_d)$ for $\ell = 1, \ldots, d$, since one can obtain the $\mathcal{E}_{1,(d,\ell)}$-MDDH matrices from a $\mathcal{E}_{1,(d,d)}$-MDDH by simply taking the last $\ell + 1$ rows of the second matrix (which is a perfect tuple fixing $w = a^{d-\ell}w$). Lemma 4.3.1 easily follows. □

We now conclude with the following lemma.

Lemma 4.3.2. *Assuming d-DDHI holds in $\mathbb{G}$, then so does $\mathcal{E}_{1,(d,d)}$-MDDH.*

Proof of Lemma 4.3.2. Let $\mathscr{D}$ be an adversary against the $\mathcal{E}_{1,(d,d)}$-MDDH problem in $\mathbb{G}$ so it has a tuple $\boldsymbol{Z} = ([bw], [abw], \ldots, \left[a^d bw\right], z)$ where $z = \left[a^{d+1}bw\right]$ or z is a random group element $[c]$. Then it chooses $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_p$ at random and computes the tuple $\boldsymbol{X} \in \mathbb{Z}_p^{d+1\times 1}$ by letting $X_\ell = X_{\ell+1}^\alpha \cdot X_\ell^\beta$ for $\ell = 0, \ldots, d-1$ and $X_d = z^\alpha \cdot X_d^\beta$. Hence, for $\ell = 0, \ldots, d-1$, it is clear that $X_\ell = \left[(\alpha a + \beta)a^\ell b\right]$. If $z = \left[a^{d+1}\right]$, we have $X_d = \left[(\alpha a + \beta)a^d b\right]$. If $z = [c]$ with $c \xleftarrow{\$} \mathbb{Z}_p$, then $X_d = \left[(\alpha c + \beta a^d)b\right]$.

Let us fix $a, c \in \mathbb{Z}_p$ and let $b' = \alpha a + \beta$ and $c' = \alpha c + \beta a^d$. Then, $[\mathbf{\Gamma}]$ and $[\boldsymbol{X}]$ form exactly an $\mathcal{E}_{1,(d,d)}$-MDDH tuple if and only if for any fixed $b', c' \in \mathbb{Z}_p$, there is a unique $(\alpha, \beta) \in \mathbb{Z}_p$ such that $b' = \alpha a + \beta$ and $c' = \alpha c + \beta a^d$. Hence, we need the determinant of the matrix $\begin{pmatrix} a & 1 \\ c & a^d \end{pmatrix}$ to be non-zero. This determinant is $D = a^{d+1} - c$ so it is non-zero if and only if $c \neq a^{d+1}$. Since c is by definition uniformly random in $\mathbb{Z}_p$, we have $D \neq 0$ with probability $\frac{p-1}{p}$.

To finish the proof, we simply need to explain how adversary $\mathscr{A}$ can get such a tuple $\boldsymbol{X}$ from its d-DDHI tuple. Let $([1], [a], \ldots, \left[a^d\right], x) \in \mathbb{G}^{d+2}$ be a d-DDHI tuple. Then the tuple $((\left[a^d\right], \left[a^{d-1}\right], \ldots, [a], [1]), x)$ is such a tuple since $h = \left[a^d\right]$ is a random generator (with overwhelming probability) of $\mathbb{G}$ (since $[1]$ is a random generator) and $\frac{1}{a}$ is random in $\mathbb{Z}_p^*$, so we have $\left[a^{d-j}\right] = h^{(\frac{1}{a})^j}$, for $j = 0, \ldots, d$ and either x is random, either $x = \left[\frac{1}{a}\right]$ and then

$x = h^{(\frac{1}{a})^{d+1}}$. Hence, this is exactly a tuple of the wanted form $\boldsymbol{X}$. The claim now easily follows. □

4.4 Security in the Generic Multilinear Group Model

In this section, and to conclude this chapter, we provide a proof that our assumptions hold in the generic k-linear group model, for any $k \geq 1$ and any $d \geq 1$. A formalization of this model (defined as the generic multilinear map model) is detailed in Section 2.4, but only a simple intuition, given below, is necessary for understanding the following.

Intuition. Informally, in the (standard) generic group model, an adversary is only able to perform group operations. That is, given a multiplicative group $\mathbb{G}$ of order p and group elements $[a], [b]$, an adversary can compute elements of the form $[a+b]$ or $[a \cdot \alpha]$ with $\alpha \in \mathbb{Z}_p$, but not $[a \cdot b]$. The generic k-linear group model is a generalization of this model. In this model, the adversary is more powerful as it has access to a k-linear map. That is, it can now compute, given $[a_1], \dots, [a_{k+1}]$, a group element whose discrete logarithm is $a_1 \cdots a_k$, so the product of k of the a_i's, but cannot compute a group element whose discrete logarithm is $a_1 \cdots a_{k+1}$.

Hence, to prove that an assumption is secure in the k-linear group model, it is sufficient to prove that there is no polynomial relation of degree *at most* k between the discrete logarithms of the group elements in the (real or random) distribution of the assumption.

We start by defining supplementary material on commutative algebra in Section 4.4.1, then state and prove our main lemma in Section 4.4.2, and conclude about the security of our assumptions in Section 4.4.3.

4.4.1 Definitions: Monomial Order and Leading Commutative Monomials

Definition 4.4.1 (Monomial order). *Let n be a positive integer. A monomial order for $\mathbb{Z}_p[T_1, \dots, T_n]$ is a total order $<$ such that, for any monomials u, v, w:*

- *if $u < v$, then $uw < vw$,*
- *$1 \leq u$.*

We write $\vec{T}^{\vec{i}} = T_1^{i_1} \cdots T_n^{i_n}$ for $\vec{i} = (i_1, \dots, i_n)$. The *leading monomial* of a polynomial $P(\vec{T}) = \sum_{\vec{i}} \alpha_{\vec{i}} \vec{T}^{\vec{i}}$ is the maximum of the set $\{\vec{T}^{\vec{i}} \mid \alpha_{\vec{i}} \neq 0\}$ for the monomial order $<$, and is denoted $\mathrm{LM}(P)$. The *leading term* of this polynomial P is $\alpha_{\vec{i}^\star} \vec{T}^{\vec{i}^\star}$, when $\mathrm{LM}(P) = \vec{T}^{\vec{i}^\star}$.

We extend this definition to non-commutative polynomials as follows: let

$$\pi\colon \ \mathbb{Z}_p\langle T_1, \dots, T_n\rangle \to \mathbb{Z}_p[T_1, \dots, T_n] \ ,$$

be the (canonical) linear map from the set of non-commutative polynomials $\mathbb{Z}_p\langle T_1, \dots, T_n\rangle$ to the set of (commutative) polynomials $\mathbb{Z}_p[T_1, \dots, T_n]$, defined by $\pi(T_{j_1} \cdots T_{j_k}) = T_{j_1} \cdots T_{j_k}$. The *leading monomials set* of a non-commutative polynomial

$$P(\vec{T}) = \sum_{\substack{k \geq 1 \\ j_1, \dots, j_k \in \{1, \dots, n\}}} \alpha_{j_1, \dots, j_k} T_{j_1} \cdots T_{j_k} \ ,$$

is the set of monomials $T_{j_1}\cdots T_{j_k}$ such that $\pi(T_{j_1}\cdots T_{j_k})$ is the maximum of

$$\{\pi(T_{j_1}\cdots T_{j_k}) \mid \alpha_{j_1,\dots,j_k} \neq 0\} \ .$$

It is denoted $\mathrm{CLM}(P)$. We say a polynomial has *unique* commutative leading monomial if $\mathrm{CLM}(P)$ is a singleton $\{T_{j_1}\cdots T_{j_k}\}$, in which case, we also often write $\mathrm{CLM}(P) = T_{j_1}\cdots T_{j_k}$, to simplify notations.

We remark that if we identify (commutative) polynomials with non-commutative polynomials (by writing them as $P = \sum_{\vec{i}} \alpha_{\vec{i}} \vec{T}^{\vec{i}} = \sum_{\vec{i}} \alpha_{\vec{i}} T_1^{i_1}\cdots T_n^{i_n}$), then polynomials have unique commutative leading monomial.

Example 4.4.2. *For $n = 2$ and $<$ the lexicographic order with $T_1 > T_2$, we have:*

$$\mathrm{LM}(5T_1^2T_2 + T_1T_2^3 + T_2) = T_1^2T_2 \qquad\qquad \mathrm{LM}(T_1^3 + 3T_1T_2^7) = T_1^3$$

for commutative polynomials, and

$$\mathrm{CLM}(5T_1^2T_2 + T_1T_2^3 + T_2) = \{T_1^2T_2\}$$
$$\mathrm{CLM}(5T_1^2T_2 + T_1T_2T_1 + T_2T_1^2 + T_2 + T_1) = \{T_1^2T_2, T_1T_2T_1, T_2T_1^2\}$$

for non-commutative polynomials.

Finally, the *partial degree* of a polynomial P in a set $S \subseteq \{T_1,\dots,T_n\}$ of indeterminates is the degree of the polynomial P seen as a polynomial with indeterminates $(T_i)_{i\in S}$ and coefficients in $\mathbb{Z}_p[(T_i)_{i\notin S}]$.

4.4.2 Main Lemma

Before stating our main lemma, we need to introduce two technical lemmata.

Lemma 4.4.3. *Let k_1, n_1, n_2, q_1 be positive integers. Let $Q_{1,1},\dots,Q_{1,q_1} \in \mathbb{Z}_p[X_{1,1},\dots,X_{1,n_1}]$ be q_1 polynomials. We suppose that, if there exists a polynomial $R \in \mathbb{Z}_p[U_{1,1},\dots,U_{1,q_1}]$ of total degree at most k_1 such that:*

$$R(Q_{1,1},\dots,Q_{1,q_1}) = 0$$

then $R = 0$.

Then, the same is true when R is in $\mathbb{Z}_p[X_{2,1},\dots,X_{2,n_2}][U_{1,1},\dots,U_{1,q_1}]$ instead of being in $\mathbb{Z}_p[U_{1,1},\dots,U_{1,q_1}]$, i.e., when coefficients of R are polynomials in $\mathbb{Z}_p[X_{2,1},\dots,X_{2,n_2}]$ instead of being scalars.

Proof of Lemma 4.4.3. Let us suppose that R is a polynomial in $\mathbb{Z}_p[X_{2,1},\dots,X_{2,n_2}][U_{1,1},\dots,U_{1,q_1}]$ of partial degree at most k_1 in $\{U_{1,1},\dots,U_{1,q_1}\}$, such that

$$R(Q_{1,1},\dots,Q_{1,q_1}) = 0 \ .$$

We want to show that $R = 0$. Seeing R as a polynomial in $\mathbb{Z}_p[U_{1,1},\dots,U_{1,q_1}][X_{2,1},\dots,X_{2,n_2}]$, we can write:

$$R = \sum_{i_1,\dots,i_{n_2}} \lambda_{i_1,\dots,i_{n_2}} \cdot R_{i_1,\dots,i_{n_2}}(U_{1,1},\dots,U_{1,q_1}) \cdot X_{2,1}^{i_{2,1}} \cdots X_{2,n_2}^{i_{2,n_2}}, \tag{4.1}$$

where $\lambda_{i_1,\dots,i_{n_2}} \in \mathbb{Z}_p$ and $R_{1,i_1,\dots,i_{n_2}} \in \mathbb{Z}_p[U_{1,1},\dots,U_{1,q_1}]$. We can then evaluate R on $(Q_{1,1},\dots,Q_{1,q_1})$ and define $R' \in \mathbb{Z}_p[X_{1,1},\dots,X_{1,n_1}][X_{2,1},\dots,X_{2,n_2}]$ as:

$$\begin{aligned} R' &= R(Q_{1,1},\dots,Q_{1,q_1}) \\ &= \sum_{i_1,\dots,i_{n_2}} \lambda_{i_1,\dots,i_{n_2}} \cdot R_{i_1,\dots,i_{n_2}}(Q_{1,1},\dots,Q_{1,q_1}) \cdot X_{2,1}^{i_{2,1}} \cdots X_{2,n_2}^{i_{2,n_2}}. \end{aligned}$$

We also know that $R' = 0$. Since $R_{i_1,\dots,i_{n_2}}(Q_{1,1},\dots,Q_{1,q_1}) \in \mathbb{Z}_p[X_{1,1},\dots,X_{1,n_1}]$, the polynomials

$$\lambda_{i_1,\dots,i_{n_2}} \cdot R_{i_1,\dots,i_{n_2}}(Q_{1,1},\dots,Q_{1,q_1})$$

can be seen as the coefficients of R' (seen as a polynomial over $\mathbb{Z}_p[X_{2,1},\dots,X_{2,n_2}]$), we have:

$$R_{i_1,\dots,i_{n_2}}(Q_{1,1},\dots,Q_{1,q_1}) = 0 \ .$$

As $R_{i_1,\dots,i_{n_2}}$ has degree at most k_1, from the assumption of the lemma, we have $R_{i_1,\dots,i_{n_2}} = 0$. It implies that $R = 0$ (from Equation (4.1)), which concludes the proof of Lemma 4.4.3. □

Lemma 4.4.4. *Let k_1,k_2,n_1,n_2,q_1,q_2 be positive integers. Let $Q_{1,1},\dots,Q_{1,q_1} \in \mathbb{Z}_p[X_{1,1},\dots,X_{1,n_1}]$ and $Q_{2,1},\dots,Q_{2,q_2} \in \mathbb{Z}_p[X_{2,1},\dots,X_{2,n_2}]$ be q_1+q_2 polynomials. We suppose that, if there exist polynomials $R_1 \in \mathbb{Z}_p[U_{1,1},\dots,U_{1,q_1}]$ of total degree at most k_1 and $R_2 \in \mathbb{Z}_p[U_{2,1},\dots,U_{2,q_2}]$ of total degree at most k_2 such that:*

$$R_1(Q_{1,1},\dots,Q_{1,q_1}) = 0 \qquad\qquad R_2(Q_{2,1},\dots,Q_{2,q_2}) = 0 \tag{4.2}$$

then $R_1 = 0$ and $R_2 = 0$. This condition is called Condition $\star$.

Let us suppose there exists a polynomial $R \in \mathbb{Z}_p[U_{1,1},\dots,U_{1,q_1},U_{2,1},\dots,U_{2,q_2}]$ such that any monomial of R is of the form

$$U_{1,1}^{i_{1,1}} \cdots U_{1,q_1}^{i_{1,q_1}} \cdot U_{2,1}^{i_{2,1}} \cdots U_{2,q_2}^{i_{2,q_2}} \qquad \textit{with} \begin{cases} i_{1,1} + \cdots + i_{1,q_1} \le k_1 \\ i_{2,1} + \cdots + i_{2,q_2} \le k_2 \ , \end{cases} ,$$

and such that

$$R(Q_{1,1},\dots,Q_{1,q_1},Q_{2,1},\dots,Q_{2,q_2}) = 0 \ .$$

Then, $R = 0$.

Proof of Lemma 4.4.4. Let us write R as follows:

$$R = \sum_{i_1+\cdots+i_{q_1} \le k_1} \lambda_{i_1,\dots,i_{q_1}} \cdot R_{2,i_1,\dots,i_{q_1}}(U_{2,1},\dots,U_{2,q_2}) \cdot U_{1,1}^{i_1} \cdots U_{1,q_1}^{i_{q_1}} \tag{4.3}$$

where $\lambda_{i_1,\dots,i_{q_1}} \in \mathbb{Z}_p$ and $R_{2,i_1,\dots,i_{q_1}} \in \mathbb{Z}_p[U_{2,1},\dots,U_{2,q_2}]$ of degree at most k_2. Let R' be the polynomial in $\mathbb{Z}_p[X_{2,1},\dots,X_{2,n_2}][U_{1,1},\dots,U_{1,q_1}]$, defined as:

$$\begin{aligned} R' &= R(U_{1,1},\dots,U_{1,q_1},Q_{2,1},\dots,Q_{2,q_2}) \\ &= \sum_{i_1+\cdots+i_{q_1} \le k_1} \lambda_{i_1,\dots,i_{q_1}} \cdot R_{2,i_1,\dots,i_{q_1}}(Q_{2,1},\dots,Q_{2,q_2}) \cdot U_{1,1}^{i_1} \cdots U_{1,q_1}^{i_{q_1}}. \end{aligned}$$

As a polynomial over $\mathbb{Z}_p[X_{2,1},\dots,X_{2,n_2}]$ in $U_{1,1},\dots,U_{1,q_1}$, R' has degree at most k_1 and we have that $R'(Q_{1,1},\dots,Q_{1,q_1}) = 0$. Therefore, $R' = 0$ thanks to Lemma 4.4.3 and the assumption that there is no non-zero polynomial R_1 of degree at most k_1 such that $R_1(Q_{1,1},\dots,Q_{1,q_1}) = 0$. This means that $R_{2,i_1,\dots,i_{q_1}}(Q_{2,1},\dots,Q_{2,q_2}) = 0$ (which is a polynomial in $\mathbb{Z}_p[X_{2,1},\dots,X_{2,n_2}]$), for all $i_1,\dots,i_{q_1}$. Since $R_{2,i_1,\dots,i_{q_1}}$ has degree at most k_2, $R_{2,i_1,\dots,i_{q_1}} = 0$. From Equation (4.3), we get that $R = 0$. □

We can now prove our main lemma, that we later use to prove the security of the $\mathcal{E}_{k,d}$-MDDH assumptions in the generic k-linear group model.

Lemma 4.4.5 (Main Lemma)**.** *Let k, n, m, q be positive integers. We suppose fixed a monomial order $<$ for $\mathbb{Z}_p[T_1, \ldots, T_n]$. Let $(P_s)_{s=1,\ldots,q}$ be a family of polynomials with distinct and unique commutative leading monomial. Let*

$$\mathscr{R} = \mathbb{Z}_p[(X_{\ell,i,j})_{\substack{\ell=1,\ldots,n,\\ i=1,\ldots,k\\ j=1,\ldots,k}}, (Y_{i,j})_{\substack{i=1,\ldots,k\\ j=1,\ldots,m}}] \ .$$

Let us define $\vec{\boldsymbol{A}} \in \left(\mathscr{R}^{k\times k}\right)^n$ a vector of $k \times k$ matrices of (commutative) polynomials with indeterminates $X_{\ell,i,j}$, such that $a_{\ell,i,j} = X_{\ell,i,j}$. Let us also define $\boldsymbol{B} \in \mathscr{R}^{k\times m}$, such that $b_{i,j} = Y_{i,j}$. In other words:

$$\boldsymbol{A}_\ell = \begin{pmatrix} X_{\ell,1,1} & \ldots & X_{\ell,1,k} \\ \vdots & & \vdots \\ X_{\ell,k,1} & \ldots & X_{\ell,k,k} \end{pmatrix} \qquad \boldsymbol{B} = \begin{pmatrix} Y_{1,1} & \ldots & Y_{1,m} \\ \vdots & & \vdots \\ Y_{k,1} & \ldots & Y_{k,m} \end{pmatrix} .$$

Let $Q_{s,i,j} \in \mathscr{R}$ be the polynomial corresponding to the coordinate $(i,j) \in \{1,\ldots,k\}\times\{1,\ldots,m\}$ of the matrix $P_s(\vec{\boldsymbol{A}}) \cdot \boldsymbol{B}$ (for any $s = 1,\ldots,q$).

Finally, let us suppose there exists a polynomial $R \in \mathbb{Z}_p[(U_{s,i,j})_{\substack{s=1,\ldots,q\\ i=1,\ldots,k\\ j=1,\ldots,m}}]$ of total degree at most k, such that

$$R((Q_{s,i,j})_{s,i,j}) = 0. \tag{4.4}$$

Then, necessarily, $R = 0$ (R is the zero polynomial).

Proof of Lemma 4.4.5. Let us assume, without loss of generality that:

$$\mathrm{CLM}(P_1) < \ldots < \mathrm{CLM}(P_q) \ .$$

We do the proof by induction over k.

Base case ($k = 1$). When $k = 1$, $Q_{s,1,1} = P_s((X_{\ell,1,1})_\ell)$ and Equation (4.4) shows there is a linear combination between the P_s's, which is impossible as their leading monomials are distinct (here everything is commutative, as matrices have size 1×1).

Inductive step. We suppose the lemma holds for some value all values lower than k, and prove it for k.

First, let us show that R contains no monomial of the form:

$$U_{s_1,i_1,j_1} \cdots U_{s_{k_1},i_{k_1},j_{k_1}} \cdot U_{s_{k_1+1},i_{k_1+1},j_{k_1+1}} \cdots U_{s_{k_1+k_2},i_{k_1+k_2},j_{k_1+k_2}} \ ,$$

with k_1 and k_2 two positive integers such that $k_1 + k_2 \le k$ and

$$i_1, \ldots, i_{k_1} \in \{1, \ldots, k_1\} \qquad\qquad i_{k_1+1}, \ldots, i_{k_1+k_2} \in \{k_1 + 1, \ldots, k\}.$$

More precisely, let k_1, k_2 be two positive integers such that $k_1 + k_2 \le k$. Let us write R as a sum $R = \tilde{R} + \hat{R}$, with $\tilde{R}$ containing the monomials of the above form, and $\hat{R}$ the other monomials. We want to show that $\tilde{R} = 0$.

Now, in Equation (4.4), we set $X_{\ell,i,j}$ to 0 for all $\ell = 1, \dots, n$ and:

$$(i,j) \notin (\{1,\dots,k_1\} \times \{1,\dots,k_1\}) \cup (\{k_1+1,\dots,k\} \times \{k_1+1,\dots,k\}) \ .$$

Concretely, this means that:

$$\boldsymbol{A}_\ell = \begin{pmatrix} \boldsymbol{A}'_\ell & 0 \\ 0 & \boldsymbol{A}''_\ell \end{pmatrix} \ ,$$

with

$$\boldsymbol{A}'_\ell = \begin{pmatrix} X_{\ell,1,1} & \dots & X_{\ell,1,k_1} \\ \vdots & & \vdots \\ X_{\ell,k_1,1} & \dots & X_{\ell,k_1,k_1} \end{pmatrix} \qquad \boldsymbol{A}''_\ell = \begin{pmatrix} X_{\ell,k_1+1,k_1+1} & \dots & X_{\ell,k_1+1,k} \\ \vdots & & \vdots \\ X_{\ell,k,k_1+1} & \dots & X_{\ell,k,k} \end{pmatrix} .$$

Let us also write

$$\boldsymbol{B} = \begin{pmatrix} \boldsymbol{B}' \\ \boldsymbol{B}'' \end{pmatrix} \ ,$$

with

$$\boldsymbol{B}' = \begin{pmatrix} Y_{1,1} & \dots & Y_{1,m} \\ \vdots & & \vdots \\ Y_{k_1,1} & \dots & Y_{k_1,m} \end{pmatrix} \qquad \boldsymbol{B}' = \begin{pmatrix} Y_{k_1+1,1} & \dots & Y_{k_1+1,m} \\ \vdots & & \vdots \\ Y_{k,1} & \dots & Y_{k,m} \end{pmatrix} \ .$$

Therefore, we have:

$$Q_{s,i,j} = \begin{cases} \text{coefficient } (i,j) \text{ of the matrix } P_s(\boldsymbol{A}') \cdot \boldsymbol{B}' & \text{if } 1 \le i \le k_1 \\ \text{coefficient } (i-k_1,j) \text{ of the matrix } P_s(\boldsymbol{A}'') \cdot \boldsymbol{B}'' & \text{if } k_1+1 \le i \le k \end{cases} \ .$$

Thus, all monomials in $\tilde{R}((Q_{s,i,j})_{s,i,j})$ have partial degree k_1 in $\{Y_{i,j}\}_{\substack{i=1,\dots,k_1 \\ j=1,\dots,m}}$ (coming from the polynomials $Q_{s_1,i_1,j_1}, \dots, Q_{s_{k_1},i_{k_1},j_{k_1}}$) and partial degree k_2 in $\{Y_{i,j}\}_{\substack{i=k_1,\dots,k \\ j=1,\dots,m}}$ (coming from the polynomials $Q_{s_{k_1+1},i_{k_1+1},j_{k_1+1}}, \dots, Q_{s_{k_1+k_2},i_{k_1+k_2},j_{k_1+k_2}}$), while no monomial in $\hat{R}((Q_{s,i,j})_{s,i,j})$ has such partial degrees. Since $R((Q_{s,i,j})_{s,i,j}) = 0$, we have $\tilde{R}((Q_{s,i,j})_{s,i,j}) = 0$.

Now we can apply Lemma 4.4.4, where

$(Q_{1,i})_i$	corresponds to	$(Q_{s,i,j})_{\substack{s=1,\dots,q \\ i=1,\dots,k_1 \\ j=1,\dots,m}}$
$(X_{1,i})_i$	corresponds to	$(X_{i,j})_{\substack{i=1,\dots,k_1 \\ j=1,\dots,k_1}} \cup (Y_{i,j})_{\substack{i=1,\dots,k_1 \\ j=1,\dots,m}}$
$(Q_{2,i})_i$	corresponds to	$(Q_{s,i,j})_{\substack{s=1,\dots,q \\ i=k_1+1,\dots,k \\ j=1,\dots,m}}$
$(X_{2,i})_i$	corresponds to	$(X_{i,j})_{\substack{i=k_1+1,\dots,k \\ j=k_1+1,\dots,k}} \cup (Y_{i,j})_{\substack{i=k_1+1,\dots,k \\ j=1,\dots,m}}$
R	corresponds to	$\tilde{R}$.

Condition $\star$ is satisfied thanks to the induction hypothesis for $k_1 < k$ and $k_2 < k$.

Second, let $1 \le j_1, \dots, j_k \le k$ be positive integers in $\{1,\dots,k\}$. These integers are fixed in all this second step. Let us show that R contains no monomial of the form $U_{s_1,1,j_1} \cdots U_{s_k,1,j_k}$.

Let us write R as a sum $R = \tilde{R} + \hat{R}$, with $\tilde{R}$ containing the monomials of the above form, and $\hat{R}$ the other monomials. We remark that all monomials in $\tilde{R}((Q_{s,i,j})_{s,i,j})$ are multiple of $Y_{i_1,j_1} \cdots Y_{i_k,j_k}$ (for some $i_1,\dots,i_k$), while monomials in $\hat{R}((Q_{s,i,j})_{s,i,j})$ are not. Since $R((Q_{s,i,j})_{s,i,j}) = 0$, $\tilde{R}((Q_{s,i,j})_{s,i,j}) = 0$. We now just need to prove that $\tilde{R} = 0$.

We order monomials of $\mathscr{R}$ using the product order on $\{Y_{i,j}\}_{i,j} \times \{X_{\ell,i,j}\}_{\ell,i,j}$, with the lexicographic order on $\{Y_{i,j}\}$ corresponding to the lexicographic order on (i,j), and with the order on $\{X_{\ell,i,j}\}$ corresponding to the lexicographic order on (i,j,ℓ):

$$Y_{i,j} < Y_{i',j'} \iff \left|\begin{array}{l} i < i' \\ \text{or } i = i' \text{ and } j < j' \end{array}\right.$$

$$X_{\ell,i,j} < X_{\ell',i',j'} \iff \left|\begin{array}{l} i < i' \\ \text{or } i = i' \text{ and } j < j' \\ \text{or } (i,j) = (i',j') \text{ and } \ell < \ell' \end{array}\right.$$

Now, we set $X_{\ell,i,j}$ to 0 for all $\ell = 1,\dots,n$ and $i \neq j$ and $i \neq 1$. We also set $X_{\ell,1,i} = X_{\ell,i,i}$. For the sake of simplicity, in this step, we write $X_{\ell,1,i} = X_{\ell,i,i} = X_{\ell,i}$, and $\overrightarrow{X}_i = (X_{1,i},\dots,X_{n,i})$. Concretely, this means that:

$$\boldsymbol{A}_\ell = \begin{pmatrix} X_{\ell,1} & X_{\ell,2} & X_{\ell,3} & \dots & X_{\ell,k} \\ 0 & X_{\ell,2} & 0 & \dots & 0 \\ 0 & 0 & X_{\ell,3} & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & X_{\ell,k} \end{pmatrix} .$$

Then, we get (easily by induction):

$$P_s(\overrightarrow{\boldsymbol{A}}) = \begin{pmatrix} P_s(\overrightarrow{X}_1) & \mathrm{LT}(P_s(\overrightarrow{X}_2)) + \dots & \mathrm{LT}(P_s(\overrightarrow{X}_3)) + \dots & \dots & \mathrm{LT}(P_s(\overrightarrow{X}_k)) + \dots \\ 0 & P_s(\overrightarrow{X}_2) & 0 & \dots & 0 \\ 0 & 0 & P_s(\overrightarrow{X}_3) & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & P_s(\overrightarrow{X}_k) \end{pmatrix},$$

where $\mathrm{LT}(P_s(\overrightarrow{X}_i)) + \dots$ corresponds to a polynomial with leading term $\mathrm{LT}(P_s(\overrightarrow{X}_i))$. Thus, we have:

$$Q_{s,1,j} = P_s(\overrightarrow{X}_1) \cdot Y_{1,j} + (\mathrm{LT}(P_s(\overrightarrow{X}_2)) + \dots) \cdot Y_{2,j} + \dots + (\mathrm{LT}(P_s(\overrightarrow{X}_k)) + \dots) Y_{k,j} \quad (4.5)$$

Let us now suppose by contradiction that $\tilde{R} \neq 0$. Let $U_{s_1,1,j_1} \cdots U_{s_k,1,j_k}$ be the monomial of $\tilde{R}$, for which the tuple $(s_{\sigma(k)},\dots,s_{\sigma(1)})$ is the highest for the lexicographic order, where σ is a permutation of $\{1,\dots,k\}$ such that $s_{\sigma(k)} \geq \dots \geq s_{\sigma(1)}$. When $\tilde{R}$ is evaluated on $(Q_{s,i,j})_{s,i,j}$ this monomial corresponds to $Q_{s_1,1,j_1} \cdots Q_{s_k,1,j_k}$. From Equation (4.5), we get that the latter expression contains the following monomial:

$$M = P_{s_{\sigma(1)}}(\overrightarrow{X}_1) \cdot Y_{1,j_{\sigma(1)}} \cdot \mathrm{LM}(P_{s_{\sigma(2)}}(\overrightarrow{X}_2)) \cdot Y_{2,j_{\sigma(2)}} \cdots \mathrm{LM}(P_{s_{\sigma(k)}}(\overrightarrow{X}_k)) \cdot Y_{k,j_{\sigma(k)}} .$$

We just need to prove that this monomial M does not appear in any other polynomial $Q_{s'_1,1,j_1} \cdots Q_{s'_k,1,j_k}$, with $(s'_{\sigma'(k)}, \dots, s'_{\sigma'(1)})$ lower or equal to $(s_{\sigma(k)}, \dots, s_{\sigma(1)})$ for the lexicographic order (we write it $(s'_{\sigma'(k)}, \dots, s'_{\sigma'(1)}) \preceq (s_{\sigma(k)}, \dots, s_{\sigma(1)})$) and $U_{s'_1,1,j'_k} \cdots U_{s'_k,1,j'_k} \neq U_{s_1,1,j_1} \cdots U_{s_k,1,j_k}$, where σ' is defined similarly to σ. This will implies that $\tilde{R}((Q_{s,i,j})_{s,i,j}) \neq 0$ (as it contains the above monomial M which does not get canceled out by other terms), which is impossible.

Let us suppose that $Q_{s'_1,1,j_1} \cdots Q_{s'_k,1,j_k}$ contains the monomial M, with $(s'_{\sigma'(k)}, \dots, s'_{\sigma'(1)}) \preceq (s_{\sigma(k)}, \dots, s_{\sigma(1)})$. We first remark that none of the terms in the "left" part of $Q_{s,i,j}$:

$$P_s(\overrightarrow{X}_1) \cdot Y_{1,j} + (\mathrm{LT}(P_s(\overrightarrow{X}_2)) + \dots) \cdot Y_{2,j} + \dots + (\mathrm{LT}(P_s(\overrightarrow{X}_{k-1})) + \dots) Y_{k-1,j}$$

contain monomials multiple of $X_{\ell,i,j}$ for $i \leq k-1$ (from the definition of the monomial order on $\mathscr{R}$). The monomial $\mathrm{LM}(P_{s_{\sigma(k)}}(\overrightarrow{X}_k)) \cdot Y_{k,j_{\sigma(k)}}$ divides the monomial M and can only come from the "right" part of one $Q_{s'_r,1,j_r}$, because of $Y_{k-1,j}$ which is only present one time in the monomial M. As in addition, $s'_{\sigma'(1)} \leq \dots \leq s'_{\sigma'(k)} \leq s_{\sigma(k)}$ (because $(s'_{\sigma'(k)}, \dots, s'_{\sigma'(1)}) \preceq (s_{\sigma(k)}, \dots, s_{\sigma(1)})$), we get that $s_{\sigma'(k)} = s_{\sigma(k)}$ and $j_{\sigma'(k)} = j_{\sigma(k)}$. We can continue like that by induction and prove that $s'_{\sigma'(k-1)} = s_{\sigma(k-1)}$, $j_{\sigma'(k-1)} = j_{\sigma(k-1)}$, $\dots$, $s'_{\sigma'(1)} = s_{\sigma(1)}$, and $j_{\sigma'(1)} = j_{\sigma(1)}$. We finally get that $U_{s'_1,1,j'_k} \cdots U_{s'_k,1,j'_k} \neq U_{s_1,1,j_1} \cdots U_{s_k,1,j_k}$. That was we wanted to prove.

Third, let us conclude by showing all the other cases come down to the first two cases after performing some permutation. More precisely, let σ be a permutation of $\{1, \dots, k\}$. Let $\Sigma \in \mathbb{Z}_p^{k \times k}$ be the corresponding permutation matrix: $\Sigma_{i,j} = 1$ if and only if $\sigma(j) = i$, and $\Sigma_{i,j} = 0$ otherwise. We also set:

$$\begin{aligned} a'_{\ell,i,j} &= X'_{\ell,i,j} = X_{\ell,\sigma(i),\sigma(j)} \\ b'_{i,j} &= Y'_{i,j} = Y'_{\sigma(i),j} \end{aligned}$$

so that:

$$\begin{aligned} \boldsymbol{A}' &= \Sigma^{-1} \cdot \boldsymbol{A} \cdot \Sigma \\ \boldsymbol{B}' &= \Sigma^{-1} \cdot \boldsymbol{B} \\ P_s(\boldsymbol{A}) \cdot \boldsymbol{B} &= \Sigma \cdot P_s(\boldsymbol{A}') \cdot \boldsymbol{B}' \\ Q_{s,i,j} &= Q_{s,\sigma(i),j}((X_{\ell,i,j} \leftarrow X'_{\ell,i,j})_{\ell,i,j}, (Y_{i,j} \leftarrow Y'_{i,j})_{i,j}) \end{aligned}$$

Let R' be the polynomial R where $U_{s,i,j}$ is replaced by $U_{s,\sigma(i),j}$. We have:

$$\begin{aligned} R((Q_{s,i,j})) &= R((Q_{s,\sigma(i),j}((X_{\ell,i,j} \leftarrow X'_{\ell,i,j})_{\ell,i,j}, (Y_{i,j} \leftarrow Y'_{i,j})_{i,j}))_{s,i,j}) \\ &= R'((Q_{s,i,j})_{s,i,j})((X_{\ell,i,j} \leftarrow X'_{\ell,i,j})_{\ell,i,j}, (Y_{i,j} \leftarrow Y'_{i,j})_{i,j}). \end{aligned}$$

Therefore, as $R((Q_{s,i,j})_{s,i,j}) = 0$, we have $R'((Q_{s,i,j})_{s,i,j}) = 0$. And it is clear that $R = 0$ if and only if $R' = 0$.

Let us now show that R contain no monomial $M = U_{s_1,i_1,j_1} \cdots U_{s_{k'},i_{k'},j_{k'}}$, with $k' \leq k$. This will prove that $R = 0$. To do that, let us suppose by contradiction that R contains such monomial M. We consider two cases:

- if $k = k'$ and $i_1 = \cdots = i_k$, then choose σ to be an arbitrary permutation such that $\sigma(i_1) = 1$. We know that the corresponding polynomial R' then contains the monomial $M' = U_{s_1,1,j_1} \cdots U_{s_k,k,j_k}$. But that is impossible according to the second step, as $R'((Q_{s,i,j})_{s,i,j}) = 0$.
- otherwise, let $k_1 = |\{p | i_p = i_1\}|$ be the number of "i" indices equal to i_1. We know that $k_1 < k$. Let $k_2 = k' - k_1$ and let (I_1, I_2) be a partition of $\{1, \ldots, k\}$ such that $|I_1| = k_1$, $|I_2| = k - k_1 \geq k_2$, $i_1 \in I_1$, and for all $i_p \neq i_1$, $i_p \in I_2$. Such a partition exist because there are only k_2 values $i_p \neq i_1$. Then, let σ be an arbitrary permutation such that $\sigma(I_1) = \{1, \ldots, k_1\}$ and $\sigma(I_2) = \{k_1 + 1, \ldots, k\}$. Finally, we remark that the corresponding polynomial R' contains the monomial

  $$U_{s_1,i'_1,j_1} \cdots U_{s_{k_1},i'_{k_1},j_{k_1}} \cdot U_{s_{k_1+1},i'_{k_1+1},j_{k_1+1}} \cdots U_{s_{k_1+k_2},i'_{k_1+k_2},j_{k_1+k_2}} \ ,$$

  such that:

  $$i'_1, \ldots, i'_{k_1} \in \{1, \ldots, k_1\} \qquad\qquad i'_{k_1+1}, \ldots, i'_{k_1+k_2} \in \{k_1 + 1, \ldots, k\},$$

  where $i'_p = \sigma(i_p)$. But the first step of our proof show it is impossible.

This concludes the proof. $\square$

4.4.3 Putting Everything Together

Similarly to the proof of Theorem 3 of [EHK+13] and the proof for uber assumptions [BBG05; Boy08], to prove the security of the $\mathcal{E}_{k,d}$-MDDH assumption in generic symmetric k-linear groups (in which there would exist a symmetric k-linear map), we just need to show that there is no (non-trivial) polynomial relation of degree k between entries of $\mathbf{\Gamma}$ and $\boldsymbol{Z}$, both when $\boldsymbol{Z} = \mathbf{\Gamma} \cdot \boldsymbol{W}$ and when $\boldsymbol{Z} = \boldsymbol{U}$, with

$$\mathbf{\Gamma} = \begin{pmatrix} \boldsymbol{B} \\ \boldsymbol{A} \cdot \boldsymbol{B} \\ \vdots \\ \boldsymbol{A}^d \cdot \boldsymbol{B} \end{pmatrix}.$$

Indeterminates are entries of $\boldsymbol{A}$ and $\boldsymbol{B}$ ($a_{i,j}$, $b_{i,j}$, for $i = 1, \ldots, k$, $j = 1, \ldots, k$), entries of $\boldsymbol{W}$ (w_i, for $i = 1, \ldots, k$), and entries of $\boldsymbol{U}$ ($u_{i,j}$, for $i = 1, \ldots, k(d + 1)$, $j = 1, \ldots, k$). The polynomial independence follows from Lemma 4.4.5, with $n = 1$, $q = d + 1$, and $P_s = T_1^{s-1}$, for $s = 1, \ldots, d + 1$.

Chapter 5

An Algebraic Framework for Pseudorandomness

In this chapter, we introduce one of the main results of this manuscript. We describe a simple, algebraic framework, that translates the pseudorandomness property of a function into a simple algebraic characterization.

We start by giving a brief intuition of our notion and by describing a few subtleties in the formal definition. Then, we formally define our new security notion, termed polynomial linear pseudorandomness security (PLP) and later state our main result, that relates this new notion to the $\mathcal{E}_{k,d}$-MDDH assumption.

The full proof of the latter statement is given in Section 5.3 and is quite technical. However, our result is only applied in a black-box manner throughout the rest of this manuscript, thus understanding the proof is not necessary for reading the rest of this work.

Yet, we make extensive use of Theorem 5.2.2 to argue about the security of pseudorandom functions, related-key secure pseudorandom functions, aggregate pseudorandom functions, as well as multilinear pseudorandom functions in the next chapter and thus encourage the reader to inspect carefully Sections 5.1 and 5.2.

Contents

5.1 Intuition and Subtleties

We start by giving some intuition about our security notion and by arguing about some subtleties before providing the formal definition in Section 5.2.

5.1.1 Intuition

Let us consider a group $\mathbb{G} = \langle g \rangle$ of order p. Intuitively, the polynomial linear pseudorandomness security notion says that for any polynomials $P_1, \ldots, P_q \in \mathbb{Z}_p[T_1, \ldots, T_n]$, the group elements

$$[P_1(\vec{a}) \cdot b], \ldots, [P_q(\vec{a}) \cdot b] \quad ,$$

with $\vec{a} \xleftarrow{\$} \mathbb{Z}_p^n$ and $b \xleftarrow{\$} \mathbb{Z}_p$, are computationally indistinguishable from the group elements:

$$[U(P_1)], \ldots, [U(P_q)] \quad ,$$

with $U \xleftarrow{\$} \mathsf{L}(\mathbb{Z}_p[T_1, \ldots, T_n]_{\leq d}, \mathbb{Z}_p)$ being a random linear function from the vector space $\mathbb{Z}_p[T_1, \ldots, T_n]_{\leq d}$ (with d the maximum degree of $P_1, \ldots, P_q$ in any indeterminate T_i) to the base field $\mathbb{Z}_p$. Our main theorem (Theorem 5.2.2) shows that this security notion holds under the $\mathcal{E}_{1,d}$-MDDH assumption (and thus also under DDH for $d = 1$ and d-DDHI for $d \geq 2$).

In particular, when $P_1, \ldots, P_q$ are linearly independent, $[U(P_1)], \ldots, [U(P_q)]$ are uniformly random and independent group elements in $\mathbb{G}$. Therefore, under the polynomial linear pseudorandomness security, any efficient function that produces on input $x \in \mathcal{D}$ an output of the form $[P_x(\vec{a}) \cdot b]$, with $\{P_x\}_{x \in \mathcal{D}}$ being a family of linearly independent polynomials, is a pseudorandom function.

We remark that, in the generic group model, the polynomial linear pseudorandomness security notion holds trivially, by definition. The difficulty of the work is to prove it under classical assumptions such as the $\mathcal{E}_{1,d}$-MDDH assumption.

5.1.2 Procedure for Testing Linear Dependence

When we want to formally define the polynomial linear pseudorandomness security notion, we quickly face a problem:

How to compute $[U(P_i)]$ for a random linear map $U \xleftarrow{\$} \mathsf{L}(\mathbb{Z}_p[T_1, \ldots, T_n]_{\leq d}, \mathbb{Z}_p)$?

Such a map can be represented by a (random) vector with $(d+1)^n$ entries. But doing so would make the game in the security notion exponential time. As already done earlier in this manuscript, the idea is to define or draw U lazily: each time we need to evaluate it on a polynomial P_i linearly independent of all the previous polynomials P_j (with $j < i$), we define $U(P_i) \xleftarrow{\$} \mathbb{Z}_p$; otherwise, we compute $U(P_i)$ as a linear combination of $U(P_j)$. More precisely, if $P_i = \sum_{j=1}^{i-1} \lambda_j \cdot P_j$, $U(P_i) = \sum_{j=1}^{i-1} \lambda_j \cdot U(P_j)$.

Then, for this simulation to be efficient, we need an efficient procedure for testing such linear dependencies. We denote by $\mathsf{TestLin}$ a procedure which takes as inputs a list $\mathcal{L}$ of polynomials $(R_1, \ldots, R_L)$ (such that $R_1, \ldots, R_L$ are linearly independent as polynomials) and a polynomial R and which outputs:

$$\begin{cases} \perp & \text{if } R \text{ is linearly independent of the set } \{R_1, \ldots, R_L\} \\ \vec{\lambda} = (\lambda_1, \ldots, \lambda_L) & \text{otherwise, so that } R = \lambda_1 R_1 + \ldots + \lambda_L R_L \end{cases} .$$

$\overrightarrow{\lambda}$ is uniquely defined since we assume that polynomials from the input list are linearly independent. However, we are facing a problem since no such procedure is known for multivariate polynomials, if we require the procedure to be deterministic and polynomial-time. Luckily, it is easy to construct such a randomized procedure which is correct with overwhelming probability, and such a statistical procedure is sufficient for our purpose. We describe such a procedure in Figure 5.1.

```
TestLin($\mathcal{L}, R$)
// $\mathcal{L}[\ell] = R_\ell$ for $\ell = 1, \dots, L$ and $L = |\mathcal{L}|$
$R_{L+1} \leftarrow R$
$N \leftarrow 2L + 4$
For $k = 1, \dots, N$
    $\overrightarrow{\gamma_k} \xleftarrow{\$} \mathbb{Z}_p^n$
$\boldsymbol{M}$ matrix over $\mathbb{Z}_p$ of $L + 1$ rows and $N$ columns
For $\ell = 1, \dots, L + 1$
    For $k = 1, \dots, N$
        $m_{\ell,k} \leftarrow R_\ell(\overrightarrow{\gamma_k})$
Apply Gaussian elimination on $\boldsymbol{M}$
If $\boldsymbol{M}$ is full-rank then
    Return $\perp$
Else
    Let $\overrightarrow{\lambda'}$ be the row vector such that $\overrightarrow{\lambda'} \cdot \boldsymbol{M} = \overrightarrow{0}$
    $\overrightarrow{\lambda} \leftarrow (\lambda'_1/\lambda'_{L+1}, \dots, \lambda'_L/\lambda'_{L+1})$
    Return $\overrightarrow{\lambda}$
```

Figure 5.1: TestLin procedure

Lemma 5.1.1. *The procedure* TestLin *given in Figure 5.1 is correct with probability at least* $\frac{p-1}{p}$ *as soon as* $nd \leq \sqrt{p}$*, where* d *is the maximum degree in one indeterminate and* n *is the number of indeterminates.*

Proof of Lemma 5.1.1. Let us prove that this approximate procedure is incorrect with probability at most $\frac{1}{p}$ (over its random coins). The polynomials $P_{\overrightarrow{\phi}_l, x_l, j}$ with $l = 1, \dots, L$ are supposed to be linearly independent. Then, there are two cases:

1. If $P_{\overrightarrow{\phi}, x, j} = P_{\overrightarrow{\phi}_{L+1}, x_{L+1}, j}$ is linearly independent from $P_{\overrightarrow{\phi}_1, x_1, j}, \dots, P_{\overrightarrow{\phi}_L, x_L, j}$, then the probability that the procedure does not return $\perp$ is (over the value of X):

$$\begin{aligned}\Pr\left[\exists \overrightarrow{\lambda} \in \mathbb{Z}_p^{(L+1)},\ \overrightarrow{\lambda} \cdot \boldsymbol{M} = 0\right] &\leq \sum_{\overrightarrow{\lambda} \in \mathbb{Z}_p^{(L+1)}} \Pr\left[\overrightarrow{\lambda} \cdot \boldsymbol{M} = 0\right] \\ &\leq \sum_{\overrightarrow{\lambda} \in \mathbb{Z}_p^{(L+1)}} \Pr\left[\forall k = 1, \dots, N,\ (\sum_{l=1}^{L+1} \lambda_l P_{\overrightarrow{\phi}_l, x_l, j})(\gamma_k) = 0\right]\end{aligned}$$

and $\sum_{l=1}^{L+1} \lambda_l P_{\overrightarrow{\phi}_l, x_l, j}$ is a non-zero polynomial of degree at most jd. Since γ_k are chosen independently and uniformly at random in $\mathbb{Z}_p^n$, according to the Schwartz-Zippel lemma, the error probability is at most:

$$\sum_{\vec{\lambda} \in \mathbb{Z}_p^{(L+1)}} \left(\frac{jd}{p}\right)^N = p^{L+1} \cdot \left(\frac{jd}{p}\right)^N \leq p^{L+1} \cdot \frac{1}{p^{L+2}} = \frac{1}{p} ,$$

since $jd \leq nd \leq \sqrt{p}$ and $N = 2L + 4$.

2. If $P_{\vec{\phi},x,j} = P_{\vec{\phi}_{l+1},x_{L+1},j}$ is not linearly independent from $P_{\vec{\phi}_1,x_1,j}, \ldots, P_{\vec{\phi}_L,x_L,j}$, then there exists $\vec{\lambda} \in \mathbb{Z}_p^L$ such that $P_{\vec{\phi},x,j} = \sum_{l=1}^L \lambda_l P_{\vec{\phi}_l,x_l,j}$, and such $\vec{\lambda}$ is unique. Let us prove that the probability that the TestLin procedure does not return $\vec{\lambda}$ is at most $\frac{1}{p}$. Let Λ be the set of $\vec{\lambda'} \in \mathbb{Z}_p^{L+1}$ such that $\lambda'_{L+1} \cdot \vec{\lambda} \neq \vec{\lambda'}_{1,\ldots,L}$. Then the error probability of the TestLin procedure is at most:

$$\Pr\left[\exists \vec{\lambda'} \in \Lambda,\, \vec{\lambda'} \cdot \boldsymbol{M} = 0\right] \leq \sum_{\vec{\lambda'} \in \Lambda} \Pr\left[\forall k = 1, \ldots, N,\, (\sum_{l=1}^{L+1} \lambda'_l P_{\vec{\phi}_l,x_l,j})(\gamma_k) = 0\right] .$$

Moreover, $\sum_{l=1}^{L+1} \lambda'_l P_{\vec{\phi}_l,x_l,j}$ is a polynomial of degree at most jd, which is non-zero because otherwise the $P_{\vec{\phi}_1,x_1,j}, \ldots, P_{\vec{\phi}_L,x_L,j}$ would not be independent. We can conclude the proof as in the first case, since $|\Lambda| \leq |\mathbb{Z}_p^{(L+1)}|$.

This concludes the proof of Lemma 5.1.1. □

5.1.3 Extension to Weaker Assumptions

Before, arguing the last subtlety and showing the formal definition and theorem, let us briefly introduce an extension of our polynomial linear pseudorandomness security notion to handle weaker assumptions, namely $\mathcal{E}_{k,d}$-MDDH, with $k \geq 2$. In that case, we need to evaluate polynomials on matrices: $[P_i(\boldsymbol{A}) \cdot \boldsymbol{B}]$, with $\boldsymbol{A} \xleftarrow{\$} \mathbb{Z}_p^{k \times k}$ and $\boldsymbol{B} \xleftarrow{\$} \mathbb{Z}_p^{k \times m}$ (with $m \geq 1$ being a positive integer).

As multiplication of matrices is not commutative, it is clear we need to make some restrictions on the form of polynomials queried. This is actually already the case in the restricted case where $k = 1$, for different reasons. A precise analysis of our restrictions is thus given in the next subsection.

5.1.4 On the Representation of Multivariate Polynomials

A last challenge is to define how the polynomials are represented. Indeed, an important but subtle point of our work is that we do not need polynomials to be given in expanded form in the polynomial linear pseudorandomness security notion. Otherwise, the theorem would be quite easy to prove but would not encompass many interesting cases. Specifically, it would restrict us to polynomials with a polynomial number of monomials and forbid polynomials such as $\prod_{i=1}^n (a_i + 1)$, for instance.

However, some representations of polynomials will even not enable us to define properly the PLP game (Figure 5.2). It is indeed at the very least necessary to be able to evaluate the polynomial at arbitrary points (which may be scalars in $\mathbb{Z}_p$ when $k = 1$, or matrices in $\mathbb{Z}_p^{k \times k}$ when $k \geq 2$). For example, it would be inconceivable to define a polynomial P by an RSA modulus N, as the polynomial $P = (X - p_1)(X - p_2)$, with p_1 and p_2 the two prime factors of N.

For this subsection only, let us write $\tilde{P}$ the representation of the polynomial, while P is the mathematical polynomial object. The same polynomial may have many representations. We always suppose that $\tilde{P}$ has a polynomial size in n and d. This assumption is reasonable and simplifies the bounds, but is not required (bounds in theorems would then need to be changed).

5.1.4.1 The Scalar Case

In the case $k = 1$, we could actually just require the following condition.

Condition 1. *It is possible to get from $\tilde{P}$ (in polynomial time):*

full evaluation *the value $P(a_1, \ldots, a_n) \in \mathbb{Z}_p$, given $a_1, \ldots, a_n \in \mathbb{Z}_p$;*

partial evaluation *for any $j = 0, \ldots, n$, a representation $\tilde{Q}$ of the j-variate polynomial $Q = P(T_1, \ldots, T_j, a_{j+1}, \ldots, a_n)$, given $a_{j+1}, \ldots, a_n \in \mathbb{Z}_p$. This representation $\tilde{Q}$ has again to verify (recursively) Condition 1.*

In all cases in our work, actually, $\tilde{P}$ is just an expression or an abstract syntax tree (AST) where internal nodes are either $+$ or $\cdot$, while leaves are either an indeterminate T_i or a scalar in $\mathbb{Z}_p$. A partial evaluation can be performed by replacing T_i by a_i (when $i > j$) in the AST, while a full evaluation can be performed by evaluating the AST (after the previous replacement, with $j = 0$). Both operations are polynomial-time in the size of the AST.

5.1.4.2 The Matrix Case

When $k > 1$, everything is more contrived because of the absence of commutativity. Intuitively, we want that all the indeterminates always appear in the same order. Without loss of generality, T_n appears before T_{n-1}, T_{n-1} before T_{n-2}, ... This has to hold not only in the polynomial as a mathematical object, but somehow also "in the representation" if we want to be able to prove something. More precisely, we do not want that, at some point, when evaluating the polynomial, we have to compute $\boldsymbol{A}_j\boldsymbol{A}_i$ with $i > j$, even if this expression does not appear in the resulting polynomial. For example, the representation $T_2T_3 - T_2(T_3 + T_1)$ is not acceptable (because of the presence of T_2T_3), while the representation T_2T_1 (which corresponds to the same polynomial) is acceptable.

More formally, we assume the representation of the polynomials satisfies the following condition.

Condition 2. *The representation of a polynomial is an expression or AST (where internal nodes are either $+$ or $\cdot$, while leaves are either an indeterminate T_i or a scalar in $\mathbb{Z}_p$) with the following additional (natural) property (to deal with non-commutativity): if $\tilde{P}_1 \cdot \tilde{P}_2$ is a sub-expression of $\tilde{P}$, and if T_j is a leaf of $\tilde{P}_1$ for some j, then for any $i > j$, T_i is not a leaf of $\tilde{P}_2$.*

We remark that, because of this condition, even if the polynomials we consider would normally be non-commutative, we can as well view them as commutative polynomials (when we evaluate a polynomial from a mathematical point of view, we perform multiplications of the indeterminates in the right order). We also remark that, when $k = 1$, Condition 2 is stronger than Condition 1.

5.2 Formal Security Notion and Main Result

Here, we define formally the polynomial linear pseudorandomness security and state our main result in Theorem 5.2.2. Once again, we consider a cyclic group $\mathbb{G} = \langle g \rangle$ of order p.

5.2.1 Formal Definition of the Polynomial Linear Pseudorandomness Security

Definition 5.2.1 (Polynomial Linear Pseudorandomness Security). *The advantage of an adversary $\mathscr{D}$ against the (n,d,k,m)-PLP security of $\mathbb{G}$ is defined as:*

$$\mathbf{Adv}_{\mathbb{G}}^{(n,d,k,m)\text{-plp}}(\mathscr{D}) := \Pr\left[(n,d,k,m)\text{-PLPReal}_{\mathbb{G}}^{\mathscr{D}} \Rightarrow 1\right] - \Pr\left[(n,d,k,m)\text{-PLPRand}_{\mathbb{G}}^{\mathscr{D}} \Rightarrow 1\right] \ ,$$

with $\mathscr{D}$ being restricted to make queries $P \in \mathbb{Z}_p[T_1, \dots, T_n]_{\leq d}$ that satisfy Condition 1 (when $k = 1$) or Condition 2 (when $k \geq 2$), with the probability being over the choice of $\overrightarrow{\boldsymbol{A}}, \boldsymbol{B}$, and the random coins used by the adversary, and where games (n,d,k,m)-PLPReal$_{\mathbb{G}}$ and (n,d,k,m)-PLPRand$_{\mathbb{G}}$ are defined in Figure 5.2.

When m is not specified, it is implicitly equal to 1. Note that when $k = m = 1$, we get exactly the intuitive security notion defined in Section 5.1.1, as in that case $\overrightarrow{\boldsymbol{A}} = \vec{a} \in \mathbb{Z}_p^n$ and $\boldsymbol{B} = b \in \mathbb{Z}_p$.

```
(n, d, k, m)-PLPReal_G
proc Initialize
    $\overrightarrow{\boldsymbol{A}} \xleftarrow{\$} (\mathbb{Z}_p^{k\times k})^n$
    $\boldsymbol{B} \xleftarrow{\$} \mathbb{Z}_p^{k\times m}$
proc Pl($P$)
    Return $\left[P(\overrightarrow{\boldsymbol{A}}) \cdot \boldsymbol{B}\right]$
proc Finalize($b$)
    Return $b$
```

```
(n, d, k, m)-PLPRand_G
proc Initialize
    $\mathcal{L}_1 \leftarrow$ empty list
    $\mathcal{L}_2 \leftarrow$ empty list
    $L \leftarrow 0$
proc Pl($P$)
    $\vec{\lambda} \leftarrow \mathsf{TestLin}(\mathcal{L}_1, P)$
    If $\vec{\lambda} = \perp$ then
        $\boldsymbol{Y} \xleftarrow{\$} \mathbb{Z}_p^{k\times m}$
        $L \leftarrow L + 1$
        $\mathcal{L}_1[L] \leftarrow P$
        $\mathcal{L}_2[L] \leftarrow \boldsymbol{Y}$
    Else
        $\boldsymbol{Y} \leftarrow \sum_{i=1}^{L} \lambda_i \cdot \mathcal{L}_2[i]$
    Return $[\boldsymbol{Y}]$
proc Finalize($b$)
    Return $b$
```

Figure 5.2: Security games for (n,d,k,m)-PLP in a group $\mathbb{G}$

Then, our main result states that the (n,d,k,m)-PLP security holds in $\mathbb{G}$ assuming the hardness of the $\mathcal{E}_{k,d}$-MDDH problem in $\mathbb{G}$, as detailed below.

5.2.2 The PLP Theorem

Theorem 5.2.2 (PLP). *Assuming $\mathcal{E}_{k,d}$-MDDH holds in $\mathbb{G}$, so does the (n,d,k,m)-PLP security. Moreover, the running time of this reduction is polynomial (in n, d, k, m, and the* *running time of the adversary).*

Proof Overview. As we directly prove the general result, the notation makes to proof slightly hard to read. However, the main idea is rather simple. Basically, we define hybrid games in which we replace partial evaluations in $\boldsymbol{A}_{i+1}, \dots, \boldsymbol{A}_n$ of the polynomials queried by uniformly random values (still preserving the linear relations) to obtain a polynomial in i indeterminates $T_1, \dots, T_i$. Then, under the $\mathcal{E}_{k,d}$-MDDH assumption, we can replace elements $\boldsymbol{A}_i^k \cdot \boldsymbol{B} \cdot \boldsymbol{W}$ to uniformly random values, which let us go to the next hybrid game. At the end, we have that the two games defining the (n, d, k, m)-PLP security are indistinguishable under the $\mathcal{E}_{k,d}$-MDDH. A detailed proof is provided in Section 5.3.

5.2.3 Immediate Corollary: the LIP Theorem

Before proving the PLP theorem, we introduce the following simple corollary that we use extensively in the next chapter. We term this corollary the LIP theorem, where LIP stands for *Linearly Independent Polynomial.* This is just a simplification of the PLP theorem in the case where *we further restrict the adversary to query only linearly independent polynomials.* In particular, under these assumptions, one can simplify the PLP security notion by defining the following LIP security notion as follows.

Definition 5.2.3 (Linearly Independent Polynomial Security)**.** *The advantage of an adversary $\mathscr{D}$ against the (n, d, k, m)-LIP security of $\mathbb{G}$ security of $\mathbb{G}$ is defined as:*

$$\mathbf{Adv}_{\mathbb{G}}^{(n,d,k,m)\text{-lip}}(\mathscr{D}) := \Pr\left[(n,d,k,m)\text{-}\mathsf{LIPReal}_{\mathbb{G}}^{\mathscr{D}} \Rightarrow 1\right] - \Pr\left[(n,d,k,m)\text{-}\mathsf{LIPRand}_{\mathbb{G}}^{\mathscr{D}} \Rightarrow 1\right] ,$$

with $\mathscr{D}$ being restricted to make queries linearly independent polynomials $P \in \mathbb{Z}_p[T_1, \dots, T_n]_{\leq d}$ that satisfy Condition 1 (when $k = 1$) or Condition 2 (when $k \geq 2$), with the probability being over the choice of $\vec{\boldsymbol{A}}, \boldsymbol{B}$, and the random coins used by the adversary, and where games (n, d, k, m)-$\mathsf{LIPReal}_{\mathbb{G}}$ and (n, d, k, m)-$\mathsf{LIPRand}_{\mathbb{G}}$ are defined in Figure 5.3.

When k or m are not specified, they are implicitly equal to 1.

(n,d,k,m)-$\mathsf{LIPReal}_{\mathbb{G}}$	(n,d,k,m)-$\mathsf{LIPRand}_{\mathbb{G}}$
proc Initialize	**proc Pl**(P)
$\vec{\boldsymbol{A}} \xleftarrow{\$} (\mathbb{Z}_p^{k\times k})^n$	$\boldsymbol{Y} \xleftarrow{\$} \mathbb{Z}_p^{k\times m}$
$\boldsymbol{B} \xleftarrow{\$} \mathbb{Z}_p^{k\times m}$	Return $[\boldsymbol{Y}]$
proc Pl(P)	**proc Finalize**(b)
Return $\left[P(\vec{\boldsymbol{A}}) \cdot \boldsymbol{B}\right]$	Return b
proc Finalize(b)	
Return b	

Figure 5.3: Security games for (n, d, k, m)-LIP in a group $\mathbb{G}$

Theorem 5.2.4 (LIP)**.** *Assuming $\mathcal{E}_{1,d}$-MDDH holds in $\mathbb{G}$, so does the (n, d)-LIP security. Moreover, the running time of this reduction is polynomial (in n, d, and the running time of the adversary).*

5.3 Proof of Theorem 5.2.2

In order to prove the PLP theorem, we will need to able to run some decomposition algorithms on the polynomials queried. The following subsection provides a few lemmata that detail the types of decomposition we require in our proof.

5.3.1 Decomposition Lemmata

Lemma 5.3.1. *Let $k \geq 2$ be an integer. There exists a polynomial-time algorithm which takes as input:*

- *an integer $j \in \{0, \dots, n\}$,*
- *$n - j$ matrices $\boldsymbol{A}_{j+1}, \dots, \boldsymbol{A}_n$ in $\mathbb{Z}_p^{k \times k}$,*
- *an expression $\tilde{P}$ of a multivariate polynomial $P \in \mathbb{Z}_p[T_1, \dots, T_n]$ satisfying Condition 2,*

and which outputs a decomposition of $\tilde{P}$ as N polynomials $Q_1, \dots, Q_N \in \mathbb{Z}_p[T_1, \dots, T_j]$ and N matrices $\boldsymbol{C}_1, \dots, \boldsymbol{C}_N \in \mathbb{Z}_p^{k \times k}$ such that:

$$P(T_1, \dots, T_j, \boldsymbol{A}_{j+1}, \dots, \boldsymbol{A}_n) = \sum_{\nu=1}^{N} \boldsymbol{C}_\nu \cdot Q_\nu(T_1, \dots, T_j).$$

In addition, N is less than the number of internal nodes in the expression or AST $\tilde{P}$; and the representations of the polynomials $Q_1, \dots, Q_N$ satisfy Condition 2.

Proof of Lemma 5.3.1. We do the proof by recursion:

- *Base case (a leaf): an indeterminate T_i or a scalar in $\mathbb{Z}_p$.* Straightforward.
- *Recursive case 1: additive node $\tilde{P}_1 + \tilde{P}_2$.* We decompose recursively $\tilde{P}_1$ and $\tilde{P}_2$.
- *Recursive case 2: multiplicative node $\tilde{P}_1 \cdot \tilde{P}_2$.* This is the most important case. We consider two sub-cases:
  - $\tilde{P}_1$ only contain leaves with scalars or indeterminates $T_{j+1}, \dots, T_n$. In that case, its decomposition is just a matrix in $\mathbb{Z}_p^{k \times k}$. The decomposition of $\tilde{P}_1 \cdot \tilde{P}_2$ then contains as many terms as in the decomposition of $\tilde{P}_2$.
  - Otherwise, $\tilde{P}_2$ does not contain indeterminates $T_{j+1}, \dots, T_n$ (otherwise that would break Condition 2), and so the decomposition of $\tilde{P}_2$ is just a polynomial (matrices are identity matrices). The decomposition of $\tilde{P}_1 \cdot \tilde{P}_2$ then contains as many terms as in the decomposition of $\tilde{P}_1$.

Lemma 5.3.1 follows. □

Lemma 5.3.2. *Let $k \geq 1$ and $j \geq 1$ be two integers. There exists a polynomial-time algorithm which takes as input an expression $\tilde{P}$ of a multivariate polynomial $P \in \mathbb{Z}_p[T_1, \dots, T_j]$ of degree at most $d < p$ in T_j and satisfying Condition 1, and which outputs $d + 1$ polynomials $Q_0, \dots, Q_d \in \mathbb{Z}_p[T_1, \dots, T_{j-1}]$ such that*

$$P = Q_0 + Q_1 \cdot T_j + \cdots + Q_d \cdot T_j^d.$$

In addition, the representations of $Q_0, \dots, Q_d$ satisfy Condition 1.

Proof of Lemma 5.3.2. We can use the Lagrange interpolation

$$P = \sum_{i=0}^{d} P(T_1, \dots, T_{j-1}, i) \prod_{\substack{i'=0,\dots,d \\ i' \neq i}} \frac{(T_j - i')}{i - i'},$$

and regroup terms correctly. □

5.3.2 The Main Proof

Preliminaries. We recall that the representations of the polynomials we consider satisfy Condition 1 (when $k = 1$) or Condition 2 (when $k \geq 2$). Please refer to Section 5.1.4 for the definition of these conditions.

Let $\vec{\boldsymbol{A}} \in (\mathbb{Z}_p^{k \times k})^n$. When $k \geq 2$, for any polynomial $P \in \mathbb{Z}_p[T_1, \dots, T_n]$ whose degree in one indeterminate is at most d and for $j = 1, \dots, n$, using Lemma 5.3.1, we can decompose $\boldsymbol{Q}_{P,\vec{\boldsymbol{A}},j} = P(T_1, \dots, T_j, \boldsymbol{A}_{j+1}, \dots, \boldsymbol{A}_n)$ as follows (in polynomial time):

$$\boldsymbol{Q}_{P,\vec{\boldsymbol{A}},j} = \sum_{\nu=1}^{N_{P,\vec{\boldsymbol{A}},j}} \boldsymbol{C}_{P,\vec{\boldsymbol{A}},j,\nu} \cdot Q_{P,\vec{\boldsymbol{A}},j,\nu}(T_1, \dots, T_j),$$

with $N_{P,\vec{A},j}$ a positive integer, $\boldsymbol{C}_{P,\vec{\boldsymbol{A}},j,\nu}$ a matrix in $\mathbb{Z}_p^{k \times k}$, and $Q_{P,\vec{\boldsymbol{A}},j,\nu}$ a polynomial in $\mathbb{Z}_p[T_1, \dots, T_j]$ (given by a representation still satisfying Condition 2), for $\nu = 1, \dots, N_{P,\vec{A},j}$. We remark that this decomposition exists and can trivially be obtained when $k = 1$ (in this case $N_{P,\vec{\boldsymbol{A}},j} = 1$ and $\boldsymbol{C}_{P,\vec{\boldsymbol{A}},j,1} = 1$. When the index $\vec{\boldsymbol{A}}$ is clear from context, it is omitted. We write N the maximum possible value of $N_{P,j}$ (when $k = 1$, $N = 1$).

Since $Q_{P,j,\nu}$ is a polynomial in $\mathbb{Z}_p[T_1, \dots, T_j]$, with degree in any indeterminate bounded by d, according to Lemma 5.3.2, we can decompose it in polynomial time as:

$$Q_{P,j,\nu} = Q_{P,j,\nu,0} + T_j \cdot Q_{P,j,\nu,1} + \dots + T_j^d \cdot Q_{P,j,\nu,d},$$

with $Q_{P,k,\nu,0}, \dots, Q_{P,k,\nu,d}$ polynomials in $\mathbb{Z}_p[T_1, \dots, T_j]$ (given by a representation satisfying Condition 1).

In particular, we have:

$$\boldsymbol{Q}_{P,j} = \sum_{\nu=1}^{N_{P,j}} \sum_{i=0}^{d} \boldsymbol{C}_{P,j,\nu} \cdot T_j^i \cdot Q_{P,j,\nu,i} \tag{5.1}$$

$$\boldsymbol{Q}_{P,j-1} = \sum_{\nu=1}^{N_{P,j}} \sum_{i=0}^{d} \boldsymbol{C}_{P,j,\nu} \cdot A_i^j \cdot Q_{P,j,\nu,i} \tag{5.2}$$

Finally, we write $\boldsymbol{C}_{P,j,z_1,\dots,z_j} \in \mathbb{Z}_p^{k \times k}$ the (matrix) coefficient of $T_j^{z_j} \cdots T_1^{z_1}$ in $\boldsymbol{Q}_{P,j}$, and we write $c_{P,j,\nu,z_1,\dots,z_j} \in \mathbb{Z}_p$ the coefficient of the previous monomial in $Q_{P,j,\nu}$ (or equivalently in

$Q_{P,j,\nu,z_j} \cdot T_j^i)$. As we have:

$$\boldsymbol{Q}_{P,j} = \sum_{z_1,\dots,z_j} \boldsymbol{C}_{P,j,z_1,\dots,z_j} \cdot T_j^{z_j} \cdots T_1^{z_1}$$
$$\boldsymbol{Q}_{P,j} = \sum_{\nu=1}^{N_{P,j}} \sum_{z_1,\dots,z_j} \boldsymbol{C}_{P,j,\nu} \cdot c_{P,j,\nu,z_1,\dots,z_j} \cdot T_j^{z_j} \cdots T_1^{z_1}$$
$$\boldsymbol{Q}_{P,j} = \sum_{\nu=1}^{N_{P,j+1}} \sum_{z_1,\dots,z_j} \sum_{i=0}^{d} \boldsymbol{C}_{P,j+1,\nu} \cdot A_{j+1}^i \cdot c_{P,j+1,\nu,z_1,\dots,z_j,i} \cdot T_j^{z_j} \cdots T_1^{z_1}$$

we have:

$$\boldsymbol{C}_{P,j,z_1,\dots,z_j} = \sum_{\nu=1}^{N_{P,j}} \boldsymbol{C}_{P,j,\nu} \cdot c_{P,j,\nu,z_1,\dots,z_j} \tag{5.3}$$
$$\boldsymbol{C}_{P,j,z_1,\dots,z_j} = \sum_{\nu=1}^{N_{P,j+1}} \sum_{i=0}^{d} \boldsymbol{C}_{P,j+1,\nu} \cdot A_{j+1}^i \cdot c_{P,j+1,\nu,z_1,\dots,z_j,i}. \tag{5.4}$$

Proof of Theorem 5.2.2. We write N the maximum of the $N_{P,j}$'s and $M = (d+1) \cdot q \cdot N \cdot m$. Let $\mathscr{A}$ be an adversary against the (n,d,k,m)-PLP security of $\mathbb{G}$ that makes q oracle queries. We prove a first statement under the $(\mathcal{E}_{k,d}, M)$-MDDH assumption, which denotes the M-fold $\mathcal{E}_{k,d}$-MDDH assumption. We can then use random self-reducibility to obtain our statement under the $\mathcal{E}_{k,d}$-MDDH assumption. Please refer to Chapter 4 for formal definitions of this intermediate assumption and of random self-reducibility.

More precisely, we first design an adversary $\mathscr{B}$ against the $(\mathcal{E}_{k,d}, N)$-MDDH problem such that:

$$\mathbf{Adv}_{\mathbb{G}}^{(n,d,k,m)\text{-}\mathsf{plp}}(\mathscr{A}) \le n \cdot \mathbf{Adv}_{\mathbb{G}}^{(\mathcal{E}_{k,d},M)\text{-}\mathsf{mddh}}(\mathscr{B}) + \frac{2n(d+1)qN}{p} + \frac{n}{p} + \frac{n}{p^2} \, . \tag{5.5}$$

The proof of the above equation is based on the sequence of games in Figure 5.4. The games are used in the following order: $\mathbf{G}_{0,1}, \mathbf{G}_{1,1}, \mathbf{G}_{0,2}, \dots, \mathbf{G}_{1,n}$. We denote by $\textsc{Succ}_i$ the event that game $\mathbf{G}_i$ output takes the value 1.

Let us start with the proof. For the sake of simplicity, let us first suppose the procedure $\mathsf{TestLin}$ is perfect. We will deal with its imperfection at the end of the proof.

We first show that game $\mathbf{G}_{0,1}$ instantiates exactly the game defining the (n,d,k,m)-PLP security of $\mathbb{G}$. For any query P, we have $Q_{P,\vec{\boldsymbol{A}},1,\nu} \in \mathbb{Z}_p[T_1]$ (for any $\nu = 1,\dots,N_{P,1}$) and according to Equation (5.1):

$$\boldsymbol{Q}_{P,1} = \sum_{\nu=1}^{N_{P,1}} \sum_{i=0}^{d} \boldsymbol{C}_{P,1,\nu,i} \cdot T_1^i \cdot Q_{P,1,\nu,i} \; ,$$

with $Q_{P,1,\nu,i} \in \mathbb{Z}_p$ and $\boldsymbol{C}_{P,1,\nu,i} \in \mathbb{Z}_p^{k \times k}$. The first time we see a non-zero coefficient $\alpha = Q_{P,1,\nu,i} \in \mathbb{Z}_p$: $\mathcal{L}[1] \leftarrow \alpha \in \mathbb{Z}_p$, $\mathsf{T}[1,0] \xleftarrow{\$} \mathbb{Z}_p^{k\times k}$ (let us write this element $\alpha \boldsymbol{A'}$), and $\mathsf{T}[1,\ell] \leftarrow \alpha \cdot \boldsymbol{A}_j^\ell \cdot \boldsymbol{A'}$ for $\ell = 1,\dots,d$. Afterwards, $\mathsf{TestLin}(\mathcal{L}, Q_{P,1,\nu,i})$ always outputs $\vec{\lambda}^{(\nu,i)} = Q_{P,1,\nu,i}/\alpha$,

```
proc Initialize // $\mathbf{G}_{0,j}$ ; $j = 1,\ldots,n$
$\vec{\boldsymbol{A}} \xleftarrow{\$} \mathbb{Z}_p^n$
$\mathcal{L} \leftarrow$ empty list
$\mathsf{T} \leftarrow$ empty 2-dimensional table
$L \leftarrow 0$ (length of $\mathcal{L}$)

proc RKFn($P$) // $\mathbf{G}_{0,j}$ ; $j = 1,\ldots,n$
$\boldsymbol{Y} \leftarrow \mathbf{0} \in \mathbb{Z}_p^{k\times m}$
For $\nu = 1,\ldots,N_{P,j}$
  For $i = 0,\ldots,d$
    $\vec{\lambda}^{(\nu,i)} \leftarrow \mathsf{TestLin}(\mathcal{L}, Q_{P,j,\nu,i})$
    If $\vec{\lambda}^{(\nu,i)} = \perp$ then
      $L \leftarrow L+1$
      $\mathcal{L}[L] \leftarrow Q_{P,j,\nu,i}$
      $\mathsf{T}[L,0] \xleftarrow{\$} \mathbb{Z}_p^{k\times m}$
      For $\ell = 1,\ldots,d$
        $\boxed{\mathsf{T}[L,\ell] \xleftarrow{\$} \boldsymbol{A}_j^{\ell}\cdot\mathsf{T}[L,0]}$
      $\vec{\lambda}^{(\nu,i)} \leftarrow (0,\ldots,0,1) \in \mathbb{Z}_p^{L+1}$
    $\boldsymbol{Y} \leftarrow \boldsymbol{Y} + \boldsymbol{C}_{P,j,\nu}\cdot\sum_{l=1}^{L}\lambda_l^{(\nu,i)}\cdot\mathsf{T}[l,i]$
Return $[\boldsymbol{Y}]$
```

```
proc Initialize // $\mathbf{G}_{1,j}$ ; $j = 1,\ldots,n$
$\vec{\boldsymbol{A}} \xleftarrow{\$} \mathbb{Z}_p^n$
$\mathcal{L} \leftarrow$ empty list
$\mathsf{T} \leftarrow$ empty 2-dimensional table
$L \leftarrow 0$ (length of $\mathcal{L}$)

proc RKFn($P$) // $\mathbf{G}_{1,j}$ ; $j = 1,\ldots,n$
$\boldsymbol{Y} \leftarrow \mathbf{0} \in \mathbb{Z}_p^{k\times m}$
For $\nu = 1,\ldots,N_{P,j}$
  For $i = 0,\ldots,d$
    $\vec{\lambda}^{(\nu,i)} \leftarrow \mathsf{TestLin}(\mathcal{L}, Q_{P,j,\nu,i})$
    If $\vec{\lambda}^{(\nu,i)} = \perp$ then
      $L \leftarrow L+1$
      $\mathcal{L}[L] \leftarrow Q_{P,j,\nu,i}$
      $\mathsf{T}[L,0] \xleftarrow{\$} \mathbb{Z}_p^{k\times m}$
      For $\ell = 1,\ldots,d$
        $\boxed{\mathsf{T}[L,\ell] \xleftarrow{\$} \mathbb{Z}_p}$
      $\vec{\lambda}^{(\nu,i)} \leftarrow (0,\ldots,0,1) \in \mathbb{Z}_p^{L+1}$
    $\boldsymbol{Y} \leftarrow \boldsymbol{Y} + \boldsymbol{C}_{P,j,\nu}\cdot\sum_{l=1}^{L}\lambda_l^{(\nu,i)}\cdot\mathsf{T}[l,i]$
Return $[\boldsymbol{Y}]$
```

Figure 5.4: Games $\mathbf{G}_{0,j}$ and $\mathbf{G}_{1,j}$ for the proof of the PLP theorem

for $i = 0,\ldots,d$. Then, the matrix $\boldsymbol{Y}$ is computed as

$$\begin{aligned}
\boldsymbol{Y} &= \sum_{\nu=1}^{N_{P,1}}\sum_{i=0}^{d}\boldsymbol{C}_{P,1,\nu}\cdot\sum_{l=1}^{1}\lambda_l^{(\nu,i)}\cdot\mathsf{T}[l,i]\\
&= \sum_{\nu=1}^{N_{P,1}}\sum_{i=0}^{d}\boldsymbol{C}_{P,1,\nu}\cdot Q_{P,1,\nu,i}\cdot\alpha^{-1}\cdot\alpha\cdot\boldsymbol{A}_1^i\cdot\boldsymbol{A'}\\
&= \sum_{\nu=1}^{N_{P,1}}\sum_{i=0}^{d}\boldsymbol{C}_{P,1,\nu}\cdot Q_{P,1,\nu,i}\cdot\boldsymbol{A}_1^i\cdot\boldsymbol{A'}\\
&= \boldsymbol{Q}_{P,0}\cdot\boldsymbol{A'} = P(\boldsymbol{A}_1,\ldots,\boldsymbol{A}_n)\cdot\boldsymbol{A'},
\end{aligned}$$

where the last-but-one equality comes from Equation (5.2). Hence, this is exactly the game (n,d,k,m)-$\mathsf{PLPReal}_{\mathbb{G}}$. Now, let us show Game $\mathbf{G}_{0,j}$ and Game $\mathbf{G}_{1,j}$ are indistinguishable under the $(\mathcal{E}_{k,d}, M)$-MDDH assumption. Afterwards, we will show that Game $\mathbf{G}_{1,j}$ and Game $\mathbf{G}_{0,j+1}$ are perfectly indistinguishable.

Indistinguishability of Game $\mathbf{G}_{0,j}$ and Game $\mathbf{G}_{1,j}$ under the $(\mathcal{E}_{1,d}, d\cdot q)$-MDDH assumption. We recall that $M = (d+1)\cdot q\cdot N\cdot m$. We design adversaries $\mathscr{B}_j$ attacking the $(\mathcal{E}_{1,d}, M)$-MDDH problem in $\mathbb{G}$ such that

$$\Pr[\,\textsc{Succ}_{0,j}\,] - \Pr[\,\textsc{Succ}_{1,j}\,] \leq \mathbf{Adv}_{\mathbb{G}}^{(\mathcal{E}_{1,d},M)\text{-mddh}}(\mathscr{B}_j)\ ;\ \forall j = 1,\ldots,n.$$

The adversary $\mathscr{B}_j$ takes as input a tuple $([\boldsymbol{\Gamma}], [\boldsymbol{Z}]) \in \mathbb{G}^{k(d+1)\times k} \times \mathbb{G}^{(d+1)\times M}$, where either $\boldsymbol{Z} = \boldsymbol{\Gamma} \cdot \boldsymbol{W}$, and $\boldsymbol{W} \xleftarrow{\$} \mathbb{Z}_p^{k\times M}$, or $\boldsymbol{Z} = \boldsymbol{U} \xleftarrow{\$} \mathbb{Z}_p^{k(d+1)\times M}$, with $\boldsymbol{\Gamma}$ defined as in Section 4.2, and has to distinguish these two cases. For that purpose, the adversary $\mathscr{B}_j$ simulates everything as in Game $\mathbf{G}_{0,j}$ or $\mathbf{G}_{1,j}$ for $\mathscr{A}$, except it sets $\mathsf{T}[l,i]$ to be the $k \times m$ matrix with $z_{\alpha+ki,\beta+lm}$ as the entry of index $(\alpha,\beta) \in \{1,\dots,k\} \times \{1,\dots,m\}$. Assuming the matrix $\boldsymbol{B} \in \mathbb{Z}_p^{k\times k}$ (in the definition of $\boldsymbol{\Gamma}$) is invertible (which happens with probability $(1-1/p)\cdots(1-1/p^k) \geq 1 - 1/p - 1/p^2$, thanks to Euler's Pentagonal Number Theorem), in the first case, everything is simulated as in Game $\mathbf{G}_{0,j}$, while in the second case, everything is simulated as in Game $\mathbf{G}_{1,j}$. In the first case, everything is simulated as in Game $\mathbf{G}_{0,j}$, while in the second case, everything is simulated as in Game $\mathbf{G}_{1,j}$.

Perfect Indistinguishability of Game $\mathbf{G}_{1,j}$ and Game $\mathbf{G}_{0,j+1}$. We introduce an intermediate Game $\mathbf{G}_{2,j}$, described in Figure 5.5. We will use it to prove that Game $\mathbf{G}_{1,j}$ is perfectly indistinguishable from Game $\mathbf{G}_{0,j+1}$ by showing that both these games are perfectly indistinguishable from game $\mathbf{G}_{2,j}$. This intermediate game is not polynomial-time, since U is a linear map from $\mathbb{Z}_p[T_1,\dots,T_j]_{\leq d}$ to $\mathbb{Z}_p^{k\times m}$ and so would be represented by a matrix with $km(d+1)^j$ entries. But this does not affect our proof since we show that it is perfectly indistinguishable from Game $\mathbf{G}_{1,j}$ and Game $\mathbf{G}_{0,j+1}$ which are both polynomial-time.

First, we prove that game $\mathbf{G}_{1,j}$ is perfectly indistinguishable from Game $\mathbf{G}_{2,j}$. For that, we remark that T in $\mathbf{G}_{1,j}$ can be seen as computed as $\mathsf{T}[l,i] = U(T_j^i \cdot Q_l)$, for $k = 0,\dots,d$, with $U \xleftarrow{\$} \mathsf{L}(\mathbb{Z}_p[T_1,\dots,T_j]_{\leq d}, \mathbb{Z}_p^{k\times m})$ and $\mathcal{L}[l] = Q_l \in \mathbb{Z}_p[T_1,\dots,T_{j-1}]$. Indeed, the polynomials $T_j^i \cdot Q_l$ are linearly independent, and so $U(Tj^i \cdot Q_l)$ are independent uniform matrices in $\mathbb{Z}_p^{k\times m}$. We have:

$$Q_{P,j,\nu,i} = \sum_{l=1}^{L} \lambda_l^{(\nu,i)} \cdot Q_l.$$

Thus, for a query P, we remark that the matrix $\boldsymbol{Y}$ is computed as:

$$\begin{aligned}
\boldsymbol{Y} &= \sum_{\nu=1}^{N_{P,j}} \sum_{i=0}^{d} \boldsymbol{C}_{P,j,\nu} \cdot \sum_{l=1}^{L} \lambda_l^{(\nu,i)} \cdot \mathsf{T}[l,i] = \sum_{\nu=1}^{N_{P,j}} \sum_{i=0}^{d} \boldsymbol{C}_{P,j,\nu} \cdot \sum_{l=1}^{L} \lambda_l^{(\nu,i)} \cdot U(T_j^i \cdot Q_l) \\
&= \sum_{\nu=1}^{N_{P,j}} \boldsymbol{C}_{P,j,\nu} \cdot U\left(\sum_{i=0}^{d} \sum_{l=1}^{L} \lambda_l^{(\nu,i)} \cdot T_j^i \cdot Q_l \right) \\
&= \sum_{\nu=1}^{N_{P,j}} \sum_{i=0}^{d} \boldsymbol{C}_{P,j,\nu} \cdot U\left(\sum_{i=0}^{d} Q_{P,j,\nu,i} \cdot T_j^i \right) \\
&= \sum_{\nu=1}^{N_{P,j}} \boldsymbol{C}_{P,j,\nu} \cdot U(\sum_{z_1,\dots,z_j} c_{P,j,\nu,z_1,\dots,z_j} \cdot T_j^{z_j} \cdots T_1^{z_1}) \\
&= \sum_{z_1,\dots,z_j} \left(\sum_{\nu=1}^{N_{P,j}} \boldsymbol{C}_{P,j,\nu} \cdot c_{P,j,\nu,z_1,\dots,z_j} \right) \cdot U(T_j^{z_j} \cdots T_1^{z_1}) \\
&= \sum_{z_1,\dots,z_j} \boldsymbol{C}_{P,j,z_1,\dots,z_j} \cdot U(T_j^{z_j} \cdots T_1^{z_1}),
\end{aligned}$$

where the last equality comes from Equation (5.3) and most other equalities come from the linearity of U. The matrix $\boldsymbol{Y}$ is computed exactly as in Game $\mathbf{G}_{2,j}$. Therefore, Games $\mathbf{G}_{1,j}$ and $\mathbf{G}_{2,j}$ are perfectly indistinguishable, for $j = 1,\dots,n$.

```
proc Initialize  // $\mathbf{G}_{2,j}$ ; $j = 1, \ldots, n$
$U \xleftarrow{\$} \mathsf{L}(\mathbb{Z}_p[T_1, \ldots, T_j]_{\leq d}, \mathbb{Z}_p^{k \times m})$

proc RKFn($P$)  // $\mathbf{G}_{2,j}$ ; $j = 1 \ldots, n$
$\boldsymbol{Y} \leftarrow \sum_{z_1,\ldots,z_j} \boldsymbol{C}_{P,j,z_1,\ldots,z_j} \cdot U(T_j^{z_j} \cdots T_1^{z_1})$
Return $[\boldsymbol{Y}]$
```

Figure 5.5: Games $\mathbf{G}_{2,j}$ for the proof of the PLP theorem

Second, we prove that Game $\mathbf{G}_{2,j}$ is perfectly indistinguishable from Game $\mathbf{G}_{0,j+1}$. The proof is similar to the previous one. For that, we remark that T in $\mathbf{G}_{1,j}$ can be seen as computed as $\mathsf{T}[l,i] = \boldsymbol{A}_{j+1}^i \cdot U(Q_l)$, for $i = 0, \ldots, d$, with $U \xleftarrow{\$} \mathsf{L}(\mathbb{Z}_p[T_1, \ldots, T_j]_{\leq d}, \mathbb{Z}_p^{k \times m})$ with $\mathcal{L}[l] = Q_l \in \mathbb{Z}_p[T_1, \ldots, T_j]$. Indeed, the polynomials Q_l are linearly independent, and so $U(Q_l)$ are independent uniform matrices in $\mathbb{Z}_p^{k \times m}$. We also have:

$$Q_{P,j+1,\nu,i} = \sum_{l=1}^{L} \lambda_l^{(\nu,i)} Q_l.$$

Thus, for a query P, we remark that the matrix $\boldsymbol{Y}$ is computed as:

$$\begin{aligned}
\boldsymbol{Y} &= \sum_{\nu=1}^{N_{P,j+1}} \sum_{i=0}^{d} \boldsymbol{C}_{P,j+1,\nu} \cdot \sum_{l=1}^{L} \lambda_l^{(\nu,i)} \cdot \mathsf{T}[l,i] \\
&= \sum_{\nu=1}^{N_{P,j+1}} \sum_{i=0}^{d} \boldsymbol{C}_{P,j+1,\nu} \cdot \sum_{l=1}^{L} \lambda_l^{(\nu,i)} \cdot A_{j+1}^i \cdot U(Q_l) \\
&= \sum_{\nu=1}^{N_{P,j+1}} \sum_{i=0}^{d} \boldsymbol{C}_{P,j+1,\nu} \cdot A_{j+1}^i \cdot U\left(\sum_{l=1}^{L} \lambda_l^{(\nu,i)} \cdot Q_l\right) \\
&= \sum_{\nu=1}^{N_{P,j+1}} \sum_{i=0}^{d} \boldsymbol{C}_{P,j,\nu} \cdot A_{j+1}^i \cdot U(Q_{P,j+1,\nu,i}) \\
&= \sum_{\nu=1}^{N_{P,j+1}} \sum_{i=0}^{d} \boldsymbol{C}_{P,j,\nu} \cdot U(\sum_{z_1,\ldots,z_j} c_{P,j+1,\nu,z_1,\ldots,z_j,i} \cdot T_j^{z_j} \cdots T_1^{z_1}) \\
&= \sum_{z_1,\ldots,z_j} \left(\sum_{\nu=1}^{N_{P,j+1}} \sum_{i=0}^{d} \boldsymbol{C}_{P,j,\nu} \cdot c_{P,j+1,\nu,z_1,\ldots,z_j,i} \right) \cdot U(T_j^{z_j} \cdots T_1^{z_1}) \\
&= \sum_{z_1,\ldots,z_j} \boldsymbol{C}_{P,j,z_1,\ldots,z_j} \cdot U(T_j^{z_j} \cdots T_1^{z_1}),
\end{aligned}$$

where the last equality comes from Equation (5.4) and most other equalities come from the linearity of U. This is computed exactly as in Game $\mathbf{G}_{2,j}$. Therefore, games $\mathbf{G}_{1,j}$ and $\mathbf{G}_{2,j}$ are perfectly indistinguishable, for $j = 1, \ldots, n$.

To conclude, let us prove that Game $\mathbf{G}_{1,n}$ is perfectly indistinguishable from the game (n,d,k,m)-$\mathsf{PLPRand}_{\mathbb{G}}$. Since Game $\mathbf{G}_{1,n}$ is perfectly indistinguishable from Game $\mathbf{G}_{2,n}$, we just need to prove the $\mathbf{G}_{2,0}$ is perfectly indistinguishable from the game (n,d,k,m)-$\mathsf{PLPRand}_{\mathbb{G}}$. This is the case, since $\boldsymbol{Q}_{P,n} = P$ is a polynomial with scalar coefficients, and in the expression of the matrix $\boldsymbol{Y} = \sum_{z_1,\ldots,z_j} \boldsymbol{C}_{P,n,z_1,\ldots,z_n} \cdot U(T_n^{z_n} \cdots T_1^{z_1})$ (in Game $\mathbf{G}_{2,0}$), $\boldsymbol{C}_{P,n,z_1,\ldots,z_n}$ can be seen as a scalar, and $\boldsymbol{Y} = U(Q_{P,n}) = U(P)$ by linearity.

Dealing with an Imperfect $\mathsf{TestLin}$. To deal with an imperfect $\mathsf{TestLin}$, we just remark that the only part where we supposed $\mathsf{TestLin}$ to be perfect in the proof was to prove the perfect indistinguishability of Game $\mathbf{G}_{1,j}$ and Game $\mathbf{G}_{0,j+1}$, and the perfect indistinguishability between Game $\mathbf{G}_{0,1}$ (respectively Game $\mathbf{G}_{1,n}$) with the game (n,d,k,m)-$\mathsf{PLPReal}_{\mathbb{G}}$ (respectively (n,d,k,m)-$\mathsf{PLPRand}_{\mathbb{G}}$). All this properties are statistical, so that it is possible to replace the real $\mathsf{TestLin}$ (with error $1/p$) by a perfect (with error 0, as used in the proof). This just loses an additive factor at most $(d+1)qN/p$ each time, and so at most $2n(d+1)qN/p$ in total.

Equation (5.5) easily follows from the bounds arising in the different game hops, and then Theorem 5.2.2 immediately follows. $\square$

Chapter 6

Applications

In this chapter, we show some applications of the PLP theorem (Theorem 5.2.2 on page 72). Specifically, we first show how the LIP theorem (Theorem 5.2.4 on page 73) can be used to prove the (standard and related-key) security of pseudorandom functions. In particular, using this theorem, we are able to design a fully algebraic framework for related-key security that allows to improve previous known results, by providing positive results for larger classes of related-key deriving functions, but also by building secure constructions based on weaker assumptions. We also apply the PLP theorem to solve many open questions in the area of aggregate pseudorandom functions and multilinear pseudorandom functions.

Our statements are directly given using the classical assumptions (rather than the $\mathcal{E}_{k,d}$-MDDH assumption). Please refer to Table 4.1 on page 56 for a summary of the relations.

Contents

6.1 Applications to Standard Pseudorandom Functions

In this section, we introduce weighted (extended) versions of the Naor-Reingold and Boneh-Montgomery-Raghunathan pseudorandom functions, termed weighted NR and weighted BMR, and respectively denoted WNR and WBMR. We later use these extended constructions in order to build related-key secure pseudorandom functions for new classes of RKD functions (Section 6.2).

These weighted versions are obtained by applying particular permutations to the key space. Then, in the standard setting, it is straightforward that the security of NR and BMR implies the security of their weighted versions. However, as detailed in Section 6.2, in the related-key setting, we can prove that some of these weighted constructions are secure against certain classes of RKD functions while both NR and BMR are not, even if we apply the framework from Chapter 3.

Finally, the security proofs of WNR and WBMR are straightforward using the LIP theorem, and so are the security proofs of NR, NR^*, and BMR, as particular cases of WNR and WBMR. Thus, as a first application of the LIP theorem theorem, we detail proofs of security for both these extended constructions.

6.1.1 Extended Number-Theoretic Pseudorandom Functions

6.1.1.1 Extended NR

Let $\vec{w} = (w_0, \dots, w_n) \in \mathbb{Z}_p^{n+1}$. We denote $\mathsf{WNR}^{\vec{w}} = (\mathsf{WNR}^{\vec{w}}.\mathsf{Setup}, \mathsf{WNR}^{\vec{w}}.\mathsf{Eval})$ the following construction:

- $\mathsf{WNR}^{\vec{w}}.\mathsf{Setup}(1^\kappa)$ generates public parameters $\mathsf{pp} = (p, \mathbb{G}, g)$ where $(p, \mathbb{G}, g)$ describes a cyclic group of prime order p generated by g;
- $\mathsf{WNR}^{\vec{w}}.\mathsf{Eval}_{\mathsf{pp}}(\cdot,\cdot)$ takes as input a key $\vec{a} = (a_0, a_1, \dots, a_n) \in \mathbb{Z}_p^{n+1}$ and an input $x = x_1 \| \dots \| x_n \in \{0,1\}^n$ and outputs $[a_0 \cdot \prod_{i=1}^n a_i^{x_i}]_g$.

Furthermore, in the case where $w_0 = 0$, we consider the key space to be $\mathbb{Z}_p^n$ (by ignoring a_0) and restrict the input x to be in $\{0,1\}^n \setminus \{0^n\}$, similarly to the definition of NR^*. As before, for simplicity, we often use the notation $\mathsf{WNR}^{\vec{w}}(\vec{a}, x)$ to denote $\mathsf{WNR}^{\vec{w}}.\mathsf{Eval}_{\mathsf{pp}}(\vec{a}, x)$.

Lemma 6.1.1. *Assuming* DDH *holds in* $\mathbb{G}$*,* $\mathsf{WNR}^{\vec{w}}$ *is a pseudorandom function, for any* $\vec{w} \in \mathbb{Z}_p^{n+1}$ *such that* w_i *and* $p-1$ *are coprime, for* $i = 0, \dots, n$.

Remark 6.1.2. We recover the standard NR and NR^* constructions by letting $\vec{w} = (1, \dots, 1)$ and $\vec{w} = (0, 1, \dots, 1)$ respectively. For a weight $\vec{w}$ such that there exists i with w_i and $p-1$ not being coprime, we could still use the LIP theorem to prove the security, but we would need to rely on a stronger assumption (as the application $a \in \mathbb{Z}_p \mapsto a^{w_i} \in \mathbb{Z}_p$ is no longer a bijection).

6.1.1.2 Extended BMR

Let $\vec{w} = (w_1, \dots, w_n) \in \mathbb{Z}_p^n$. We denote $\mathsf{WBMR}^{\vec{w}} = (\mathsf{WBMR}^{\vec{w}}.\mathsf{Setup}, \mathsf{WBMR}^{\vec{w}}.\mathsf{Eval})$ the following construction:

- $\mathsf{WBMR}^{\vec{w}}.\mathsf{Setup}(1^\kappa)$ generates public parameters $\mathsf{pp} = (p, \mathbb{G}, g)$ where $(p, \mathbb{G}, g)$ describes a cyclic group of prime order p generated by g;
- $\mathsf{WBMR}^{\vec{w}}.\mathsf{Eval}_{\mathsf{pp}}(\cdot,\cdot)$ takes as input a key $\vec{a} = (a_1, \dots, a_n) \in \mathbb{Z}_p^n$ and an input $x = x_1 \| \dots \| x_n \in \{0, \dots, d\}^n$ and outputs $\left[\prod_{i=1}^n \frac{1}{a_i+w_i+x_i}\right]_g$.

Again, we often use the notation $\mathsf{WBMR}^{\vec{w}}(\vec{a}, x)$ to denote $\mathsf{WBMR}^{\vec{w}}.\mathsf{Eval}_{\mathsf{pp}}(\vec{a}, x)$.

Lemma 6.1.3. *Assuming d-DDHI holds in $\mathbb{G}$, $\mathsf{WBMR}^{\vec{w}}$ is a pseudorandom function, for any $\vec{w} \in \mathbb{Z}_p^n$.*

Remark 6.1.4. We recover the standard BMR construction by letting $\vec{w} = (0, \dots, 0)$.

6.1.2 Simple Proofs of Security

6.1.2.1 Intuition

Here, we detail how the LIP theorem can be used to provide a very simple proof of security of the above extended constructions. Let us first provide a brief intuition with the WBMR construction, the proof of security for WNR being even simpler. Formal proofs are given right after.

Concretely, for $\mathsf{WBMR}^{\vec{w}}$, we start by revealing as public parameters the group $\mathbb{G}$, its order p, and the generator h to the adversary where

$$h = \left[\left(\prod_{i=1}^n \prod_{k\in\{0,\dots,d\}} (a_i + w_i + k)\right) \cdot b\right]_g = [P(\vec{a}) \cdot b]_g$$

which is a generator with overwhelming probability. Then, when the adversary makes a query x, it is clear that

$$\left[\prod_{i=1}^n \frac{1}{a_i + w_i + x_i}\right]_h = \left[\left(\prod_{i=1}^n \prod_{k\in\{0,\dots,d\}\setminus\{x_i\}} (a_i + w_i + k)\right) \cdot b\right]_g = [P_x(\vec{a}) \cdot b]_g \quad .$$

As each polynomial P_x is null on every input $-x'$ for $x' \in \{0, \dots, d\}^n$, seen as a vector of $\mathbb{Z}_p^n$, except when $x' = x$, and as P is null on all $-x'$, P and $(P_x)_x$ are linearly independent. Then, we conclude the security proof of $\mathsf{WBMR}^{\vec{w}}$ by applying the LIP theorem.

6.1.2.2 Extended NR

Proof of Lemma 6.1.1. Since the application $a \in \mathbb{Z}_p \mapsto a^w \in \mathbb{Z}_p$ is a bijection as long as w and $p-1$ are coprime, it is clear that if NR is a secure pseudorandom, then so is $\mathsf{WNR}^{\vec{w}}$, as long as w_i and $p-1$ are coprime, for $i = 0, \dots, n$, since we just apply a permutation to the key space.

Let $\mathscr{A}$ be an adversary against the pseudorandom function security of NR that makes q oracle queries. We can assume without loss of generality that $\mathscr{A}$ never repeats a query. Then we can construct an adversary $\mathscr{B}$ against the $(n, 1)$-LIP security of $\mathbb{G}$ that makes q queries $P_0, P_1, \dots, P_q$, defined as follows.

First, $\mathscr{B}$ sends the public parameters $(p, \mathbb{G}, g)$ to $\mathscr{A}$. By doing so, it implicitly sets $a_0 = b$.

Next, $\mathscr{B}$ runs adversary $\mathscr{A}$. When the latter makes a query $x \in \{0,1\}^n$, adversary $\mathscr{B}$ makes the query $P_x(T_1, \dots, T_n) = \prod_{i=1}^n T_i^{x_i}$ to its oracle and returns the value it gets to $\mathscr{A}$. When $\mathscr{A}$ halts, $\mathscr{B}$ halts with the same output. If $\mathscr{B}$'s oracle responds to a query P by the value $[P(\vec{a}) \cdot b]$, then $\mathscr{B}$ sends $[P_x(\vec{a}) \cdot b] = [b \cdot \prod_{i=1}^n a_i^{x_i} b] = \mathsf{NR}((b, \vec{a}), x)$ to $\mathscr{A}$ when the latter makes a query x, while if $\mathscr{B}$'s oracle responds to a query P by a uniformly random value, then $\mathscr{B}$ sends uniformly random values to $\mathscr{A}$. Hence, $\mathscr{B}$ simulates exactly the game defining the pseudorandom function security of NR for the key $\vec{a}$ which is taken at random over $\mathbb{Z}_p^{n+1}$.

The only thing we need to prove is that all these queries are allowed to $\mathscr{B}$, meaning that for any distinct queries $x_1, \dots, x_q$ of $\mathscr{A}$, polynomials $P_{x_1}, \dots, P_{x_q}$ are linearly independent over $\mathbb{Z}_p$, which is straightforward. Hence, we have

$$\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{NR}}(\mathscr{A}) \leq \mathbf{Adv}^{(n,1)\text{-}\mathsf{lip}}_{\mathbb{G}}(\mathscr{B}) \ .$$

We now conclude the proof by applying the LIP theorem.

This also proves the security of $\mathsf{WNR}^{\vec{w}}$ for any $\vec{w}$ such that w_i and $p-1$ are coprime, for $i = 0, \dots, n$. Please also note that if $w_0 = 0$ (for instance for NR^*), we can do a similar proof using the $(n,1)$-LIP security of $\mathbb{G}$, by querying 1 at first and revealing $[b]$ as the generator instead of g. □

6.1.2.3 Extended BMR

Proof of Lemma 6.1.3. Let $\mathscr{A}$ be an adversary against the pseudorandom function security of $\mathsf{WBMR}^{\vec{w}}$ that makes q oracle queries. We can assume without loss of generality that $\mathscr{A}$ never repeats a query. Then we can construct an adversary $\mathscr{B}$ against the (n,d)-LIP security of $\mathbb{G}$ that makes $q+1$ queries $P_0, P_1, \dots, P_q$, defined as follows.

First, $\mathscr{B}$ queries polynomial $P_0(\vec{T}) = \prod_{i=1}^n \prod_{k \in \{0,\dots,d\}} (T_i + w_i + k)$ and gets

$$h = \left[\left(\prod_{i=1}^n \prod_{k \in \{0,\dots,d\}} (a_i + w_i + k) \right) \cdot b \right]_g = [P_0(\vec{a}) \cdot b]_g \ ,$$

that it sends to $\mathscr{A}$ along with $\mathbb{G}$ and p as public parameters (h is a generator with probability $(\frac{p-d-1}{p})^n$).

Next, $\mathscr{B}$ runs adversary $\mathscr{A}$. When the latter makes a query $x \in \{0, \dots, d\}^n$, adversary $\mathscr{B}$ makes the query $P_x(T_1, \dots, T_n) = \prod_{i=1}^n \prod_{k \in \{0,\dots,d\} \setminus \{x\}} (T_i + w_i + k)$ to its oracle and returns the value he gets to $\mathscr{A}$. When $\mathscr{A}$ halts, $\mathscr{B}$ halts with the same output. If $\mathscr{B}$'s oracle responds to a query P by the value $[P(\vec{a}) \cdot b]_g$, then $\mathscr{B}$ sends

$$\begin{aligned}
[P_x(\vec{a}) \cdot b]_g &= \left[\left(\prod_{i=1}^n \prod_{k \in \{0,\dots,d\} \setminus \{x\}} (a_i + w_i + k) \right) \cdot b \right]_g \\
&= \left[\frac{P_0(\vec{a}) \cdot b}{\prod_{i=1}^n (a_i + w_i + x_i)} \right]_g \\
&= \left[\prod_{i=1}^n \frac{1}{a_i + w_i + x_i} \right]_h \ ,
\end{aligned}$$

to $\mathscr{A}$ when the latter makes a query x, while if $\mathscr{B}$'s oracle responds to a query P by a uniformly random value, then $\mathscr{B}$ sends uniformly random values to $\mathscr{A}$. Hence, $\mathscr{B}$ simulates exactly the game defining the pseudorandom function security of $\mathsf{WNR}^{\vec{w}}$ for the key $\vec{a}$ which is taken at random over $\mathbb{Z}_p^n$ and the generator h.

Then, since $P_x(T_1, \ldots, T_n) = \prod_{i=1}^n \prod_{k \in \{0,\ldots,d\} \setminus \{x\}} (T_i + w_i + k)$ is a degree at most d polynomial in each indeterminate, the only thing that remains to prove is that all these queries are allowed to $\mathscr{B}$, meaning that for any distinct queries $x^{(1)}, \ldots, x^{(q)}$ of $\mathscr{A}$, polynomials $P_0, P_{x^{(1)}}, \ldots, P_{x^{(q)}}$ are linearly independent over $\mathbb{Z}_p$. By contradiction, let us assume there is a sequence of distinct queries $x^{(1)}, \ldots, x^{(q)}$ such that there is a linear combination:

$$\lambda_0 P_0 + \sum_{m=1}^{q} \lambda_k P_{x^{(k)}}$$

with $\lambda_k \neq 0$ for all $k = 1, \ldots, q$.

By evaluating this sum in $-x^{(k)} - \vec{w} = (-x_1^{(k)} - w_1, \ldots, -x_n^{(k)} - w_n) \in \mathbb{Z}_p^n$, we get $P_0(-x^{(k)} - \vec{w}) = 0$ for all k and $P_{x^{(i)}}(-x^{(k)} - \vec{w}) = 0$ for all $i \neq k$ and then $\lambda_k P_{x^{(k)}}(-x^{(k)} - \vec{w}) = 0$, which directly implies $\lambda_k = 0$, since $P_{x^{(k)}}(-x^{(k)} - \vec{w}) \neq 0$ for all $k = 1, \ldots, q$. Then, we have proven that $\lambda_k = 0$, for all $k = 1, \ldots, q$. Finally, since P_0 is not the zero polynomial and $\lambda_0 P_0 = 0$, we have also $\lambda_0 = 0$.

Then, for any distinct queries $x^{(1)}, \ldots, x^{(q)}$ of $\mathscr{A}$, polynomials $P_0, P_{x^{(1)}}, \ldots, P_{x^{(q)}}$ are linearly independent over $\mathbb{Z}_p$. Hence, we have

$$\mathsf{Adv}^{\mathsf{prf}}_{\mathsf{WBMR}^{\vec{w}}}(\mathscr{A}) \leq \left(\frac{p}{p-d-1}\right)^n \cdot \mathbf{Adv}^{(n,d)\text{-}\mathsf{lip}}_{\mathbb{G}}(\mathscr{B}) \ .$$

We conclude the proof by applying the LIP theorem. □

6.2 Applications to Related-Key Security

In this section, we show how the LIP theorem can also be used to prove the related-key security of a pseudorandom function $F = (\mathsf{F.Setup}, \mathsf{F.Eval})$ such that $\mathsf{F.Eval}$ takes a key $\vec{a}$ and an input x and outputs a group element $\mathsf{F.Eval}_{\mathsf{pp}}(\vec{a}, x) = [P_x(\vec{a})]$ over a group $\mathbb{G} = \langle g \rangle$ of prime order p, described by the $\mathsf{F.Setup}$ procedure, where P_x is a certain multivariate polynomial depending on x. For simplicity, let us once again simply denote by $F(\vec{a}, x)$ the output $\mathsf{F.Eval}_{\mathsf{pp}}(\vec{a}, x)$.

For simplicity, we only consider the case $k = 1$ in the following, but similar constructions can be obtained with $k > 1$ (except when permutations are involved). A brief extension is provided at the end of this section.

Let Φ be a class of RKD functions, where functions $\vec{\phi} = (\phi_1, \ldots, \phi_n) \in \Phi$ are such that ϕ_i are multivariate polynomials in $\mathbb{Z}_p[T_1, \ldots, T_n]$. Then, for an RKD function $\vec{\phi}$ and an input x, the pseudorandom function outputs $F(\vec{\phi}(\vec{a}), x) = \left[P_{\vec{\phi},x}(\vec{a})\right]$, where the polynomial $P_{\vec{\phi},x}(\vec{T}) = P_x(\vec{\phi}(\vec{T})) = P_x(\phi_1(\vec{T}), \ldots, \phi_n(\vec{T}))$ depends on $\vec{\phi}$ and x. In particular, $P_{\mathsf{id},x} = P_x$ for all x, where id is the identity function.

When all polynomials $P_{\vec{\phi},x}$ and the constant polynomial 1 are linearly independent, the LIP theorem directly shows that F is Φ-related-key secure under the hardness of the $\mathcal{E}_{1,d}$-MDDH problem (with d being the maximum degree in one indeterminate occurring in the family $\{P_{\vec{\phi},x}\}_{\vec{\phi},x}$). To illustrate this, we use this method to construct a related-key secure pseudorandom function for the class of permutations in Section 6.2.1.

However, to assume that all polynomials $P_{\vec{\phi},x}$ are linearly independent is a very strong property and, in general, this is not the case for all x and $\vec{\phi}$. Hence, in Section 6.2.2, we consider the less restrictive case where the polynomials $P_{\vec{\phi}_1,x_1},\dots,P_{\vec{\phi}_q,x_q}$ are linearly independent as long as the inputs $x_1,\dots,x_q$ are distinct (so the adversary is unique-input). Specifically, we design a new algebraic framework that extends the one from Section 3.6, when the pseudorandom function F outputs elements of the form $[P_x(\vec{a})]$ and when the RKD functions are multivariate polynomials. The proof of this framework follows easily from the LIP theorem. We then use this algebraic framework to construct related-key secure pseudorandom functions for new and larger classes of RKD functions as well as weaker assumptions.

6.2.1 Direct Constructions

In this section, we show how the LIP theorem can be used to prove the Φ-related-key security of a pseudorandom function F of the above form in the particular case where all polynomials $P_{\vec{\phi},x}$ are linearly independent, for any $\vec{\phi} \in \Phi$ and any input x.

Here, we consider the class $\Phi_{\mathfrak{S}_n}$ of functions defined as $\{\sigma \mid \sigma \in \mathfrak{S}_n\}$ such that, applying a function $\sigma \in \Phi_{\mathfrak{S}_n}$ to a key $\vec{a} = (a_1,\dots,a_n) \in \mathbb{Z}_p^n$ leads to the key $\sigma(\vec{a}) = (a_{\sigma^{-1}(1)},\dots,a_{\sigma^{-1}(n)})$, so the i-th component of $\vec{a}$ becomes the $\sigma(i)$-th component of the key $\sigma(\vec{a})$.

It is clear that BMR is not $\Phi_{\mathfrak{S}_n}$-related-key secure, since we can distinguish BMR from a random function with only 2 queries. Indeed, let id be the identity function and (12) be the permutation which switches the first two components of the key. Then, one can just query $(\mathsf{id}, 100\dots0)$ and $((12), 010\dots0)$ and check whether the output of these queries are the same, which is the case in the real case while they are independent in the random case. However, we show in what follows that a particular case of WBMR, defined below, is a $\Phi_{\mathfrak{S}_n}$-related-key secure PRF.

Linear WBMR PRF. We define $\mathsf{WBMR}^{\mathsf{lin}}$ as the particular case of WBMR, where $w_i = (i-1)(d+1)$, for $i = 1,\dots,n$. Please refer to Section 6.1.1.2 for details.

Lemma 6.2.1. *Assuming $(n(d+1)-1)$-DDHI holds in $\mathbb{G}$, $\mathsf{WBMR}^{\mathsf{lin}}$ is a $\Phi_{\mathfrak{S}_n}$-related-key secure pseudorandom function.*

Moreover, the time overhead of this reduction is polynomial in n, d and in the running time of the adversary.

The proof is given below and is very similar to the proof of security of WBMR in Section 6.1.

Proof of Lemma 6.2.1. Let $\mathscr{A}$ be a $\Phi_{\mathfrak{S}_n}$-restricted adversary against the related-key security of $\mathsf{WBMR}^{\mathsf{lin}}$ that makes q queries. We design an adversary $\mathscr{B}$ against the $(n, n(d+1)-1)$-LIP security of $\mathbb{G}$ that makes $q+1$ queries as follows.

First, $\mathscr{B}$ queries polynomial $P_0(\vec{T}) = \prod_{i=1}^n \prod_{k\in\{0,\dots,n(d+1)-1\}} (T_i + k)$ and gets h with

$$h = \left[\left(\prod_{i=1}^{n} \prod_{k\in\{0,\dots,n(d+1)-1\}} (a_i + k)\right) \cdot b\right]_g = [P_0(\vec{a}) \cdot b]_g\,,$$

or a random group element, that it sends to $\mathscr{A}$ as the generator along with $\mathbb{G}$ and p as the public parameters for $\mathsf{WBMR}^{\mathsf{lin}}$ (h is a generator with probability at least $(\frac{p-n(d+1)-1}{p})^n$).

Next, $\mathscr{B}$ runs adversary $\mathscr{A}$. When the latter makes a query (σ, x), $\mathscr{B}$ computes the polynomial

$$P_{\sigma,x}(\vec{T}) = \frac{\prod_{i=1}^{n}\prod_{k=0}^{n(d+1)-1}(T_i + k)}{\prod_{i=1}^{n}(T_{\sigma^{-1}(i)} + w_i + x_i)} = \frac{P_0(\vec{T})}{\prod_{i=1}^{n}(T_i + w_{\sigma(i)} + x_{\sigma(i)})}$$

and queries $P_{\sigma,x}$. This is possible since $\prod_{i=1}^{n}(T_i + w_{\sigma(i)} + x_{\sigma(i)})$ divides P_0 and since $P_{\sigma,x}$ is a degree $n(d+1)-1$ polynomial in each indeterminate. It then returns the value it gets to $\mathscr{A}$.

It is clear that if $\mathscr{B}$'s oracle returns uniformly random values, $\mathscr{B}$ simulates exactly the game $\text{RKPRFRand}_{\mathsf{WBMR}^{\mathsf{lin}}}$, whereas if it returns $[P(\vec{a}) \cdot b]$ for a query P, then $\mathscr{B}$ simulates exactly game $\text{RKPRFReal}_{\mathsf{WBMR}^{\mathsf{lin}}}$ with the generator defined above. Indeed, in that case, let $h = [P_0(\vec{a}) \cdot b]_g$ be the generator given to $\mathscr{A}$. Then for a query (σ, x), $[P_{\sigma,x}(\vec{a}) \cdot b]_g = \left[\prod_{i=1}^{n} \frac{1}{\vec{a}[\sigma^{-1}(i)]+w_i+x_i}\right]_h = \mathsf{WBMR}^{\mathsf{lin}}(\sigma(\vec{a}), x)$, where $\mathsf{WBMR}^{\mathsf{lin}}$ is defined using the generator h. Then, the only thing that remains to prove is that all the polynomials queried by $\mathscr{B}$ to its oracle are linearly independent over $\mathbb{Z}_p$.

Hence, let us now prove that all polynomials $P_{\sigma,x}(\vec{T})$ corresponding to queries (σ, x) are linearly independent. By contradiction, let us assume that there exists a sequence of distinct queries $(\sigma^{(1)}, x^{(1)}), \ldots, (\sigma^{(q)}, x^{(q)})$ such that there exists a linear combination:

$$\lambda_0 P_0 + \sum_{k=1}^{q} \lambda_k \cdot P_{\sigma^{(k)},x^{(k)}} = 0$$

with $\lambda_k \neq 0$ for all $k = 1, \ldots, q$.

Then, by evaluating the above sum of polynomials on points

$$(-w_{\sigma^{(k)}(1)} - x^{(k)}_{\sigma^{(k)}(1)}, \ldots, -w_{\sigma^{(k)}(n)} - x^{(k)}_{\sigma^{(k)}(n)}),$$

since all the above polynomials except $P_{\sigma^{(k)},x^{(k)}}$ take the value 0 on these points, we obtain, for $k = 1, \ldots, q$, $\lambda_k = 0$. Then, we have $\lambda_0 P_0 = 0$ and since $P_0 \neq 0$, we have $\lambda_0 = 0$. Hence, all these polynomials are linearly independent. Then, Lemma 6.2.1 easily follows from the above and from the LIP theorem. $\square$

Remark 6.2.2. By slightly changing the construction, we can also achieve $\Phi_{\mathfrak{S}_n,+_k}$-related-key security, where $\Phi_{\mathfrak{S}_n,+_k}$ is the class of functions $\sigma_{\vec{b}}$ that, when applied to a key $\vec{a} \in \mathbb{Z}_p^n$, leads to the key $(a_{\sigma^{-1}(1)}+b_1, \ldots, a_{\sigma^{-1}(n)}+b_n)$, for any $\sigma \in \mathfrak{S}_n$ and any $\vec{b} \in \{0, \ldots, k\}^n$. In other words, $\Phi_{\mathfrak{S}_n,+_k}$ also tolerates small additive factors in addition to permutations. Indeed, considering the function $\mathsf{WBMR}_k^{\mathsf{lin}}$ as the particular case of WBMR where $w_i = (i-1)(d+1)(k+1)$ for $i = 1, \ldots, n$ one can prove that this construction is $\Phi_{\mathfrak{S}_n,+_k}$-related-key secure under the $(d(k+1)+k+(n-1)(d+1)(k+1))$-$\mathsf{DDHI}$ assumption in $\mathbb{G}$ and the proof follows closely the proof of Lemma 6.2.1. In particular, with $k = 0$, this is exactly the result from Lemma 6.2.1.

6.2.2 Constructions From Unique-Input-Related-Key Security, Algebraically

In this section, we address the less restrictive case in which we only require the polynomials $P_{\vec{\phi}_1,x_1}, \ldots, P_{\vec{\phi}_q,x_q}$ to be linearly independent for any $\vec{\phi}_1, \ldots, \vec{\phi}_q$ when the inputs $x_1, \ldots, x_q$

are all distinct. Please notice that this is the case for all the classes considered in previous sections and chapters. We now denote by $M = (\mathsf{M.Setup}, \mathsf{M.Eval})$ the "original" pseudorandom function (that we want to transform into a related-key secure one) such that for any x, $M(\vec{a}, x) = \mathsf{M.Eval}_{\mathsf{pp}}(\vec{a}, x) = [P_x(\vec{a})]$ for a certain multivariate polynomial P_x depending on x.

In order to build related-key secure pseudorandom functions from such pseudorandom functions, we would like to apply the generic framework from Section 3.6 that allows to transform a unique-input-related-key secure pseudorandom function Minto a related-key secure one, F. Recall that the framework is simply the following: F keeps the same setup ($\mathsf{F.Setup} = \mathsf{M.Setup}$) and its outputs are defined as:

$$\mathsf{F.Eval}_{\mathsf{pp}}(K, x) = \mathsf{M.Eval}_{\mathsf{pp}}(K, H(x, M(K, \vec{\omega}))),$$

where H is a compatible collision-resistant hash function, and where $\vec{\omega}$ is a strong key fingerprint. Please refer to Chapter 3 and especially Sections 3.4 and 3.6 for further details.

Unfortunately, if we consider $\mathsf{WNR}^{\vec{w}}$ with some $w_i > 1$, then it is not clear how to find a strong key fingerprint, which can be used to apply the above framework. Furthermore, this framework requires to prove several non-algebraic properties (statistical or computational), namely key-collision, statistical-key-collision, and unique-input-related-key securities.

For this reason, we design a new algebraic framework, that generalizes this framework in the particular case of pseudorandom functions of the form $M(\vec{a}, x) = [P_x(\vec{a})]$ and considering classes of multivariate polynomials. For completeness, a more general framework, which does not make any assumptions about the outputs of the pseudorandom function, is also given in Section 6.2.5. Afterwards, we use our algebraic framework to design new related-key secure pseudorandom functions based on WNR for larger classes for which previous constructions are not secure.

6.2.2.1 An Algebraic Framework for Related-Key Security

Here, we describe a new framework that transforms any pseudorandom function of the above form that satisfies that $P_{\vec{\phi}_1,x_1}, \dots, P_{\vec{\phi}_q,x_q}$ are linearly independent, for any $\vec{\phi}_1, \dots, \vec{\phi}_q$ as long as $x_1, \dots, x_q$ are all distinct inputs, into a related-key secure pseudorandom function. To do so, we first introduce three new notions, termed *algebraic fingerprint*, *helper information*, and *expansion function*, and defined as follows.

Algebraic Fingerprint. In order to overcome the possible lack of a strong key fingerprint, we introduce the notion of algebraic fingerprint, which will be used to replace $M(K, \vec{\omega})$ from the previous framework, where $\vec{\omega}$ is a strong fingerprint. An algebraic fingerprint is simply an injective function $\vec{\Omega}$: $\mathbb{Z}_p^n \to \mathbb{G}^m$ such that the image $\vec{\Omega}(\vec{a})$ is a vector of group elements $([\Omega_1(\vec{a}) \cdot b], \dots, [\Omega_m(\vec{a}) \cdot b])$ with $\Omega_1, \dots, \Omega_m$ being polynomials in $\mathbb{Z}_p[T_1, \dots, T_n]$ and $b \in \mathbb{Z}_p$. In our applications, we will simply have $\vec{\Omega}(\vec{a}) = ([a_1], \dots, [a_n])$, so $m = n$ and $\Omega_i(\vec{T}) = T_i$ for $i = 1, \dots, n$.

Helper Information. In order to prove the security of our framework, we need to be able to compute the image of the algebraic fingerprint, $\vec{\Omega}(\vec{\phi}(\vec{a})) = ((\Omega_1 \circ \vec{\phi})(\vec{a}), \dots, (\Omega_m \circ \vec{\phi})(\vec{a}))$, for any related key $\vec{\phi}(\vec{a}) \in \mathbb{Z}_p^n$, with $\vec{\phi} \in \Phi$, from some information which can somehow be made public without hurting security. We call this information a helper information, write it $\mathsf{Help}_\Phi(\vec{a})$, and call Help_Φ the helper function. We suppose that $\mathsf{Help}_\Phi(\vec{a}) = ([\mathsf{help}_1(\vec{a}) \cdot b], \dots, [\mathsf{help}_\ell(\vec{a}) \cdot b])$, with $\mathsf{help}_1, \dots, \mathsf{help}_\ell$ linearly independent polynomials which generate a vector subspace of $\mathbb{Z}_p[T_1, \dots, T_n]$ containing the polynomials $\Omega_i \circ \vec{\phi}$ for $i = 1, \dots, m$, and $\vec{\phi} \in \Phi$.

Hash Function and Expansion Function. Let $\overline{\mathcal{D}} = \mathcal{D} \times \mathbb{G}^m$ where $\mathcal{D}$ is the domain of M, and let h be a collision-resistant hash function $h\colon \overline{\mathcal{D}} \to \mathsf{hSp}$, where hSp is a large enough space. The last thing we need to define is an expansion function, which is simply an injective function $\mathsf{E}\colon \mathsf{hSp} \to \mathcal{S} \subseteq \mathcal{D}$ such that for any sequence $(\vec{\phi}_1, x_1), \dots, (\vec{\phi}_q, x_q)$ where $x_1, \dots, x_q$ are distinct inputs in $\mathcal{S}$ and $\vec{\phi}_1, \dots, \vec{\phi}_q$ are RKD functions, polynomials $\mathsf{help}_1, \dots, \mathsf{help}_\ell$ and polynomials $P_{\vec{\phi}_1, x_1}, \dots, P_{\vec{\phi}_q, x_q}$ and 1 (which needs to be queried to define $[b]$) are linearly independent over $\mathbb{Z}_p$. In particular, E has to be injective.

Using these new tools, we obtain the following framework.

Theorem 6.2.3. *Let $M = (\mathsf{M.Setup}, \mathsf{M.Eval})$ be a pseudorandom function whose key space, domain, and range are respectively $\mathbb{Z}_p^n, \mathcal{D}$, and $\mathbb{G}$, where p and $\mathbb{G}$ are publicly revealed by $\mathsf{M.Setup}$, and such that $M(\vec{a}, x) = [P_x(\vec{a})]$. Let $\Phi \subseteq \mathbb{Z}_p[T_1, \dots, T_n]$ be a class of RKD functions. Let d be a upper bound for the maximum degree in any indeterminate of polynomials in $\{\mathsf{help}_1, \dots, \mathsf{help}_\ell\} \cup \{P_{x,\vec{\phi}} \mid x \in \mathcal{S}, \vec{\phi} \in \Phi\}$. Define $F = (\mathsf{F.Setup}, \mathsf{F.Eval})$ as $\mathsf{F.Setup} = \mathsf{M.Setup}$ and:*

$$\mathsf{F.Eval}_{\mathsf{pp}}(\vec{a}, x) = \mathsf{M.Eval}_{\mathsf{pp}}(\vec{a}, \mathsf{E}(h(x, \vec{\Omega}(\vec{a}))))$$

for all $\vec{a} \in \mathbb{Z}_p^n$ and $x \in \mathcal{D}$.

Then, assuming d-DDHI holds in $\mathbb{G}$ and that h is a collision-resistance hash function, F is a Φ-related-key secure pseudorandom function.

Moreover, the running time of this reduction is polynomial in n, d, q, and in the running time of the adversary.

Proof Overview. The proof of the above theorem is detailed below and relies on the sequence of 10 games (games $\mathbf{G}_0 - \mathbf{G}_9$) described in Figure 6.2 and on Lemma 6.2.4. We first prove an intermediate statement whose proof is very similar to the proof of Theorem 3.4.2, under a notion termed extended key-collision security (that states the hardness of finding key collisions given the helper information and oracle access to the pseudorandom function) which is defined below. Afterwards, we reduce this notion to the hardness of the d-SDL problem in $\mathbb{G}$ (which is trivially implied by the (n, d)-LIP security). Here we provide a brief overview of the proof of this intermediate statement.

We start by giving the generator along with $\mathbb{G}$ and p as public parameters for the pseudorandom function by querying polynomial 1. Hence, the generator is simply $[b]$. Since we may have key collisions (i.e., two RKD functions $\phi_1 \neq \phi_2$, such that $\phi_1(\vec{a}) = \phi_2(\vec{a})$), we start by dealing with possible collisions on the related keys in the game $\mathsf{RKPRFReal}_F$, using the extended key-collision notion (games $\mathbf{G}_0 - \mathbf{G}_2$). These claws can be detected by looking for collisions on images of $\vec{\Omega}$ for different RKD functions.

Then, in games $\mathbf{G}_3 - \mathbf{G}_4$, we deal with possible collisions on hash values in order to ensure that the inputs $t = \mathsf{E}(h(x, \vec{\Omega}(\vec{a})))$ used to compute the output y are distinct (recall that E is injective). Next, we use the (n, d)-LIP security notion to show that it is hard to distinguish the output of F and the helper information from uniformly random values (games $\mathbf{G}_5 - \mathbf{G}_6$).

Finally, we use once again the extended-key-collision security notion to deal with possible key collisions in the game $\mathsf{RKPRFRand}_F$ (games $\mathbf{G}_7 - \mathbf{G}_9$) so that $\mathbf{G}_9$ matches exactly the description of $\mathsf{RKPRFRand}_F$. These key collisions can still be detected in these games by making crucial use of the helper information.

Proof of Theorem 6.2.3. Let $\mathscr{A}$ denote an adversary against the Φ-related-key security of F. We assume without loss of generality that it never repeats a query. We first introduce a new security notion termed *extended* key-collision security, whose security game is depicted in Figure 6.1 below, which extends the key-collision security notion defined in Section 3.4.2 but where **Initialize** also leaks Help_Φ to the adversary.

proc Initialize	**proc RKFn**$(\overrightarrow{\phi}, x)$	**proc Finalize**$(\overrightarrow{\phi}_1, \overrightarrow{\phi}_2)$
$\overrightarrow{a} \xleftarrow{\$} \mathbb{Z}_p^n$	$y \leftarrow \mathsf{M.Eval}_{\mathsf{pp}}(\overrightarrow{\phi}(\overrightarrow{a}), x)$	Return $(\overrightarrow{\phi}_1 \neq \overrightarrow{\phi}_2$ and $\overrightarrow{\phi}_1(\overrightarrow{a}) = \overrightarrow{\phi}_2(\overrightarrow{a}))$
$\mathsf{pp} \xleftarrow{\$} \mathsf{M.Setup}(1^\kappa)$	Return y	
Return $(\mathsf{pp}, \mathsf{Help}_\Phi(\overrightarrow{a}))$		

Figure 6.1: Security game for extended Φ-key-collision of a pseudorandom function M for a class Φ

Then, we show that:

$$\mathsf{Adv}^{\mathsf{rka\text{-}prf}}_{\Phi,F}(\mathscr{A}) \;\leq\; \mathbf{Adv}^{(n,d)\text{-}\mathsf{lip}}_{\mathbb{G}}(\mathscr{B}) + \mathbf{Adv}^{\mathsf{cr}}_{H}(\mathscr{C}) + 2 \cdot \mathbf{Adv}^{\mathsf{ext\text{-}kc}}_{\Phi,M}(\mathscr{D}) \enspace . \tag{6.1}$$

Afterwards, we the extended key-collision security notion to the d-SDL assumption in $\mathbb{G}$, which is implied by the d-DDHI assumption. This proves our statement.

Proof of this Intermediate Statement. The proof is based on the sequence of games in Figure 6.2. We denote by $\textsc{Succ}_i$ the event that game $\mathbf{G}_i$ output takes the value 1.

$\mathbf{G}_0$ matches the description of the $\mathsf{RKPRFReal}_F$ game, so:

$$\Pr[\,\textsc{Succ}_0\,] = \Pr\left[\mathsf{RKPRFReal}^{\mathscr{A}}_F \Rightarrow 1\right] \enspace .$$

Game $\mathbf{G}_1$ introduces storage of used RKD functions and values of images $\overrightarrow{C}$ of $\overrightarrow{\Omega}$ in sets D and E respectively and sets flag_1 to true if the same value of $\overrightarrow{C}$ arises for two different RKD functions. Since this storage does not affect the values returned by **RKFn**

$$\Pr[\,\textsc{Succ}_0\,] = \Pr[\,\textsc{Succ}_1\,] \enspace .$$

Game $\mathbf{G}_2$ adds the boxed code which changes how the repetition of a value $\overrightarrow{C}$ is handled, by picking instead a random value from $\mathbb{G}^m \setminus E$ that will not repeat any previous one. Games $\mathbf{G}_1$ and $\mathbf{G}_2$ are identical until flag_1 is set to true, hence we have

$$\Pr[\,\textsc{Succ}_1\,] \leq \Pr[\,\textsc{Succ}_2\,] + \Pr[\,E_1\,] \enspace ,$$

where E_1 denotes the event that the execution of $\mathscr{A}$ with game $\mathbf{G}_1$ sets flag_1 to true. We design an adversary $\mathscr{D}$ attacking the extended Φ-key-collision security of M such that

$$\Pr[\,E_1\,] \leq \mathbf{Adv}^{\mathsf{ext\text{-}kc}}_{\Phi,M}(\mathscr{D}) \enspace .$$

Adversary $\mathscr{D}$ gets helper information $\mathsf{help}_\Phi = \mathsf{Help}_\Phi(\overrightarrow{a})$, then runs $\mathscr{A}$. When the latter makes an **RKFn**-query $(\overrightarrow{\phi}, x)$, adversary $\mathscr{D}$ computes $\overrightarrow{C} = \overrightarrow{\Omega}(\overrightarrow{\phi}(\overrightarrow{a}))$ using its helper information, $z = h(x, \overrightarrow{C})$ and $t = \mathsf{E}(z)$ and finally queries $(\overrightarrow{\phi}, t)$ to its oracle and sends the value it gets to $\mathscr{A}$. When $\mathscr{A}$ halts, $\mathscr{D}$ searches for two different RKD functions $\overrightarrow{\phi}_1, \overrightarrow{\phi}_2$ queried by $\mathscr{A}$ that lead to the same value $\overrightarrow{C}$ and returns these two functions if found. Since $\overrightarrow{\Omega}$ is a Algebraic Fingerprint, such two functions lead to the same key, so $\mathscr{D}$ wins if he finds such two functions.

```
proc Initialize // $\mathbf{G}_0$
$\vec{a} \xleftarrow{\$} \mathbb{Z}_p^n$

proc RKFn($\vec{\phi}, x$) // $\mathbf{G}_0$
$\vec{C} \leftarrow \vec{\Omega}(\vec{\phi}(\vec{a}))$
$z \leftarrow h(x, \vec{C})$
$t \leftarrow \mathsf{E}(z)$
$y \leftarrow M(\vec{\phi}(\vec{a}), t)$
Return $y$

proc Finalize(b') // All Games
Return $b'$
```

```
proc Initialize // $\mathbf{G}_1, \mathbf{G}_2$
$\vec{a} \xleftarrow{\$} \mathbb{Z}_p^n$ ; $D \leftarrow \emptyset$ ; $E \leftarrow \emptyset$

proc RKFn($\vec{\phi}, x$) // $\mathbf{G}_1$, $\boxed{\mathbf{G}_2}$
$\vec{C} \leftarrow \vec{\Omega}(\vec{\phi}(\vec{a}))$
If $\vec{C} \in E$ and $\vec{\phi} \notin D$ then
$\mathsf{flag}_1 \leftarrow \mathsf{true}$ ; $\boxed{\vec{C} \xleftarrow{\$} \mathbb{G}^m \setminus E}$
$D \leftarrow D \cup \{\vec{\phi}\}$ ; $E \leftarrow E \cup \{\vec{C}\}$
$z \leftarrow h(x, \vec{C})$
$t \leftarrow \mathsf{E}(z)$
$y \leftarrow M(\vec{\phi}(\vec{a}), t)$
Return $y$
```

```
proc Initialize // $\mathbf{G}_3, \mathbf{G}_4$
$\vec{a} \xleftarrow{\$} \mathbb{Z}_p^n$ ; $D \leftarrow \emptyset$ ; $E \leftarrow \emptyset$ ; $G \leftarrow \emptyset$

proc RKFn($\vec{\phi}, x$) // $\mathbf{G}_3$, $\boxed{\mathbf{G}_4}$
$\vec{C} \leftarrow \vec{\Omega}(\vec{\phi}(\vec{a}))$
If $\vec{C} \in E$ and $\vec{\phi} \notin D$ then $\vec{C} \xleftarrow{\$} \mathbb{G}^m \setminus E$
$D \leftarrow D \cup \{\vec{\phi}\}$ ; $E \leftarrow E \cup \{\vec{C}\}$
$z \leftarrow h(x, \vec{C})$
If $z \in G$ then $\mathsf{flag}_2 \leftarrow \mathsf{true}$
    $\boxed{z \xleftarrow{\$} \mathsf{hSp} \setminus G}$
$G \leftarrow G \cup \{z\}$
$t \leftarrow \mathsf{E}(z)$
$y \leftarrow M(\vec{\phi}(\vec{a}), t)$
Return $y$
```

```
proc Initialize // $\mathbf{G}_5$
$\vec{a} \xleftarrow{\$} \mathbb{Z}_p^n$ ; $D \leftarrow \emptyset$ ; $E \leftarrow \emptyset$ ; $G \leftarrow \emptyset$

proc RKFn($\vec{\phi}, x$) // $\mathbf{G}_5$
$\vec{C} \leftarrow \vec{\Omega}(\vec{\phi}(\vec{a}))$
If $\vec{C} \in E$ and $\vec{\phi} \notin D$
    then $\vec{C} \xleftarrow{\$} \mathbb{G}^m \setminus E$
$D \leftarrow D \cup \{\vec{\phi}\}$ ; $E \leftarrow E \cup \{\vec{C}\}$
$z \leftarrow h(x, \vec{C})$
If $z \in G$ then $z \xleftarrow{\$} \mathsf{hSp} \setminus G$
$G \leftarrow G \cup \{z\}$
$t \leftarrow \mathsf{E}(z)$
$y \xleftarrow{\$} \mathbb{G}$
Return $y$
```

```
proc Initialize // $\mathbf{G}_6$
$\vec{a} \xleftarrow{\$} \mathbb{Z}_p^n$ ; $D \leftarrow \emptyset$ ; $E \leftarrow \emptyset$
    $G \xleftarrow{\$} \mathsf{Fun}(\mathbb{Z}_p^n, \mathcal{D}, \mathbb{G})$

proc RKFn($\vec{\phi}, x$) // $\mathbf{G}_6$
$y \xleftarrow{\$} \mathbb{G}$
Return $y$
```

```
proc Initialize // $\mathbf{G}_9$
$\vec{a} \xleftarrow{\$} \mathbb{Z}_p^n$ ; $G \xleftarrow{\$} \mathsf{Fun}(\mathbb{Z}_p^n, \mathcal{D}, \mathbb{G})$

proc RKFn($\vec{\phi}, x$) // $\mathbf{G}_9$
$y \leftarrow G(\vec{\phi}(\vec{a}), x)$
Return $y$
```

```
proc Initialize // $\mathbf{G}_7$, $\mathbf{G}_8$
$\vec{a} \xleftarrow{\$} \mathbb{Z}_p^n$ ; $D \leftarrow \emptyset$ ; $E \leftarrow \emptyset$
    $G \xleftarrow{\$} \mathsf{Fun}(\mathbb{Z}_p^n, \mathcal{D}, \mathbb{G})$

proc RKFn($\vec{\phi}, x$) // $\boxed{\mathbf{G}_7}$, $\mathbf{G}_8$
If $\vec{\phi}(\vec{a}) \in E$ and $\vec{\phi} \notin D$ then
    $\boxed{y \xleftarrow{\$} \mathbb{G}}$ ; $\mathsf{flag}_3 \leftarrow \mathsf{true}$
$\boxed{\text{else}}$ $y \leftarrow G(\vec{\phi}(\vec{a}), x)$
$D \leftarrow D \cup \{\vec{\phi}\}$ ; $E \leftarrow E \cup \{\vec{\phi}(\vec{a})\}$
Return $y$
```

Figure 6.2: Games for the proof of Theorem 6.2.3

Game $\mathbf{G}_3$ introduces the storage of hash values in a set G and sets flag_2 to true if the same hash output arises twice. Since this storage does not affect the values returned by **RKFn**, we have

$$\Pr[\,\textsc{Succ}_2\,] = \Pr[\,\textsc{Succ}_3\,] \ .$$

Game $\mathbf{G}_4$ adds the boxed code which changes how repetition of hash values is handled, by picking instead a random value z from $\mathsf{hSp} \setminus G$ that will not repeat any previously used hash value. Games $\mathbf{G}_3$ and $\mathbf{G}_4$ are identical until flag_2 is set to true, hence we have

$$\Pr[\,\text{Succ}_3\,] \le \Pr[\,\text{Succ}_4\,] + \Pr[\,E_2\,]\ ,$$

where E_2 denotes the event that the execution of $\mathscr{A}$ with game $\mathbf{G}_3$ sets flag_2 to true. We design an adversary $\mathscr{C}$ attacking the collision-resistance security of h such that

$$\Pr[\,E_2\,] \le \mathbf{Adv}_h^{\mathsf{cr}}(\mathscr{C})\ .$$

Adversary $\mathscr{C}$ starts by picking $\vec{a} \xleftarrow{\$} \mathbb{Z}_p^n$ and initializes $j \leftarrow 0$. It runs $\mathscr{A}$. When the latter makes an **RKFn**-query $(\vec{\phi}, x)$, adversary $\mathscr{C}$ responds via

```
$\vec{C} \leftarrow \vec{\Omega}(\vec{\phi}(\vec{a}))$
$j \leftarrow j+1$ ; $\vec{\phi}_j \leftarrow \vec{\phi}$ ; $x_j \leftarrow x$
If $\vec{C} \in E$ and $\vec{\phi} \notin D$ then $\vec{C} \xleftarrow{\$} \mathbb{G}^m \setminus E$          $(*)$
$D \leftarrow D \cup \{\vec{\phi}\}$ ; $E \leftarrow E \cup \{\vec{C}\}$
$\vec{C}_j \leftarrow \vec{C}$
$z \leftarrow h(x, \vec{C})$
$z_j \leftarrow z$
$t \leftarrow \mathsf{E}(z)$ $y \leftarrow M(\vec{\phi}(\vec{a}), t)$
Return $y$.
```

When $\mathscr{A}$ halts, $\mathscr{C}$ searches for a, b satisfying $1 \le a < b \le j$ such that $z_a = z_b$ and, if it finds them, outputs $(x_a, \vec{C}_a), (x_b, \vec{C}_b)$ and halts. The pairs $(x_a, \vec{C}_a)$ and $(x_b, \vec{C}_b)$ are distinct. Indeed, consider two cases: first, if $\vec{\phi}_a = \vec{\phi}_b$ then since $\mathscr{A}$ never repeats an oracle query, $x_a \ne x_b$ hence $(x_a, \vec{C}_a) \ne (x_b, \vec{C}_b)$. Second, if $\vec{\phi}_a \ne \vec{\phi}_b$, then condition $(*)$ ensures that $\vec{C}_a \ne \vec{C}_b$. Hence once again, $(x_a, \vec{C}_a) \ne (x_b, \vec{C}_b)$, and then

$$\Pr[\,\text{Succ}_3\,] \le \Pr[\,\text{Succ}_4\,] + \mathbf{Adv}_h^{\mathsf{cr}}(\mathscr{C})\ .$$

In game $\mathbf{G}_5$, instead of returning the value $M(\vec{\phi}(\vec{a}), t)$, we always return a random value. To show that games $\mathbf{G}_4$ and $\mathbf{G}_5$ are indistinguishable, we design an adversary $\mathscr{B}$ against the (n,d)-LIP security of $\mathbb{G}$ such that

$$\Pr[\,\text{Succ}_4\,] \le \Pr[\,\text{Succ}_5\,] + \mathbf{Adv}_{\mathbb{G}}^{(n,d)\text{-}\mathsf{lip}}(\mathscr{B})\ .$$

Adversary $\mathscr{B}$ starts by querying polynomial 1 to $[b]$ and returns it as the generator used for the PRF to $\mathscr{A}$. Next, it initializes sets $D \leftarrow \emptyset$, $E \leftarrow \emptyset$, $G \leftarrow \emptyset$. Then $\mathscr{B}$ queries $\mathsf{help}_1, \ldots, \mathsf{help}_\ell$ such that $\mathsf{Help}_\Phi(\vec{a}) = ([\mathsf{help}_1(\vec{a}) \cdot b], \ldots, [\mathsf{help}_\ell(\vec{a}) \cdot b])$. Afterwards, it runs $\mathscr{A}$. When the latter makes an **RKFn**-query $(\vec{\phi}, x)$, $\mathscr{B}$ responds as follows. First, it computes $\vec{C} = \vec{\Omega}(\vec{\phi}(\vec{a}))$ using its helper information. Then, $\mathscr{B}$ checks if $\vec{C} \in E$ and $\vec{\phi} \in D$. If they do, $\mathscr{B}$ picks $\vec{C} \xleftarrow{\$} \mathbb{G}^m \setminus E$ at random. $\mathscr{B}$ then sets $D \leftarrow D \cup \{\vec{\phi}\}$ and $E \leftarrow E \cup \{\vec{C}\}$. Next, $\mathscr{B}$ computes $z \leftarrow h(x, \vec{C})$ and checks if $z \in G$. If it does, $\mathscr{B}$ picks $z \xleftarrow{\$} \mathsf{hSp} \setminus G$ at random. Notice that this step guarantees that all values z are distinct as long as $\mathscr{A}$ makes at most $|\mathsf{hSp}|$ queries. Finally, $\mathscr{B}$ sets $G \leftarrow G \cup \{z\}$, computes $t = \mathsf{E}(z)$ and makes the query $P_{\vec{\phi},t}$ to its oracle, where $P_{\vec{\phi},t}$ is the polynomial such that $M(\vec{\phi}(\vec{a}), t) = \left[P_{\vec{\phi},t}(\vec{a})\right]$, and returns the value it gets, which is either $\left[P_{\vec{\phi},t}(\vec{a}) \cdot b\right]$ or a uniformly random value, to $\mathscr{A}$. When $\mathscr{A}$ halts, $\mathscr{B}$ halts with the same output. It is clear that all the polynomials queried by $\mathscr{B}$ are linearly independent via the definition of E. Finally, it is clear that if $\mathscr{B}$'s oracle answers to a query P by the value $[P(\vec{a}) \cdot b]$, then it simulates exactly game $\mathbf{G}_4$ (where the generator used for the PRF construction is $[b]$, and if $\mathscr{B}$'s oracle gives uniformly random values, then the outputs are correctly simulated, since they are uniformly random. This concludes the proof of the above statement.

In game $\mathbf{G}_6$, we simply set the value y to a uniformly random value. Clearly, $\mathbf{G}_5$ and $\mathbf{G}_6$ are identical since the value returned is a uniformly random value for any query. Then, we have

$$\Pr[\,\textsc{Succ}_5\,] = \Pr[\,\textsc{Succ}_6\,] \ .$$

In game $\mathbf{G}_7$, we check if two different queries can lead to a key collision. Since the "If" test ensures that the returned value is still uniformly random over $\mathbb{G}$ even when two different queries result in the same key, games $\mathbf{G}_6$ and $\mathbf{G}_7$ are identical. Hence,

$$\Pr[\,\textsc{Succ}_6\,] = \Pr[\,\textsc{Succ}_7\,] \ .$$

In game $\mathbf{G}_8$, we compute the output of **RKFn** using a random function G in $\mathsf{Fun}(\mathbb{Z}_p^n, \mathcal{D}, \mathbb{G})$. Since games $\mathbf{G}_7$ and $\mathbf{G}_8$ are identical until flag_3 is set to true, we have

$$\Pr[\,\textsc{Succ}_7\,] \leq \Pr[\,\textsc{Succ}_8\,] + \Pr[\,E_3\,] \ ,$$

where E_3 denotes the event that the execution of $\mathscr{A}$ with game $\mathbf{G}_8$ sets flag_3 to true. To bound the probability of event E_3, we design an adversary $\mathscr{D}$ attacking the extended Φ-key-collision security of M such that

$$\Pr[\,E_3\,] \leq \mathbf{Adv}^{\mathsf{ext\text{-}kc}}_{\Phi,M}(\mathscr{D}) \ .$$

Adversary $\mathscr{D}$ starts by initializing a list $\mathcal{L} \leftarrow$ empty list and by choosing an element $\vec{\psi}$ in Φ and by setting $\vec{\psi}_1 \leftarrow \vec{\psi}$ and $\vec{\psi}_2 \leftarrow \vec{\psi}$. Then, it runs $\mathscr{A}$. When the latter makes an **RKFn**-query $(\vec{\phi}, x)$, if $\vec{\psi}_1 = \vec{\psi}_2$, adversary $\mathscr{D}$ does the following: it first computes $\vec{C} \leftarrow \Omega(\vec{\phi}(\vec{a}))$ using its helper information and searches for all tuples $(\vec{\phi}_i, \vec{C}_i) \in \mathcal{L}$ such that $\vec{C}_i = \vec{C}$. If it does find such tuples, it checks for all of them if $\vec{\phi} \neq \vec{\phi}_i$. If it does for a certain i, it sets $\vec{\psi}_1 \leftarrow \vec{\phi}_i$ and $\vec{\psi}_2 \leftarrow \vec{\phi}$. Then, it adds $(\vec{\phi}, \vec{C})$ to $\mathcal{L}$. Finally, $\mathscr{D}$ picks $y \xleftarrow{\$} \mathbb{G}$ at random, returns y to $\mathscr{A}$. When $\mathscr{A}$ halts, $\mathscr{D}$ halts and outputs $(\vec{\psi}_1, \vec{\psi}_2)$. If the execution of $\mathscr{A}$ sets flag_3 to true, then $\mathscr{A}$ has queried $\vec{\phi}_1 \neq \vec{\phi}_2$ such that $\vec{\phi}_1(\vec{a}) = \vec{\phi}_2(\vec{a})$ and assuming it has first queried $\vec{\phi}_1$, when $\mathscr{D}$ computes $\Omega(\vec{\phi}_2(\vec{a}))$ and checks if this value is already in $\mathcal{L}$, it finds that this value matches $\Omega(\vec{\phi}_1(\vec{a}))$ and since $\vec{\phi}_1 \neq \vec{\phi}_2$, it sets $\vec{\psi}_1 = \vec{\phi}_1$ and $\vec{\psi}_2 = \vec{\phi}_2$, so $\mathscr{D}$ wins. Then, we have

$$\Pr[\,E_3\,] \leq \mathbf{Adv}^{\mathsf{ext\text{-}kc}}_{\Phi,M}(\mathscr{D}) \ .$$

Since A does not repeat oracle queries and since key collisions are dealt with in a similar way, it follows that games $\mathbf{G}_8$ and $\mathbf{G}_9$ are identical. Thus,

$$\Pr[\,\textsc{Succ}_8\,] = \Pr[\,\textsc{Succ}_9\,] \ .$$

Finally, $\mathbf{G}_9$ matches the description of the $\mathsf{RKPRFRand}_F$ game, so:

$$\Pr[\,\textsc{Succ}_9\,] = \Pr\left[\mathsf{RKPRFRand}_F^{\mathscr{A}} \Rightarrow 1\right] \ .$$

Equation (6.1) now follows by combining the bounds arising in the different game hops. □

From Extended-Key-Collision Security to SDL. In the following lemma, we reduce the extended-key-collision security to the hardness of the d-SDL problem in $\mathbb{G}$.

Lemma 6.2.4. *Let d be an integer such that, for any polynomial in $\{\mathsf{help}_1, \dots, \mathsf{help}_\ell\} \cup \{P_{x,\vec{\phi}} \mid x \in \mathcal{S}, \vec{\phi} \in \Phi\}$, its maximum degree in any indeterminate is at most d. Then, assuming d-SDL holds in $\mathbb{G}$, so does the extended Φ-key-collision security of M.*

Moreover, the running time of this reduction is polynomial in n, d, q, and in the running time of the adversary.

Proof of Lemma 6.2.4. Adversary $\mathscr{B}$ receives a d-SDL tuple $([1], [a], \dots, \left[a^d\right])$ where $a \xleftarrow{\$} \mathbb{Z}_p$ and where $[1]$. Adversary $\mathscr{B}$ then picks $j \xleftarrow{\$} \{1, \dots, n\}$ at random.

$\mathscr{B}$ start by picking b at random in $\mathbb{Z}_p$ and by sending $[b]$ to $\mathscr{A}$ as the generator being part of public parameters of the PRF construction. Then $\mathscr{B}$ picks $a_i \xleftarrow{\$} \mathbb{Z}_p$ for $i = 1, \dots, n, i \neq j$ at random. Implicitly, $\mathscr{B}$ sets $a_j = a$.

Then, $\mathscr{B}$ computes the helper information $\mathsf{Help}_\Phi(\vec{a}) = ([\mathsf{help}_1(\vec{a}) \cdot b], \dots, [\mathsf{help}_\ell(\vec{a}) \cdot b])$ using its d-SDL tuple and the known values b, a_i for $i \neq j$ and sends this helper information to $\mathscr{A}$. When $\mathscr{A}$ makes a query $(\vec{\phi}, x)$, $\mathscr{B}$ wants to return to $\mathscr{A}$ the value $M(\vec{\phi}(\vec{a}), x) = \left[P_x(\vec{\phi}(\vec{a})) \cdot b\right] = \left[P_{\vec{\phi},x}(\vec{a}) \cdot b\right]$.

To do so, $\mathscr{B}$ starts by computing $P_{\vec{\phi},x}(a_1, \dots, a_{j-1}, T_j, a_{j+1}, \dots, a_n)$ using chosen values $a_i, i \neq j$. It then gets a degree at most d polynomial P_j in T_j. Hence, using its d-SDL tuple and the known value b, it can now easily compute $[P_j(a_j) \cdot b] = \left[P_{\vec{\phi},x}(\vec{a}) \cdot b\right]$ and returns this value to $\mathscr{A}$.

At the end, $\mathscr{A}$ sends $(\vec{\phi}_1, \vec{\phi}_2)$ to $\mathscr{B}$ and wins if $\vec{\phi}_1 \neq \vec{\phi}_2$ and $\vec{\phi}_1(\vec{a}) = \vec{\phi}_2(\vec{a})$. Hence, $\mathscr{B}$ computes $\vec{\psi} = \vec{\phi}_1 - \vec{\phi}_2$. Since $\vec{\phi}_1 \neq \vec{\phi}_2$, there is at least a component of $\vec{\psi}$ which is a non-zero multivariate polynomial. Let us call this component $R \in \mathbb{Z}_p[T_1, \dots, T_n]$. Since we chose j at random in $\{1, \dots, n\}$, the indeterminate T_j appears in R with probability at least $\frac{1}{n}$. $\mathscr{B}$ now evaluates R in $(a_1, \dots, a_{j-1}, T_j, a_{j+1}, \dots, a_n)$ to obtain a univariate polynomial $S \in \mathbb{Z}_p[T_j]$. This polynomial is a non-zero polynomial with probability at least $\frac{p-(n-1)d}{p}$, via the Schwartz-Zippel lemma.

Finally, $\mathscr{B}$ simply factorizes S (for instance using the Kedlaya-Umans algorithm), and outputs the unique root r of S such that $[r] = [a]$. Hence, we have:

$$\mathbf{Adv}^{\mathsf{ext\text{-}kc}}_{\Phi,M}(\mathscr{A}) \leq \frac{p}{p-(n-1)d} \cdot n \cdot \mathsf{Adv}^{d\text{-}\mathsf{sdl}}_{\mathbb{G}}(\mathscr{B}) , \tag{6.2}$$

and Lemma 6.2.4 follows. □

Theorem 6.2.3 now follows from combining Equation (6.1), Equation (6.2), and from the LIP theorem (and from the fact that the hardness of d-SDL problem is implied by the $\mathcal{E}_{1,d}$-MDDH assumption). □

6.2.2.2 Related-Key Security for Permutations of Univariate Polynomials

We now apply our framework to a particular case of WNR and build the first related-key secure pseudorandom function for the class of permutations of univariate polynomials. We choose to set w_0 to 0 in our construction in order to ease the readability so that the key space of the pseudorandom function stays $\mathbb{Z}_p^n$, but similar results can be proven with $w_0 = 1$ or set to a prime number $p_0 > d$ (and distinct to $p_1, \dots, p_n$ defined below).

For $d \geq 1$, let Φ_d be the class of degree at most d non-constant univariate polynomials defined as $\Phi_d = \{\vec{\phi}\colon \mathbb{Z}_p^n \to \mathbb{Z}_p^n \mid \phi_i : \vec{T} \mapsto \sum_{j=0}^{d} \alpha_{i,j} T_i^j, (\alpha_{i,1}, \dots, \alpha_{i,d}) \neq 0^d, \forall i = 1, \dots, n\}$.

Then we consider the class $\Phi_{\mathfrak{S}_n,d}$ of permutations of degree at most d non-constant univariate polynomials, defined as follows:

$$\Phi_{\mathfrak{S}_n,d} = \{\sigma \circ \vec{\phi} \mid (\sigma, \vec{\phi}) \in \mathfrak{S}_n \times \Phi_d\} \ .$$

For a key $\vec{a} = (a_1, \ldots, a_n) \in \mathbb{Z}_p^n$, applying an RKD function $\sigma \circ \vec{\phi} \in \Phi_{\mathfrak{S}_n,d}$, where $\vec{\phi} = (\phi_1, \ldots, \phi_n)$ leads to the key $(\phi_{\sigma^{-1}(1)}(\vec{a}), \ldots, \phi_{\sigma^{-1}(n)}(\vec{a})) \in \mathbb{Z}_p^n$, so the i-th component a_i of the key is changed into $\phi_i(\vec{a})$ and becomes the $\sigma(i)$-th component of the related key.

Before explaining our construction, we would like to point out that, even if we just consider the simple class of permutations $\Phi_{\mathfrak{S}_n} \subset \Phi_{\mathfrak{S}_n,1}$ introduced in Section 6.2.1, we can already show that NR and NR^* are not $\Phi_{\mathfrak{S}_n}$-related-key secure, even with respect to unique-input adversaries. Indeed, let us consider NR^*: let id be the identity function and (12) be the permutation which switches the first two components of the key. Then, the output of the queries $(\mathsf{id}, 100\ldots0)$ and $((12), 010\ldots0)$ will be the same in the real case and independent in the random case.

In fact, we can generalize the attack above to show that there even exists a compatible collision-resistant hash function h such that the pseudorandom function that one obtains when applying the Bellare-Cash transform (or one of the transforms from Chapter 3) to NR^* would not be related-key secure with respect to the class of permutations. Indeed, let h' be a collision-resistant hash function. The counter-example for h could be as follows (where x_1 and x_2 are two arbitrary distinct inputs):

$$h(x, [a_1], \ldots, [a_n]) = \begin{cases} 1110 \,\|\, h'(x_1, [a_1], \ldots, [a_n]) & \text{if } x = x_1 \\ 1101 \,\|\, h'(x_1, [a_2], [a_1], [a_3], \ldots, [a_n]) & \text{if } x = x_2 \\ 1111 \,\|\, h'(x, [a_1], \ldots, [a_n]) & \text{otherwise.} \end{cases}$$

Note that h is a compatible collision-resistant hash function. It is easy to see that the output of the queries (id, x_1) and $((12), x_2)$ will be the same in the real case and independent in the random case. The same kind of attack can be mounted against NR.

However, while NR and NR^* are not related-key secure against permutations attacks, we show in what follows that a particular case of WNR, defined below, yields a $\Phi_{\mathfrak{S}_n,d}$-related-key secure pseudorandom function.

Linear WNR. Let $d \geq 1$. Let $p_1 < p_2 < \cdots < p_n$ be distinct prime numbers such that $p_1 > d$. We define $\mathsf{WNR}^{d\text{-}\mathsf{lin}}$ as the particular case of WNR, where $w_0 = 0$ and $w_i = p_i$. Please refer to Section 6.1.1.2 for details. Using standard inequalities over prime numbers, it is easy to see that we can find $p_1, \ldots, p_n$ such that $p_n = \tilde{O}(d + n)$.

In order to apply the framework from Theorem 6.2.3 to $\mathsf{WNR}^{d\text{-}\mathsf{lin}}$ and $\Phi_{\mathfrak{S}_n,d}$, we define:

- $[b] \in \mathbb{G}$ denote the generator output by the setup procedure along with $\mathbb{G}$ and p
- $\vec{\Omega}$: $\vec{a} \in \mathbb{Z}_p^n \mapsto ([a_1], \ldots, [a_n]) \in \mathbb{G}^n$
- $\mathsf{Help}_{\Phi_{\mathfrak{S}_n,d}}$: $\vec{a} \in \mathbb{Z}_p^n \mapsto ([1], [a_1], \ldots, \left[a_1^d\right], \ldots, [a_n], \ldots, \left[a_n^d\right]) \in \mathbb{G}^{nd+1}$
- h can be any collision-resistant hash function h: $\{0,1\}^n \times \mathbb{G}^n \to \{0,1\}^{n-2}$
- E: $z \in \{0,1\}^{n-2} \mapsto 11 \,\|\, z \in \{0,1\}^n$

We just need to prove that E satisfies the linear independence property required to apply the framework, which is done below, and sketched here. We order monomials of multivariate polynomials, with any order respecting the total degree of polynomials (e.g., the graded lexicographic order). The leading monomial (i.e., the first monomial for that order) of the polynomial $P_{\vec{\phi},x}$ is $T_1^{x_{\sigma(1)}p_{\sigma(1)}d_1} \cdots T_n^{x_{\sigma(n)}p_{\sigma(n)}d_n}$, with $d_i > 0$ the degree of ϕ_i. The polynomials for the helper information (help_k) are T_i^j. Therefore, the leading monomials of $\mathsf{help}_1, \dots, \mathsf{help}_\ell$, $P_{\vec{\phi_1},x_1}, \dots, P_{\vec{\phi_q},x_q}, 1$ are all distinct, when $x_1, \dots, x_q$ are distinct inputs. This means that the matrix whose columns correspond to monomials (ordered as specified above) and whose rows correspond to the polynomials $\mathsf{help}_1, \dots, \mathsf{help}_\ell, P_{\vec{\phi_1},x_1}, \dots, P_{\vec{\phi_q},x_q}, 1$ (ordered according to their leading monomial) is in echelon form. Hence, the latter polynomials are linearly independent. Finally, using Theorem 6.2.3, we obtain the following result.

Lemma 6.2.5. *Assuming $p_n d$-DDHI holds in $\mathbb{G}$ and h is a collision-resistant hash function, the construction obtained with the above ingredients and Theorem 6.2.3 is a $\Phi_{\mathfrak{S}_n,d}$-related-key secure pseudorandom function.*

Moreover, the running time of the reduction is polynomial in n, d, p_n, and in the running time of the adversary.

Proof of Lemma 6.2.5. Let $P_{\sigma\circ\vec{\phi},x}(\vec{T}) = \prod_{i=1}^n \phi_i(\vec{T})^{p_{\sigma(i)}\cdot x_{\sigma(i)}}$ for $\sigma \circ \vec{\phi} \in \Phi_{\mathfrak{S}_n,d}$ and $x \in \{0,1\}^n$. Let $\mathcal{S} = \{11 \,\|\, z \mid z \in \{0,1\}^{n-2}\}$. The only thing we need to prove is that, for any sequence $(x_1, \sigma_1 \circ \vec{\phi}_1), \dots, (x_q, \sigma_q \circ \vec{\phi}_q)$, polynomials 1, T_i^j for $i = 1, \dots, n$ and $j = 1, \dots, d$ (revealed in the helper information) and polynomials $P_{\sigma_1\circ\vec{\phi}_1,x_1}, \dots, P_{\sigma_q\circ\vec{\phi}_q,x_q}$ are linearly independent, as long as $x_1, \dots, x_q$ are all distinct in $\mathcal{S}$.

By contradiction, let us assume that there exists a sequence of distinct polynomials $P_1, \dots, P_q$, where for $k = 1, \dots, q$, $P_k = P_{\sigma\circ\vec{\phi},x}$ for distinct $x \in \mathcal{S}$ or $P_k = T_i^j$ for some $(i,j) \in \{1, \dots, n\} \times \{1, \dots, d\}$ or $P_k = 1$, such that:

$$\sum_{k=1}^{q} \lambda_k \cdot P_k = 0$$

with $\lambda_k \neq 0$ for all k. Since polynomials T_i^j, for $i = 1, \dots, n$ and $j = 1, \dots, d$ and polynomial 1 are clearly linearly independent over $\mathbb{Z}_p$, then there is at least one polynomial P_k in the above sum such that P_k corresponds to $P_{\sigma\circ\vec{\phi},x}$ for a query $(\sigma \circ \vec{\phi}, x)$ with $x \in \mathcal{S}$.

Let $P_{\sigma\circ\vec{\phi},x}(\vec{T})$ denote a polynomial in the above sum such that $x \in \mathcal{S}$ has the maximum Hamming weight amongst all the bitstrings $z \in \mathcal{S}$ such that there exists $k \in \{1, \dots, q\}$ and $(\sigma \circ \vec{\phi}) \in \Phi_{\mathfrak{S}_n,d}$ such that $P_k = P_{\sigma\circ\vec{\phi},z}$. Since $x \in \mathcal{S}$, $\mathsf{hw}(x) \geq 2$. Since ϕ_i is a non-constant polynomial for all $i = 1, \dots, n$, letting d_i denote the degree of ϕ_i, the monomial $\prod_{i=1}^n T_i^{d_i p_{\sigma(i)}\cdot x_{\sigma(i)}}$ appears in $P_{\sigma\circ\vec{\phi},x}(\vec{T})$. Hence, since $\sum_{k=1}^q \lambda_k \cdot P_k$ is the zero polynomial and $\lambda_k \neq 0$ for all k, there exists another query $P_{\sigma'\circ\vec{\phi}',x'}$ such that the monomial $\prod_{i=1}^n T_i^{d_i p_{\sigma(i)}\cdot x_{\sigma(i)}}$ also appears in $P_{\sigma'\circ\vec{\phi}',x'}$. Hence $\mathsf{hw}(x') \geq \mathsf{hw}(x)$.

But x has maximum Hamming weight by assumption, so $\mathsf{hw}(x') \leq \mathsf{hw}(x)$ and then $\mathsf{hw}(x') = \mathsf{hw}(x)$. We note that the degree in T_i in this monomial is either 0, if $x_{\sigma(i)} = 0$, or $d_i p_{\sigma(i)}$ otherwise, where $1 \leq d_i \leq d$. But by definition of $P_{\sigma'\circ\vec{\phi}',x}$, the possible degrees in T_i in a monomial that appears in $P_{\sigma'\circ\vec{\phi}',x}$ is either 0 if $x'_{\sigma'(i)} = 0$ or a non-zero multiple $l \cdot p_{\sigma'(i)}$ of $p_{\sigma'(i)}$ with $1 \leq l \leq d$ otherwise.

However, $p_1, \ldots, p_n$ are clearly coprime and $d < p_1 < \ldots < p_n$ by assumption. Hence, for all $i = 1, \ldots, n$ such that $x_{\sigma(i)} = 1$, we have, by Gauss's Lemma, $p_{\sigma'(i)} = p_{\sigma(i)}$ and $x_{\sigma(i)} = 1$ implies $x'_{\sigma'(i)} = 1$. Finally, since $p_{\sigma'(i)} = p_{\sigma(i)}$ implies $\sigma'(i) = \sigma'(i)$ for all i such that $x_{\sigma(i)} = 1$, and since they have both same Hamming weight, it implies that $x = x'$ and then to have such a linear combination, we need to use twice the same entry $x \in \mathcal{S}$, which is not possible.

The linear independence property follows, and so does Lemma 6.2.5. $\square$

6.2.3 Other Applications to Related-Key Security

Here, we describe how our new framework can be used to build related-key secure pseudorandom functions for two other classes. The first class we address, in Section 6.2.3.1, is the class Φ_d of degree at most d univariate polynomials. We use our framework to show that for any choice of weights, WNR is Φ_d-related-key secure. The second class we address, in Section 6.2.3.2, is the class $\Phi_{n+1,\mathsf{multi\text{-}aff}}$ of non-constant affine multivariate functions. However, this construction is of limited interest and should only be seen as a first step towards building related-key secure pseudorandom functions for large classes of RKD functions. All the proofs of statements are detailed in Section 6.2.3.3.

6.2.3.1 Related-Key Security for Univariate Polynomials

Here, we apply our framework to WNR, for the class of RKD functions $\Phi_d = \{\vec{\phi}\colon \mathcal{K} \to \mathcal{K} \mid \vec{\phi}_i : \vec{T} \mapsto \sum_{j=0}^{d} \alpha_{i,j} \cdot T_i^j, (\alpha_{i,1}, \ldots, \alpha_{i,d}) \neq 0^d, \forall i = 0, \ldots, n\}$, for any choice of weights. In what follows, we assume that $w_0 \neq 0$, but similar results can easily be proven if $w_0 = 0$.

In order to apply our framework to WNR and Φ_d, we need to define associated algebraic fingerprint, helper function, collision-resistant hash function and expansion function. We define these as follows:

- $[b] \in \mathbb{G}$ denote the generator output by the setup procedure along with $\mathbb{G}$ and p
- $\vec{\Omega}\colon\ \vec{a} \in \mathbb{Z}_p^n \mapsto ([a_1], \ldots, [a_n]) \in \mathbb{G}^n$
- $\mathsf{Help}_{\Phi_d}\colon\ \vec{a} \in \mathbb{Z}_p^n \mapsto ([1], [a_1], \ldots, \left[a_1^d\right], \ldots, [a_n], \ldots, \left[a_n^d\right]) \in \mathbb{G}^{nd+1}$
- h can be any collision-resistant hash function $h\colon\ \{0,1\}^n \times \mathbb{G}^n \to \{0,1\}^{n-2}$
- $\mathsf{E}\colon\ z \in \{0,1\}^{n-2} \mapsto 11 \,\|\, z \in \{0,1\}^n$

We just need to prove that E satisfies the linear independence property required to apply the framework, which is done in Section 6.2.3.3, and then, using Theorem 6.2.3, we obtain the following lemma.

Lemma 6.2.6. *Let $m = \max(w_0, \ldots, w_n)$ be the maximum component of $\vec{w}$. Assuming md-DDHI holds in $\mathbb{G}$ and h is a collision-resistant hash function, the construction obtained with the above ingredients and Theorem 6.2.3 is a Φ_d-related-key secure pseudorandom function.*

Moreover, the running time of the reduction is polynomial in n, d, m, and in the running time of the adversary.

6.2.3.2 Related-Key Security for Affine Multivariate Functions

In order to deal with larger classes of related-key attacks, it would be important to consider multivariate RKD functions in which the attacker is allowed to mix different components of the secret key. In fact, the construction $\mathsf{WNR}^{d\text{-}\mathsf{lin}}$ given in the previous section seems like a plausible candidate if we assume that the RKD functions that are being applied to each sub-key are linearly independent. However we have not been able to prove or disprove this statement. We construct an alternative pseudorandom function that can be shown to be related-key secure against an adversary that can modify the key by applying functions from $\Phi_{n+1,\mathsf{multi\text{-}aff}}$, defined as:

$$\{(\boldsymbol{M}, \vec{b}) \mid (\boldsymbol{M}, \vec{b}) \in \mathbb{Z}_p^{(n+1)\times(n+1)} \times \mathbb{Z}_p^{n+1} \text{ s.t. } \vec{M_i} \neq 0^{n+1}, \forall i = 0, \ldots, n\}$$

where $\vec{M_i}$ denote the i-th row of $\boldsymbol{M}$, for $i = 0, \ldots, n$. Hence, for a key $\vec{a} = (a_0, \ldots, a_n) \in \mathbb{Z}_p^{n+1}$, applying an RKD function $(\boldsymbol{M}, \vec{b}) \in \Phi_{n+1,\mathsf{multi\text{-}aff}}$, leads to the key $\boldsymbol{M} \cdot \vec{a} + \vec{b} = (\vec{M_0} \cdot \vec{a} + b_0, \ldots, \vec{M_n} \cdot \vec{a} + b_n) \in \mathbb{Z}_p^{n+1}$.

Since the new construction is based on WNR with exponential weights $w_i = 2^i$, it is of limited interest given that its security relies on an assumption whose input is of exponential size. Moreover, in settings in which it is acceptable to have security assumptions with exponential-size inputs, much simpler constructions are possible. Nevertheless, we still believe that the new construction provides a first small step towards building related-key secure pseudorandom functions for large classes of RKD functions such as multivariate affine functions or even multivariate polynomial functions.

To construct a $\Phi_{n+1,\mathsf{multi\text{-}aff}}$-related-key secure pseudorandom function, we apply our framework to a particular case of WNR, defined as follows:

Exponential WNR. We define $\mathsf{WNR}^{\mathsf{exp}}$ as the particular case of WNR, where $w_i = 2^i$, for $i = 0, \ldots, n$. Please refer to Section 6.1.1.2 for details.

In order to apply our framework to $\mathsf{WNR}^{\mathsf{exp}}$ and $\Phi_{n+1,\mathsf{multi\text{-}aff}}$, we need to define associated algebraic fingerprint, helper function, collision-resistant hash function and expansion function. We define these as follows:

- $[b] \in \mathbb{G}$ denote the generator output by the setup procedure along with $\mathbb{G}$ and p
- $\vec{\Omega}$: $\vec{a} \in \mathbb{Z}_p^n \mapsto ([a_1], \ldots, [a_n]) \in \mathbb{G}^n$
- $\mathsf{Help}_{\Phi_{n+1,\mathsf{multi\text{-}aff}}}$: $\vec{a} \in \mathbb{Z}_p^n \mapsto ([1], [a_1], \ldots, [a_n]) \in \mathbb{G}^{n+1}$
- h can be any collision-resistant hash function h: $\{0,1\}^n \times \mathbb{G}^n \to \{0,1\}^{n-2}$
- E: $z \in \{0,1\}^{n-2} \mapsto 11 \,\|\, z \in \{0,1\}^n$

We just need to prove that E satisfies the linear independence property required to apply the framework, which is done in Section 6.2.3.3, and then, combining the above ingredients with Theorem 6.2.3, we obtain the following theorem.

Theorem 6.2.7. *Assuming 2^{n+1}-DDHI holds in $\mathbb{G}$ and h is a collision-resistant hash function, the construction obtained with the above ingredients and Theorem 6.2.3 is a $\Phi_{n+1,\mathsf{multi\text{-}aff}}$-related-key secure pseudorandom function.*

Moreover, the running time of the reduction is polynomial in $n, d, 2^n$, and in the running time of the adversary.

Remark 6.2.8. The above construction could be easily generalized to multivariate polynomial RKD functions of degree at most d, by properly changing the exponential sequence used in the construction to $w_i = (d+1)^i$ in order to guarantee that, for any two different values of x and x' and any $d_i, d'_i \in \{1, \ldots, d\}$ for $i = 0, \ldots, d$, the sums $w_0 d_0 + \sum_{i=1}^{n} w_i d_i x_i$ and $w_0 d'_0 + \sum_{i=1}^{n} w_i d'_i x'_i$ are distinct. We can then use exactly the same helper function than in Section 6.2.2.2.

6.2.3.3 Proof of Linearly Independence Properties for Section 6.2.3.1 and Section 6.2.3.2

For Univariate Polynomials. Let $P_{\vec{\phi},x}(\vec{T}) = \phi_0(\vec{T})^{w_0} \cdot \prod_{i=1}^{n} \phi_i(\vec{T})^{w_i x_i}$ for $\vec{\phi} \in \Phi_d$ and $x \in \{0,1\}^n$. Let $\mathcal{S} = \{11 \,\|\, h \mid z \in \{0,1\}^{n-2}\}$. The only thing we need to prove is that, for any sequence $(x_1, \vec{\phi}_1), \ldots, (x_q, \vec{\phi}_q)$, polynomials 1, T_i^j for $i = 0, \ldots, n$ and $j = 1, \ldots, d$ and polynomials $P_{\vec{\phi}_1,x_1}, \ldots, P_{\vec{\phi}_q,x_q}$ are linearly independent, as long as $x_1, \ldots, x_q$ are distinct in $\mathcal{S}$.

By contradiction, let us assume that there exists a sequence of distinct polynomials $P_1, \ldots, P_q$, where for $k = 1, \ldots, q$, $P_k = P_{\vec{\phi},x}$ for distinct $x \in \mathcal{S}$ or $P_k = T_i^j$ for some $(i,j) \in \{0, \ldots, n\} \times \{1, \ldots, d\}$ or $P_k = 1$, such that:

$$\sum_{k=1}^{q} \lambda_k \cdot P_k = 0$$

with $\lambda_k \neq 0$ for all k.

Since polynomials T_i^j, for $i = 0, \ldots, n$ and $j = 1, \ldots, d$ and polynomial 1 are clearly linearly independent over $\mathbb{Z}_p$, there is at least one polynomial P_k in the above sum such that P_k corresponds to $P_{\vec{\phi},x}$ for a query $(\vec{\phi}, x)$ with $x \in \mathcal{S}$.

We now consider an arbitrary monomial $T_0^{z_0} \cdots T_n^{z_n}$ appearing in at least one of the polynomials in this combination and such that z has highest Hamming weight. It is clear that $\mathsf{hw}(z_1 \,\|\, \ldots \,\|\, z_n) \geq 2$, since the Hamming weight of $x \in \mathcal{S}$ is at least 2. But, since the sum is the zero polynomial, there must exist at least two distinct polynomials $P_{\vec{\phi}_1,x_1}$ and $P_{\vec{\phi}_2,x_2}$ containing this monomial $T_0^{z_0} \cdots T_n^{z_n}$ in the above sum.

Let $\hat{z}$ denote the n-bit string such that $\hat{z}_i = 0$ if $z_i = 0$, while $\hat{z}_i = 1$ otherwise, for $i = 1, \ldots, n$. It is clear that $\mathsf{hw}(\hat{z}) \geq 2$ and then $\hat{z} \in \mathcal{S}$. Also, since $\hat{z}$ has the highest possible Hamming weight, $x_1 = x_2 = \hat{z}$ (from the definitions of $P_{\vec{\phi}_1,x_1}$ and $P_{\vec{\phi}_2,x_2}$ and since the first components of $\vec{\phi}_1$ and $\vec{\phi}_2$, which are the polynomials that apply to A_0, have degree at least 1). This means $\hat{z} \in \mathcal{S}$ has been used twice, which is forbidden. Hence, all the polynomials are linearly independent. The linear independence property follows. $\square$

For Multivariate Affine Functions. Let $P_{(\boldsymbol{M},\vec{b}),x}(\vec{T}) = (\vec{M}_0 \cdot \vec{T} + b_0) \prod_{i=1}^{n} (\vec{M}_i \cdot \vec{T} + b_i)^{2^i \cdot x_i}$ for $(\boldsymbol{M}, \vec{b}) \in \Phi_{n+1,\mathsf{multi\text{-}aff}}$ and $x \in \{0,1\}^n$, where $\vec{M}_i$ denote the i-th row of matrix $\boldsymbol{M}$. Let $\mathcal{S} = \{11 \,\|\, h \mid z \in \{0,1\}^{n-2}\}$. The only thing we need to prove is that, for any sequence $(x_1, (\boldsymbol{M}_1, \vec{b}_1)), \ldots, (x_q, (\boldsymbol{M}_q, \vec{b}_q))$ with, for $k = 1, \ldots, q$, x_k all distinct in $\mathcal{S}$ and $(\boldsymbol{M}_k, \vec{b}_k) \in \Phi_{n+1,\mathsf{multi\text{-}aff}}$, polynomials 1, T_i for $i = 0, \ldots, n$ and polynomials $P_{(\boldsymbol{M}_1,\vec{b}_1),x_1}, \ldots, P_{(\boldsymbol{M}_q,\vec{b}_q),x_q}$ are linearly independent.

Polynomial 1 is a degree 0 polynomial and polynomials T_i, for $i = 0, \ldots, n$ are degree 1 polynomials, and they are all linearly independent. Then, we just prove that for any

$((\boldsymbol{M}, \vec{b}), x)$ and $((\boldsymbol{M}', \vec{b}'), x')$ with $x, x' \in \mathcal{S}$ and $x \neq x'$, the degrees of polynomials $P_{(\boldsymbol{M}, \vec{b}), x}$ and $P_{(\boldsymbol{M}', \vec{b}'), x'}$ are always distinct and greater than 2.

Let $((\boldsymbol{M}, \vec{b}), x)$ with $x \in \mathcal{S}$, then $P_{(\boldsymbol{M}, \vec{b}), x}(\vec{T}) = (\vec{M}_0 \cdot \vec{T} + b_0) \prod_{i=1}^{n} (\vec{M}_i \cdot \vec{T} + b_i)^{2^i \cdot x_i}$. Since for any $i = 0, \ldots, n$, $\vec{M}_i \neq 0^{n+1}$, $\vec{M}_i \cdot \vec{T} + b_i$ is always a multivariate polynomial of degree 1, $(\vec{M}_i \cdot \vec{T} + b_i)^{2^i}$ is always a polynomial of degree 2^i. Hence, $P_{(\boldsymbol{M}, \vec{b}), x}(\vec{T})$ is a multivariate polynomial of degree $\sum_{i=0}^{n} 2^i x_i$, which is clearly greater than 2 since $\mathsf{hw}(x) \geq 2$ for any $x \in \mathcal{S}$.

Finally, since the entries x used in distinct queries has to be always distinct, and since for any $x \neq x'$, $\sum_{i=0}^{n} 2^i x_i \neq \sum_{i=0}^{n} 2^i x'_i$ by the uniqueness of the binary decomposition, the degrees of polynomials $P_{(\boldsymbol{M}, \vec{b}), x}$ and $P_{(\boldsymbol{M}', \vec{b}'), x'}$ are always distinct and at least 2 for any queries $((\boldsymbol{M}, \vec{b}), x)$ and $((\boldsymbol{M}', \vec{b}'), x')$ with $x, x' \in \mathcal{S}$ and $x \neq x'$. The linear independence property follows. $\square$

6.2.4 Extension to Weaker Assumptions

One can easily extend part of the above results to k-Lin-based pseudorandom functions designing a more general framework and still using the LIP theorem. One just needs to be careful that the RKD functions do not mix the matrices that are components of the key, such that Condition 2 on page 71 is always satisfied. In particular, one cannot prove related-key security for multivariate affine functions or permutations, but the result for univariate polynomials can easily be extended to k-Lin. We do not detail more this possibility as it is rather immediate.

6.2.5 A Further Generalization of the Framework

While the above framework (Section 6.2.2.1) is already quite powerful, it makes a strong assumption about the form of the outputs of the pseudorandom functions, which prevents it to being applied for other forms of pseudorandom functions, contrarily to the frameworks proposed in Chapter 3. Nevertheless, the latter frameworks require the use of a strong key fingerprint. The existence of such a tool is not always clear. For instance, given $\vec{w} \in \mathbb{Z}_p \times (\mathbb{Z}_p^*)^n$, if $w_0 \neq 0$ or if there exists $i \in \{1, \ldots, n\}$ such that $w_i > 1$, there is no practical strong key fingerprint for $\mathsf{WNR}^{\vec{w}}$. Also, for any weight, it is not very clear that there exists a strong key-fingerprint for WBMR. Thus, it seems one cannot apply any of the frameworks given in Chapter 3 to WNR or WBMR, in general.

Therefore, in this section, we design a new and generalized framework, that generalizes in particular the frameworks given in Chapter 3 and does not make any specific assumption about the form of the output of the pseudorandom function. This generic framework encompasses in particular the framework we propose in Section 3.6 as well as the framework given above in Section 6.2.2.1 (using the the LIP theorem). However, the latter algebraic framework (Section 6.2.2.1) is significantly easier to use, which explains why we chose to give it first and propose this generic framework only as an extension, for completeness.

We introduce new notions, defined as follows, that extends the notions introduced in Section 6.2.2.

Perfectly Binding Key-Commitment. In order to overcome the lack of a strong key fingerprint, we introduce perfectly binding key-commitment. A perfectly binding key-commitment is a (deterministic) algorithm $\mathsf{Com}\colon \mathcal{K} \to \mathsf{ComSp}$ that takes a key $K \in \mathcal{K}$ as input and outputs a value $\mathsf{Com}(K)$ such that for any K, K' in $\mathcal{K}$, we have $\mathsf{Com}(K) = \mathsf{Com}(K')$ if and only if $K = K'$ (perfectly binding). As we will see later, we also want that for $K \in \mathcal{K}$, $\mathsf{Com}(K)$ hides the value of K. However, we do not need special requirement for this in the present definition, since this requirement will be implied by the extended definitions of the key-collision and unique-input-related-key pseudorandom function security notions, defined below.

Helper Information. In order to prove the security of our framework, we need to be able to compute commitments of any related key from some (public) information. Then, we enable the adversary to have access to some helper information $\mathsf{help}_\Phi = \mathsf{Help}_\Phi(K) \in \mathsf{HelpSp}$ about the secret K, where Help_Φ is a function from $\mathcal{K}$ to HelpSp. The helper function Help_Φ depends on the class Φ of RKD functions we are interested in. We suppose that it is possible to compute $\mathsf{Com}(\phi(K))$ just by knowing ϕ and $\mathsf{Help}_\Phi(K)$ but not K.

Then, we use the extended version of the key-collision and unique-input-rka-prf security games depicted in Figure 6.3 and Figure 6.4, where **Initialize** also leaks help_Φ to the adversary. We remark that unique-input-related-key pseudorandom function security implies that help_Φ hides K, otherwise it would be trivial to win the game defining this security notion. This directly implies that the commitment of K is hiding, since it can be computed from help_Φ.

Remark 6.2.9. We do not need a statistical-key-collision security notion, because it is implied by the extended key-collision security. The compatibility of the hash function is also simplified. We just require that it is a collision-resistant hash function and that its range $\mathcal{S}$ is such that the extended $(\mathcal{S}, \Phi)$-unique-input-related-key pseudorandom function security holds.

```
proc Initialize
$\mathsf{pp} \xleftarrow{\$} \mathsf{M.Setup}(1^\kappa)$
$K \xleftarrow{\$} \mathcal{K}$
$\mathsf{help}_\Phi \leftarrow \mathsf{Help}_\Phi(K)$
Return $(\mathsf{pp}, \mathsf{help}_\Phi)$

proc RKFn$(\phi, x)$
$y \leftarrow \mathsf{M.Eval}_{\mathsf{pp}}(\phi(K), x)$
Return $y$

proc Finalize$(\phi_1, \phi_2)$
Return $(\phi_1 \neq \phi_2$ and $\phi_1(K) = \phi_2(K))$
```

Figure 6.3: Game defining the extended Φ-key-collision security off M with helper function Help.

```
proc Initialize
$K \xleftarrow{\$} \mathcal{K}$ ; $b \xleftarrow{\$} \{0,1\}$
$\mathsf{help}_\Phi \leftarrow \mathsf{Help}_\Phi(K)$
Return $\mathsf{help}_\Phi$

proc Finalize$(b')$
Return $b' = b$

proc RKFn$(\phi, x)$
If $x \in \mathcal{S}$ then
    If $b = 0$ then $y \leftarrow M(\phi(K), x)$
    Else $y \xleftarrow{\$} \mathcal{R}$
Else $y \leftarrow \perp$
Return $y$
```

Figure 6.4: Security game for extended $(\mathcal{S}, \Phi)$-unique-input-related-key pseudorandom function security of a pseudorandom function M with helper function Help

Using these new tools, we obtain the following framework, which generalizes our previous frameworks.

Theorem 6.2.10. *Let $M = (\mathsf{M.Setup}, \mathsf{M.Eval})$ be a pseudorandom function whose key space, domain, and range are denoted $\mathcal{K}, \mathcal{D}, \mathcal{R}$, respectively. Let $\mathsf{Com}\colon \mathcal{K} \to \mathsf{ComSp}$ be a perfectly binding key-commitment. Let $\mathsf{Help}_{\Phi}\colon \mathcal{K} \to \mathsf{HelpSp}$ be the helper function associated to Com and Φ. Let $\overline{\mathcal{D}} = \mathcal{D} \times \mathsf{ComSp}$ and let $H\colon \overline{\mathcal{D}} \to \mathcal{S}$ be a compatible collision-resistant hash function, with $\mathcal{S} \subseteq \mathcal{D}$. Define $F = (\mathsf{F.Setup}, \mathsf{F.Eval})$ as $\mathsf{F.Setup} = \mathsf{M.Setup}$ and:*

$$\mathsf{F.Eval}_{\mathsf{pp}}(K, x) := \mathsf{M.Eval}_{\mathsf{pp}}(K, H(x, \mathsf{M.Eval}_{\mathsf{pp}}(K, \mathsf{Com}(K))))$$

for all $K \in \mathcal{K}$ and $x \in \mathcal{D}$.

Then, assuming M is an $(\mathcal{S}, \Phi)$-unique-input-related-key secure pseudorandom function and is extended Φ-key-collision secure, and H is a collision-resistant hash function, then F is a Φ-related-key secure pseudorandom function.

Proof Overview. The proof of the above theorem is detailed below and relies on the sequence of 10 games (games $\mathbf{G}_0 - \mathbf{G}_9$) described in Figure 6.5. It is very similar to the proof other frameworks. Here we provide a brief overview. Since the RKD functions that we consider in our case may have claws, we start by dealing with possible collisions on the related keys in the $\mathsf{RKPRFReal}_F$ case, using the extended key-collision notion (games $\mathbf{G}_0 - \mathbf{G}_2$). These claws can be detected by looking for collisions of perfectly binding key-commitments for different RKD functions. Then, in games $\mathbf{G}_3 - \mathbf{G}_4$, we deal with possible collisions on hash values in order to ensure that the hash values $h = H(x, \mathsf{Com}(K))$ used to compute the output y are distinct. Then, we use the new extended $(\mathcal{S}, \Phi)$-unique-input-related-key pseudorandom function security notion to show that it is hard to distinguish the output of F from a uniformly random output (games $\mathbf{G}_5 - \mathbf{G}_6$). Finally, we use once again the extended key-collision security notion to deal with possible key collisions in the $\mathsf{RKPRFRand}_F$ case (games $\mathbf{G}_7 - \mathbf{G}_9$) so that $\mathbf{G}_9$ matches the description of the $\mathsf{RKPRFRand}_F$ Game. These key collisions can still be detected in these games by making crucial use of the helper function.

Proof of Theorem 6.2.10. The proof is based on the sequence of games in Figure 6.5. Much of the proof is similar to the proof of previously introduced frameworks. We denote by $\textsc{Succ}_i$ the event that game $\mathbf{G}_i$ output takes the value 1. Boolean flags are assumed initialized to false. Games $\mathbf{G}_i$, $\mathbf{G}_j$ are said to be identical until flag if their code differs only in statements that follow the setting of flag to true. Let $\mathscr{A}$ be an adversary against the Φ-related-key security of F. We assume without loss of generality that adversary $\mathscr{A}$ never repeats an oracle query.

$\mathbf{G}_0$ matches the description of the $\mathsf{RKPRFReal}_F$ game, so:

$$\Pr[\textsc{Succ}_0] = \Pr\left[\mathsf{RKPRFReal}_F^{\mathscr{A}} \Rightarrow 1\right] .$$

Game $\mathbf{G}_1$ introduces storage of used RKD functions and values of key-commitment com in sets D and E respectively and sets flag_1 to true if the same value of com arises for two different RKD functions. Since this storage does not affect the values returned by **RKFn**

$$\Pr[\textsc{Succ}_1] = \Pr[\textsc{Succ}_0] .$$

Game $\mathbf{G}_2$ adds the boxed code which changes how the repetition of a commitment value com is handled, by picking instead a random value from $\mathsf{ComSp}\backslash E$ that will not repeat any previous one. Games $\mathbf{G}_1$ and $\mathbf{G}_2$ are identical until flag_1 is set to true, hence we have

$$\Pr[\textsc{Succ}_1] \leq \Pr[\textsc{Succ}_2] + \Pr[E_1] ,$$

```
proc Initialize  // $\mathbf{G}_0$
$K \xleftarrow{\$} \mathcal{K}$
proc RKFn($\phi, x$)  // $\mathbf{G}_0$
$\mathsf{com} \leftarrow \mathsf{Com}(\phi(K))$
$h \leftarrow H(x, \mathsf{com})$
$y \leftarrow M(\phi(K), h)$
Return $y$
proc Finalize(b')  // All Games
Return $b'$
```

```
proc Initialize  // $\mathbf{G}_1, \mathbf{G}_2$
$K \xleftarrow{\$} \mathcal{K}$ ; $D \leftarrow \emptyset$ ; $E \leftarrow \emptyset$
proc RKFn($\phi, x$)  // $\mathbf{G}_1$, $\boxed{\mathbf{G}_2}$
$\mathsf{com} \leftarrow \mathsf{Com}(\phi(K))$
If $\mathsf{com} \in E$ and $\phi \notin D$ then
$\mathsf{flag}_1 \leftarrow \mathsf{true}$ ; $\boxed{\mathsf{com} \xleftarrow{\$} \mathsf{ComSp} \backslash E}$
$D \leftarrow D \cup \{\phi\}$ ; $E \leftarrow E \cup \{\mathsf{com}\}$
$h \leftarrow H(x, \mathsf{com})$
$y \leftarrow M(\phi(K), h)$
Return $y$
```

```
proc Initialize  // $\mathbf{G}_3, \mathbf{G}_4$
$K \xleftarrow{\$} \mathcal{K}$ ; $D \leftarrow \emptyset$ ; $E \leftarrow \emptyset$ ; $G \leftarrow \emptyset$
proc RKFn($\phi, x$)  // $\mathbf{G}_3$, $\boxed{\mathbf{G}_4}$
$\mathsf{com} \leftarrow \mathsf{Com}(\phi(K))$
If $\mathsf{com} \in E$ and $\phi \notin D$
    then $\mathsf{com} \xleftarrow{\$} \mathsf{ComSp} \backslash E$
$D \leftarrow D \cup \{\phi\}$ ; $E \leftarrow E \cup \{\mathsf{com}\}$
$h \leftarrow H(x, \mathsf{com})$
If $h \in G$ then $\mathsf{flag}_2 \leftarrow \mathsf{true}$
    $\boxed{h \xleftarrow{\$} \mathcal{S} \backslash G}$
$G \leftarrow G \cup \{r\}$
$y \leftarrow M(\phi(K), h)$
Return $y$
```

```
proc Initialize  // $\mathbf{G}_5$
$K \xleftarrow{\$} \mathcal{K}$ ; $D \leftarrow \emptyset$ ; $E \leftarrow \emptyset$ ; $G \leftarrow \emptyset$
proc RKFn($\phi, x$)  // $\mathbf{G}_5$
$\mathsf{com} \leftarrow \mathsf{Com}(\phi(K))$
If $\mathsf{com} \in E$ and $\phi \notin D$
    then $\mathsf{com} \xleftarrow{\$} \mathsf{ComSp} \backslash E$
$D \leftarrow D \cup \{\phi\}$ ; $E \leftarrow E \cup \{\mathsf{com}\}$
$h \leftarrow H(x, \mathsf{com})$
If $h \in G$ then $h \xleftarrow{\$} \mathcal{S} \backslash G$
$G \leftarrow G \cup \{r\}$
$y \xleftarrow{\$} \mathcal{R}$
Return $y$
```

```
proc Initialize  // $\mathbf{G}_6$
$K \xleftarrow{\$} \mathcal{K}$ ; $D \leftarrow \emptyset$ ; $E \leftarrow \emptyset$
$G \xleftarrow{\$} \mathsf{Fun}(\mathcal{K}, \mathcal{D}, \mathcal{R})$
proc RKFn($\phi, x$)  // $\mathbf{G}_6$
$y \xleftarrow{\$} \mathcal{R}$
Return $y$
```

```
proc Initialize  // $\mathbf{G}_9$
$K \xleftarrow{\$} \mathcal{K}$ ; $G \xleftarrow{\$} \mathsf{Fun}(\mathcal{K}, \mathcal{D}, \mathcal{R})$
proc RKFn($\phi, x$)  // $\mathbf{G}_9$
$y \leftarrow G(\phi(K), x)$
Return $y$
```

```
proc Initialize  // $\mathbf{G}_7$, $\mathbf{G}_8$
$K \xleftarrow{\$} \mathcal{K}$ ; $D \leftarrow \emptyset$ ; $E \leftarrow \emptyset$
$G \xleftarrow{\$} \mathsf{Fun}(\mathcal{K}, \mathcal{D}, \mathcal{R})$
proc RKFn($\phi, x$)  // $\boxed{\mathbf{G}_7}$, $\mathbf{G}_8$
If $\phi(K) \in E$ and $\phi \notin D$ then
    $\boxed{y \xleftarrow{\$} \mathcal{R}}$ ; $\mathsf{flag}_3 \leftarrow \mathsf{true}$
$\boxed{\text{else}}$ $y \leftarrow G(\phi(K), x)$
$D \leftarrow D \cup \{\phi\}$ ; $E \leftarrow E \cup \{\phi(K)\}$
Return $y$
```

Figure 6.5: Games for the proof of Theorem 6.2.10

where E_1 denotes the event that the execution of $\mathscr{A}$ with game $\mathbf{G}_1$ sets flag_1 to true. We design an adversary $\mathscr{D}$ attacking the extended Φ-key-collision security of M such that

$$\Pr[E_1] \leq \mathbf{Adv}^{\mathsf{ext\text{-}kc}}_{\Phi,M}(\mathscr{D}) \ .$$

Adversary $\mathscr{D}$ gets helper information $\mathsf{help}_\Phi = \mathsf{Help}_\Phi(K)$, then runs $\mathscr{A}$. When the latter makes an **RKFn**-query (ϕ, x), adversary $\mathscr{D}$ computes $\mathsf{com} = \mathsf{Com}(\phi(K))$ using its helper information and then $h = H(x, \mathsf{com})$ and finally queries (ϕ, h) to its oracle and sends the value it gets to $\mathscr{A}$. When $\mathscr{A}$ halts, $\mathscr{D}$ searches for two different RKD functions ϕ queried by $\mathscr{A}$ that lead to the same commitment value com and returns these two functions if found.

Since Com is a perfectly binding key-commitment, two such functions lead to the same key, so $\mathscr{D}$ wins if he finds such two functions.

Game $\mathbf{G}_3$ introduces the storage of hash values in a set G and sets flag_2 to true if the same hash output arises twice. Since this storage does not affect the values returned by **RKFn**, we have

$$\Pr[\,\textsc{Succ}_3\,] = \Pr[\,\textsc{Succ}_2\,] \ .$$

Game $\mathbf{G}_4$ adds the boxed code which changes how repetition of hash values is handled, by picking instead a random value h from $\mathcal{S}\backslash G$ that will not repeat any previously used hash value. Games $\mathbf{G}_3$ and $\mathbf{G}_4$ are identical until flag_2 is set to true, hence we have

$$\Pr[\,\textsc{Succ}_3\,] \leq \Pr[\,\textsc{Succ}_4\,] + \Pr[\,E_2\,] \ ,$$

where E_2 denotes the event that the execution of $\mathscr{A}$ with game $\mathbf{G}_3$ sets flag_2 to true. We design an adversary $\mathscr{C}$ attacking the cr-security of H such that

$$\Pr[\,E_2\,] \leq \mathbf{Adv}^{\mathsf{cr}}_{H}(\mathscr{C}) \ .$$

Adversary $\mathscr{C}$ starts by picking $K \xleftarrow{\$} \mathcal{K}$ and initializes $j \leftarrow 0$. It runs $\mathscr{A}$. When the latter makes an **RKFn**-query (ϕ, x), adversary $\mathscr{C}$ responds via

```
$\mathsf{com} \leftarrow \mathsf{Com}(\phi(K))$
$j \leftarrow j+1$ ; $\phi_j \leftarrow \phi$ ; $x_j \leftarrow x$
If $\mathsf{com} \in E$ and $\phi \notin D$ then $\mathsf{com} \xleftarrow{\$} \mathsf{ComSp}\backslash E$          $(*)$
$D \leftarrow D \cup \{\phi\}$ ; $E \leftarrow E \cup \{\mathsf{com}\}$
$\mathsf{com}_j \leftarrow \mathsf{com}$
$h \leftarrow H(x, \mathsf{com})$
$h_j \leftarrow h$
$y \leftarrow M(\phi(K), h)$
Return $y$.
```

When $\mathscr{A}$ halts, $\mathscr{C}$ searches for a, b satisfying $1 \leq a < b \leq j$ such that $h_a = h_b$ and, if it finds them, outputs $(x_a, \mathsf{com}_a), (x_b, \mathsf{com}_b)$ and halts. The pairs (x_a, com_a) and (x_b, com_b) are distinct. Indeed, consider two cases: first, if $\phi_a = \phi_b$ then since $\mathscr{A}$ never repeats an oracle query, $x_a \neq x_b$ hence $(x_a, \mathsf{com}_a) \neq (x_b, \mathsf{com}_b)$. Second, if $\phi_a \neq \phi_b$, then condition $(*)$ ensures that $\mathsf{com}_a \neq \mathsf{com}_b$. Hence once again, $(x_a, \mathsf{com}_a) \neq (x_b, \mathsf{com}_b)$, and then

$$\Pr[\,\textsc{Succ}_3\,] \leq \Pr[\,\textsc{Succ}_4\,] + \mathbf{Adv}^{\mathsf{cr}}_{H}(\mathscr{C}) \ .$$

In game $\mathbf{G}_5$, instead of returning the value $M(\phi(K), h)$, we always return a random value. To show that games $\mathbf{G}_4$ and $\mathbf{G}_5$ are indistinguishable, we design an adversary $\mathscr{B}$ against the extended $(\mathcal{S}, \Phi)$-unique-input-related-key pseudorandom function security of M such that

$$\Pr[\,\textsc{Succ}_4\,] \leq \Pr[\,\textsc{Succ}_5\,] + \mathbf{Adv}^{\mathsf{ext\text{-}ui\text{-}prf\text{-}rka}}_{\Phi,\mathcal{S},M}(\mathscr{B}) \ .$$

Adversary $\mathscr{B}$ starts by initializing sets $D \leftarrow \emptyset$, $E \leftarrow \emptyset$, $G \leftarrow \emptyset$. Then $\mathscr{B}$ gets helper information $\mathsf{help}_\Phi = \mathsf{Help}_\Phi(K)$, then runs $\mathscr{A}$. When the latter makes an **RKFn**-query (ϕ, x), $\mathscr{B}$ responds as follows. First, it computes $\mathsf{com} = \mathsf{Com}(\phi(K))$ using its helper information. Then, $\mathscr{B}$ checks if $\mathsf{com} \in E$ and $\phi \in D$. If they do, $\mathscr{B}$ picks $\mathsf{com} \xleftarrow{\$} \mathsf{ComSp}\backslash E$ at random. $\mathscr{B}$

then sets $D \leftarrow D \cup \{\phi\}$ and $E \leftarrow E \cup \{\mathsf{com}\}$. Next, $\mathscr{B}$ computes $h \leftarrow H(x, \mathsf{com})$ and checks if $h \in G$. If it does, $\mathscr{B}$ picks $h \overset{\$}{\leftarrow} \mathcal{S} \backslash G$ at random. Notice that this step guarantees that all values h are in $\mathcal{S}$ and are all distinct as long as $\mathscr{A}$ makes at most $|\mathcal{S}|$ queries. Finally, $\mathscr{B}$ sets $G \leftarrow G \cup \{h\}$, makes the query (ϕ, h) to its oracle, and returns the value it gets, which is either $M(\phi(K), h)$ or a uniformly random value, to $\mathscr{A}$. When $\mathscr{A}$ halts, $\mathscr{B}$ halts with the same output. It follows from these observations that $\mathscr{B}$ is a unique-input adversary for queries in $\mathcal{S}$. Finally, it is clear that if $\mathscr{B}$'s oracle gives real outputs of M for queries in $\mathcal{S}$, then it simulates exactly game $\mathbf{G}_4$ and if $\mathscr{B}$'s oracle gives uniformly random values for queries in $\mathcal{S}$, then it simulates exactly game $\mathbf{G}_5$.

In game $\mathbf{G}_6$, we simply set the value y to a uniformly random value. Clearly, $\mathbf{G}_5$ and $\mathbf{G}_6$ are identical since the value returned is a uniformly random value for any query. Then, we have

$$\Pr[\,\textsc{Succ}_5\,] = \Pr[\,\textsc{Succ}_6\,] \ .$$

In game $\mathbf{G}_7$, we check if two different queries can lead to a key collision. Since the "If" test ensures that the returned value is still uniformly random over $\mathcal{R}$ even when two different queries result in the same key, games $\mathbf{G}_6$ and $\mathbf{G}_7$ are identical. Hence,

$$\Pr[\,\textsc{Succ}_6\,] = \Pr[\,\textsc{Succ}_7\,] \ .$$

In game $\mathbf{G}_8$, we compute the output of **RKFn** using a random function G in $\mathsf{Fun}(\mathcal{K}, \mathcal{D}, \mathcal{R})$. Since games $\mathbf{G}_7$ and $\mathbf{G}_8$ are identical until flag_3 is set to true, we have

$$\Pr[\,\textsc{Succ}_7\,] \le \Pr[\,\textsc{Succ}_8\,] + \Pr[\,E_3\,] \ ,$$

where E_3 denotes the event that the execution of $\mathscr{A}$ with game $\mathbf{G}_8$ sets flag_3 to true. To bound the probability of event E_3, we design an adversary $\mathscr{D}$ attacking extended Φ-key-collision security of M such that

$$\Pr[\,E_3\,] \le \mathbf{Adv}^{\mathsf{ext\text{-}kc}}_{\Phi,M}(\mathscr{D}) \ .$$

Adversary $\mathscr{D}$ starts by initializing a list $\mathcal{L} \leftarrow$ empty list and by choosing an element ψ in Φ and by setting $\psi_1 \leftarrow \psi$ and $\psi_2 \leftarrow \psi$. Then, it runs $\mathscr{A}$. When the latter makes an **RKFn**-query (ϕ, x), adversary $\mathscr{D}$ does the following: it first computes $\mathsf{com} \leftarrow \mathsf{Com}(\phi(K))$ using its helper information and searches for all tuples $(\phi_i, \mathsf{com}_i) \in \mathcal{L}$ such that $\mathsf{com}_i = \mathsf{com}$. If it does find such tuples, it checks for all of them if $\phi \ne \phi_\ell$ and in that case sets $\psi_1 \leftarrow \phi_\ell$ and $\psi_2 \leftarrow \phi$. Finally, $\mathscr{D}$ picks $y \overset{\$}{\leftarrow} \mathcal{R}$ at random, returns y to $\mathscr{A}$ and adds (ϕ, com) to $\mathcal{L}$. When $\mathscr{A}$ halts, $\mathscr{B}$ halts and outputs (ψ_1, ψ_2). If the execution of $\mathscr{A}$ sets flag_3 to true, then $\mathscr{A}$ has queried $\phi_1 \ne \phi_2$ such that $\phi_1(K) = \phi_2(K)$ and assuming it has first queried ϕ_1, when $\mathscr{D}$ computes $\mathsf{Com}(\phi_2(K))$ and checks if this value is already in $\mathcal{L}$, it finds that this value matches $\mathsf{Com}(\phi_1(K))$ and since $\phi_1 \ne \phi_2$, it sets $\psi_1 = \phi_1$ and $\psi_2 = \phi_2$, so $\mathscr{D}$ wins. Then, we have

$$\Pr[\,E_3\,] \le \mathbf{Adv}^{\mathsf{ext\text{-}kc}}_{\Phi,M}(\mathscr{D}) \ .$$

Since A does not repeat oracle queries and since key collisions are dealt with in a similar way, it follows that games $\mathbf{G}_8$ and $\mathbf{G}_9$ are identical. Thus,

$$\Pr[\,\textsc{Succ}_8\,] = \Pr[\,\textsc{Succ}_9\,] \ .$$

Finally, $\mathbf{G}_9$ matches the description of the $\mathsf{RKPRFRand}_F$ game, so:

$$\Pr[\,\textsc{Succ}_9\,] = \Pr\left[\mathsf{RKPRFRand}_F^{\mathscr{A}} \Rightarrow 1\right] \ .$$

Theorem 6.2.10 now follows by combining the bounds arising in the different game hops. □

6.3 Applications to Aggregate Pseudorandom Functions

In this section, we show that for all constructions proposed in [CGV15], one can prove the aggregate pseudorandom function security with a polynomial-time reduction, while proofs proposed in this seminal paper suffered from an exponential (in the input size) overhead in the running time of the reduction. Moreover, our reductions are almost straightforward via the PLP theorem.

A first attempt to solve the issue of the exponential time of the original reductions was done in [CH15]. By introducing multilinear pseudorandom functions and giving a particular instantiation, Cohen and Holmgren showed that one can prove the aggregate pseudorandom function security of NR with a polynomial-time reduction for hypercubes and decision trees aggregation. However, their technique does not extend to the more general case of read-once formulas aggregation. Also, as we show in what follows, their construction can be seen as a particular case of our main theorem, and then can be proven secure very easily using our result.

Here, we provide a polynomial-time reduction for the general case of read-once formulas. This implies in particular the previous results on hypercubes and decision trees which are particular cases of read-once formulas.

Intuitively, if we consider the PLP security for $k = 1$ and aggregation with the Naor-Reingold pseudorandom function, our PLP theorem (Theorem 5.2.2) implicitly says that as long as the aggregate values can be computed as a group element whose discrete logarithm is the evaluation of a multivariate polynomial on the key, then, if the corresponding polynomials have a small representation, the PLP theorem guarantees the security (with a polynomial-time reduction), even if the number of points aggregated is super-polynomial. Please notice that if these polynomials do not have any small representation (e.g. the smallest representation is exponential in the input size), then there is no point of considering such aggregation, since the whole point of aggregate pseudorandom function lies in the possibility of aggregating super-polynomially many outputs with a very efficient computation.

6.3.1 Read-Once Formula Aggregation

Read-Once Formulas. A read-once formula is a circuit on $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ composed of only AND, OR and NOT gates with fan-out 1, so that each input literal is fed into at most one gate and each gate output is fed into at most one other gate. We denote by ROF_n the family of all read-once boolean formulas over $x_1, \ldots, x_n$ variables. In order to ease the reading, we restrict these circuits to be in a standard form, so that they are composed of fan-in 2 and fan-out 1 AND and OR gates, and NOT gates occurring only at the inputs. This common restriction can be done without loss of generality. Hence, one can see such a circuit as a binary tree where each leaf is labeled by a variable x_i or its negation $\bar{x}_i$ and where each internal node has a label C and has two children with labels C_L and C_R and represents either an AND or an OR gate (with fan-in 2). We identify a formula (and the set it represents) with the label of its root C_ϕ.

Aggregation for Read-Once Formulas. We recall the definition of read-once formula aggregation used in [CGV15]. For the sake of simplicity, we only consider the case of the Naor-Reingold pseudorandom function. We define the aggregation function for read-once formulas of length n as follows.

The collection $\mathcal{S}_{\mathsf{rof}} \subseteq \{0,1\}^n$ corresponds to all the subsets of $S \subseteq \{0,1\}^n$ such that there exists a read-once formula $C_\phi \in \mathrm{ROF}_n$ such that $S = \{x \in \{0,1\}^n \mid C_\phi(x) = 1\}$.

The aggregation function Γ_{rof} is defined as the product (assuming the group is a multiplicative group) of the values on such a subset. Hence, we have:

$$\begin{aligned}\mathrm{AGG}_{\mathsf{NR},\mathcal{S}_{\mathsf{rof}},\Gamma_{\mathsf{rof}}}(C_\phi) &= \prod_{x \mid C_\phi(x)=1} \left[a_0 \prod_{i=1}^n a_i^{x_i}\right] = \left[a_0 \sum_{x \mid C_\phi(x)=1} \prod_{i=1}^n a_i^{x_i}\right] \\ &= \left[a_0 \cdot A_{C_\phi,1}(\vec{a})\right] ,\end{aligned}$$

where $A_{C,b}$ is the polynomial $\sum_{x \in \{0,1\}^n \mid C(x)=b} \prod_{i=1}^n T_i^{x_i}$ for any $C \in \mathrm{ROF}_n$ and $b \in \{0,1\}$.

Efficient Evaluation of $A_{C,b}$. One can efficiently compute $A_{C,b}$ recursively as follows:

- If C is a literal for variable x_i, then $A_{C,1} = T_i$ and $A_{C,0} = 1$ if $C = x_i$; and $A_{C,1} = 1$ and $A_{C,0} = T_i$ if $C = \bar{x}_i$;
- If C is an AND gate with C_L and C_R its two children, then we have:
  $A_{C,1} = A_{C_L,1} \cdot A_{C_R,1}$
  $A_{C,0} = A_{C_L,0} \cdot A_{C_R,0} + A_{C_L,1} \cdot A_{C_R,0} + A_{C_L,0} \cdot A_{C_R,1}$;
- If C is an OR gate with C_L and C_R its two children, then we have:
  $A_{C,1} = A_{C_L,1} \cdot A_{C_R,1} + A_{C_L,1} \cdot A_{C_R,0} + A_{C_L,0} \cdot A_{C_R,1}$
  $A_{C,0} = A_{C_L,0} \cdot A_{C_R,0}$.

Now we have introduced everything, we can prove that NR (or more general constructions) is an $(\mathcal{S}_{\mathsf{rof}}, \Gamma_{\mathsf{rof}})$-aggregate pseudorandom function under the standard DDH assumption, as stated in the lemma below.

Lemma 6.3.1. *Assuming* DDH *holds in* $\mathbb{G}$, NR *is an* $(\mathcal{S}_{\mathsf{rof}}, \Gamma_{\mathsf{rof}})$*-aggregate pseudorandom function.*

Moreover, the running time of the reduction is polynomial in n and in the running time of the adversary.

The proof is straightforward using the PLP theorem: all queries in the security game for the aggregate PRF can be seen as a queries of the form $\mathbf{Pl}(P)$ for some polynomial P with a small representation: $\mathbf{Fn}(x)$ returns $\mathbf{Pl}(T_0 \prod_{i=1}^n T_i^{x_i})$ and $\mathrm{AGG}(C_\phi)$ returns $\mathbf{Pl}(T_0 \cdot A_{C_\phi,1}(\vec{T}))$. Details are given below.

Proof of Lemma 6.3.1. Let $\mathscr{A}$ be an adversary against the $(\mathscr{S}_{\mathsf{rof}}, \Gamma_{\mathsf{rof}})$-aggregate pseudorandom function security of NR that makes q oracle queries. We design an adversary $\mathscr{B}$ against the $(n,1,1)$-PLP security in $\mathbb{G}$ as follows, where we denote by $a_1, \ldots, a_n$ and $a_0 = b$ the secret values chosen at random in the **Initialize** procedure of the game defining the $(n,1,1)$-PLP security (it corresponds to $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_n$ and $\boldsymbol{B}$ but since $k = 1$, these values are simply scalars in $\mathbb{Z}_p$).

$\mathscr{B}$ runs $\mathscr{A}$. The latter can make two types of queries, so let us first describe how $\mathscr{B}$ responds to both these types of queries. The first type consists in standard PRF queries. When adversary $\mathscr{A}$ makes a query x, then $\mathscr{B}$ makes the query $P_x = \prod_{i=1}^n T_i^{x_i}$ and returns the value it gets to $\mathscr{A}$. The second type of queries consists in aggregate PRF values. When $\mathscr{A}$ makes an aggregate query $C_\phi \in \mathrm{ROF}_l$, for some $l \leq n$, the adversary $\mathscr{B}$ computes

$A_{C_\phi,1}(\overrightarrow{T})$ using the efficient recursive evaluation, as described in Section 6.3, then queries this polynomial $P_{C_\phi} = A_{C_\phi,1}(\overrightarrow{T})$, and returns the value it gets to $\mathscr{A}$. As $A_{C_\phi,1}(\overrightarrow{T})$ is a multivariate polynomial in $T_1, \dots, T_l$ with degree at most 1 in any indeterminate (since it is a sum of such multivariate polynomials), clearly, P_{C_ϕ} is a multivariate polynomial with degree at most 1 in any indeterminate.

The only thing we need to prove is that $\mathscr{B}$ simulates correctly either $\mathsf{AGGPRFReal}_{\mathsf{NR}}$ or $\mathsf{AGGPRFRand}_{\mathsf{NR}}$, which is almost straightforward by definition of the PLP security. On the one hand, it is clear that if $\mathscr{B}$'s oracle responds to a query P by $[P(\overrightarrow{a}) \cdot b]$, then $\mathscr{A}$ gets exactly the (standard or aggregate) values of the Naor-Reingold PRF defined with the generator $g = [1]$ and with the key $(b, a_1, \dots, a_n) \in \mathbb{Z}_p^{n+1}$. On the other hand, if $\mathscr{B}$'s oracle responds to a query P by random values computed taking into account related between P and previously queried polynomials, then the values $\mathscr{A}$ gets are statistically indistinguishable from the values it would get from the $\mathsf{AGGPRFRand}_{\mathsf{NR}}$ oracles. Indeed, the only difference lies in the fact that any value sent to $\mathscr{A}$ is random if the corresponding polynomial is linearly independent from previous queries, or is computed from previous values if the corresponding polynomial is linearly dependent, but these dependence are tested using a statistical procedure (which is correct with probability at least $\frac{p-1}{p}$) while in $\mathsf{AGGPRFRand}_{\mathsf{NR}}$, no error can occur.

Hence, in the first case, $\mathscr{B}$ simulates perfectly $\mathsf{AGGPRFReal}_{\mathsf{NR}}$, while in the second game, the simulation is statistically indistinguishable from $\mathsf{AGGPRFRand}_{\mathsf{NR}}$. Thus, we have shown that

$$\mathsf{Adv}^{\mathsf{agg\text{-}prf}}_{\mathsf{NR},\mathscr{S}_{\mathsf{rof}},\Gamma_{\mathsf{rof}}}(\mathscr{A}) \le \frac{p}{p-1} \cdot \mathbf{Adv}^{(n,1,1)\text{-}\mathsf{plp}}_{\mathbb{G}}(\mathscr{B})$$

and Lemma 6.3.1 now follows from the PLP theorem. □

6.3.2 Impossibility Results

Impossibility Result for CNF (Conjunctive Normal Form) and DNF (Disjunctive Normal Form) Formulas. In [CGV15], the authors show that, unless NP=BPP, there does not exist an $(\mathcal{S}, \Gamma)$-aggregate pseudorandom function[1], with $\mathcal{D} = \{0,1\}^n$, $\mathcal{S}$ containing the following sets:

$$S_\phi = \{x \in \{0,1\}^n \mid \phi(x) = 1\}$$

with ϕ a CNF formula with n-bit input, and Γ a "reasonable" aggregate function, e.g., Γ_{rof} (assuming $\mathcal{R}$ is a cyclic group $\mathbb{G}$ of prime order p). The proof consists in showing that if such aggregate pseudorandom function exists, then we can solve SAT in polynomial time. More precisely, given a SAT instance, i.e., a CNF formula ϕ, we can compute $\mathrm{AGG}(\phi)$. If ϕ is not satisfiable, $\mathrm{AGG}(\phi) = 1 \in \mathbb{G}$, while otherwise $\mathrm{AGG}(\phi) = \prod_{x \in \{0,1\}^n,\ \phi(x)=1} F(K, x)$. This latter value is not 1 with high probability, otherwise we would get a non-uniform distinguisher against aggregate pseudorandomness.

The case of DNF formulas (or more generally of any class for which satisfiability is tractable) was left as an important open problem in [CGV15]. Here, we show that unless NP=BPP, there also does not exist an $(\mathcal{S}, \Gamma)$-aggregate pseudorandom function as above, when $\mathcal{S}$ contains S_ϕ for any DNF (instead of CNF) formula ϕ with n-bit input. For that, we first remark that the formula $\top$, always true, is a DNF formula (it is the disjunction of all the possible literals),

[1]We suppose that the aggregate pseudorandomness security property holds non-uniformly. When $\mathcal{S}$ is expressive enough, we can also do the proof when this security property holds uniformly, see [CGV15, Section 2.2] for details.

and that the negation $\bar{\phi}$ of a CNF formula ϕ is a DNF formula. Then, given a SAT instance, a CNF formula ϕ, we compute $\mathrm{AGG}(\bar{\phi})$ and $\mathrm{AGG}(\top)$. If ϕ is not satisfiable, $\bar{\phi}$ is always true and $\mathrm{AGG}(\bar{\phi}) = \mathrm{AGG}(\top)$, while otherwise, $\mathrm{AGG}(\bar{\phi}) = \mathrm{AGG}(\top)/\prod_{x\in\{0,1\}^n,\,\phi(x)=1} F(K,x)$. This latter value is not $\mathrm{AGG}(\top)$ with high probability, otherwise we would get a non-uniform distinguisher against aggregate pseudorandomness.

6.3.3 Extension to Weaker Assumptions

One can easily extend this result for k-Lin-based pseudorandom functions using our main theorem. Also, one can easily use our PLP theorem (Theorem 5.2.2) to prove the security for any aggregate as soon as the aggregate values can be represented as group elements whose discrete logarithms are the evaluation of a (multivariate) polynomial on the key (and that this polynomial is efficiently computable). We do not detail further this as it is immediate.

6.4 Applications to Multilinear Pseudorandom Functions

To conclude this chapter, we explain how the PLP theorem can be used to prove directly the security of multilinear pseudorandom functions.

6.4.1 Cohen-Holmgren Construction

We first explain how to proof the security of CH. Please refer to Section 2.3.4.2 for the definition of this construction. Using the PLP theorem, one can easily obtain the following lemma.

Lemma 6.4.1. *Assuming* DDH *holds in* $\mathbb{G}$, CH *is a multilinear pseudorandom function.*

Moreover, the running time of the reduction is polynomial in n, in the dimension of the vector spaces, and in the running time of the adversary.

The proof is rather immediate. We give an informal proof below.

Proof of Lemma 6.4.1. Let $\vec{T} = (T_{1,1},\dots,T_{1,\ell_1},\dots,T_{n,1},\dots,T_{n,\ell_n})$ be a vector of indeterminates, and let $\vec{T_i} = (T_{i,1},\dots,T_{i,\ell_i})$. The PLP theorem shows that $\mathsf{CH}(\vec{a_1},\dots,\vec{a_n},\vec{x_1},\dots,\vec{x_n})$ (using a random key $\vec{a}$) is computationally indistinguishable from

$$\left[U\left(\prod_{i=1}^{n}\langle\vec{T_i},\vec{x_i}\rangle\right)\right] = [f(\vec{x_1},\dots,\vec{x_n})]$$

with $U \xleftarrow{\$} \mathsf{L}(\mathbb{Z}_p[\vec{T}]_{\le 1},\mathbb{Z}_p)$ and

$$f : \begin{pmatrix} \mathbb{Z}_p^{\ell_1}\times\cdots\times\mathbb{Z}_p^{\ell_n} & \to & \mathbb{Z}_p \\ (\vec{x_1},\dots,\vec{x_n}) & \mapsto & U(\prod_{i=1}^n\langle\vec{T_i},\vec{x_i}\rangle) \end{pmatrix}.$$

To conclude, we just need to prove that f is a random n-linear function in $\mathsf{L}(\mathbb{Z}_p^{\ell_1}\otimes\cdots\otimes\mathbb{Z}_p^{\ell_n},\mathbb{Z}_p)$. For that purpose, let us introduce the following n-linear application:

$$\psi : \begin{pmatrix} \mathbb{Z}_p^{\ell_1}\times\cdots\times\mathbb{Z}_p^{\ell_n} & \to & \mathbb{Z}_p[\vec{T}]_{\le 1} \\ (\vec{x_1},\dots,\vec{x_n}) & \mapsto & \prod_{i=1}^n\langle\vec{T_i},\vec{x_i}\rangle \end{pmatrix}.$$

We remark that f is the composition of U and ψ: $f = U \circ \psi$. Furthermore, if we write $\overrightarrow{e_{i,\ell}} = (0, \dots, 0, 1, 0, \dots, 0)$ the i-th vector of the canonical base of $\mathbb{Z}_p^\ell$, then:

$$\psi(\overrightarrow{e_{i_1,\ell_1}}, \dots, \overrightarrow{e_{i_n,\ell_n}}) = T_{1,i_1} \cdots T_{n,i_n};$$

and as the monomials $T_{1,i_1} \cdots T_{n,i_n}$ are linearly independent, ψ is injective. Since $f = U \circ \psi$ and $U \overset{\$}{\leftarrow} \mathsf{L}(\mathbb{Z}_p[\overrightarrow{T}]_{\leq 1}, \mathbb{Z}_p)$, the function f is a uniform random linear function from $\mathsf{L}(\mathbb{Z}_p^{\ell_1} \otimes \cdots \otimes \mathbb{Z}_p^{\ell_n}, \mathbb{Z}_p)$. This is exactly what we wanted to show. $\square$

6.4.2 Symmetric Multilinear Pseudorandom Functions

In [CGV15], constructing symmetric multilinear pseudorandom functions was left as an open problem. The definition of this notion is the same as the notion of multilinear pseudorandom function, except that we only require the function to be indistinguishable from a random *symmetric* multilinear function. In that case, we suppose that $\ell_1 = \cdots = \ell_n = \ell$, i.e., all the vectors $\overrightarrow{x_1}, \dots, \overrightarrow{x_n}$ have the same size ℓ. The authors wrote in [CGV15] that the natural modification of the CH construction to obtain a symmetric construction consisting in setting $\overrightarrow{a_1} = \overrightarrow{a_2} = \cdots = \overrightarrow{a_n}$ (simply denoted $\overrightarrow{a}$ in what follows) leads to a symmetric multilinear pseudorandom function whose security is less clear, but claimed that it holds under the $\mathcal{E}_{1,n}$-MDDH assumption (which is exactly the n-DDHI assumption), when $\ell = |\overrightarrow{a}| = 2$. We show that this construction is actually secure under the same assumption for any $\ell = |\overrightarrow{a}| \geq 2$ as stated in the following lemma, whose proof is almost the same as the proof of Lemma 6.4.1. Let us denote by $\mathsf{CH}_{\mathsf{sym}}$ this symmetric construction, whose key space and domain are respectively $\mathbb{Z}_p^\ell$ and $(\mathbb{Z}_p^\ell)^n$.

Lemma 6.4.2. *Assuming n-DDHI holds in $\mathbb{G}$, $\mathsf{CH}_{\mathsf{sym}}$ is a symmetric multilinear pseudorandom function.*

Moreover, the running time of the reduction is polynomial in n, ℓ, and in the running time of the adversary.

Proof of Lemma 6.4.2. Let $\mathscr{A}$ be an adversary against the symmetric multilinear pseudorandom function security of $\mathsf{CH}_{\mathsf{sym}}$ that makes q oracle queries. We design an adversary $\mathscr{B}$ against the $(\ell, 1, n)$-PLP security in $\mathbb{G}$ as follows, where we denote by $a_1, \dots, a_\ell$ and b the secret values chosen at random in the **Initialize** procedure of the game defining the $(\ell, 1, n)$-PLP security (it corresponds to $\boldsymbol{A}_1, \dots, \boldsymbol{A}_\ell$ and $\boldsymbol{B}$ but since $k = 1$, these values are simply scalars in $\mathbb{Z}_p$).

$\mathscr{B}$ runs $\mathscr{A}$. When the latter makes a query $\overrightarrow{x} = (\overrightarrow{x_1}, \dots, \overrightarrow{x_n})$ where $\overrightarrow{x_i} \in \mathbb{Z}_p^{l_i}$, $\mathscr{B}$ does the following. First, it starts by computing $P_{\overrightarrow{x}} = \prod_{i=1}^n \langle \overrightarrow{T}, \overrightarrow{x_i} \rangle$, where $\overrightarrow{T} = (T_1, \dots, T_\ell)$ is a vector of indeterminates. Hence, it computes a multivariate polynomial in $\mathbb{Z}_p[T_1, \dots, T_\ell]$ with degree at most n in any indeterminate.

Afterwards, $\mathscr{B}$ queries $P_{\overrightarrow{x}}$ to its oracle and sends the value it gets to $\mathscr{A}$. Hence, on the one hand, if $\mathscr{B}$'s oracle outputs $[P_{\overrightarrow{x}}(a_1, \dots, a_\ell) \cdot b]$, then the value $\mathscr{A}$ gets is exactly the evaluation of $\mathsf{CH}_{\mathsf{sym}}$ in $\overrightarrow{x}$ with the key $\overrightarrow{a} \in \mathbb{Z}_p^\ell$ and with the generator $[b]$ (which is a generator as soon as $b \neq 0$). On the other hand, if $\mathscr{B}$'s oracle responds to a query P by random values computed taking into account the linear relations between P and the previously queried polynomials, then the values $\mathscr{A}$ gets are statistically indistinguishable from the values it would get from the symmetric $\mathsf{MPRFRand}$ oracle, as this is exactly how the values output by the symmetric $\mathsf{MPRFRand}$ oracle are computed in order to obtain a polynomial-time simulation.

Hence, we have shown that

$$\mathsf{Adv}^{\mathsf{mprf}}_{\mathsf{CH}_{\mathsf{sym}}}(\mathscr{A}) \leq \frac{p}{p-1} \cdot \mathbf{Adv}^{(\ell,1,n)\text{-}\mathsf{plp}}_{\mathbb{G}}(\mathscr{B})$$

and Lemma 6.4.1 now follows from the PLP theorem (Theorem 5.2.2) and from the fact that the $\mathcal{E}_{1,n}$-MDDH is implied by the n-DDHI assumption. □

6.4.3 Skew-Symmetric Multilinear Pseudorandom Function

In [CGV15], the author left as an open problem the construction of a skew-symmetric multilinear pseudorandom function. The definition of this notion is the same as the notion of multilinear pseudorandom function, except that we only require the function to be indistinguishable from a random *skew-symmetric* multilinear function. We assume that $\ell_1 = \cdots = \ell_n = \ell = n$, i.e., all the vectors $\overrightarrow{x_1}, \ldots, \overrightarrow{x_n}$ have the same size $\ell = n$. We need $\ell = n$ because there is no skew-symmetric n-multilinear map from $\left(\mathbb{Z}_p^\ell\right)^n$ to $\mathbb{Z}_p$, when $\ell < n$.

We know that any skew-symmetric n-multilinear map f is of the form:

$$f(\overrightarrow{x_1}, \ldots, \overrightarrow{x_n}) = c \cdot \det(\overrightarrow{x_1}, \ldots, \overrightarrow{x_n}),$$

with c being a scalar in $\mathbb{Z}_p$ and det being the determinant function. Therefore, the function

$$F(a, (\overrightarrow{x_1}, \ldots, \overrightarrow{x_n})) = [a \cdot \det(\overrightarrow{x_1}, \ldots, \overrightarrow{x_n})]$$

is a skew-symmetric multilinear pseudorandom function with key $a \in \mathbb{Z}_p$.

The proof is trivial since, $(\overrightarrow{x_1}, \ldots, \overrightarrow{x_n}) \mapsto F(a, (\overrightarrow{x_1}, \ldots, \overrightarrow{x_n}))$ is actually a random skew-symmetric n-multilinear map when a is a random scalar in $\mathbb{Z}_p$. No assumption is required. Our analysis shows that skew-symmetric multilinear pseudorandom functions are of limited interest, but our construction still solves an interesting open problem in [CGV15].

6.4.4 Extension to Weaker Assumptions

As for related-key secure pseudorandom functions and aggregate pseudorandom functions, it is very easy to build multilinear pseudorandom functions under k-Lin and to prove their security applying our PLP theorem (Theorem 5.2.2), for instance using the same construction but changing the key components from elements in $\mathbb{Z}_p$ to elements in $\mathbb{Z}_p^{k\times k}$ while keeping the same inputs space, and by defining $\langle \overrightarrow{\boldsymbol{A}}, \overrightarrow{x} \rangle = \sum_{i=1}^{\ell} x_i \cdot \boldsymbol{A_i}$, with $\overrightarrow{\boldsymbol{A}} = (\boldsymbol{A}_1, \ldots, \boldsymbol{A}_\ell) \in (\mathbb{Z}_p^{k\times k})^\ell$ and $x = (x_1, \ldots, x_\ell) \in \mathbb{Z}_p^\ell$. This leads to the following construction, where $\mathsf{F.Setup}$ outputs $(p, \mathbb{G}, g)$ and dimensions $\ell_1, \ldots, \ell_n$, and where the outputs are computed via:

$$\mathsf{F.Eval}_{\mathsf{pp}} : \left(\begin{array}{rcl} \mathbb{Z}_p^{\ell_1} \times \cdots \times \mathbb{Z}_p^{\ell_n} & \to & \mathbb{G}^{k\times m} \\ (\overrightarrow{x_1}, \ldots, \overrightarrow{x_n}) & \mapsto & \left[(\prod_{i=1}^n \langle \overrightarrow{\boldsymbol{A}_i}, \overrightarrow{x_i} \rangle) \cdot \boldsymbol{B} \right] \end{array} \right)$$

with $(\overrightarrow{\boldsymbol{A}_1}, \ldots, \overrightarrow{\boldsymbol{A}_n}) \in (\mathbb{Z}_p^{k\times k})^{\ell_1} \times \cdots \times (\mathbb{Z}_p^{k\times k})^{\ell_n}$ and $\boldsymbol{B} \in \mathbb{Z}_p^{k\times m}$.

Once again, such a result is immediate and we do not provide further details.

Chapter 7

A Provably-Secure Pseudorandom Function For XOR-Relations

In this chapter, we construct the first XOR-related-key secure pseudorandom function that is provably-secure, assuming the existence of a weak form of multilinear maps. As the existence of multilinear maps is still an important controversial matter, this result should mainly be seen as a proof of concept.

However, despite recent devastating attacks against current multilinear maps, we prove that none of these attacks affect the security of our construction. Then, an important contribution of this work is also to explore a different way to prove security of multilinear-map-based constructions by taking attacks into account. In particular, as our construction is secure even with current instantiations, we only require a weak form of multilinear maps, and hope that such a weak form might exist under on standard assumptions.

Contents

7.1 Additional Material

Most of the definitions required for this chapter have already been provided in Chapters 2 and 3, especially definitions of multilinear maps and of the multilinear map model. Please refer to this chapter if necessary. Here, we only recall the definition of the class of XOR relations and provide some conventions and additional material.

7.1.1 Related-Key Security for XOR Relations

XOR Relations. Consider a key space $\mathcal{K} = \{0,1\}^k$ for some integer $k \geq \kappa$. We define the class of XOR relations $\Phi_\oplus$ as the following set of functions:

$$\Phi_\oplus = \{\phi_s\colon\ K \in \{0,1\}^k \mapsto K \oplus s \in \{0,1\}^k \mid s \in \{0,1\}^k\} \ ,$$

and say that a pseudorandom function F is XOR-related-key secure if it is a $\Phi_\oplus$-related-key secure pseudorandom function (see Definition 3.1.1 for details).

7.1.2 Multilinear Maps

Conventions. In order to ease the reading, we adopt the following conventions in the rest of this chapter:

- Scalars are noted with lowercase letter, e.g., $a, b, \dots$
- Encodings are noted either as their encoding at index set $\mathcal{S}$, $[a]_{\mathcal{S}}$ or simply with a hat, when the index set is clear from the context, e.g., $\hat{a}, \hat{b}, \dots$ In particular, $\hat{a}$ is an encoding of the scalar a.
- Index sets and formal variables are noted with uppercase letters, e.g., $X, S, \mathscr{S}, \dots$.
- We denote by $S_1 \cdot S_2$ the union of sets S_1 and S_2. This notation implicitly assumes that the two sets are disjoint. If S_1 is an element, then $S_1 \cdot S_2$ stands for $\{S_1\} \cdot S_2$.
- The top-level index set is refered as $\mathscr{U}$.

We also naturally extend these notations when clear from the context, so for instance $\hat{a} + \hat{b} = \mathsf{MM.Add}(\mathsf{mm.pp}, \hat{a}, \hat{b})$ and $\hat{a} \cdot \hat{b} = \mathsf{MM.Mult}(\mathsf{mm.pp}, \hat{a}, \hat{b})$.

Variant of the Multilinear Map Model. In order to ease the reading, we actually use a slightly different and more intuitive characterization of the generic multilinear map oracle (defined in Section 2.4.2) in our proofs. Informally, instead of considering queries to $\mathsf{MM.ZeroTest}$ as nonces, we consider these as formal polynomials (that can be computed easily), whose formal variables are substituted with their join value distribution from the real game. In our construction, formal variables are $\hat{a}_{i,b}, \hat{c}_{j,b}, \hat{z}_{i_1,i_2,b_1,b_2}$ —please refer to the construction in Section 7.2.2 for details. This variant characterization follows the formalization from [Zim15, Appendix B].

Actual Instantiations of Multilinear Maps. While Definition 2.4.3 is a very natural definition and is what we actually would like as a multilinear map, up to now, we still do not know any such construction. Known constructions [GGH13; CLT13; GGH15; CLT15] of multilinear maps are actually "noisy" variants of our formal definition. That is, each encoding includes a random error term, and similarly to what happens for lattice-based constructions, this error term grows when performing operations (addition or multiplication). Eventually, this error term becomes too big and the MM.ZeroTest can no longer recover the correct answer. This noise implicitly restricts the number of operations that can be performed. Intuitively, in current constructions, the errors are added when performing an addition and multiplied when performing a multiplication. However, the fact that current instantiations are noisy does not pose any problem regarding our construction, as the number of operations for evaluating our pseudorandom function is fixed and well defined.

7.1.3 Straddling Sets

Our construction and its proofs use straddling sets, as described by Barak *et al.* in [BGK+14], in order to prevent the adversary from mixing too many encodings and creating encodings of zero. We recall their definition below. We first recall that, for a set $\mathscr{S}$, we say that $\{S_1, \ldots, S_k\}$, for some integer k, is a partition of $\mathscr{S}$, if and only if $\cup_{i=1}^k S_i = \mathscr{S}$, $S_i \neq \emptyset$ and $S_i \cap S_j = \emptyset$, for any $1 \leq i, j \leq k$, $i \neq j$.

Definition 7.1.1 (Straddling Set System). *For $k \in \mathbb{N}$, a k-*straddling set system *over a set $\mathscr{S}$ consists of two partitions $S_0 = \{S_{0,1}, \ldots, S_{0,k}\}$ and $S_1 = \{S_{1,1}, \ldots, S_{1,k}\}$ of $\mathscr{S}$ such that the following holds: for any $\mathscr{T} \subseteq \mathscr{S}$, if T_0, T_1 are distinct subsequences of $S_{0,1}, \ldots, S_{0,k}, S_{1,1}, \ldots, S_{1,k}$ such that T_0 and T_1 are partitions of $\mathscr{T}$, then $\mathscr{T} = \mathscr{S}$ and $T_0 = S_b$ and $T_1 = S_{1-b}$ for some $b \in \{0,1\}$.*

Intuitively, this implies that the only two solutions to build a partition of $\mathscr{S}$ from combining sets in S_0 or S_1 are to use either every element in S_0 or every element in S_1.

Construction 7.1.2 (Construction of Straddling Set Systems). . *Let k be a fixed integer and let $\mathscr{S} = \{1, \ldots, 2k-1\}$. Then, the following partitions form a k-straddling set system over $\mathscr{S}$:*

$$S_0 = (S_{0,1}, \ldots, S_{0,k}) = (\{1\}, \{2,3\}, \{4,5\}, \ldots, \{2k-4, 2k-3\}, \{2k-2, 2k-1\})$$
$$S_1 = (S_{1,1}, \ldots, S_{1,k}) = (\{1,2\}, \{3,4\}, \ldots, \{2k-3, 2k-2\}, \{2k-1\}) .$$

This construction naturally extends to any set with $2k-1$ elements.

The proof is immediate and thus not detailed here. Please refer to [BGK+14] or [Zim15] for details.

7.2 Our Construction

Let us now describe our construction of an XOR-related-key secure pseudorandom function, for security parameter κ, key space $\mathcal{K} = \{0,1\}^k$, with $k = 2\kappa$, and domain $\mathcal{D} = \{0,1\}^n$ for some integer n.

7.2.1 Intuition

The main idea behind our construction is very natural. The starting point is the Naor-Reingold pseudorandom function, whose evaluation function is defined as $\mathsf{NR.Eval}_{\mathsf{pp}} : (\vec{a}, x) \in$

$\mathbb{Z}_p^{2n} \times \{0,1\}^n \mapsto [\prod_{i=1}^n a_{i,x_i}]$, where $\vec{a}$ is the secret key. As we are interested in XOR relations, we want the key to be a bitstring. A simple solution is to tweak this construction by considering the evaluation $f_{\vec{a},\vec{c}} : (K,x) \in \{0,1\}^k \times \{0,1\}^n \mapsto \left[\prod_{i=1}^k a_{i,K_i} \cdot \prod_{j=1}^n c_{i,x_i}\right]$, with $\vec{a} \in \mathbb{Z}_p^k$ and $\vec{c} \in \mathbb{Z}_p^n$. It is easy to see that without knowing $\vec{a}$ nor $\vec{c}$, the outputs of this function are computationally indistinguishable from random (they correspond to NR evaluations with key $(\vec{a},\vec{c})$ and on input (K,x)). However, given a key $K \in \{0,1\}^k$, one needs the values $\vec{a},\vec{c}$ in order to be able to evaluate this function, so these values need to be made public. Then, it becomes very easy, even without knowing K, to distinguish this function from a random one.

Therefore, our idea is to use a multilinear map: this allows us to publicly reveal low-level encodings of elements in $\vec{a}$ and $\vec{c}$. These encodings let anyone evaluate the function on any key K and any input x, while keeping the outputs of the function computationally indistinguishable from random to an adversary that does not know the secret key K. Formally, we let $\mathscr{U} = \{1,\dots,k+n\}$ be the set of indices for a multilinear map, $\{a_{i,b}\}_{i\in[k],b\in\{0,1\}}$ and $\{c_{j,b}\}_{j\in[n],b\in\{0,1\}}$ be random scalars in $\mathbb{Z}_p$ and $\hat{a}_{i,b} = [a_{i,b}]_{\{i\}}$ be the encoding of $a_{i,b}$ at index index i and $\hat{c}_{j,b} = [c_{j,b}]_{\{j+k\}}$ be the encoding of $c_{j,b}$ at index level $j+k$. We then consider the function whose evaluation is:

$$\mathsf{F.Eval}_{\mathsf{pp}}(K,x) = \left[\prod_{i=1}^k a_{i,K_i} \prod_{j=1}^n c_{j,x_i}\right]_{\mathscr{U}} ,$$

with public parameters pp including $\{\hat{a}_{i,b}\}_{i\in[k],b\in\{0,1\}}$ and $\{\hat{c}_{j,b}\}_{j\in[n],b\in\{0,1\}}$ as well as the public parameters of the multilinear map.

This construction can be easily proven to be an XOR-related-key secure pseudorandom function in the generic multilinear map model, and it is also easy to show that it does not let an adversary create encodings of zero. However, it seems very hard to prove that this construction is secure under a non-interactive assumption. Hence, we slightly modify this construction by using a more complex set of indices and straddling sets. While this makes the proof in the generic multilinear map model a bit harder, this allows us to prove the security of our construction under non-interactive assumptions, whose hardness seems plausible even with current instantiations of multilinear maps. In particular, we prove in Sections 7.5 and 7.6 that these assumptions are secure in the generic multilinear map model and do not let an adversary generate (non-trivial) encodings of zero.

7.2.2 Actual Construction

We define our pseudorandom function $F = (\mathsf{F.Setup}, \mathsf{F.Eval})$ as follows.

Index set. First, similarly to [Zim15], for each $i \in \{0,\dots,k\}$, we construct a k-straddling set system over a set $\mathscr{S}_i$ of $2k-1$ fresh formal symbols. We denote by $S_{i,b}$ the two partitions forming each of this straddling set, for $b \in \{0,1\}$, and by $S_{i,b,j}$ their elements, for $1 \le j \le k$. We also define:

$$\mathsf{BitCommit}_{i,b} = S_{i,b,i} \qquad\qquad \mathsf{BitFill}_{i_1,i_2,b_1,b_2} = S_{i_1,b_1,i_2} \cdot S_{i_2,b_2,i_1}$$

for any $i, i_1, i_2 \in \{1,\dots,k\}$ and $b, b_1, b_2 \in \{0,1\}$. Intuitively, the straddling set systems $\mathscr{S}_i$ for $i \ge 1$ play the same role as in [Zim15] (preventing the adversary from mixing an exponential number of inputs), while $\mathscr{S}_0$ is used in the proof to prevent the adversary from mixing the private representation of the key from the parameters.

Let X_j be fresh formal symbols for $j \in \{1, \dots, n\}$. We then define the top-level index set as follows:

$$\mathscr{U} = \prod_{i=0}^{k} \mathscr{S}_i \prod_{j=1}^{n} X_j \ .$$

Setup. The algorithm $\mathsf{F.Setup}$ first generates the parameters $(\mathsf{mm.pp}, \mathsf{mm.sp}, p)$ for the multilinear map by running $\mathsf{MM.Setup}(1^\kappa, \mathscr{U})$. Then it generates the following elements:

$$\begin{aligned}
a_{i,b} &\xleftarrow{\$} \mathbb{Z}_p && \text{for } i \in \{1, \dots, k\} \text{ and } b \in \{0,1\} \\
\hat{a}_{i,b} &\leftarrow [a_{i,b}]_{S_{0,0,i}\mathsf{BitCommit}_{i,b}} && \text{for } i \in \{1, \dots, k\} \text{ and } b \in \{0,1\} \\
c_{j,b} &\xleftarrow{\$} \mathbb{Z}_p && \text{for } j \in \{1, \dots, n\} \text{ and } b \in \{0,1\} \\
\hat{c}_{j,b} &\leftarrow [c_{i,b}]_{X_i} && \text{for } j \in \{1, \dots, n\} \text{ and } b \in \{0,1\} \\
\hat{z}_{i_1,i_2,b_1,b_2} &\leftarrow [1]_{\mathsf{BitFill}_{i_1,i_2,b_1,b_2}} && \text{for } i_1, i_2 \in \{1, \dots, k\} \text{ and } b_1, b_2 \in \{0,1\} \\
& && \text{with } i_1 < i_2 \,,
\end{aligned}$$

and outputs the following parameters:

$$\mathsf{pp} = \Big(\mathsf{mm.pp}, (\hat{a}_{i,b})_{i,b}, (\hat{c}_{j,b})_{j,b}, (\hat{z}_{i_1,i_2,b_1,b_2})_{i_1,i_2,b_1,b_2}\Big) \ .$$

Intuitively, $\mathsf{BitCommit}_{i,b} = S_{i,b,i}$ is associated to the (public) encoding used to evaluate the function if the i-th bit of the key is b. By definition of a straddling set, the only way to reach the top-level $\mathscr{U}$, which contains $\mathscr{S}_i$, once we have used an encoding with index $S_{i,b,i}$ is to use every index $S_{i,b,j}$ with $j \neq i$. These is done by multiplying the terms $\hat{z}_{i,j,K_i,K_j}$. Therefore, using $K_i = b$ is like "committing" to the partition $S_{i,b}$ of $\mathscr{S}_i$, and terms $\hat{z}_{i,j,K_i,K_j}$ are then used to "fill" this partition.

Remark 7.2.1. For the sake of simplicity, we set $\hat{z}_{i_1,i_2,b_1,b_2}$ to be encodings of 1, but one could also simply set it to encodings of a random value, as soon as they are all encodings of the same value.

Evaluation. The output of the pseudorandom function on a key $K \in \{0,1\}^k$ and an input $x \in \{0,1\}^n$ is

$$\mathsf{F.Eval}_{\mathsf{pp}}(K, x) = \mathsf{MM.Extract}(\prod_{i=1}^{k} \hat{a}_{i,K_i} \prod_{j=1}^{n} \hat{c}_{j,x_j} \prod_{i_1=1}^{k} \prod_{i_2=i_1+1}^{k} \hat{z}_{i_1,i_2,K_{i_1},K_{i_2}}) \ .$$

We can re-write it as:

$$\mathsf{F.Eval}_{\mathsf{pp}}(K, x) = \mathsf{MM.Extract}(\left[\prod_{i=1}^{k} a_{i,K_i} \prod_{j=1}^{n} c_{j,x_i}\right]_{\mathscr{U}}) \ .$$

Extraction. As explained in Remark 2.4.4, the role of extraction ($\mathsf{MM.Extract}$) is dual. First, it ensures the correctness of the pseudorandom function, as in currently known instantiations of multilinear maps [GGH13; CLT13; GGH15; CLT15], a scalar has many different encodings.

Second, it is used in our proof of security under non-interactive assumptions in Section 7.4.2, as in the security proof we change the way the group element

$$\prod_{i=1}^{k} \hat{a}_{i,K_i} \prod_{j=1}^{n} \hat{c}_{j,x_j} \prod_{i_1=1}^{k} \prod_{i_2=i_1+1}^{k} \hat{z}_{i_1,i_2,K_{i_1},K_{i_2}}$$

is computed. We indeed recall that due to the fact that a scalar has many different encodings, any group element (as the above one) leaks information on the exact computation used to obtain it, instead of just depending on its discrete logarithm. The usual way to solve this issue is to randomize the resulting group element using encodings of zero. However, we do not want to use any encoding of zero, hence the requirement for this extraction. For the proof in the generic multilinear map model in Section 7.3, this is not an issue, and we just ignore the extraction (see Remark 2.4.4).

As discussed in Remark 2.4.4, with current instantiations of multilinear maps [GGH13; CLT13; GGH15; CLT15], the output of the pseudorandom function has to be the extracted value of the group elements previously indicated, for the evaluation to be deterministic.

7.3 Security in the Generic Multilinear Map Model

In this section, we prove the security of our construction in the generic multilinear map model. As already explained in Remark 2.4.4, we suppose in this section that no extraction is performed. We actually even prove that no polynomial-time adversary can construct a (non-trivial) encoding of zero, in any of the two experiments $\mathsf{RKPRFReal}_F$ and $\mathsf{RKPRFRand}_F$, with non-negligible probability. This implies, in particular, that these two experiments cannot be distinguished by a polynomial-time adversary, in the generic multilinear map model, as an adversary only sees handles which only leak information when two of them correspond to the same (top-level) element.

This section is mainly a warm-up to familiarize with the model, since the fact that we prove our construction under some assumptions that are proven secure in the generic multilinear map model and proven to not let an adversary generate encodings of zero also implies the results below. However, it is very simple to modify the proof below in order to prove the security of the simplified construction proposed in Section 7.2.1, which is of independent interest.

We first need to formally define the notion of (non-trivial) encoding of zero. We follow the definition of Badrinarayanan *et al.* [BMSZ16].

Definition 7.3.1 ((Non-trivial) encoding of zero). *An adversary $\mathscr{A}$ in the generic multilinear map model with multilinear map oracle $\mathscr{M}$ returns a* (non-trivial) encoding of zero *if it returns a handle h (output by $\mathscr{M}$) such that h corresponds to the element* 0 *in $\mathscr{M}$'s table and the polynomial corresponding to the handle is not identically null.*

Theorem 7.3.2 (Impossibility of constructing encodings of zero). *In the generic multilinear map model with oracle $\mathscr{M}$, for any adversary $\mathscr{A}$ making at most $q_{\mathscr{M}}$*

*queries to the oracle $\mathscr{M}$ and $q_{\boldsymbol{RKFn}}$ queries to the oracle **RKFn**, we have:*

$$\Pr\left[\mathsf{PRFReal}_F^{\mathscr{A}} \Rightarrow \textit{an encoding of } 0\right] \le q_{\mathscr{M}}\left(\frac{q_{\boldsymbol{RKFn}}}{2^k} + \frac{k+n}{p}\right)$$

and

$$\Pr\left[\mathsf{PRFRand}_F^{\mathscr{A}} \Rightarrow \textit{an encoding of } 0\right] \le q_{\mathscr{M}}\frac{k+n}{p} \quad .$$

Proof of Theorem 7.3.2. We first introduce a technical lemma.

Lemma 7.3.3. *Let k and n be two positive integers. Let $\mathscr{U}$ be the index defined in Section 7.2.2. Let $\hat{z}_{i_1,i_2,b_1,b_2} = [1]_{\mathsf{BitFill}_{i_1,i_2,b_1,b_2}}$ for $1 \le i_1 < i_2 \le k$ and $b_1, b_2 \in \{0,1\}$. Let Z_1 and Z_2 be two subsets of $\{(i_1,i_2) \mid 1 \le i_1 < i_2 \le k\} \times \{0,1\}^2$. If $t_1 = \prod_{(i_1,i_2,b_1,b_2)\in Z_1} \hat{z}_{i_1,i_2,b_1,b_2}$ and $t_2 = \prod_{(i_1,i_2,b_1,b_2)\in Z_2} \hat{z}_{i_1,i_2,b_1,b_2}$ have the same index set, then $Z_1 = Z_2$.*

Proof of Lemma 7.3.3. Let T denote the index set of t_1 and t_2. For $i = 1,\dots,k$, one can interstect T with $\mathscr{S}_i$. As T cannot contain $\mathscr{S}_i$ (since it cannot contain $S_{i,b,i}$ for any $b \in \{0,1\}$ as it is not contained in any index set of any $\hat{z}_{i_1,i_2,b_1,b_2}$), we have $T \cap \mathscr{S}_i \ne \mathscr{S}_i$. Therefore, for any $(i_1,i_2,b_1,b_2) \in Z_1$ with $i_1 < i_2$, as S_{i_1,b_1,i_2} is in the index set of t_1 and $T \cap \mathscr{S}_{i_1} \ne \mathscr{S}_{i_1}$, there exists b' such that $S_{i_1,b_1,i_2,b'}$ is in Z_2 (so that S_{i_1,b_1,i_2} is in the index set of t_2), by definition of a straddling set system. Then, there are two possibilities: either $b' = b_2$ and $(i_1,i_2,b_1,b_2) \in Z_2$, or $b' = 1 - b_2$. In the latter case, this means that S_{i_2,b_2,i_1} is contained in the index set of t_1 and $S_{i_2,1-b_2,i_1}$ is contained in the index set of t_2. As $T \cap \mathscr{S}_{i_2} \ne \mathscr{S}_{i_2}$, this contradicts the fact that t_1 and t_2 have the same index set. Lemma 7.3.3 immediately follows. □

We need to show that the adversary cannot generate a non-trivial encoding of zero.

$\mathsf{RKPRFRand}_F$. We start by proving it in the game $\mathsf{RKPRFRand}_F$. In this game, except for $\hat{z}_{i_1,i_2,b_1,b_2}$, all the handles the adversary sees correspond to fresh new formal variables, as the oracle **RKFn** only returns fresh new formal variables (of index $\mathscr{U}$). The only polynomials the adversary can generate are therefore of the form:

$$P = \sum_{\ell=1}^{L} Q_\ell \prod_{(i_1,i_2,b_1,b_2)\in Z_\ell} \hat{z}_{i_1,i_2,b_1,b_2} \quad ,$$

where Q_ℓ are polynomials over all the elements except $\hat{z}_{i_1,i_2,b_1,b_2}$, and Z_ℓ are distinct subsets of $\{(i_1,i_2) \mid 1 \le i_1 < i_2 \le k\} \times \{0,1\}^2$ (L might be exponential in $q_{\mathscr{M}}$, but that does not matter for what follows).

Let us now show that if P is not the zero polynomial, then when replacing $\hat{z}_{i_1,i_2,b_1,b_2}$ by 1, the resulting polynomial is still a non-zero polynomial. From Lemma 7.3.3, one can assume that elements $\prod_{(i_1,i_2,b_1,b_2)\in Z_\ell} \hat{z}_{i_1,i_2,b_1,b_2}$ all have distinct indices. Therefore, the polynomials Q_ℓ all have distinct indices too. No monomial in two different Q_ℓ of the sum $\sum_\ell Q_\ell$ (when forgetting the indices) can therefore cancel out, otherwise this would mean that the adversary can construct two equal monomials (without $\hat{z}_{i,b}$) with two different indices. This is impossible as, except for $\hat{z}_{i,b}$, two distinct handles correspond to two fresh variables (or in other words, all the handles except $\hat{z}_{i,b}$ are encodings of scalars chosen uniformly and independently at random).

We therefore simulate the oracle $\mathscr{M}$ as follows: we do everything as normal, but we make the zero-testing oracle always output "non-zero" except when its input corresponds to the zero polynomial. The Schwarz-Zippel lemma ensures that any non-zero polynomial of degree at most $k+n$ and whose variables are fixed to uniformly random values in $\mathbb{Z}_p$ does not evaluate to zero, except with probability at most $(k+n)/p$. In other words, the zero-testing oracle outputs "zero" on a non-zero polynomial with probability at most $(k+n)/p$, as this polynomial remains non-zero and has degree at most $(k+n)$, when we replace $\hat{z}_{i_1,i_2,b_1,b_2}$ by 1. As we can suppose that the zero-testing oracle is queried with the output of the adversary without loss of generality, using at most $q_{\mathscr{M}}$ hybrid games (replacing one-by-one every output of the zero-testing oracle with "non-zero") we get that:

$$\Pr\left[\mathsf{PRFRand}_F^{\mathscr{A}} \Rightarrow \text{an encoding of } 0\right] \leq q_{\mathscr{M}} \frac{k+n}{p} \quad .$$

$\mathsf{RKPRFReal}_F$. Let us now look at the game $\mathsf{RKPRFReal}_F$. The analysis is more complicated as the adversary has access to new formal variables

$$\hat{y}_{s,x} = F_{\mathsf{pp}}(K \oplus s, x)$$

returned by queries $\mathbf{RKFn}(\phi_s, x)$.

We use the same simulator as in the previous case. We need to show that if a polynomial P produced by the adversary is not zero, it remains non-zero when $\hat{z}_{i_1,i_2,b_1,b_2}$ is replaced by 1 and $\hat{y}_{s,x}$ is replaced by its value.

We first consider the case where P is not a top-level polynomial. In this case, P cannot contain these new variables $\hat{y}_{s,x}$ as these variables are top-level. Then, as in $\mathsf{RKPRFRand}_F$, the zero-testing oracle outputs "non-zero" except with probability at most $(k+n)/p$.

Let us now suppose that P is a top-level polynomial. This polynomial has the form:

$$P = \sum_{\ell=1}^{L} Q_\ell \prod_{i=1}^{k} \hat{a}_{i,K'_{\ell,i}} \prod_{i_1=1}^{k} \prod_{i_2=i_1+1}^{k} \hat{z}_{i_1,i_2,K'_{\ell,i_1},K'_{\ell,i_2}} + \sum_{j=1}^{q'} \lambda_j \hat{y}_{s_j,x_j} \ ,$$

with L being a non-negative integer (possibly exponential in $q_{\mathscr{M}}$), Q_ℓ being non-zero polynomials in the formal variables $\hat{c}_{j,b}$, K'_ℓ being distinct bitstrings in $\{0,1\}^k$ (chosen by the adversary), q' an integer less or equal to $q_{\mathbf{RKFn}}$, $(s_1,x_1),\ldots,(s_{q'},x_{q'})$ queries to $\mathbf{RKFn}$, and λ_j some scalar in $\mathbb{Z}_p$. Indeed, the adversary can ask for an encoding of any polynomial of the form $Q_\ell \prod_{i=1}^k \hat{a}_{i,K'_{\ell,i}}$, and by definition of straddling set systems, the unique way to obtain a top-level encoding from such a polynomial is by multiplying it with an encoding of $\prod_{i_1=1}^k \prod_{i_2=i_1+1}^k \hat{z}_{i_1,i_2,K'_{\ell,i_1},K'_{\ell,i_2}}$.

Let us suppose that P is not zero but becomes zero when $\hat{z}_{i_1,i_2,b_1,b_2}$ is replaced by 1 and $\hat{y}_{s,x}$ is replaced by its value. In this case, in particular, the first monomial (for any order) of the term

$$Q_1 \prod_{i=1}^{k} \hat{a}_{i,K'_{1,i}} \prod_{i_1=1}^{k} \prod_{i_2=i_1+1}^{k} \hat{z}_{i_1,i_2,K'_{1,i_1},K'_{1,i_2}}$$

necessarily needs to be canceled out by some $\hat{y}_{s_j,x_j}$. The probability over $K \xleftarrow{\$} \{0,1\}^k$ that this happens is at most:

$$\Pr\left[\, \exists j' \in \{1,\ldots,q'\},\ K'_1 = K \oplus s_{j'} \,\right] \leq \frac{q'}{2^k} \leq \frac{q_{\mathbf{RKFn}}}{2^k} \quad .$$

As before, thanks to the Schwarz-Zippel lemma, we get that the zero-testing oracle outputs "zero", on input a non-zero polynomial, with probability at most:

$$\frac{q_{\mathbf{RKFn}}}{2^k} + \frac{k+n}{p} \ .$$

This concludes the proof of Theorem 7.3.2. □

Remark 7.3.4. We never use the properties of the straddling set system $\mathscr{S}_0$ in this proof. These properties are only used in our proof under non-interactive assumptions in Section 7.4.2.

We obtain the following immediate corollary.

Corollary 7.3.5 (Security in the generic multilinear map model)**.** *F is an XOR-related-key secure pseudorandom function in the generic multilinear map model.*

Proof of Corollary 7.3.5. We just consider an intermediate game where we simulate everything as before except the zero-testing oracle which always outputs "non-zero" unless its input is zero, as a polynomial. This game is indistinguishable from both $\mathsf{RKPRFReal}_F$ and $\mathsf{RKPRFRand}_F$ according to Theorem 7.3.2 (up to the bounds in this lemma). Thus, we have:

$$\mathsf{Adv}^{\mathsf{rka\text{-}prf}}_{\Phi_\oplus,F}(\mathscr{A}) \le \frac{q_{\mathscr{M}} q_{\mathbf{RKFn}}}{2^k} + \frac{2q_{\mathscr{M}}(k+n)}{p} \ ,$$

and Corollary 7.3.5 follows. □

7.4 Security under Non-Interactive Assumptions

7.4.1 Two Non-Interactive Assumptions

We use two assumptions that we call the (k,n,X,Y)-XY-DDH assumption, which is roughly a generalization of the standard DDH assumption, and the (k,n)-Sel-Prod assumption. We show in Sections 7.5 and 7.6 that both these assumptions are secure in the generic multilinear map model, and even that an adversary against these assumptions cannot generate encodings of zero. Hence, contrary to most assumptions considered on multilinear maps (e.g., classical DDH-like assumptions and the multilinear subgroup assumption [GLW14; GLSW15]), these assumptions are therefore plausible at least with two current instantiations of multilinear maps [CLT13; GGH15].

To ensure the impossibility of generating encodings of zero, in these two assumptions, we restrict the adversary's capabilities as follows: it is only provided parameters mm.pp, so it can only run MM.Add, MM.Mult, and MM.ZeroTest (and of course use the elements generated by the assumption), but we do not allow the adversary to generate new encodings of a chosen scalar. In particular, this forces us to let the assumption contains the group elements $\hat{z}_{i_1,i_2,b_1,b_2}$. It is straightforward to get rid of these additional elements by allowing the adversary to generate any element of the multilinear map, at the cost of getting an implausible assumption under current instantiations of multilinear maps.

Finally, our assumption implicitly contains a list $\mathcal{L}$ of a polynomial number of encodings of random values at non-zero indices, indices being implicit parameters of the assumption. We could avoid this artifact with the previous proposition as well, or by giving a sufficient number of encodings of 0 and 1, but once again, in that case, the assumption would most likely not hold with currently known multilinear maps instantiations. We believe this is a small prize to pay to get plausible assumptions, as the resulting assumptions are still non-interactive.

Definition 7.4.1 ((k,n,X,Y)-$\mathsf{XY\text{-}DDH}$). *Let k and n be two positive integers. Let X and Y be two non-empty and disjoint indices in the index set $\mathscr{U}$ of our construction in Section 7.2.2. The advantage of an adversary $\mathscr{D}$ against the (k,n,X,Y)-$\mathsf{XY\text{-}DDH}$ problem is:*

$$\mathsf{Adv}^{(k,n,X,Y)\text{-}\mathsf{XY\text{-}DDH}}(\mathscr{D}) := \Pr\left[\,(k,n,X,Y)\text{-}\mathsf{XY\text{-}DDHReal}^{\mathscr{D}} \Rightarrow 1\,\right] - \Pr\left[\,(k,n,X,Y)\text{-}\mathsf{XY\text{-}DDHRand}^{\mathscr{D}} \Rightarrow 1\,\right],$$

where the games (k,n,X,Y)-$\mathsf{XY\text{-}DDHReal}^{\mathscr{D}}$ and (k,n,X,Y)-$\mathsf{XY\text{-}DDHRand}^{\mathscr{D}}$ are defined in Figure 7.1.

This assumption is very close to the classical DDH assumption with indices, with two main differences: the presence of elements $\hat{z}_{i_1,i_2,b_1,b_2}$ which are necessary to prove our construction and the implicit presence of encodings of random values at non-zero indices (list $\mathcal{L}$ described previously) instead of a polynomial number of encodings of 0 and 1. Without the elements $\hat{z}_{i_1,i_2,b_1,b_2}$, the proof of this assumption in the generic multilinear map model would be completely straightforward. The difficulty of the proof is to deal with these elements.

In the security proof of our construction, this assumption is used in a similar way as the DDH assumption in the proof of the Naor-Reingold pseudorandom function.

Definition 7.4.2 ((k,n)-$\mathsf{Sel\text{-}Prod}$). *Let k and n be two positive integers. The advantage of an adversary $\mathscr{D}$ against the (k,n,X,Y)-$\mathsf{Sel\text{-}Prod}$ problem is:*

$$\mathsf{Adv}^{(k,n)\text{-}\mathsf{Sel\text{-}Prod}}(\mathscr{D}) := \Pr\left[\,(k,n)\text{-}\mathsf{Sel\text{-}ProdReal}^{\mathscr{D}} \Rightarrow 1\,\right] - \Pr\left[\,(k,n)\text{-}\mathsf{Sel\text{-}ProdRand}^{\mathscr{D}} \Rightarrow 1\,\right],$$

where the games (k,n)-$\mathsf{Sel\text{-}ProdReal}^{\mathscr{D}}$ and (k,n)-$\mathsf{Sel\text{-}ProdRand}^{\mathscr{D}}$ are defined in Figure 7.1.

Intuitively, under this assumption, we can replace the encodings $\hat{a}_{i,K_i \oplus s_i}$ at index from the first partition of the straddling set, used in the computation of the output with relation s, to encodings of uniformly random scalars at index from the second partition. In particular, doing this change, we no longer need to know the key K to simulate correctly the output, but only the relations s for each query.

Remark 7.4.3. For the sake of simplicity, we do not explicitly specify the noise level in our assumptions. It can easily be made to work with our proof.

7.4.2 Security of our Construction

In this whole section, we set $\mathscr{S} = \prod_{i=0}^{k} \mathscr{S}_i$ and $\mathscr{S}' = \prod_{i=1}^{k} \mathscr{S}_i$, so $\mathscr{S} = \mathscr{S}_0 \cdot \mathscr{S}'$. Please refer to Section 7.2.2 for notation.

Theorem 7.4.4 (Security Under Non-Interactive Assumptions). *Assuming the (k,n)-$\mathsf{Sel\text{-}Prod}$ and the $(k,n,\mathscr{S}'\prod_{i=1}^{i'-1} S_{0,1,i}, S_{0,1,i'})$-$\mathsf{XY\text{-}DDH}$ assumptions hold, then F is an XOR-related-key secure pseudorandom function.*

Moreover, the running time of this reduction is polynomial in n and in the running time of the adversary.

We provide a sketch of the proof before describing the full proof below.

Proof Overview. The proof follows a sequence of hybrid games. The first hybrid corresponds exactly to $\mathsf{RKPRFReal}_F$, while the last game corresponds to $\mathsf{RKPRFRand}_F$. Here is how we

(k, n, X, Y)-XY-DDHReal$^{\mathscr{D}}$

```
proc Initialize
$(\mathsf{mm.sp}, \mathsf{mm.pp}) \xleftarrow{\$} \mathsf{MM.Setup}(1^\kappa, \mathscr{U})$
$d_0, d_1, \boxed{e \xleftarrow{\$} \mathbb{Z}_p}$
$\hat{d}_0 \leftarrow [d_0]_X$;    $\hat{d}_1 \leftarrow [d_1]_X$
$\boxed{\hat{e}_0 \leftarrow [ed_0]_{XY}}$;    $\boxed{\hat{e}_1 \leftarrow [ed_1]_{XY}}$
For $i_1, i_2 \in \{1, \dots, k\}$ and $b_1, b_2 \in \{0, 1\}$
    $\hat{z}_{i_1,i_2,b_1,b_2} \leftarrow [1]_{\mathsf{BitFill}_{i_1,i_2,b_1,b_2}}$
Return $(\mathcal{L}, \mathsf{mm.pp}, \hat{d}_0, \hat{d}_1, \hat{e}_0, \hat{e}_1, (\hat{z}_{i_1,i_2,b_1,b_2}))$

proc Finalize($b$)
Return $b$
```

(k, n, X, Y)-XY-DDHRand$^{\mathscr{D}}$

```
proc Initialize
$(\mathsf{mm.sp}, \mathsf{mm.pp}) \xleftarrow{\$} \mathsf{MM.Setup}(1^\kappa, \mathscr{U})$
$d_0, d_1, \boxed{e_0, e_1 \xleftarrow{\$} \mathbb{Z}_p}$
$\hat{d}_0 \leftarrow [d_0]_X$;    $\hat{d}_1 \leftarrow [d_1]_X$
$\boxed{\hat{e}_0 \leftarrow [e_0]_{XY}}$;    $\boxed{\hat{e}_1 \leftarrow [e_1]_{XY}}$
For $i_1, i_2 \in \{1, \dots, k\}$ and $b_1, b_2 \in \{0, 1\}$
    $\hat{z}_{i_1,i_2,b_1,b_2} \leftarrow [1]_{\mathsf{BitFill}_{i_1,i_2,b_1,b_2}}$
Return $(\mathcal{L}, \mathsf{mm.pp}, \hat{d}_0, \hat{d}_1, \hat{e}_0, \hat{e}_1, (\hat{z}_{i_1,i_2,b_1,b_2}))$

proc Finalize($b$)
Return $b$
```

(k, n)-Sel-ProdReal$^{\mathscr{D}}$

```
proc Initialize
$(\mathsf{mm.sp}, \mathsf{mm.pp}) \xleftarrow{\$} \mathsf{MM.Setup}(1^\kappa, \mathscr{U})$
$\boxed{K \xleftarrow{\$} \{0, 1\}^k}$
For $i \in \{1, \dots, k\}$ and $b \in \{0, 1\}$
    $a_{i,b} \xleftarrow{\$} \mathbb{Z}_p$;    $\boxed{\gamma_{i,K_i \oplus b} \leftarrow a_{i,b}}$
    $\hat{a}_{i,b} \leftarrow [a_{i,b}]_{S_{0,0,i}\mathsf{BitCommit}_{i,b}}$
    $\hat{\gamma}_{i,K_i \oplus b} \leftarrow [\gamma_{i,K_i \oplus b}]_{S_{0,1,i}\mathsf{BitCommit}_{i,K_i \oplus b}}$
For $i_1, i_2 \in \{1, \dots, k\}$ and $b_1, b_2 \in \{0, 1\}$
    $\hat{z}_{i_1,i_2,b_1,b_2} \leftarrow [1]_{\mathsf{BitFill}_{i_1,i_2,b_1,b_2}}$
Return $(\mathcal{L}, \mathsf{mm.pp}, (\hat{a}_{i,b}), (\hat{\gamma}_{i,b}), (\hat{z}_{i_1,i_2,b_1,b_2}))$

proc Finalize($b$)
Return $b$
```

(k, n)-Sel-ProdRand$^{\mathscr{D}}$

```
proc Initialize
$(\mathsf{mm.sp}, \mathsf{mm.pp}) \xleftarrow{\$} \mathsf{MM.Setup}(1^\kappa, \mathscr{U})$

For $i \in \{1, \dots, k\}$ and $b \in \{0, 1\}$
    $a_{i,b} \xleftarrow{\$} \mathbb{Z}_p$;    $\boxed{\gamma_{i,b} \xleftarrow{\$} \mathbb{Z}_p}$
    $\hat{a}_{i,b} \leftarrow [a_{i,b}]_{S_{0,0,i}\mathsf{BitCommit}_{i,b}}$
    $\hat{\gamma}_{i,b} \leftarrow [\gamma_{i,b}]_{S_{0,1,i}\mathsf{BitCommit}_{i,b}}$
For $i_1, i_2 \in \{1, \dots, k\}$ and $b_1, b_2 \in \{0, 1\}$
    $\hat{z}_{i_1,i_2,b_1,b_2} \leftarrow [1]_{\mathsf{BitFill}_{i_1,i_2,b_1,b_2}}$
Return $(\mathcal{L}, \mathsf{mm.pp}, (\hat{a}_{i,b}), (\hat{\gamma}_{i,b}), (\hat{z}_{i_1,i_2,b_1,b_2}))$

proc Finalize($b$)
Return $b$
```

Figure 7.1: Games defining the XY-DDH and Sel-Prod problems

proceed. First, instead of computing the output using encodings $\hat{a}_{i,b}$ of $a_{i,b}$ with index $S_{0,0,i}\mathsf{BitCommit}_{i,b}$, we use encodings $\hat{\gamma}_{i,b}$ of $a_{i,K_i \oplus b}$ with index $S_{0,1,i}\mathsf{BitCommit}_{i,K_i \oplus b}$. That is, we use the second partition $S_{0,1}$ of the straddling set $\mathscr{S}_0$ instead of the first one ($S_{0,0}$) to reach top-level index (which contains $\mathscr{S}_0$). Also, we now compute the output using only the relation s instead of the key K. More precisely, the output on a query (s, x) is computed as:

$$\mathsf{MM.Extract}(\prod_{i=1}^{k} \hat{\gamma}_{i,s_i} \prod_{j=1}^{n} \hat{c}_{j,x_j} \prod_{i_1=1}^{k} \prod_{i_2=i_1+1}^{k} \hat{z}_{i_1,i_2,K_{i_1},K_{i_2}}),$$

which can be computed without knowing K. This does not change anything regarding the output (thanks to the extraction), so these two games are indistinguishable.

However, using the (k, n)-Sel-Prod assumption, we can now switch the encodings $\hat{\gamma}_{i,b}$ to encodings of fresh random scalars $\gamma_{i,b} \in \mathbb{Z}_p$. The rest of the proof is very similar to the proof of the Naor-Reingold pseudorandom function. We do $k+n$ hybrid games, where in the j-th hybrid, we just switch products of encodings $\prod_{i=1}^{j} \hat{\gamma}_{i,s_i}$ to encodings of uniformly fresh random values using the $\mathsf{XY\text{-}DDH}$ assumption with the proper indices. These modifications are done in a lazy fashion to obtain a polynomial-time reduction.

Proof of Theorem 7.4.4. The proof is based on the games in Figure 7.2 and Figure 7.3. We consider the games in this order:

$$\mathbf{G}_0, \mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_3, \mathbf{G}_{4,2}, \ldots, \mathbf{G}_{4,k}, \mathbf{G}_{5,1}, \ldots, \mathbf{G}_{5,n} \ .$$

For the sake of simplicity, and as already mentioned in Remark 7.4.3, we assume that the multilinear map is clean and the representation is unique. In particular, these assumptions allow us to avoid using an extractor and to assume that two equivalent games are perfectly indistinguishable (and not only statistically indistinguishable). Our proof could be easily adapted to avoid this assumption.

Game $\mathbf{G}_0$. This game exactly corresponds to $\mathsf{RKPRFReal}_F$:

$$\Pr\left[\mathsf{RKPRFReal}_F^{\mathscr{A}} \Rightarrow 1\right] = \Pr\left[\mathbf{G}_0 \Rightarrow 1\right] \ .$$

Game $\mathbf{G}_1$. In this game, outputs of $\mathbf{RKFn}(\phi_s, x)$ are generated using $\hat{\gamma}_{i,s_i}$ instead of $\hat{a}_{i,K_i \oplus s_i}$, where:

$$\hat{a}_{i,b} = [a_{i,b}]_{S_{0,0,i}\mathsf{BitCommit}_{i,b}}$$
$$\hat{\gamma}_{i,b} = [a_{i,K_i \oplus b}]_{S_{0,1,i}\mathsf{BitCommit}_{i,b}} \ .$$

This game is perfectly indistinguishable from the previous one, so:

$$\Pr\left[\mathbf{G}_0 \Rightarrow 1\right] = \Pr\left[\mathbf{G}_1 \Rightarrow 1\right] \ .$$

Game $\mathbf{G}_2$. In this game, $\hat{\gamma}_{i,b}$ are now chosen uniformly at random and independently of the public parameters $\hat{a}_{i,b}$. It is immediate to see that games $\mathbf{G}_1$ and $\mathbf{G}_2$ are indistinguishable under the (k,n)-$\mathsf{Sel\text{-}Prod}$ assumption. More precisely, we can construct an adversary $\mathscr{D}$ running in time similar to $\mathscr{A}$ such that:

$$\Pr\left[\mathbf{G}_1 \Rightarrow 1\right] - \Pr\left[\mathbf{G}_2 \Rightarrow 1\right] \leq \mathsf{Adv}^{(k,n)\text{-}\mathsf{Sel\text{-}Prod}}(\mathscr{D}) \ .$$

Game $\mathbf{G}_3$. In this game, we just slightly change the indices of $\hat{\gamma}_{1,0}$ and $\hat{\gamma}_{1,1}$, so that combining elements $\hat{\gamma}_{i,b}$ and $\hat{c}_{j,x_j}$ directly leads to a top-level encoding (without any need of the $\hat{z}_{i_1,i_2,b_1,b_2}$'s). As elements $\hat{\gamma}_{1,0}$ and $\hat{\gamma}_{1,1}$ are never directly revealed to the adversary, this game is perfectly indistinguishable from the previous one, so:

$$\Pr\left[\mathbf{G}_2 \Rightarrow 1\right] = \Pr\left[\mathbf{G}_3 \Rightarrow 1\right] \ .$$

We also call this game: Game $\mathbf{G}_{4,0}$.

Game $\mathbf{G}_{4,i'}$. In this game, $\prod_{i=1}^{i'} \hat{\gamma}_{i,s_i}$ are replaced by independent random elements for each tuple $(s_1, \ldots, s_{i'}) \in \{0,1\}^{i'}$ queried by the adversary (so only for polynomially many such tuples). Game $\mathbf{G}_{4,i'}$ is indistinguishable from Game $\mathbf{G}_{4,i'-1}$, under the $(k, n, \mathscr{S}' \prod_{i=1}^{i'-1} S_{0,1,i}, S_{0,1,i'})$-

$\mathsf{XY\text{-}DDH}$ assumption. More precisely, we can construct an adversary $\mathscr{B}_{i'}$ running in time similar to $\mathscr{A}$ such that:

$$\Pr[\mathbf{G}_{4,i'-1} \Rightarrow 1] - \Pr[\mathbf{G}_2 \Rightarrow 1] \le q_{\mathbf{RKFn}} \cdot \mathsf{Adv}^{(k,n,\mathscr{S}' \prod_{i=1}^{i'-1} S_{0,1,i}, S_{0,1,i'})\text{-}\mathsf{XY\text{-}DDH}}(\mathscr{B}_{i'}) \ . \quad (7.1)$$

The reduction is immediate from the definition of the $(k, n, \mathscr{S}' \prod_{i=1}^{i'-1} S_{0,1,i}, S_{0,1,i'})$-$\mathsf{XY\text{-}DDH}$ assumption. We also call Game $\mathbf{G}_{4,k}$, Game $\mathbf{G}_{5,0}$.

Game $\mathbf{G}_{5,j'}$. In this game, $\prod_{i=1}^{k} \hat{\gamma}_{i,s_i} \prod_{j=1}^{j'} \hat{b}_{i,x_i}$ are also replaced by independent random elements for each tuple $(s_1, \dots, s_k, x_1, \dots, x_{j'}) \in \{0,1\}^{k+j'}$ queried by the adversary (so only for polynomially many such tuples). Game $\mathbf{G}_{5,j'}$ is indistinguishable from Game $\mathbf{G}_{5,j'-1}$, under the $(k, n, \mathscr{S} \prod_{j=1}^{j'-1} X_j, X_{j'})$-$\mathsf{XY\text{-}DDH}$ assumption. More precisely, we can construct an adversary $\mathscr{C}_{j'}$ running in time similar to $\mathscr{A}$ such that:

$$\Pr[\mathbf{G}_{4,i'-1} \Rightarrow 1] - \Pr[\mathbf{G}_2 \Rightarrow 1] \le q_{\mathbf{RKFn}} \cdot \mathsf{Adv}^{(k,n,\mathscr{S} \prod_{j=1}^{j'-1} X_j, X_{j'})\text{-}\mathsf{XY\text{-}DDH}}(\mathscr{C}_{j'}) \ .$$

Once again, the proof is immediate from the definition of the $(k, n, \mathscr{S} \prod_{j=1}^{j'-1} X_j, X_{j'})$-$\mathsf{XY\text{-}DDH}$ assumption.

Finally, we remark that Game $\mathbf{G}_{5,n}$ is exactly $\mathsf{RKPRFRand}_F^{\mathscr{A}}$ and therefore:

$$\Pr[\mathbf{G}_{5,n} \Rightarrow 1] = \Pr\left[\mathsf{RKPRFRand}_F^{\mathscr{A}} \Rightarrow 1\right] \ .$$

By summing all the previous bounds, we obtain the following bound:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{rka\text{-}prf}}_{\Phi_\oplus,F}(\mathscr{A}) \le \mathsf{Adv}^{(k,n)\text{-}\mathsf{Sel\text{-}Prod}}(\mathscr{D}) + \\ \sum_{i'=2}^{k} q_{\mathbf{RKFn}} \cdot \mathsf{Adv}^{(k,n,\mathscr{S}' \prod_{i=1}^{i'-1} S_{0,1,i}, S_{0,1,i'})\text{-}\mathsf{XY\text{-}DDH}}(\mathscr{B}_{i'}) + \\ \sum_{j'=1}^{n} q_{\mathbf{RKFn}} \cdot \mathsf{Adv}^{(k,n,\mathscr{S} \prod_{j=1}^{j'-1} X_j, X_{j'})\text{-}\mathsf{XY\text{-}DDH}}(\mathscr{C}_{j'}) \ ,\end{aligned}$$

where the running time of every adversary is approximately the same as the running time of $\mathscr{A}$. Theorem 7.4.4 immediately follows. ☐

```
proc Initialize // $\mathbf{G}_0$
$(\mathsf{mm.sp}, \mathsf{mm.pp}) \xleftarrow{\$} \mathsf{MM.Setup}(1^\kappa, \mathscr{U})$
$K \xleftarrow{\$} \{0,1\}^k$
For $i \in \{1, \dots, k\}$ and $b \in \{0,1\}$
    $a_{i,b} \xleftarrow{\$} \mathbb{Z}_p$
    $\hat{a}_{i,b} \leftarrow [a_{i,b}]_{S_{0,0,i}\mathsf{BitCommit}_{i,b}}$
For $j \in \{1, \dots, n\}$ and $b \in \{0,1\}$
    $c_{j,b} \xleftarrow{\$} \mathbb{Z}_p$
    $\hat{c}_{j,b} \leftarrow [c_{i,b}]_{X_i}$
For $i_1, i_2 \in \{1, \dots, k\}$ and $b_1, b_2 \in \{0,1\}$
    $\hat{z}_{i_1,i_2,b_1,b_2} \leftarrow [1]_{\mathsf{BitFill}_{i_1,i_2,b_1,b_2}}$
$\mathsf{pp} \leftarrow (\mathsf{mm.pp}, (\hat{a}_{i,b}), (\hat{c}_{j,b}), (\hat{z}_{i_1,i_2,b_1,b_2}))$
Return $\mathsf{pp}$

proc RKFn($\phi_s, x$) // $\mathbf{G}_0$
$\hat{y}' \leftarrow \prod_{i=1}^{k} \hat{a}_{i,K_i \oplus s_i} \prod_{j=1}^{n} \hat{c}_{j,x_j}$
$\hat{y} \leftarrow \hat{y}' \prod_{i_1=1}^{k} \prod_{i_2=i_1+1}^{k} \hat{z}_{i_1,i_2,K_{i_1}\oplus s_{i_1},K_{i_2}\oplus s_{i_2}}$
Return $\mathsf{MM.Extract}(\hat{y})$

proc Finalize($b$) // All games
Return $b$
```

```
proc Initialize // $\mathbf{G}_1$
$(\mathsf{mm.sp}, \mathsf{mm.pp}) \xleftarrow{\$} \mathsf{MM.Setup}(1^\kappa, \mathscr{U})$
$K \xleftarrow{\$} \{0,1\}^k$
For $i \in \{1, \dots, k\}$ and $b \in \{0,1\}$
    $a_{i,b} \xleftarrow{\$} \mathbb{Z}_p$
    $\hat{a}_{i,b} \leftarrow [a_{i,b}]_{S_{0,0,i}\mathsf{BitCommit}_{i,b}}$
    $\boxed{\gamma_{i,K_i \oplus b} \leftarrow a_{i,b}}$
    $\boxed{\hat{\gamma}_{i,K_i \oplus b} \leftarrow [\gamma_{i,K_i \oplus b}]_{S_{0,1,i}\mathsf{BitCommit}_{i,K_i \oplus b}}}$
For $j \in \{1, \dots, n\}$ and $b \in \{0,1\}$
    $c_{j,b} \xleftarrow{\$} \mathbb{Z}_p$
    $\hat{c}_{j,b} \leftarrow [c_{i,b}]_{X_i}$
For $i_1, i_2 \in \{1, \dots, k\}$ and $b_1, b_2 \in \{0,1\}$
    $\hat{z}_{i_1,i_2,b_1,b_2} \leftarrow [1]_{\mathsf{BitFill}_{i_1,i_2,b_1,b_2}}$
$\mathsf{pp} \leftarrow (\mathsf{mm.pp}, (\hat{a}_{i,b}), (\hat{c}_{j,b}), (\hat{z}_{i_1,i_2,b_1,b_2}))$
Return $\mathsf{pp}$

proc RKFn($\phi_s, x$) // $\mathbf{G}_1$
$\boxed{\hat{y}' \leftarrow \prod_{i=1}^{k} \hat{\gamma}_{i,s_i} \prod_{j=1}^{n} \hat{c}_{j,x_j}}$
$\boxed{\hat{y} \leftarrow \hat{y}' \prod_{i_1=1}^{k} \prod_{i_2=i_1+1}^{k} \hat{z}_{i_1,i_2,s_{i_1},s_{i_2}}}$
Return $\mathsf{MM.Extract}(\hat{y})$
```

```
proc Initialize // $\mathbf{G}_2$
$(\mathsf{mm.sp}, \mathsf{mm.pp}) \xleftarrow{\$} \mathsf{MM.Setup}(1^\kappa, \mathscr{U})$
For $i \in \{1, \dots, k\}$ and $b \in \{0,1\}$
    $a_{i,b} \xleftarrow{\$} \mathbb{Z}_p$
    $\hat{a}_{i,b} \leftarrow [a_{i,b}]_{S_{0,0,i}\mathsf{BitCommit}_{i,b}}$
    $\boxed{\gamma_{i,b} \xleftarrow{\$} \mathbb{Z}_p}$
    $\hat{\gamma}_{i,b} \leftarrow [\gamma_{i,b}]_{S_{0,1,i}\mathsf{BitCommit}_{i,b}}$
For $j \in \{1, \dots, n\}$ and $b \in \{0,1\}$
    $c_{j,b} \xleftarrow{\$} \mathbb{Z}_p$
    $\hat{c}_{j,b} \leftarrow [c_{i,b}]_{X_i}$
For $i_1, i_2 \in \{1, \dots, k\}$ and $b_1, b_2 \in \{0,1\}$
    $\hat{z}_{i_1,i_2,b_1,b_2} \leftarrow [1]_{\mathsf{BitFill}_{i_1,i_2,b_1,b_2}}$
$\mathsf{pp} \leftarrow (\mathsf{mm.pp}, (\hat{a}_{i,b}), (\hat{c}_{j,b}), (\hat{z}_{i_1,i_2,b_1,b_2}))$
Return $\mathsf{pp}$

proc RKFn($\phi_s, x$) // $\mathbf{G}_2$
$\hat{y}' \leftarrow \prod_{i=1}^{k} \hat{\gamma}_{i,s_i} \prod_{j=1}^{n} \hat{c}_{j,x_j}$
$\hat{y} \leftarrow \hat{y}' \prod_{i_1=1}^{k} \prod_{i_2=i_1+1}^{k} \hat{z}_{i_1,i_2,s_{i_1},s_{i_2}}$
Return $\mathsf{MM.Extract}(\hat{y})$
```

```
proc Initialize // $\mathbf{G}_3$
$(\mathsf{mm.sp}, \mathsf{mm.pp}) \xleftarrow{\$} \mathsf{MM.Setup}(1^\kappa, \mathscr{U})$
For $i \in \{1, \dots, k\}$ and $b \in \{0,1\}$
    $a_{i,b} \xleftarrow{\$} \mathbb{Z}_p$
    $\hat{a}_{i,b} \leftarrow [a_{i,b}]_{S_{0,0,i}\mathsf{BitCommit}_{i,b}}$
    $\gamma_{i,b} \xleftarrow{\$} \mathbb{Z}_p$
    $\boxed{\text{If } i = 1}$
        $\boxed{\hat{\gamma}_{i,b} \leftarrow [\gamma_{i,b}]_{S_{0,1,i}\mathscr{S}'}}$
    $\boxed{\text{Else}}$
        $\boxed{\hat{\gamma}_{i,b} \leftarrow [\gamma_{i,b}]_{S_{0,1,i}}}$
For $j \in \{1, \dots, n\}$ and $b \in \{0,1\}$
    $c_{j,b} \xleftarrow{\$} \mathbb{Z}_p$
    $\hat{c}_{j,b} \leftarrow [c_{i,b}]_{X_i}$
For $i_1, i_2 \in \{1, \dots, k\}$ and $b_1, b_2 \in \{0,1\}$
    $\hat{z}_{i_1,i_2,b_1,b_2} \leftarrow [1]_{\mathsf{BitFill}_{i_1,i_2,b_1,b_2}}$
$\mathsf{pp} \leftarrow (\mathsf{mm.pp}, (\hat{a}_{i,b}), (\hat{c}_{j,b}), (\hat{z}_{i_1,i_2,b_1,b_2}))$
Return $\mathsf{pp}$

proc RKFn($\phi_s, x$) // $\mathbf{G}_3$
$\boxed{\hat{y} \leftarrow \prod_{i=1}^{k} \hat{\gamma}_{i,s_i} \prod_{j=1}^{n} \hat{c}_{j,x_j}}$
Return $\mathsf{MM.Extract}(\hat{y})$
```

Figure 7.2: Games $\mathbf{G}_0, \mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_3$ for the proof of Theorem 7.4.4

```
proc Initialize  // $\mathbf{G}_{4,i'}$
$(\mathsf{mm.sp}, \mathsf{mm.pp}) \xleftarrow{\$} \mathsf{MM.Setup}(1^\kappa, \mathscr{U})$
$\boxed{T \leftarrow \text{empty table}}$
For $i \in \{1, \ldots, k\}$ and $b \in \{0, 1\}$
    $a_{i,b} \xleftarrow{\$} \mathbb{Z}_p$
    $\hat{a}_{i,b} \leftarrow [a_{i,b}]_{S_{0,0,i}\mathsf{BitCommit}_{i,b}}$
    $\gamma_{i,b} \xleftarrow{\$} \mathbb{Z}_p$
    If $i = 1$
        $\hat{\gamma}_{i,b} \leftarrow [\gamma_{i,b}]_{S_{0,1,i}\mathscr{S}'}$
    Else
        $\hat{\gamma}_{i,b} \leftarrow [\gamma_{i,b}]_{S_{0,1,i}}$
For $j \in \{1, \ldots, n\}$ and $b \in \{0, 1\}$
    $c_{j,b} \xleftarrow{\$} \mathbb{Z}_p$
    $\hat{c}_{j,b} \leftarrow [c_{i,b}]_{X_i}$
For $i_1, i_2 \in \{1, \ldots, k\}$ and $b_1, b_2 \in \{0, 1\}$
    $\hat{z}_{i_1,i_2,b_1,b_2} \leftarrow [1]_{\mathsf{BitFill}_{i_1,i_2,b_1,b_2}}$
$\mathsf{pp} \leftarrow (\mathsf{mm.pp}, (\hat{a}_{i,b}), (\hat{c}_{j,b}), (\hat{z}_{i_1,i_2,b_1,b_2}))$
Return $\mathsf{pp}$

proc RKFn($\phi_s, x$)  // $\mathbf{G}_{4,i'}$
    If $T[(s_1, \ldots, s_{i'-1})] \neq \perp$
        $r \xleftarrow{\$} \mathbb{Z}_p$
        $T[(s_1, \ldots, s_{i'-1})] \leftarrow [r]_{\mathscr{S}' \prod_{i=1}^{i'-1} S_{0,1,i}}$
    $\hat{y} \leftarrow T[(s_1, \ldots, s_{i'-1})] \cdot \prod_{i=i'}^{k} \hat{\gamma}_{i,s_i} \prod_{j=1}^{n} \hat{c}_{j,x_j}$
Return $\mathsf{MM.Extract}(\hat{y})$
```

```
proc Initialize  // $\mathbf{G}_{5,j'}$
$(\mathsf{mm.sp}, \mathsf{mm.pp}) \xleftarrow{\$} \mathsf{MM.Setup}(1^\kappa, \mathscr{U})$
$\boxed{T \leftarrow \text{empty table}}$
For $i \in \{1, \ldots, k\}$ and $b \in \{0, 1\}$
    $a_{i,b} \xleftarrow{\$} \mathbb{Z}_p$
    $\hat{a}_{i,b} \leftarrow [a_{i,b}]_{S_{0,0,i}\mathsf{BitCommit}_{i,b}}$
    $\gamma_{i,b} \xleftarrow{\$} \mathbb{Z}_p$
    If $i = 1$
        $\hat{\gamma}_{i,b} \leftarrow [\gamma_{i,b}]_{S_{0,1,i}\mathscr{S}'}$
    Else
        $\hat{\gamma}_{i,b} \leftarrow [\gamma_{i,b}]_{S_{0,1,i}}$
For $j \in \{1, \ldots, n\}$ and $b \in \{0, 1\}$
    $c_{j,b} \xleftarrow{\$} \mathbb{Z}_p$
    $\hat{c}_{j,b} \leftarrow [c_{i,b}]_{X_i}$
For $i_1, i_2 \in \{1, \ldots, k\}$ and $b_1, b_2 \in \{0, 1\}$
    $\hat{z}_{i_1,i_2,b_1,b_2} \leftarrow [1]_{\mathsf{BitFill}_{i_1,i_2,b_1,b_2}}$
$\mathsf{pp} \leftarrow (\mathsf{mm.pp}, (\hat{a}_{i,b}), (\hat{c}_{j,b}), (\hat{z}_{i_1,i_2,b_1,b_2}))$
Return $\mathsf{pp}$

proc RKFn($\phi_s, x$)  // $\mathbf{G}_{5,j'}$
    If $T[(s, x_1, \ldots, x_{j'-1})] \neq \perp$
        $r \xleftarrow{\$} \mathbb{Z}_p$
        $T[(s, x_1, \ldots, x_{j'-1})] \leftarrow [r]_{\mathscr{S} \prod_{j=1}^{j'-1} X_j}$
    $\hat{y} \leftarrow T[(s, x_1, \ldots, x_{j'-1})] \cdot \prod_{j=j'}^{n} \hat{c}_{j,x_j}$
Return $\mathsf{MM.Extract}(\hat{y})$
```

Figure 7.3: Games $\mathbf{G}_{4,i'}, \mathbf{G}_{5,j'}$ for the proof of Theorem 7.4.4

7.5 Security of the XY-DDH Assumption in the Generic Multilinear Map Model

Theorem 7.5.1 (Impossibility of constructing encodings of zero)**.** *Let k and n be two positive integers. Let X and Y be two non-empty and disjoint indices from the index set $\mathscr{U}$ of our construction in Section 7.2.2. In the generic multilinear map model with oracle $\mathscr{M}$, for any adversary $\mathscr{D}$ making at most $q_{\mathscr{M}}$ queries to the oracle $\mathscr{M}$, we have:*

$$\Pr\Big[(k,n,X,Y)\text{-}\mathsf{XY\text{-}DDHReal}^{\mathscr{D}} \Rightarrow \textit{an encoding of } 0\Big] \le q_{\mathscr{M}}\frac{|\mathscr{U}|}{p}$$

and

$$\Pr\Big[(k,n,X,Y)\text{-}\mathsf{XY\text{-}DDHRand}^{\mathscr{D}} \Rightarrow \textit{an encoding of } 0\Big] \le q_{\mathscr{M}}\frac{|\mathscr{U}|}{p} \ .$$

Proof of Theorem 7.5.1. We consider the two cases separately.

(k,n,X,Y)-$\mathsf{XY\text{-}DDHRand}$**.** Except for $\hat{z}_{i_1,i_2,b_1,b_2}$, all the handles the adversary sees corresponds to fresh variables: $\hat{d}_0, \hat{d}_1, \hat{e}_0, \hat{e}_1$, in addition to the ones it can generate using the list $\mathcal{L}$. We can therefore conclude exactly as in the case $\mathsf{RKPRFRand}_F$ of the proof of Theorem 7.3.2. The only difference is that the maximum degree of the polynomials the adversary can create is at most $|\mathscr{U}|$ (instead of $k+n$), as the adversary does not have access to elements of index $\emptyset$ (i.e., scalars).

(k,n,X,Y)-$\mathsf{XY\text{-}DDHReal}$**.** Similarly to the case $\mathsf{RKPRFReal}_F$ of the proof of Theorem 7.3.2, we just need to prove that any non-zero polynomial P remains non-zero when it remains non-zero when $\hat{z}_{i_1,i_2,b_1,b_2}$ is replaced by 1 and $\hat{e}_0$ and $\hat{e}_1$ are replaced respectively by ed_0 and ed_1 (where e is a fresh formal variable).

Such a non-zero polynomial P can be written $Q_0\hat{d}_0 + Q_1\hat{d}_1 + Q_2\hat{e}_0 + Q_3\hat{e}_1$, where Q_0, Q_1, Q_2, Q_3 are four polynomials over all the formal variables except $\hat{d}_0, \hat{d}_1, \hat{e}_0, \hat{e}_1$, because indices prevent the multiplication of two of the variables $\hat{d}_0, \hat{d}_1, \hat{e}_0, \hat{e}_1$. Furthemore at least one of the polynomials Q_0, Q_1, Q_2, Q_3 is non-zero, and it remains non-zero when $\hat{z}_{i_1,i_2,b_1,b_2}$ is replaced by 1 (with a proof similar to the one of the case $\mathsf{RKPRFRand}_F$ of the proof of Theorem 7.3.2, using Lemma 7.3.3). Thus, even when replacing $\hat{z}_{i_1,i_2,b_1,b_2}$ by 1 and $(\hat{e}_0, \hat{e}_1)$ by (ed_0, ed_1), P remains non-zero.

This concludes the proof. □

Similarly to Corollary 7.3.5, we have the following corollary.

Corollary 7.5.2 (Security in the generic multilinear map model)**.** *Let k and n be two positive integers. Let X and Y be two non-empty and disjoint indices in the index set $\mathscr{U}$ of our construction in Section 7.2.2. Then, the (k,n,X,Y)-$\mathsf{XY\text{-}DDH}$ assumptions holds in the generic multilinear map model.*

Proof of Corollary 7.5.2. We just consider an intermediate game where we simulate everything as before except the zero-testing oracle which always outputs "non-zero" unless its input is zero, as a polynomial. Then, we have:

$$\mathsf{Adv}^{(k,n,X,Y)\text{-}\mathsf{XY\text{-}DDH}}(\mathscr{D}) \le 2q_{\mathscr{M}}\frac{|\mathscr{U}|}{p} \ ,$$

and Corollary 7.5.2 easily follows. □

7.6 Security of the Sel-Prod Assumption in the Generic Multilinear Map Model

The proof of the Sel-Prod assumption is subtle and first requires the introduction of the notion of profiles and some lemmata.

7.6.1 Proof Ingredient: Index Sets and Profiles

We adapt the proof in [Zim15, Section 3.4] to our case. In the whole section, monomials and polynomials are over the formal variables $\hat{a}_{i,b}, \hat{\gamma}_{i,b}, \hat{c}_{j,b}, \hat{z}_{i_1,i_2,b_1,b_2}$, for $i, i_1, i_2 \in \{1,\dots,k\}$, $j \in \{1,\dots,n\}$, $b, b_1, b_2 \in \{0,1\}$. Furthermore, we use the index set $\mathscr{U}$ defined in Section 7.2.2.

We also write $\hat{a}^0_{i,b} = \hat{a}_{i,b}$ and $\hat{a}^1_{i,b} = \hat{\gamma}_{i,b}$, for $i \in \{1,\dots,k\}$ and $b \in \{0,1\}$.

Definition 7.6.1 (Profile of a monomial). *Let t be a monomial. For any $i \in \{1,\dots,k\}$ and any bit $b \in \{0,1\}$, t* incorporates *b as its i-th bit if and only if t contains one of the following formal variables: $\hat{a}^{b'}_{i,b}, \hat{z}_{i,i',b,b'}, \hat{z}_{i',i,b',b}$ for $i' \in \{1,\dots,k\}$ and $b' \in \{0,1\}$. The* profile *of t is:*

- *$\perp$ if t incorporates both 0 and 1 as its i-th bit;*
- *otherwise, the tuple $\mathsf{prof}(t) = ((\mathsf{prof}(t))_1, \dots, (\mathsf{prof}(t))_k) \in \{0,1,*\}^k$ such that $\mathsf{prof}(t)_i = b_i = b$ if t incorporates b as its i-th bit, and $\mathsf{prof}(t)_i = b_i = *$ if t does not incorporate 0 nor 1 as its i-th bit.*

Definition 7.6.2 (Partial profile, conflict, and merge). *Let t_1 and t_2 be two monomials, such that $\mathsf{prof}(t_1) \neq \perp$ and $\mathsf{prof}(t_2) \neq \perp$. We say that:*

- *the profile $\mathsf{prof}(t_1)$ is* partial *if $\mathsf{prof}(t_1)_i = *$ for some $i \in \{1,\dots,k\}$;*
- *the profiles $\mathsf{prof}(t_1)$ and $\mathsf{prof}(t_2)$* conflict *if there exists $i \in \{1,\dots,k\}$ such that $\mathsf{prof}(t_1)_i$, $\mathsf{prof}(t_2)_i \in \{0,1\}$ and $\mathsf{prof}(t_1)_i \neq \mathsf{prof}(t_2)_i$.*

We also define the merge *of two* non-conflicting *profiles $\mathsf{prof}(t_1)$ and $\mathsf{prof}(t_2)$ as the tuple $(b_1,\dots,b_n)$ such that:*

$$b_i = \begin{cases} \mathsf{prof}(t_1)_i & \text{if } \mathsf{prof}(t_1)_i \in \{0,1\} \\ \mathsf{prof}(t_2)_i & \text{if } \mathsf{prof}(t_2)_i \in \{0,1\} \\ * & \text{otherwise.} \end{cases}$$

Definition 7.6.3 (Profile of a polynomial). *Let P be a polynomial. The* profile *of a polynomial P is:*

- *$\perp$ if some monomial in the formal expansion (without cancellation) of P has a profile $\perp$;*
- *the set $\{\mathsf{prof}(t) \mid t$ is in the formal expansion of $P\}$ otherwise.*

Let us now characterize the polynomials that appears in Sel-Prod.

Lemma 7.6.4 (Characterization of polynomials in Sel-Prod). *Let $\mathscr{A}$ be an adversary in the generic multinear map model, making at most q queries to the multilinear map oracle $\mathscr{M}$, and returning a handle h corresponding to some polynomial P. Then, P satisfies one of these two conditions:*

1. *$\mathsf{prof}(P) = \perp$ or $\mathsf{prof}(P)$ is a singleton:*
   - *if the index of P does not contain $\mathscr{S}_0$, there exists a polynomial Q in the formal variables $\hat{c}_{i,b}$ for $i \in \{1,\dots,k\}$ and $b \in \{0,1\}$, together with some string $\beta \in \{0,1\}$, some tuple $K' \in \{0,1,*\}^k$ and some set $Z \subseteq \{(i_1,i_2) \mid 1 \le i_1 < i_2 \le k\} \times \{0,1\}^2$ such that:*
   $$P = Q \cdot \prod_{\substack{i=1\\K'_i \neq *}}^{k} \hat{a}_{i,K'_i}^{\beta_i} \cdot \prod_{(i_1,i_2,b_1,b_2)\in Z} \hat{z}_{i_1,i_2,b_1,b_2}\ ;$$
   - *if the index of P does contain $\mathscr{S}_0$, there exist two polynomials Q_0 and Q_1 in the formal variables $\hat{c}_{i,b}$ for $i \in \{1,\dots,k\}$ and $b \in \{0,1\}$, together with some string $\beta \in \{0,1\}$, some tuple $K' \in \{0,1,*\}^k$ and some set $Z \subseteq \{(i_1,i_2) \mid 1 \le i_1 < i_2 \le k\} \times \{0,1\}^2$ such that:*
   $$P = \left(Q_0 \cdot \prod_{\substack{i=1\\K'_i \neq *}}^{k} \hat{a}_{i,K'_i}^{0} + Q_1 \cdot \prod_{\substack{i=1\\K'_i \neq *}}^{k} \hat{a}_{i,K'_i}^{1} \right) \cdot \prod_{(i_1,i_2,b_1,b_2)\in Z} \hat{z}_{i_1,i_2,b_1,b_2}\ ;$$

   in both case, in addition the index set of P is not $\mathscr{U}$;
2. *$\mathsf{prof}(P)$ is a set containing at least 2 and at most q non-partial profiles: $\mathsf{prof}(P) = \{K'_1,\dots,K'_{q'}\}$, with $q' \le q$; in which case, there exists polynomials $Q_{1,0}, Q_{1,1},\dots,Q_{q',0}$, $Q_{q',1}$ in the formal variables $\hat{c}_{i,b}$ for $i \in \{1,\dots,k\}$ and $b \in \{0,1\}$, such that:*
   $$P = \sum_{\ell=1}^{q'} Q_{\ell,0} \cdot \prod_{i=1}^{k} \hat{a}_{i,K'_{\ell,i}}^{0} \prod_{i_1=1}^{k} \prod_{i_2=i_1+1}^{k} \hat{z}_{i_1,i_2,K'_{\ell,i_1},K'_{\ell,i_2}} + Q_{\ell,1} \cdot \prod_{i=1}^{k} \hat{a}_{i,K'_{\ell,i}}^{1} \prod_{i_1=1}^{k} \prod_{i_2=i_1+1}^{k} \hat{z}_{i_1,i_2,K'_{\ell,i_1},K'_{\ell,i_2}}\ .$$

   In addition, when the index set of P does not contain $\mathscr{S}' = \prod_{i=1}^{k} \mathscr{S}_i$, then necessarily $q' = 1$.

Furthermore $\mathsf{prof}(P)$ and the above decompositions of P can be efficiently computed (i.e., in a time polynomial in κ and q).

The proof is by induction over the expression of the polynomials formed by the adversary. We first need to introduce some intermediate lemmata.

Lemma 7.6.5 (Impossibility to add partial terms)**.** *Let t_1 and t_2 be two monomials, such that $\mathsf{prof}(t_1) \neq \perp$ and $\mathsf{prof}(t_2) \neq \perp$. If $\mathsf{prof}(t_1)$ or $\mathsf{prof}(t_2)$ is partial and $\mathsf{prof}(t_1) \neq \mathsf{prof}(t_2)$, then t_1 and t_2 do not have the same index set.*

Proof of Lemma 7.6.5. We can assume without loss of generality that, for a fixed bit $b \in \{0,1\}$ and some $i \in \{1,\dots,k\}$, $\mathsf{prof}(t_1)_i = b$ and $\mathsf{prof}(t_2)_i \neq b$. Hence, as $\mathsf{prof}(t_2)_i \neq b$ and as $\mathsf{prof}(t_2) \neq \perp$, t_2 cannot contain a factor of $\hat{a}_{i,b}$, $\hat{z}_{i,i',b,b'}$ or $\hat{z}_{i',i,b',b}$ for any $i' \in \{1,\dots,k\}$ and $b \in \{0,1\}$. We now prove Lemma 7.6.5 by contradiction. The possible cases are the following:

- Suppose t_1 contains a factor of $\hat{a}_{i,b}$. Then its index set was formed using $\mathsf{BitCommit}_{i,b} = S_{i,b,i}$. Since t_2 does not contain a factor of $\hat{a}_{i,b}$, its index set was not formed using $\mathsf{BitCommit}_{i,b} = S_{i,b,i}$. Therefore, if t_1 and t_2 have the same index set, by definition of a straddling set, this index set has to contain the whole set $\mathscr{S}_i$. However, for every $i' \in \{1, \ldots, k\}$ with $i \neq i'$, the only encodings that contain $S_{i,b,i'}$, for any $b \in \{0,1\}$, are $\hat{z}_{i,i',b,b'}$ and $\hat{z}_{i',i,b',b}$ for some $b' \in \{0,1\}$. This implies that t_1 contains a factor of $\hat{z}_{i,i',b,b'}$ or $\hat{z}_{i',i,b',b}$ for some $b' \in \{0,1\}$, for every $i' \in \{1, \ldots, k\}$ and similarly, that t_2 contains a factor of $\hat{z}_{i,i',1-b,b'}$ or $\hat{z}_{i',i,b',1-b}$ for some $b' \in \{0,1\}$ and for every $i' \in \{1, \ldots, k\}$. This contradicts the fact that at least one of the two profiles is partial.
- Suppose t_1 does not contain any factor of $\hat{a}_{i,b}$ but does contain a factor of $\hat{z}_{i,i',b,b'}$ or $\hat{z}_{i',i,b',b}$ for some $i' \in \{1, \ldots, k\}$ and some $b \in \{0,1\}$. Then, its index set was formed using $\mathsf{BitFill}_{i,i',b,b'} = S_{i,b,i'} S_{i',b',i}$ or $\mathsf{BitFill}_{i',i,b',b} = S_{i',b',i} S_{i,b,i'}$, so in particular using $S_{i,b,i'}$. Then, one can apply the same argument than in the previous case to exclude this case.

Lemma 7.6.5 immediately follows. □

Similarly to the above lemma, we can prove the following three lemmata. As their proofs are very similar, we only give brief intuitions.

Lemma 7.6.6 (Impossibility to add $\perp$-profile and non-$\perp$-profile)**.** *Let t_1 and t_2 be two monomials. If $\mathsf{prof}(t_1) = \perp$ and $\mathsf{prof}(t_2) \neq \perp$, then t_1 and t_2 do not have the same index set.*

Proof of Lemma 7.6.6. The technique is similar to the proof of Lemma 7.6.5, thus we just provide the main ideas. As $\mathsf{prof}(t_1) = \perp$, there exists $i \in \{1, \ldots, k\}$ such that t_1 incorporates both 0 and 1 as its i-th bit, and $\mathsf{prof}(t_2)_i \in \{0, 1, *\}$. Assume $\mathsf{prof}(t_2)_i = *$. Then t_2 does not contain any $\hat{a}_{i,b}$ nor $\hat{z}_{i,i',b,b'}$ nor $\hat{z}_{i',i,b',b}$, for any b, b', i', so its index set cannot be form with any of the $S_{i,b,i'}$, for any b, i', while t_1 is formed using at least one such index. Then t_1 and t_2 do not have the same index set. Assuming now that $\mathsf{prof}(t_2)_i = b$, one can reiterate the same technique with some index $S_{i',1-b,i}$ that cannot be used to form its index set. Lemma 7.6.6 follows. □

Lemma 7.6.7. *Let Q_1, Q_2 be two polynomials in the formal variables $\hat{c}_{i,b}$, for $i \in \{1, \ldots, k\}$ and $b \in \{0,1\}$, β_1, β_2 two bit strings in $\{0,1\}^k$, K'_1, K'_2 two tuples in $\{0, 1, *\}^k$, and Z_1, Z_2 two subsets of $\{(i_1, i_2) \mid 1 \leq i_1 < i_2 \leq k\} \times \{0,1\}^2$. Let P_1 and P_2 be the two polynomials defined by*

$$P_\ell = Q_\ell \prod_{\substack{i=1 \\ K'_{\ell,i} \neq *}}^{k} \hat{a}_{i,K'_{\ell,i}}^{\beta_\ell} \prod_{(i_1,i_2,b_1,b_2) \in Z_\ell} \hat{z}_{i_1,i_2,b_1,b_2} \,,$$

for $\ell \in \{1,2\}$. If the index of P_1 and P_2 is the same but does not contains $\mathscr{S}' = \prod_{i=1}^{k} \mathscr{S}_i$ nor $\mathscr{S}_0$, then $K_1 = K_2$ and $Z_1 = Z_2$. The latter condition (on the index not containing $\mathscr{S}'$) is in particular satisfied when the profile of P_1 (and so of P_2 too) is $\perp$ or contains a partial profile.

Proof of Lemma 7.6.7. Once again, the proof is similar to that of Lemma 7.6.5. The fact that the index of P_1 (and P_2) does not contain $\mathscr{S}'$ nor $\mathscr{S}_0$ guarantees that there exists $i \in \{1, \ldots, k\}$ such that $\mathscr{S}_i$ is not contained in this index, and $\mathscr{S}_0$ is not contained either. For any such i, as the indices of P_1 and P_2 are the same, they have to contain exactly the same $S_{i,b,i'}$'s from $\mathscr{S}_i$ (since they do not contain $\mathscr{S}_i$). □

Lemma 7.6.8. *Let $Q_{1,0}, Q_{1,1}, Q_{2,0}, Q_{2,1}$ be four polynomials in the formal variables $\hat{c}_{i,b}$, for $i \in \{1, \dots, k\}$ and $b \in \{0,1\}$, K'_1, K'_2 two tuples in $\{0,1,*\}^k$, and Z_1, Z_2 two subsets of $\{(i_1, i_2) \mid 1 \le i_1 < i_2 \le k\} \times \{0,1\}^2$. Let P_1 and P_2 be the two polynomials defined by*

$$P_\ell = \left(Q_{\ell,0} \prod_{\substack{i=1 \\ K'_{\ell,i} \neq *}}^{k} \hat{a}^0_{i,K'_{\ell,i}} + Q_{\ell,1} \prod_{\substack{i=1 \\ K'_{\ell,i} \neq *}}^{k} \hat{a}^1_{i,K'_{\ell,i}} \right) \cdot \prod_{(i_1,i_2,b_1,b_2) \in Z_\ell} \hat{z}_{i_1,i_2,b_1,b_2} ,$$

for $\ell \in \{1,2\}$. If the index of P_1 and P_2 is the same but does not contains $\mathscr{S}' = \prod_{i=1}^k \mathscr{S}_i$ nor $\mathscr{S}_0$, then $K_1 = K_2$ and $Z_1 = Z_2$. The latter condition (on the index not containing $\mathscr{S}'$) is in particular satisfied when the profile of P_1 (and so of P_2 too) is $\perp$ or contains a partial profile.

Proof of Lemma 7.6.8. The proof is similar to that of Lemma 7.6.7. □

We now have every tool to prove Lemma 7.6.4.

Proof of Lemma 7.6.4. In this proof, the integers i, i_1, i_2 are in $\{1, \dots, k\}$ and b is a bit. We set $\mathscr{S}' = \prod_{i=1}^k \mathscr{S}_i$. Furthermore:

- Q and Q_ℓ always denote polynomials in the formal variables $\hat{c}_{i,b}$,
- β and β_ℓ always denot bit strings in $\{0,1\}^k$,
- K' and K'_ℓ always denote tuples in $\{0,1,*\}^k$, and
- Z and Z_ℓ always denote subsets of $\{(i_1, i_2) \mid 1 \le i_1 < i_2 \le k\} \times \{0,1\}^2$.

We prove the above lemma by induction on the sequence of formal polynomials P formed by the adversary $\mathscr{A}$ via oracle queries. Let P denote a newly formed polynomial. We consider the following cases:

- Suppose $P = t$ with t being a monomial: clearly, either $\mathsf{prof}(P) = \perp$, or it is a singleton containing either a partial profile or a non-partial profile. Moreover, as a monomial, it is straightforward that it has the expected form detailed in the statement of Lemma 7.6.4 (with $q' = 1$ in the third case).
- Suppose $P = P_1 + P_2$ with P_1, P_2 some polynomials already formed. We have three possible cases:
  1. $\mathsf{prof}(P_1) = \perp$ or $\mathsf{prof}(P_2) = \perp$: immediately, we have also $\mathsf{prof}(P) = \perp$, by definition of the profile of a polynomial. Furthermore, we necessarily have $\mathsf{prof}(P_1) = \mathsf{prof}(P_2) = \perp$, otherwise, the formal expansion of $P_1 + P_2$ would contain one monomial of profile $\perp$ and another one of profile different than $\perp$, which is impossible according to Lemma 7.6.6. We consider two sub-cases:
     - the index of P_1 (and also of P_2) does not contains $\mathscr{S}_0$. By induction hypothesis, there exists two polynomials Q_1 and Q_2, two strings β_1 and β_2, two tuples K'_1 and K'_2, and two sets Z_1 and Z_2, such that:

$$P_\ell = Q_\ell \prod_{\substack{i=1 \\ K'_{\ell,i} \neq *}}^{k} \hat{a}^{\beta_{\ell,i}}_{i,K'_{\ell,i}} \prod_{(i_1,i_2,b_1,b_2) \in Z_\ell} \hat{z}_{i_1,i_2,b_1,b_2} ,$$

for $\ell \in \{1, 2\}$. According to Lemma 7.6.7, this implies that $K'_1 = K'_2$ and $Z_1 = Z_2$ and that $P = P_1 + P_2$ has the expected form.

– the index of P_1 (and also of P_2) contains $\mathscr{S}_0$. We conclude again using Lemma 7.6.8 and the induction hypothesis.

2. $\mathsf{prof}(P_1)$ or $\mathsf{prof}(P_2)$ is a singleton containing one partial profile. We have necessarily $\mathsf{prof}(P_2) = \mathsf{prof}(P_1)$, otherwise P_1 and P_2 cannot be added via Lemma 7.6.5, since there must exist two monomials t_1 and t_2 in the formal expansion of P_1 and P_2 respectively such that $\mathsf{prof}(t_1) \neq \mathsf{prof}(t_2)$ and with $\mathsf{prof}(t_1)$ or $\mathsf{prof}(t2)$ being partial. We can conclude this case similarly to the previous case.
3. both $\mathsf{prof}(P_1)$ and $\mathsf{prof}(P_2)$ are sets containing at least 2 non-partial profiles. Then $\mathsf{prof}(P) = \mathsf{prof}(P_1) \cup \mathsf{prof}(P_2)$ and does not contain any partial profile. Furthermore, if the index of P (which is the same as the one of P_1 and P_2) contains $\mathscr{S}'$, it is straightforward that P has the expected form, with $q' = |\mathsf{prof}(P)| \leq |\mathsf{prof}(P_1)| + |\mathsf{prof}(P_2)|$.

   It remains to show that if the index of P does not contain $\mathscr{S}'$, P has the expected form with $q' = 1$. By hypothesis induction, as the indexes of P_1 and P_2 do not contain $\mathscr{S}'$: there exists two polynomials Q_1 and Q_2, two tuples K'_1 and K'_2, and two sets Z_1 and Z_2, such that $\mathsf{prof}(P_1) = \{K'_1\}$ and $\mathsf{prof}(P_2) = \{K'_2\}$, such that:

$$P_\ell = Q_\ell \prod_{\substack{i=1 \\ K'_{\ell,i} \neq *}}^{k} \hat{a}_{i,K'_{\ell,i}} \prod_{(i_1,i_2,b_1,b_2) \in Z_\ell} \hat{z}_{i_1,i_2,b_1,b_2} \,,$$

   for $\ell \in \{1, 2\}$. According to Lemma 7.6.7, this implies that $K'_1 = K'_2$ and $Z_1 = Z_2$ and this clearly implies that $P = P_1 + P_2$ has the expected form.

- Suppose $P = P_1 P_2$ with P_1, P_2 some formal polynomials already formed. Once again, we have three possible cases:

  1. $\mathsf{prof}(P_1) = \perp$ or $\mathsf{prof}(P_2) = \perp$: immediately, we have also $\mathsf{prof}(P) = \perp$. We assume without losss of generality that $\mathsf{prof}(P_1) = \perp$.

     Let us show that necessarily $\mathsf{prof}(P_2)$ is either $\perp$ or a singleton containing a partial profile. Otherwise the index of P_2 would contain $\mathscr{S}'$, which implies that the index of P_1 cannot contain any element in $\mathscr{S}'$ and that $P_1 = Q$ for some polynomial Q. But in that case, the profile of P_1 would be $\{(*, \dots, *)\} \neq \perp$ which is impossible.

     We now consider two sub-cases:

     – the index of P does not contain $\mathscr{S}_0$, in this case, P_1 nor P_2 can contain $\mathscr{S}_0$ either. By induction hypothesis, there exist two strings β_1, β_2, two polynomials Q_1 and Q_2, two tuples K'_1 and K'_2, and two sets Z_1 and Z_2, such that:

$$P_\ell = Q_\ell \prod_{\substack{i=1 \\ K'_{\ell,i} \neq *}}^{k} \hat{a}_{i,K'_{\ell,i}}^{\beta_{\ell,i}} \prod_{(i_1,i_2,b_1,b_2) \in Z_\ell} \hat{z}_{i_1,i_2,b_1,b_2} \,,$$

     for $\ell \in \{1, 2\}$; thus $P_1 P_2$ has clearly the expected form.

- the index of P does contain $\mathscr{S}_0$. Clearly, at most one polynomial among P_1 and P_2 has an index containing $\mathscr{S}_0$. Using the induction hypothesis, it is easy to see that there exist two strings β_1, β_2, two polynomial Q_0 and Q_1, one tuple K', and one set Z, such that:

$$P = \left(Q_0 \cdot \prod_{\substack{i=1 \\ K'_i \neq *}}^{k} \hat{a}_{i,K'_i}^{\beta_{1,i}} + Q_1 \cdot \prod_{\substack{i=1 \\ K'_i \neq *}}^{k} \hat{a}_{i,K'_i}^{\beta_{1,i}} \right) \cdot \prod_{(i_1,i_2,b_1,b_2) \in Z} \hat{z}_{i_1,i_2,b_1,b_2} \ .$$

  To conclude, we just need to prove that $\beta_1, \beta_2 \in \{(0,\dots,0),(1,\dots,1)\}$. This directly comes from the property of straddling sets, the existence of the above polynomial P implies that $(S_{0,\beta_{0,1},1}, \dots, S_{0,\beta_{0,k},k})$ and $(S_{0,\beta_{1,1},1}, \dots, S_{0,\beta_{1,k},k})$ are two partitions of $\mathscr{S}_0$.

2. none of the index of P_1 and P_2 contains $\mathscr{S}'$. The profiles $\mathsf{prof}(P_1)$ and $\mathsf{prof}(P_2)$ are singletons, and the profile $\mathsf{prof}(P)$ is either $\perp$ or a singleton containing the merge of the two profiles contained in $\mathsf{prof}(P_1)$ and $\mathsf{prof}(P_2)$. We conclude similarly to the previous case.
3. P_1 or P_2 contains $\mathscr{S}'$. We assume without loss of generality that the index of P_1 contains $\mathscr{S}'$. This implies that the index of P_2 cannot contain any element of $\mathscr{S}'$ and so that there exists a polynomial Q such that $P_2 = Q$. We therefore get that $\mathsf{prof}(P) = \mathsf{prof}(P_1)$ is a set of non-partial profiles. Furthermore, using the induction hypothesis on P_1, it is straightforward to see that P has the expected form (with the same number of terms q' as P_1).

$\square$

7.6.2 Security of the Sel-Prod Assumption

Theorem 7.6.9 (Impossibility of constructing encodings of zero)**.** *Let k and n be two positive integers. In the generic multilinear map model with oracle $\mathscr{M}$, for any adversary $\mathscr{D}$ making at most $q_{\mathscr{M}}$ queries to the oracle $\mathscr{M}$, we have:*

$$\Pr\left[(k,n)\text{-}\mathsf{Sel\text{-}ProdReal}^{\mathscr{D}} \Rightarrow \textit{an encoding of 0}\right] \leq q_{\mathscr{M}} \left(\frac{q_{\mathscr{M}}}{2^k} + \frac{k+n}{p} \right)$$

and

$$\Pr\left[(k,n)\text{-}\mathsf{Sel\text{-}ProdRand}^{\mathscr{D}} \Rightarrow \textit{an encoding of 0}\right] \leq \frac{q_{\mathscr{M}}^2}{2^k} + \frac{2q_{\mathscr{M}}(k+n)}{p} \ .$$

Proof of Theorem 7.6.9. Once again, we consider the two cases separately.

(k,n)-**Sel-Prod.** The proof is similar to the case $\mathsf{RKPRFRand}_F$ of the proof of Theorem 7.3.2 and to the case (k,n,X,Y)-$\mathsf{XY\text{-}DDHRand}$ of the proof of Theorem 7.5.1.

(k,n)-**Sel-Prod.** Similarly to the case $\mathsf{RKPRFReal}_F$ of the proof of Theorem 7.3.2, we just need to prove that any non-zero polynomial P remains non-zero when it remains non-zero when $\hat{z}_{i_1,i_2,b_1,b_2}$ is replaced by 1 and $\hat{\gamma}_{i,b} = \hat{a}_{i,b}^1$ is replaced by $a_{i,K_i \oplus b}$ where K is a random bit string in $\{0,1\}$. We call this replacement: *the partial evaluation.*

According to Lemma 7.6.4, we have three possible cases (we use the notation of this lemma):

- P has the following form:

$$P = Q \cdot \prod_{\substack{i=1\\K'_i \neq *}}^{k} \hat{a}_{i,K'_i}^{\beta_i} \cdot \prod_{(i_1,i_2,b_1,b_2)\in Z} \hat{z}_{i_1,i_2,b_1,b_2} \enspace .$$

  In this case, the polynomial clearly remains non-zero after the partial evaluation.

- P has the following form:

$$P = \left(Q_0 \cdot \prod_{\substack{i=1\\K'_i \neq *}}^{k} \hat{a}_{i,K'_i}^{0} + Q_1 \cdot \prod_{\substack{i=1\\K'_i \neq *}}^{k} \hat{a}_{i,K'_i}^{1} \right) \cdot \prod_{(i_1,i_2,b_1,b_2)\in Z} \hat{z}_{i_1,i_2,b_1,b_2} \enspace .$$

  In this case, since P contains $\mathscr{S}_0$, we have that $K' \in \{0,1\}^k$ and thus if this polynomial P is zero adter the partial evaluation, then for any $i\{1,\ldots,k\}$, $a_{i,K'_i} = a_{i,K_i \oplus K'_i}$, i.e., $K' = K \oplus K'$, or in other words $K = 0$. This happens with probability $1/2^k$.

- P has the following form:

$$P = \sum_{\ell=1}^{q'} Q_{\ell,0} \cdot \prod_{i=1}^{k} \hat{a}_{i,K'_{\ell,i}}^{0} \prod_{i_1=1}^{k} \prod_{i_2=i_1+1}^{k} \hat{z}_{i_1,i_2,K'_{\ell,i_1},K'_{\ell,i_2}} + \\ Q_{\ell,1} \cdot \prod_{i=1}^{k} \hat{a}_{i,K'_{\ell,i}}^{1} \prod_{i_1=1}^{k} \prod_{i_2=i_1+1}^{k} \hat{z}_{i_1,i_2,K'_{\ell,i_1},K'_{\ell,i_2}} \enspace .$$

  If $Q_{\ell,0}$ is zero for all ℓ, then P is clearly still non-zero after the partial evaluation. Let us now suppose without loss of generality that $Q_{\ell,0} \neq 0$. Let us suppose that P becomes the zero polynomial after the partial evaluation. The first (for any order) monomial of the term

$$Q_{1,0} \cdot \prod_{i=1}^{k} \hat{a}_{i,K'_{1,i}}^{0} \prod_{i_1=1}^{k} \prod_{i_2=i_1+1}^{k} \hat{z}_{i_1,i_2,K'_{1,i_1},K'_{1,i_2}}$$

  needs to be canceled by one of the monomials of the terms for some ℓ:

$$Q_{\ell,1} \cdot \prod_{i=1}^{k} \hat{a}_{i,K'_{\ell,i}}^{1} \prod_{i_1=1}^{k} \prod_{i_2=i_1+1}^{k} \hat{z}_{i_1,i_2,K'_{\ell,i_1},K'_{\ell,i_2}} \enspace .$$

  This implies that for some $\ell \in \{1,\ldots,q'\}$, $K'_1 = K \oplus K'_\ell$. This happens with probability at most $q_{\mathscr{M}}$.

Therefore probability that a non-zero polynomial P generated by the adversary $\mathscr{D}$ remains non-zero when $\hat{z}_{i_1,i_2,b_1,b_2}$ is replaced by 1 and $\hat{\gamma}_{i,b}$ is replaced by $a_{i,K_i \oplus b}$ where K is a random bit string in $\{0,1\}$, is at most $q_{\mathscr{M}}^2/2^K$ (as the adversary $\mathscr{D}$ generates at most $\mathscr{M}$ polynomials). We conclude as usual using the Schwarz-Zippel lemma and an hybrid argument. ☐

Similarly to Corollary 7.3.5 and Corollary 7.6.10, we have the following corollary.

Corollary 7.6.10 (Security in the generic multilinear map model)**.** *Let k and n be two positive integers. In the generic multilinear map model with oracle $\mathscr{M}$, for any adversary $\mathscr{D}$ making at most $q_{\mathscr{M}}$ queries to the oracle $\mathscr{M}$, we have:*

Proof of Corollary 7.6.10. We just consider an intermediate game where we simulate everything as before except the zero-testing oracle which always outputs “non-zero” unless its input is zero, as a polynomial. Thus, we have

$$\mathsf{Adv}^{(k,n)\text{-}\mathsf{Sel}\text{-}\mathsf{Prod}}(\mathscr{D}) \leq 2q_{\mathscr{M}} \frac{|\mathscr{U}|}{p} \ ,$$

and Corollary 7.6.10 easily follows. □

Chapter 8

Conclusion and Open Questions

8.1 Conclusion

One of the main contribution of this thesis is to introduce an algebraic framework to analyze the security of certain forms of pseudorandom functions or extensions. Specifically, the PLP theorem and its weaker variant the LIP theorem allow us to translate in a very simple fashion the security of pseudorandom functions (or associated primitives) into simple algebraic properties. This leads to different applications for pseudorandom functions in the standard as well as in the related-key setting, but also for aggregate or multilinear pseudorandom functions. In particular, thanks to the generality of the PLP theorem, we are able to provide constructions with weaker assumptions or, in the case of related-key security, that handles larger classes of related-key deriving functions.

Concerning the related-key setting, we also define new frameworks to tranform a pseudorandom function into a related-key secure version. In particular, we repair and extend the Bellare-Cash framework, which allows us to both recover and strengthen previously known results in this area.

However, despite these new results, we have not been able to construct a related-key secure pseudorandom function for real-world classes, such as XOR relations, using any of these frameworks. Hence, we target this problem in the last part of this thesis, and provide the first provably-secure pseudorandom function for XOR relations, assuming the existence of a weak form of multilinear maps. Unfortunately, this is a strong assumption, and this result should be only seen as a proof of concept.

8.2 Open Questions

There are still several open problems that should be tackled in this area. Despite significant progress over the last three years, we are still far from solving any of them. Here is a non-exhaustive list of open questions that we would love to solve at some point.

A first series of questions is related to our main result, the PLP theorem. We would be interested in showing that it can be extended to more advanced primitives or to different assumptions. Here are a few examples of open questions.

Question 8.1. *Can we extend the* PLP *theorem to handle primitives that offer more functionalities, for instance verifiable random functions?*

Question 8.2. *In particular, can we propose an algebraic framework to transform the form of pseudorandom functions we study into verifiable random functions, generically?*

Question 8.3. *Can we develop a lattice-based variant of the* PLP *theorem, that would translate the pseudorandomness of a construction into simple properties?*

Also, targeting more specifically the related-key setting, building an XOR-related-key secure pseudorandom function based on standard assumptions remains an open problem, and probably the most important open problem in this area. Therefore, here are two natural open questions that we would be interested in solving.

Question 8.4. *Can we build an XOR-related-key secure pseudorandom function without multilinear maps (or any other powerful tool such as indistinguishability obfuscation)? In particular, can we construct one under standard assumptions?*

Question 8.5. *Alternatively, can we construct the weak form of multilinear maps we need for our XOR-related-key secure pseudorandom function under standard assumptions, such as lattice-based assumptions?*

Notation

Mathematical Notation

$\mathbb{N}$	set of non-negative integers
$\mathbb{Z}$	set of integers
$p, p_1, p_2, \ldots$	prime numbers
$(\mathbb{Z}_N, +, \cdot)$ or $\mathbb{Z}_N$	ring of integers modulo N, with $N \geq 1$ being an integer
$(\mathbb{Z}_N, +)$	$\mathbb{Z}_N$ seen as an additive group
$(\mathbb{Z}_N^*, \cdot)$ or $\mathbb{Z}_N^*$	multiplicative subgroup of $\mathbb{Z}_N$
$\mathbb{G}, \mathbb{G}_1, \ldots$	cyclic groups
$(N, \mathbb{G}, g)$	cyclic group of order N and generated by g
$[a]_g$	group element g^a
$\lvert\mathscr{S}\rvert$	cardinal of the set $\mathscr{S}$
$\mathcal{D} \times \mathcal{R}$	cartesian product of $\mathcal{D}$ and $\mathcal{R}$
$\mathcal{D} \otimes \mathcal{R}$	tensor product of $\mathcal{D}$ and $\mathcal{R}$
$\mathsf{Fun}(\mathcal{D}, \mathcal{R})$	the set of functions with domain $\mathcal{D}$ and range $\mathcal{R}$
$\mathsf{L}(\mathcal{D}, \mathcal{R})$	the vector space of linear functions from $\mathcal{D}$ to $\mathcal{R}$
$\vec{u}, \vec{x}, \ldots$	column vectors: $\vec{u} = (u_i)_{i=1,\ldots,n}$
$\lvert\vec{v}\rvert$	length of $\vec{v}$
$\langle \vec{u}, \vec{v} \rangle$	inner product of $\vec{u}$ and $\vec{v}$
$\boldsymbol{A}, \boldsymbol{B}, \ldots$	matrices: $\boldsymbol{A} = (a_{i,j})_{\substack{i=1,\ldots,n \\ j=1,\ldots,m}}$
$R[T]$	ring of univariate polynomials over ring R
$R[T_1, \ldots, T_n]$	ring of multivariate polynomials in $T_1, \ldots, T_n$ over ring R
$R[T_1, \ldots, T_n]_{\leq d}$	its subspace of degree at most d (in one indeterminate) multivariate polynomials

Algorithmic Concepts

$\{0,1\}^*$	the set of all bitstrings
$\{0,1\}^n$	the set of all bitstrings of length n
$\lvert x\rvert$	length of x
$x \oplus y$	exclusive or between $x, y \in \{0,1\}^n$
$y \overset{\$}{\leftarrow} \mathscr{S}$	y is assigned to a uniform element from the set $\mathscr{S}$
$y \overset{\$}{\leftarrow} A(x)$	x is the output of A on input x with fresh random coins
$y \leftarrow A(x)$	same when A is deterministic

Provable Security

$\kappa, 1^\kappa$	security parameter and its unary representation
$\mathscr{A}, \mathscr{B}, \ldots$	adversaries
$\mathbf{G}, \mathbf{G}_1, \ldots$	games
$\mathsf{Adv}^{\mathsf{Exp}}(\mathscr{A}, \kappa)$	advantage of $\mathscr{A}$ in an experiment Exp or in distinguishing the experiments Exp^0 and Exp^1, with security parameter κ

Abbreviations

Assumptions	
DDH	Decisional Diffie-Hellman
d-DDHI	d-Decisional Diffie-Hellman Inversion
DL	Discrete Logarithm
DLin	Decisional Linear
k-Lin	k-Linear
MDDH	Matrix Decisional Diffie-Hellman
d-SDL	d-Strong Discrete Logarithm
Cryptographic Notions	
AGG-PRF	Aggregate Pseudorandom Function
CR	Collision Resistance
KC	Key-Collision
LIP	Linearly Iindependent Polynomial
MMAP	Multilinear Map
MPRF	Multilinear Pseudorandom Function
PLP	Polynomial Linear Pseudorandomness
PPT	Probabilistic Polynomial-Time
PRF	Pseudorandom Function
RKA	Related-Key Attack
RKA-PRF	Related-Key Pseudorandom Function
RKD	Related-Key Deriving
SKC	Statistical-Key-Collision

List of Illustrations

Figures

Tables

Bibliography

[01] *Advanced Encryption Standard (AES).* National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department of Commerce. Nov. 2001 (cit. on p. 4).

[ACF09] Michel Abdalla, Dario Catalano, and Dario Fiore. “Verifiable Random Functions from Identity-Based Key Encapsulation”. In: *EUROCRYPT 2009.* Ed. by Antoine Joux. Vol. 5479. LNCS. Springer, Heidelberg, Apr. 2009, pp. 554–571 (cit. on p. 6).

[AJ15] Prabhanjan Ananth and Abhishek Jain. “Indistinguishability Obfuscation from Compact Functional Encryption”. In: *CRYPTO 2015, Part I.* Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9215. LNCS. Springer, Heidelberg, Aug. 2015, pp. 308–326. DOI: `10.1007/978-3-662-47989-6_15` (cit. on p. 11).

[BBD+15] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. “Verified Proofs of Higher-Order Masking”. In: *EUROCRYPT 2015, Part I.* Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. LNCS. Springer, Heidelberg, Apr. 2015, pp. 457–485. DOI: `10.1007/978-3-662-46800-5_18` (cit. on p. 11).

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. “Hierarchical Identity Based Encryption with Constant Size Ciphertext”. In: *EUROCRYPT 2005.* Ed. by Ronald Cramer. Vol. 3494. LNCS. Springer, Heidelberg, May 2005, pp. 440–456 (cit. on p. 65).

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. “Short Group Signatures”. In: *CRYPTO 2004.* Ed. by Matthew Franklin. Vol. 3152. LNCS. Springer, Heidelberg, Aug. 2004, pp. 41–55 (cit. on p. 6).

[BC10a] Mihir Bellare and David Cash. *Pseudorandom Functions and Permutations Provably Secure Against Related-Key Attacks.* Cryptology ePrint Archive, Report 2010/397. `http://eprint.iacr.org/2010/397`. 2010 (cit. on p. 34).

[BC10b] Mihir Bellare and David Cash. “Pseudorandom Functions and Permutations Provably Secure against Related-Key Attacks”. In: *CRYPTO 2010.* Ed. by Tal Rabin. Vol. 6223. LNCS. Springer, Heidelberg, Aug. 2010, pp. 666–684 (cit. on pp. 6, 8, 9, 29, 33, 35, 36, 38, 41, 45).

[BCM11] Mihir Bellare, David Cash, and Rachel Miller. “Cryptography Secure against Related-Key Attacks and Tampering”. In: *ASIACRYPT 2011.* Ed. by Dong Hoon Lee and Xiaoyun Wang. Vol. 7073. LNCS. Springer, Heidelberg, Dec. 2011, pp. 486–503 (cit. on pp. 7, 29, 32).

[BDK+10] Alex Biryukov, Orr Dunkelman, Nathan Keller, Dmitry Khovratovich, and Adi Shamir. “Key Recovery Attacks of Practical Complexity on AES-256 Variants with up to 10 Rounds”. In: *EUROCRYPT 2010.* Ed. by Henri Gilbert. Vol. 6110. LNCS. Springer, Heidelberg, May 2010, pp. 299–319 (cit. on p. 7).

[BDK05] Eli Biham, Orr Dunkelman, and Nathan Keller. “Related-Key Boomerang and Rectangle Attacks”. In: *EUROCRYPT 2005*. Ed. by Ronald Cramer. Vol. 3494. LNCS. Springer, Heidelberg, May 2005, pp. 507–525 (cit. on p. 7).

[BDK08] Eli Biham, Orr Dunkelman, and Nathan Keller. “A Unified Approach to Related-Key Attacks”. In: *FSE 2008*. Ed. by Kaisa Nyberg. Vol. 5086. LNCS. Springer, Heidelberg, Feb. 2008, pp. 73–96 (cit. on p. 7).

[Ber06] Daniel J. Bernstein. “Curve25519: New Diffie-Hellman Speed Records”. In: *PKC 2006*. Ed. by Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin. Vol. 3958. LNCS. Springer, Heidelberg, Apr. 2006, pp. 207–228 (cit. on p. 19).

[BGK+14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. “Protecting Obfuscation against Algebraic Attacks”. In: *EUROCRYPT 2014*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Vol. 8441. LNCS. Springer, Heidelberg, May 2014, pp. 221–238. DOI: `10.1007/978-3-642-55220-5_13` (cit. on p. 115).

[Bih94] Eli Biham. “New Types of Cryptoanalytic Attacks Using related Keys (Extended Abstract)”. In: *EUROCRYPT’93*. Ed. by Tor Helleseth. Vol. 765. LNCS. Springer, Heidelberg, May 1994, pp. 398–409 (cit. on p. 7).

[BK03] Mihir Bellare and Tadayoshi Kohno. “A Theoretical Treatment of Related-Key Attacks: RKA-PRPs, RKA-PRFs, and Applications”. In: *EUROCRYPT 2003*. Ed. by Eli Biham. Vol. 2656. LNCS. Springer, Heidelberg, May 2003, pp. 491–506 (cit. on pp. 7, 29–31).

[BK09] Alex Biryukov and Dmitry Khovratovich. “Related-Key Cryptanalysis of the Full AES-192 and AES-256”. In: *ASIACRYPT 2009*. Ed. by Mitsuru Matsui. Vol. 5912. LNCS. Springer, Heidelberg, Dec. 2009, pp. 1–18 (cit. on p. 7).

[BKN09] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolic. “Distinguisher and Related-Key Attack on the Full AES-256”. In: *CRYPTO 2009*. Ed. by Shai Halevi. Vol. 5677. LNCS. Springer, Heidelberg, Aug. 2009, pp. 231–249 (cit. on p. 7).

[BLR+15] Dan Boneh, Kevin Lewi, Mariana Raykova, Amit Sahai, Mark Zhandry, and Joe Zimmerman. “Semantically Secure Order-Revealing Encryption: Multi-input Functional Encryption Without Obfuscation”. In: *EUROCRYPT 2015, Part II*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9057. LNCS. Springer, Heidelberg, Apr. 2015, pp. 563–594. DOI: `10.1007/978-3-662-46803-6_19` (cit. on p. 10).

[BMR10] Dan Boneh, Hart William Montgomery, and Ananth Raghunathan. “Algebraic pseudorandom functions with improved efficiency from the augmented cascade”. In: *ACM CCS 10*. Ed. by Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov. ACM Press, Oct. 2010, pp. 131–140 (cit. on pp. 6, 21).

[BMSZ16] Saikrishna Badrinarayanan, Eric Miles, Amit Sahai, and Mark Zhandry. “Post-zeroizing Obfuscation: New Mathematical Tools, and the Case of Evasive Circuits”. In: *EUROCRYPT 2016, Part II*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9666. LNCS. Springer, Heidelberg, May 2016, pp. 764–791. DOI: `10.1007/978-3-662-49896-5_27` (cit. on p. 118).

[Boy08] Xavier Boyen. “The Uber-Assumption Family (Invited Talk)”. In: *PAIRING 2008*. Ed. by Steven D. Galbraith and Kenneth G. Paterson. Vol. 5209. LNCS. Springer, Heidelberg, Sept. 2008, pp. 39–56 (cit. on p. 65).

[BR06] Mihir Bellare and Phillip Rogaway. “The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs”. In: *EUROCRYPT 2006*. Ed. by Serge Vaudenay. Vol. 4004. LNCS. Springer, Heidelberg, May 2006, pp. 409–426 (cit. on p. 16).

[BV15] Nir Bitansky and Vinod Vaikuntanathan. “Indistinguishability Obfuscation from Functional Encryption”. In: *56th FOCS*. Ed. by Venkatesan Guruswami. IEEE Computer Society Press, Oct. 2015, pp. 171–190. DOI: 10.1109/FOCS.2015.20 (cit. on p. 11).

[BW04] Andrej Bogdanov and Hoeteck Wee. “A Stateful Implementation of a Random Function Supporting Parity Queries over Hypercubes”. In: *Approximation, Randomization, and Combinatorial Optimization, Algorithms and Techniques, 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2004, and 8th International Workshop on Randomization and Computation, RANDOM 2004, Cambridge, MA, USA, August 22-24, 2004, Proceedings*. Ed. by Klaus Jansen, Sanjeev Khanna, José D. P. Rolim, and Dana Ron. Vol. 3122. Lecture Notes in Computer Science. Springer, 2004, pp. 298–309. ISBN: 3-540-22894-2. DOI: 10.1007/978-3-540-27821-4_27. URL: http://dx.doi.org/10.1007/978-3-540-27821-4_27 (cit. on p. 24).

[CGV15] Aloni Cohen, Shafi Goldwasser, and Vinod Vaikuntanathan. “Aggregate Pseudorandom Functions and Connections to Learning”. In: *TCC 2015, Part II*. Ed. by Yevgeniy Dodis and Jesper Buus Nielsen. Vol. 9015. LNCS. Springer, Heidelberg, Mar. 2015, pp. 61–89. DOI: 10.1007/978-3-662-46497-7_3 (cit. on pp. 8, 9, 22, 106, 108, 110, 111).

[CH15] Aloni Cohen and Justin Holmgren. “Multilinear Pseudorandom Functions”. In: *ICALP 2015, Part I*. Ed. by Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann. Vol. 9134. LNCS. Springer, Heidelberg, July 2015, pp. 331–342. DOI: 10.1007/978-3-662-47672-7_27 (cit. on pp. 8, 23, 24, 106).

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. “Practical Multilinear Maps over the Integers”. In: *CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 476–493. DOI: 10.1007/978-3-642-40041-4_26 (cit. on pp. 26, 114, 117, 118, 121).

[CLT15] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. “New Multilinear Maps Over the Integers”. In: *CRYPTO 2015, Part I*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9215. LNCS. Springer, Heidelberg, Aug. 2015, pp. 267–286. DOI: 10.1007/978-3-662-47989-6_13 (cit. on pp. 26, 114, 117, 118).

[EHK+13] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. “An Algebraic Framework for Diffie-Hellman Assumptions”. In: *CRYPTO 2013, Part II*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8043. LNCS. Springer, Heidelberg, Aug. 2013, pp. 129–147. DOI: 10.1007/978-3-642-40084-1_8 (cit. on pp. 6, 53, 55, 65).

[GGH13] Sanjam Garg, Craig Gentry, and Shai Halevi. “Candidate Multilinear Maps from Ideal Lattices”. In: *EUROCRYPT 2013*. Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. LNCS. Springer, Heidelberg, May 2013, pp. 1–17. DOI: 10.1007/978-3-642-38348-9_1 (cit. on pp. 26, 114, 117, 118).

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. “Graph-Induced Multilinear Maps from Lattices”. In: *TCC 2015, Part II*. Ed. by Yevgeniy Dodis and Jesper Buus Nielsen. Vol. 9015. LNCS. Springer, Heidelberg, Mar. 2015, pp. 498–527. DOI: 10.1007/978-3-662-46497-7_20 (cit. on pp. 26, 114, 117, 118, 121).

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. “How to Construct Random Functions (Extended Abstract)”. In: *25th FOCS*. IEEE Computer Society Press, Oct. 1984, pp. 464–479 (cit. on pp. 4, 5).

[GL89] Oded Goldreich and Leonid A. Levin. “A Hard-Core Predicate for all One-Way Functions”. In: *21st ACM STOC*. ACM Press, May 1989, pp. 25–32 (cit. on p. 4).

[GLSW15] Craig Gentry, Allison Bishop Lewko, Amit Sahai, and Brent Waters. “Indistinguishability Obfuscation from the Multilinear Subgroup Elimination Assumption”. In: *56th FOCS*. Ed. by Venkatesan Guruswami. IEEE Computer Society Press, Oct. 2015, pp. 151–170. DOI: 10.1109/FOCS.2015.19 (cit. on p. 121).

[GLW14] Craig Gentry, Allison B. Lewko, and Brent Waters. “Witness Encryption from Instance Independent Assumptions”. In: *CRYPTO 2014, Part I*. Ed. by Juan A. Garay and Rosario Gennaro. Vol. 8616. LNCS. Springer, Heidelberg, Aug. 2014, pp. 426–443. DOI: 10.1007/978-3-662-44371-2_24 (cit. on p. 121).

[GOR11] Vipul Goyal, Adam O’Neill, and Vanishree Rao. “Correlated-Input Secure Hash Functions”. In: *TCC 2011*. Ed. by Yuval Ishai. Vol. 6597. LNCS. Springer, Heidelberg, Mar. 2011, pp. 182–200 (cit. on p. 6).

[HW10] Susan Hohenberger and Brent Waters. “Constructing Verifiable Random Functions with Large Input Spaces”. In: *EUROCRYPT 2010*. Ed. by Henri Gilbert. Vol. 6110. LNCS. Springer, Heidelberg, May 2010, pp. 656–672 (cit. on p. 6).

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. “Private Circuits: Securing Hardware against Probing Attacks”. In: *CRYPTO 2003*. Ed. by Dan Boneh. Vol. 2729. LNCS. Springer, Heidelberg, Aug. 2003, pp. 463–481 (cit. on p. 11).

[KHP07] Jongsung Kim, Seokhie Hong, and Bart Preneel. “Related-Key Rectangle Attacks on Reduced AES-192 and AES-256”. In: *FSE 2007*. Ed. by Alex Biryukov. Vol. 4593. LNCS. Springer, Heidelberg, Mar. 2007, pp. 225–241 (cit. on p. 7).

[Knu93] Lars R. Knudsen. “Cryptanalysis of LOKI91”. In: *AUSCRYPT’92*. Ed. by Jennifer Seberry and Yuliang Zheng. Vol. 718. LNCS. Springer, Heidelberg, Dec. 1993, pp. 196–208 (cit. on p. 7).

[Kob87] Neal Koblitz. “Elliptic curve cryptosystems”. In: *Mathematics of computation* 48.177 (1987), pp. 203–209 (cit. on p. 19).

[KR01] Joe Kilian and Phillip Rogaway. “How to Protect DES Against Exhaustive Key Search (an Analysis of DESX)”. In: *Journal of Cryptology* 14.1 (2001), pp. 17–35 (cit. on p. 16).

[LW09] Allison B. Lewko and Brent Waters. “Efficient pseudorandom functions from the decisional linear assumption and weaker variants”. In: *ACM CCS 09*. Ed. by Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis. ACM Press, Nov. 2009, pp. 112–120 (cit. on p. 6).

[Mil86] Victor S. Miller. “Use of Elliptic Curves in Cryptography”. In: *CRYPTO’85*. Ed. by Hugh C. Williams. Vol. 218. LNCS. Springer, Heidelberg, Aug. 1986, pp. 417–426 (cit. on p. 19).

[NR97] Moni Naor and Omer Reingold. “Number-theoretic Constructions of Efficient Pseudo-random Functions”. In: *38th FOCS*. IEEE Computer Society Press, Oct. 1997, pp. 458–467 (cit. on pp. 6, 9, 21).

[Sha71] Daniel Shanks. “Class number, a theory of factorization, and genera”. In: *Proc. Symp. Pure Math.* Vol. 20. 1971, pp. 415–440 (cit. on p. 19).

[Sho01] Victor Shoup. “OAEP Reconsidered”. In: *CRYPTO 2001*. Ed. by Joe Kilian. Vol. 2139. LNCS. Springer, Heidelberg, Aug. 2001, pp. 239–259 (cit. on p. 16).

[Sho97] Victor Shoup. “Lower Bounds for Discrete Logarithms and Related Problems”. In: *EUROCRYPT’97*. Ed. by Walter Fumy. Vol. 1233. LNCS. Springer, Heidelberg, May 1997, pp. 256–266 (cit. on p. 26).

[Zim15] Joe Zimmerman. “How to Obfuscate Programs Directly”. In: *EUROCRYPT 2015, Part II*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9057. LNCS. Springer, Heidelberg, Apr. 2015, pp. 439–467. DOI: `10.1007/978-3-662-46803-6_15` (cit. on pp. 25, 114–116, 129).

Résumé

Dans cette thèse, nous étudions la structure algébrique sous-jacente aux fonctions pseudo-aléatoires basées sur la théorie des nombres. Précisément, nous montrons que le caractère pseudo-aléatoire de fonctions d'une forme particulière est équivalent au fait que ces fonctions vérifient une propriété algébrique simple. Cette caractérisation algébrique s'applique à la plupart des constructions connues et s'étend naturellement aux généralisations des fonctions pseudo-aléatoires, par exemple aux fonctions pseudo-aléatoires sûres contre les attaques par clés liées, multilinéaires ou supportant l'agrégation.

Cette caractérisation repose sur une famille d'hypothèses MDDH, qui contient en particulier différentes hypothèses classiques, comme DDH, DLin, ou k-Lin. Ainsi, en choisissant les paramètres des constructions selon, ce résultat permet de construire des fonctions pseudo-aléatoires (ou leurs généralisations) prouvées sûres sous ces différentes hypothèses.

Enfin, nous étudions également plus précisément le cas de la sécurité contre les attaques par clés liées. D'une part, nous corrigeons et étendons la transformation proposée par Bellare et Cash pour ensuite construire de nouvelles fonctions pseudo-aléatoires sûres contre des attaques par clés liées plus puissantes. D'autre part, nous construisons la première fonction pseudo-aléatoire prouvée sûre contre des attaques par XOR. Cette construction repose sur l'existence d'une forme faible d'applications multilinéaires.

Abstract

In this thesis, we study the algebraic structure underlying number-theoretic pseudorandom functions. Specifically, we define an algebraic framework that translates the pseudorandomness of a particular form of functions into a simple algebraic property. The resulting generic framework encompasses most of existing constructions and naturally extends to related primitives, such as related-key secure, aggregate, and multilinear pseudorandom functions.

This framework holds under a family of MDDH assumptions, that contains especially different classical assumptions, such as DDH, DLin, or k-Lin. Therefore, setting accordingly the parameters in the constructions, our framework can be used to construct secure pseudorandom functions (or related primitives) whose security holds under these assumptions.

Finally, we also study more specifically the case of related-key security. On the one hand, we propose a fix and some extensions to the Bellare-Cash framework and then build pseudorandom functions secure against larger classes of attacks. On the other hand, we construct the first pseudorandom function provably secure against XOR attacks. The latter construction relies on the existence of a weak form of multilinear maps.

Mots Clés

cryptographie, fonctions pseudo-aléatoires, sécurité prouvée, sécurité par clés liées, constructions algébriques, hypothèses calculatoires.

Keywords

cryptography, pseudorandom functions, provable security, related-key security, algebraic constructions, computational assumptions.