Pseudo-Random Synthesizers, Functions and Permutations

Thesis for the Degree of

DOCTOR of PHILOSOPHY

by

Omer Reingold

Department of Applied Mathematics and Computer Science
Weizmann Institute of Science

Submitted to the Scientific Council of
The Weizmann Institute of Science
Rehovot 76100, Israel

November 1998

Thesis Advisor

Professor Moni Naor

Department of Applied Mathematics and Computer Science
Weizmann Institute of Science
Rehovot 76100, Israel

Acknowledgments

Writing these lines, I realize how many people took part in my initiation into the world of science. In the following few paragraphs I will try to acknowledge some of these many contributions. Naturally, my warmest thanks are to Moni Naor my advisor (and much more than that).

Thanking Moni for all he has done and been to me these last four years is extremely easy and extremely hard at the same time. Extremely easy since I have so much I want to thank him for and I'm grateful for this opportunity to do so (a cynic might say that this is the most important function of the dissertation). Extremely hard since I have so much I want to thank him for ... what chance is there to do a good enough job? I'm sure that, in the silent understanding between us, Moni is more aware of the extent of friendship, admiration and gratitude I have for him than I am able to express in words. But this is not only for his eyes ... so let me do my best.

Thank you Moni for your close guidance in all different aspects of the scientific process. In all things, large and small, I always knew that I can count on you!! Thank you for your constant encouragement, for treating me as a colleague from the first time I stepped into your office (when I deserved it even less than now) and at the same time for sheltering me in a parental manner. Thank you for pushing me forward when possible and for laying off in times work could not have been a high priority for me. Thank you for sharing with me, in hours and hours of conversation, your deep understanding and amazing knowledge of computer science as well as your scientific philosophy and an abundance of ideas (all of which will surely find their way into my research for years to come). Last but certainly not least, thank you for your friendship (which was most explicitly articulated by Yael but was very clear to me all along). The nest you have built for me is so warm that I hate leaving it. Therefore, (at least in my mind) I will not!

Before picking up the pace, special thanks are due to one other friend, teacher and colleague of mine. Although my joint research with Ran Raz is not presented in this dissertation, I enjoyed it enormously and have learned much from it. I enjoyed viewing Ran's superb research capabilities and creativity in action (hope some of it has rubbed off on me too). I admire his relentless optimism and enthusiasm. I treasure his friendship so very deeply. I wish for many more years of all of that.

I thank my teachers at the Weizmann Institute for the invaluable lessons they have taught me both inside and outside of the class room. Special thanks to each member of the Cryptography group: Oded Goldreich, Shafi Goldwasser, Moni Naor and Adi Shamir. I take great pride and pleasure in the fact that I have learned from this incredible group of researchers (and even in being associated with them). I often toy with the idea of not

graduating in order to continue learning in the presence of these experts. I also thank Itai Benjamini, Uri Feige, Yoram Moses and David Peleg for their enlightening courses.

A group of researchers to whom I have special sentiments are my teachers from undergraduate studies at Tel-Aviv University: Noga Alon, Yossi Azar, Amos Fiat, Ron Shamir and Uri Zwick. I thank them for installing in me the love for the foundations of computer science.

Over the years I have learned with and from many of my fellow students. I will list only a few but I thank them all for many hours of work and fun. Special thanks are due to my close "brothers" at the house of Moni, namely Kobbi Nissim and Benny Pinkas. Having you guys around was priceless. Many thanks are also due to Avishai Wool and to the new recruit Yehuda Lindell. I also thank Tal Malkin for our joint work at the coffee shops of Manhattan, and Tal Yadid for an extremely valuable time we spent together in undergraduate school.

There are so many other people to whom I am in debt for sharing their knowledge and ideas with me. The partial list that comes to mind right now includes Eli Biham, Dan Boneh, Ran Canetti, Cynthia Dwork, Russell Impagliazzo, Daniele Micciancio, Noam Nisan, Steven Rudich, Amnon Ta-Shma, Avi Wigderson and Shiyu Zhou. I apologize to anyone I have forgetfully omitted and thank you all for many stimulating conversations. Special thanks to Cynthia, Russell and Steven for extending to me part of their sentiments for Moni. I would also like to thank Joan Feigenbaum, Christos Papadimitriou, Charlie Rackoff and Umesh Vazirani for their warm hospitality during a few memorable visits.

I would like to acknowledge now the direct contributions to the work presented in this dissertation. Most of the work is based on my joint research with Moni [101, 102, 103]. Section 3.2 is based on joint work with Eli Biham and Dan Boneh [22]. My research during this time was supported by a Clore Scholars award and a grant from the Israel Science Foundation administered by the Israeli Academy of Sciences. We thank Shafi Goldwasser and Jon Sorenson who brought [121] to our attention and Mihir Bellare for his observation described in Section 4.1.6. Special thanks to Victor Shoup for suggesting the improved proof of Lemma 4.2.4 and for pointing out [143]. We thank Ran Canetti, Oded Goldreich, Joe Kilian, Kobbi Nissim, Benny Pinkas, Amnon Ta-Shma and the anonymous referees of the relevant journals and conferences for many helpful comments and for their diligent reading of some of the work included here. It is difficult to overestimate Oded's contribution to the presentation of the work described in Chapter 5. I would also like to thank Oded Goldreich, Joe Kilian, Yehuda Lindell and Alon Rosen for their comments on several parts of the dissertation.

Finally, many thanks are due to my family and friends. However, they deserve their thanks in person (and hopefuly I did not let them wait till now).

Abstract

The research reflected in this dissertation is a study of (computational) pseudo-randomness. More specifically, the main objective of this research is the efficient and simple construction of *pseudo-random functions* and *permutations* [62, 90], where efficiency refers both to the sequential and parallel time complexity of the computation. Pseudo-random functions and permutations are fundamental cryptographic primitives with many applications in cryptography and more generally in computational complexity.

Constructions of Pseudo-Random Functions

For our constructions of pseudo-random functions, we introduce and study a new cryptographic primitive which we call a *pseudo-random synthesizer* and a generalization of this primitive which we call a *k-dimensional pseudo-random synthesizer*. These primitives are of independent interest as well. In addition, we consider various applications of our constructions and study some of the underlying cryptographic assumptions used in these constructions. The main results obtained by this research are:

- Introducing new cryptographic primitives called pseudo-random synthesizer and k-dimensional pseudo-random synthesizer.
- Using pseudo-random synthesizers for a parallel construction of a pseudo-random function (the depth of the functions is larger by a logarithmic factor than the depth of the synthesizers).
- Showing several NC^1 implementations of synthesizers based on concrete intractability assumptions such as *factoring* and the *computational* Diffie-Hellman assumption.
- Showing a very simple, parallel construction of synthesizers based on what we call weak pseudo-random functions which implies simple constructions of synthesizers based on trapdoor one-way permutations and based on any hard-to-learn problem (under the definition of [23]).

These results yield the first parallel pseudo-random functions (based on computational intractability assumptions) and the first alternative to the original construction of Goldreich, Goldwasser and Micali [62]. In addition, we show two new constructions of pseudo-random functions (that are related to the construction based on synthesizers). The pseudo-randomness of one construction is proven under the assumption that *factoring* is hard while

the other construction is pseudo-random if the *decisional version of the Diffie-Hellman* assumption holds. These functions have the following properties:

- They are much more efficient than previous proposals: Computing the value of our functions at any given point involves two subset products.
- They are in TC^0 (the class of functions computable by constant depth circuits consisting of a polynomial number of threshold gates). This fact has several interesting applications.
- They have a simple algebraic structure that implies additional features. In particular, we show a zero-knowledge proof for statements of the form "$y = f_s(x)$" and "$y \neq f_s(x)$" given a commitment to a key s of a pseudo-random function f_s.

We discuss some applications of our constructions in cryptography (including applications in public-key cryptography) as well as their consequences in computational complexity and in computational learning-theory.

Constructions of Pseudo-Random Permutations

Luby and Rackoff [90] showed a method for constructing a pseudo-random permutation from a pseudo-random function. The method is based on composing four (or three for weakened security) so called Feistel permutations, each of which requires the evaluation of a pseudo-random function. We reduce somewhat the complexity of the construction and simplify its proof of security by showing that two Feistel permutations are sufficient together with initial and final pair-wise independent permutations. The revised construction and proof provide a framework in which similar constructions may be designed and their security can be easily proved. We demonstrate this by presenting some additional adjustments of the construction that achieve the following:

- Reduce the success probability of the adversary.
- Provide a construction of pseudo-random permutations with *large* input-length using pseudo-random functions with *small* input length.

A Study of Some Number-Theoretical Assumptions

Our research includes a study of two number-theoretical assumptions that are related to the Diffie-Hellman key-exchange protocol and that are used in our constructions of pseudo-random functions. The first is the *decisional* version of the Diffie-Hellman assumption (DDH-Assumption). This assumption is relatively new, or more accurately, was *explicitly considered* only recently. We therefore survey some of the different applications of the assumption and the current knowledge on its security. Furthermore, we show a randomized reduction of the worst-case DDH-Assumption to its average case (based on the random-self-reducibility of the DDH-Problem that was previously used by Stadler [143]). We consider our research of the DDH-Assumption to be of independent importance given that the assumption was recently used in quite a few interesting applications (e.g., [45]).

The second assumption we study is the *generalized* Diffie-Hellman assumption (GDH-Assumption). This assumption was originally considered in the context of a generalization of the Diffie-Hellman key-exchange protocol to $k > 2$ parties. We prove that breaking this assumption modulo a so called Blum-integer would imply an efficient algorithm for factoring Blum-integers. Therefore, both the generalized key-exchange protocol and our pseudo-random function (that is based on the GDH-Assumption) are secure as long as factoring Blum-integers is hard. Our reduction strengthens a previous "worst-case" reduction of Shmuely [139].

Contents

Chapter 1

Introduction

The research reflected in this dissertation is a study of (computational) pseudo-randomness. More specifically, the main objective of this research is the efficient and simple construction of *pseudo-random functions* and *permutations* [62, 90], where efficiency refers both to the sequential and parallel time complexity of the computation. Pseudo-random functions and permutations are fundamental cryptographic primitives with many applications in cryptography and more generally in computational complexity.

1.1 Pseudo-Randomness and Computational Indistinguishability

Since pseudo-randomness is the main notion considered in this work, we start by reviewing some of the central ideas of this area. We also discuss the definitions, applications and constructions of the relevant pseudo-random objects, with a focus on pseudo-random functions and permutations. Good references for additional reading are Goldreich [60, 61] and Luby [89].

To understand the notion of computational *pseudo-randomness* we must first consider the role of *randomness* in computations. A randomized algorithm is an algorithm that is allowed to "flip coins". A coin flip is modeled as a random (i.e., uniformly distributed) bit. Therefore, at each step of its computation, a randomized algorithm can obtain a bit which is 0 with probability half and 1 with probability half (and is independent of previous coin-flips). An important question, though less relevant to our discussion, is to determine the *applicability* of this model: What kind of random sources are available to computers? We note that this question is non-trivial even to those who believe that God (or Nature) does indeed flip coins.

Many algorithms and protocols use randomness to perform computational tasks. These algorithms are often faster and simpler than the corresponding, currently known, deterministic algorithms. For some tasks, as those of achieving cryptographic security or simulating probabilistic events, randomness is essential. Thus, it is natural to view randomness as a computational resource and to study its relations with other computational resources like time and space. More specifically, a natural question is that of *derandomization*: study-

ing ways of reducing or eliminating the amount of randomness used by algorithms without significantly enlarging their usage of other resources.

An important tool for derandomization is pseudo-randomness (though, as we argue below, the role of pseudo-randomness is not limited to derandomization). Consider an algorithm $\mathcal{A}$ that uses a random string of length ℓ (i.e., $\mathcal{A}$ uses ℓ random bits). One way to reduce the amount of randomness used by $\mathcal{A}$, (without extensively changing its internal structure), is to replace its uniformly distributed random string with a string sampled from a different distribution D. That is, to use an algorithm $\mathcal{A}'$ that samples a string $\vec{r}$ from D and invokes $\mathcal{A}$ with $\vec{r}$ as its random string. For this technique to be useful, D must have the following properties:

1. Sampling a string from D should require significantly less than ℓ random bits (to achieve our original goal of derandomizing $\mathcal{A}$).

2. D is efficiently samplable: For the derandomization of $\mathcal{A}$ (using D) not to be too costly, the resources required to sample a string from D should be comparable with those needed to execute $\mathcal{A}$ (otherwise running $\mathcal{A}'$ is substantially more expensive than running $\mathcal{A}$).

3. The "behavior" of $\mathcal{A}$ should be practically the same when using a uniformly distributed string and when using a string sampled from D. For example, if $\mathcal{A}$ solves some computational problem its probability to answer correctly (on every input or on most inputs) should be almost the same for both distributions of its random string. In this sense $\mathcal{A}$ does not distinguish D from the uniform distribution (put differently, $\mathcal{A}$ cannot be used to distinguish the two distributions).

By Property 1, D has much less entropy than the uniform distribution and is therefore statistically very different from it. However, Property 3, means that for the specific computational task at hand D is almost as good as the uniform distribution. This contrast between a large statistical difference and *computational indistinguishability* is in the essence of pseudo-randomness (see also [59, 64] for additional insight on the role of Properties 1-3). To better understand the definition of computational indistinguishability (due to Goldwasser and Micali [68] and to Yao [150]) we can think of the algorithm $\mathcal{A}$ not as trying to perform just *any* computational task but rather as trying to *distinguish* between two distributions D and D': with probability half $\mathcal{A}$ gets an input sampled from D and with probability half an input sampled from D'. The algorithm $\mathcal{A}$ tries to guess from which distribution its input was sampled. D and D' are indistinguishable to $\mathcal{A}$ if its success probability is at most negligibly larger than half (which is the success probability $\mathcal{A}$ can easily achieve for every D and D').

Usually however we would be interested in derandomizing a class of algorithms (or protocols) instead of a single algorithm $\mathcal{A}$ as in the discussion above. In such a case, we want the pseudo-random distribution to "fool" *every algorithm from the class* that we are trying to derandomize. That is, to be indistinguishable from random for this class. In cryptography, the class of algorithms we want to fool is that of "all efficient algorithms" (which should capture all possible adversaries). For convenience, "efficient algorithms" are usually modeled as probabilistic polynomial-time algorithms. However, the results presented in cryptographic

literature are usually more concrete (i.e., they imply statements such as "If there is an algorithm that distinguishes these functions from random in time t and advantage α then there is and algorithm that inverts that function in time t' and success probability α'.") as is the case for the work presented here.

1.1.1 Pseudo-Random (Bit) Generators

Pseudo-random (bit) generators were introduced by Blum and Micali [27] and Yao [150]. A pseudo-random generator is an efficiently computable (deterministic) function such that: (1) Its output is longer than its input and (2) For a uniformly distributed input its output is indistinguishable from uniform to any efficient algorithm. In other words, pseudo-random generators define a pseudo-random distribution of bit-sequences (their output distribution) which is efficiently samplable using a relatively small truly random bit-sequence (usually referred to as the seed).

Unfortunately, the existence of pseudo-random generators is unproven in itself (such a proof would be a tremendous breakthrough in computational complexity, since in particular it implies that $P \neq NP$). However, the existence of pseudo-random generators can be reduced to other computational assumptions. Most notably, Hastad, Impagliazzo, Levin and Luby [71] showed how to construct a pseudo-random generator from *any* one-way function (informally, a function is one-way if it is easy to compute its value on any input but hard to invert it on a random input). In fact, pseudo-random generators as well as pseudo-random functions and permutations exist iff one-way functions exist [62, 71, 73, 86, 90].

As suggested above, a pseudo-random generator can be used for derandomization: A trivial way of derandomizing an algorithm is going over all possible random strings (looking for a "good point" or taking majority over all possible outputs). By using a pseudo-random generator, the number of strings we have to go through can be substantially reduced. Thus, it was shown by Yao [150] that if pseudo-random generators (that fool non-uniform polynomial-size circuits) exist then

$$RP, BPP \subset \bigcap_{\varepsilon>0} DTIME(2^{n^{\varepsilon}}),$$

where RP (resp. BPP) is the class of languages that can be accepted by a probabilistic polynomial-time algorithm with one-sided (resp. two-sided) error. Stronger results in this direction were given by [76, 77, 109]. A different line of work [2, 9, 10, 75, 107, 108, 109, 110, 125], deals with the construction of bit-generators that fool an observer of restricted computational power (e.g. generators against polynomial-size constant-depth circuits) or with bounded space and *read-once access* to the random tape. Most of these constructions need no unproven assumptions.

In our discussion we are mostly interested in the role of pseudo-randomness in cryptography. A representative scenario is the following: Assume that two parties $\mathcal{A}$ and $\mathcal{B}$ share a secret n-bit long random string r and that $\mathcal{A}$ wants to communicate to $\mathcal{B}$ an n-bit long message m in a way that prevents an eavesdropper from learning anything about m. In this case $\mathcal{A}$ can use r as a *one time pad* and send $m \oplus r$ (bit-wise XOR of m and r) to $\mathcal{B}$. From Shannon's work [138] it is known that this solution is optimal in the number of random bits shared by $\mathcal{A}$ and $\mathcal{B}$. Thus, in order to obtain perfect security in the information theoretical

sense, every two parties in a distributed network need to share in advance a random string of length proportional to the total length of the messages that will be communicated between them ever. This is rarely practical, hence $\mathcal{A}$ and $\mathcal{B}$ should consider trading the requirement "impossible to learn (anything about m)" with the weaker requirement "computationally infeasible to learn (anything about m)". This can be achieved using a pseudo-random generator G with output-length $\ell = \ell(n)$ as follows: the encryption of an ℓ-bit message m (that is the value $\mathcal{A}$ sends to $\mathcal{B}$) can be defined to be $E_r(m) = m \oplus G(r)$. For every message m, the encryption $E_r(m)$ is indistinguishable from random. It follows that for any two messages m and m' the encryptions $E_r(m)$ and $E_r(m')$ are indistinguishable from each other. We therefore have a way to encrypt an ℓ-bit message m such that the content of m is concealed while *using less than ℓ bits*. In fact, for any constant $c > 1$, if there exists any pseudo-random generator then there also exists one that outputs n^c bits on an n-bit input. Thus, in the scenario discussed above if $\mathcal{A}$ and $\mathcal{B}$ communicate n^c bits (where n is the security parameter) it is enough for them to share an n-bit truly random secret.

The examples we saw show that pseudo-random generators can save random bits and can reduce the length of keys used in cryptographic settings. We note that there are many other (perhaps more striking) examples where tasks that are impossible in the information theoretical sense have a computational analogue that can be performed using pseudo-random generators. One such example are bit-commitment schemes. Loosely speaking, these are two-phase protocols between $\mathcal{A}$ and $\mathcal{B}$. At the first phase $\mathcal{A}$ commits itself to a bit b and at the second phase $\mathcal{A}$ reveals its commitment. The requirements are that at the end of the first phase (a polynomial-time bounded) $\mathcal{B}$ would not be able to distinguish a commitment to 1 from a commitment to 0 and at the second stage (even an all powerful) $\mathcal{A}$ would not be able to reveal its commitment both as a commitment to 1 and as a commitment to 0. Naor [98] showed how to use the fact that the output of a pseudo-random generator and the uniform distribution are simultaneously computationally indistinguishable and statistically very different in order to obtain a bit commitment scheme.

1.1.2 Pseudo-Random Function Ensembles

As mentioned above, the research presented herein is focused on the construction of *pseudo-random functions* and *permutations*. These are efficient pseudo-random distributions of functions (resp. permutations) that can replace uniformly-chosen functions (resp. permutations) in many applications. They are the key component of private-key cryptography and have many additional applications in cryptography and in computational complexity in general. In this section, we discuss the definitions and original constructions of these powerful cryptographic primitives as well as some of their applications.

Definition

Pseudo-random functions were introduced by Goldreich, Goldwasser and Micali [62]. These are efficient distributions of functions that are indistinguishable from the uniform distribution to an efficient (i.e., polynomial-time bounded) observer. There are two subtleties in the definition of a pseudo-random distribution of *functions*, $F = \{f_s\}$, compared with the definition of a pseudo-random distribution of *bit sequences*. The first issue is the exact meaning

of an "efficient distribution" in this context. Here the answer is rather straightforward: It should be easy to sample a key s from the distribution and given such a key it should be easy to evaluate the corresponding function f_s on any input. The second issue is what kind of access does the distinguisher have to the function. This is a more delicate question:

Assume that $F = \{f_s\}$ is a distribution of functions in B_n, where B_n is the set of all $\{0,1\}^n \mapsto \{0,1\}$ functions (i.e., the set of Binary functions with domain $\{0,1\}^n$). Consider an algorithm $\mathcal{A}$ that tries to distinguish F from the uniform distribution over B_n. The algorithm $\mathcal{A}$ has "some access" to a function $g \in B_n$ which with probability half is sampled from F and with probability half is uniformly distributed. $\mathcal{A}$ distinguishes F from random if it has a non-negligible advantage over half in guessing from which of the distributions g was sampled. The question is *what kind of access should $\mathcal{A}$ have?* In the actual definition of pseudo-random functions (given in [62]) the distinguisher $\mathcal{A}$ has a *black-box access* to g. This means that at each step $\mathcal{A}$ can specify an input x to g and obtain the value $g(x)$. One may be tempted to strengthen the definition by giving the distinguisher additional access to the functions. A first idea might be to give the 2^n-long bit-sequence of all the outputs of g as an input to $\mathcal{A}$. However, this is an exponentially long input which makes $\mathcal{A}$ (which is polynomial in its *input length*) too powerful (for example this might allow $\mathcal{A}$ to verify whether $g \in F$ by going over all possible keys in F). Another idea is to give $\mathcal{A}$ a key s whenever $g = f_s \in F$. However, given a short description, g can no longer be confused to be random.[1] We stress that pseudo-random functions are only guaranteed to look random when the distinguisher *does not get the key of the function.* This means that usually pseudo-random functions cannot be used in a scenario where all the parties (including the "bad" ones) should be able to compute the functions on their own.

It still remains to determine how much freedom the distinguisher $\mathcal{A}$ should have in selecting the values it asks to see. In [62], the distinguisher is has a *adaptive* (black-box) access to g. The meaning of "adaptive" here is that each input x that $\mathcal{A}$ specifies may depend on the value of g on previous queries. This adaptiveness is part of what makes pseudo-random functions so strong. In some cases it makes sense to consider weaker notions. For examples, one might consider a *static* attack where the distinguisher has to specify all its queries in advance. Another interesting example, is a *random* attack where the distinguisher gets the value of g on uniformly distributed inputs. We call such functions weak pseudo-random functions and further consider them in Section 4.1.7. In particular we show a relatively efficient way of constructing full fledged pseudo-random functions from weak pseudo-random functions. We note that pseudo-random functions as well as the weaker notions mentioned above exist iff one-way functions exist [62, 71, 73]. However, weaker notions are still interesting since they may be easier to construct efficiently (more on definitions of function families that are weaker than pseudo-random functions can be found in [104]).

[1] Still it is a worthwhile goal to to come up with interesting definitions for functions that look random *to an algorithm that gets their key* and with distributions of functions that satisfy such definitions (see [34, 36, 37] for some of the recent relevant work).

Applications

An equivalent formulation of pseudo-random functions is as a distribution of exponentially long bit-sequences that is indistinguishable from random to an efficient observer that has direct access to the sequence. This formulation stresses some of the properties that make pseudo-random functions so strong and so easy to work with compared to pseudo-random generators: Sharing or fixing a key of a pseudo-random function is equivalent in many scenarios to sharing or fixing a huge amount of randomness. What makes pseudo-random functions so powerful is that we do not need to know in advance *how many* bits will be used or the *location* of these bits. Indeed pseudo-random functions have numerous applications (a few examples appear in [17, 25, 31, 40, 57, 58, 60, 63, 89, 90, 102]).

Probably, the most notable applications of pseudo-random functions are in private-key cryptography. They provide parties who share a common key straightforward protocols for sending secret messages to each other, for identifying themselves and for authenticating messages [63]. We now describe the basic schemes for performing these three most common tasks of private-key cryptography (more elaborated schemes that achieve better security also exist). Consider a group of parties that share a pseudo-random function f_s. They may perform:

Authentication To authenticate a message m, append the authentication tag $f_s(m)$ to the message.

From the definition of pseudo-random functions we have that this authentication scheme is existentially unforgeable against a chosen message attack: No efficient adversary that adaptively queries for the tags of chosen messages $m_1, m_2, \ldots m_{q-1}$ can produce the tag of any new message m.

Identification A member of the group, $\mathcal{V}$, determines if $\mathcal{A}$ is also a member by issuing a random challenge r and verifying that the response of $\mathcal{A}$ is $f_s(r)$.

No efficient adversary (that is even allowed to participate in executions of the protocol as the verifier) can impersonate a member of the group.

Encryption The encryption of a message m is defined to be $\langle r, f_s(r) \oplus m \rangle$, where r is a uniformly chosen input.

This scheme is semantically secure against an efficient adversary that performs a chosen ciphertext attack in the preprocessing mode. See [13, 51] for the relevant terminology on attacks (chosen plaintext, chosen ciphertext in the preprocessing and postprocessing modes) and notions of security (semantic and non-malleability). Intuitively it means that an adversary that is allowed to ask for encryptions or descriptions and *later* gets an encryption of a new message m, cannot learn anything about m.[2]

[2]For any implementation of f this scheme is *malleable* (i.e. a ciphertext of related message can be generated by an adversary) and hence not secure against a chosen ciphertext attack in the *postprocessing mode* (i.e. when the adversary queries for decryptions *after getting the challenge*). A way to amend this problem was proposed in the full version of [51].

Note that it may be possible to use pseudo-random (bit) generators directly to enable private-key cryptography between *two parties* $\mathcal{A}$ and $\mathcal{B}$. The idea is that at each interaction $\mathcal{A}$ and $\mathcal{B}$ they can use a different part of a pseudo-random sequence they share for their private-key operations (such as authentication, identification and encryption). However, for such an idea to work, care should be taken to synchronize the pointers $\mathcal{A}$ and $\mathcal{B}$ have to the sequence (which might be non-trivial in the presence of an adversary). Even if such synchronization can be achieved for two parties it is impossible in the case of *multi-party private-key cryptography*. In this case pseudo-random functions are indeed essential.

We stress that pseudo-random functions have many additional applications including in public-key cryptography. For example, Bellare and Goldwasser [17] showed how to use pseudo-random functions and a non-interactive zero-knowledge of their values to construct digital-signatures. Another interesting example was given by Goldreich [57] who showed how to eliminate the state in the Goldwasser-Micali-Rivest signature scheme (the technique of [57] is very general). In addition, the existence of pseudo-random functions computable in some complexity class has profound consequences to computational complexity and to computational learning-theory:

- As was observed by Valiant [146], if a concept class contains pseudo-random functions then it cannot be learned: There exists a distribution of concepts, computable in this class, that is hard for *every* efficient learning algorithms, for *every* "non-trivial" distribution on the instances *even* when membership-queries are allowed. Note that such an unlearnability result is very strong (see Section 4.1.9 for more details).

- As shown by Razborov and Rudich [122] if a circuit-class contains pseudo-random functions (that are secure against a subexponential-time adversary), then there are no, what they called, *Natural Proofs* (which include all known lower bound techniques) for separating this class from $P/poly$.

As we discuss in Section 1.2.1, these results imply interesting consequences of our more efficient constructions of pseudo-random functions.

The GGM-Construction

In addition to introducing pseudo-random functions, Goldreich, Goldwasser and Micali [62] have suggested a construction of such functions from pseudo-random generators: Let G be a pseudo-random generator that expands the input by a factor of two (e.g. [24, 70, 74]). Define G^0 and G^1 such that for any n-bit string x, both $G^0(x)$ and $G^1(x)$ are n-bit strings and $G(x) = \langle G^0(x), G^1(x) \rangle$. Under the GGM-Construction, the key of a pseudo-random function $f_s : \{0,1\}^n \mapsto \{0,1\}^n$ is a uniformly chosen n-bit string, s. For any n-bit input, $x = x_1 x_2 \cdots x_n$, the function f_s is defined by:

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

For roughly a decade, this was the only known construction of pseudo-random functions based on standard computational assumptions (including concrete assumptions such as "factoring is hard" as well as general assumptions such as the existence of one-way functions).

Their construction is sequential in nature since computing f_s consists of n *successive invocations* of G. The goal of the research described in Chapter 4 is to present alternative constructions for pseudo-random functions that are more efficient either in the parallel-time complexity or the sequential-time complexity of evaluating f (and preferably in both).

1.1.3 Pseudo-Random Permutation Ensembles

Pseudo-random permutations were introduced by Luby and Rackoff [90]. The main motivation to consider pseudo-random permutations is that they formalize the well established cryptographic notion of block ciphers. These are *length-preserving* private-key encryption schemes: the private key of a block-cipher determines a permutation E such that the encryption of a message m is $E(m)$ and the decryption of a ciphertext c is $E^{-1}(c)$.[3] A highly influential example of a block cipher is the Data Encryption Standard (DES) [106].

When a block cipher E is indistinguishable from a random permutation then the only information that is leaked from a sequence of ciphertexts $\{E(m_1), E(m_2), \ldots, E(m_\ell)\}$ on the corresponding plaintexts is whether or not $m_i = m_j$ for pairs $i \neq j$. Still, when using pseudo-random functions for private-key encryption even this information is concealed. However, using a length-preserving encryption scheme has the advantage that the plaintext and ciphertext are of the same length. This property saves memory and prevents waste of communication bandwidth. Furthermore, it enables the easy incorporation of the encryption scheme into existing protocols or hardware components.

The definition Luby and Rackoff gave to pseudo-random permutation ensembles is closely related to the definition of pseudo-random functions described above: Pseudo-random permutation ensembles are efficient distributions of permutations that are indistinguishable from the uniform distribution to an efficient observer that has *adaptive black-box access* to the permutations (i.e. the distinguisher can only access the permutation by specifying inputs and obtaining the value of the permutation on these inputs). In addition, Luby and Rackoff considered a stronger notion of pseudo-randomness which they call *super pseudo-random permutation generators*. Here the distinguisher also has adaptive black-box access to the *inverse permutation*. Following [60] we use the term *strong pseudo-random permutation ensembles* instead. The two notions of security are analogous to the different attacks considered in the context of block ciphers:

- Pseudo-random permutations can be interpreted as block ciphers that are secure against an adaptive *chosen-plaintext attack*. Informally, this means that an (efficient) adversary, with access to the encryptions of messages of its choice, cannot tell apart those encryptions from the values of a truly random permutation.

- Strong pseudo-random permutations can be interpreted as block ciphers that are secure

[3] Block-ciphers cannot always be directly used to encrypt the entire message since often they have fixed (and relatively small) input length. In such a case, the block-cipher is used in some mode of operation that enables the encryption of longer messages. The standard modes of operation (proposed in the context of DES) are ECB, CBC, CFB and OFB. Unfortunately, all these modes reveal information on the message or on relations between different messages. In Section 5.6, we propose a different way to use block-ciphers that does not have this problem.

against an adaptive *chosen plaintext and ciphertext attack*. Here, the adversary has the additional power to ask for the decryption of ciphertexts of its choice.

The LR-Construction

Luby and Rackoff [90] provided a construction of strong pseudo-random permutations (the LR-Construction) which was motivated by the structure of DES. The basic building block is the so called Feistel permutation based on a pseudo-random function. A Feistel permutation for a function $f : \{0,1\}^n \mapsto \{0,1\}^n$ is a permutation on $\{0,1\}^{2n}$ defined by $\mathbf{D}_f(L,R) \stackrel{\text{def}}{=} (R, L \oplus f(R))$, where $|L| = |R| = n$. Each of the 16 rounds of DES involves a Feistel permutation of a function determined by the 56 key bits. The Luby-Rackoff design of pseudo-random permutations (resp. strong pseudo-random permutations) is $\mathbf{D}_{f_3} \circ \mathbf{D}_{f_2} \circ \mathbf{D}_{f_1}$ (resp. $\mathbf{D}_{f_4} \circ \mathbf{D}_{f_3} \circ \mathbf{D}_{f_2} \circ \mathbf{D}_{f_1}$) where all f_is are independent pseudo-random functions (see Figure 5.1.a for an illustration). This elegant construction inspired a considerable amount of research. The work described in Chapter 5 provides a study of the LR-Construction.

1.2 Overview of Research Objectives and Results

The research included here is composed of two main parts: (1) Several constructions of pseudo-random functions. (2) A study of the LR-Construction and resultant constructions of pseudo-random permutations. In Section 1.1, we have described the importance of pseudo-random functions and permutations in cryptography and part of their applications. We also described the original constructions of these primitives that were given in [62, 90]. The many applications of pseudo-random functions and permutations motivated our search for more efficient and simpler constructions, where efficiency refers both to the sequential and parallel time complexity of the computation. In this section we describe some of our results in that direction as well as some of the consequences of our constructions. In addition, we briefly describe our research of some of the underlying cryptographic assumptions used in our constructions.

1.2.1 Constructions of Pseudo-Random Functions

For our constructions of pseudo-random functions described in Chapter 4, we introduce and study a new cryptographic primitive which we call a *pseudo-random synthesizer* and a generalization of this primitive which we call a *k-dimensional pseudo-random synthesizer*. These primitives are of independent interest as well. In addition, we consider various applications of our constructions and study some of the underlying cryptographic assumptions used in these constructions. Motivated by the inherent sequentiality of the GGM-Construction, we show (in Section 4.1) a *parallel* construction of pseudo-random functions from a pseudo-random synthesizer and parallel constructions of pseudo-random synthesizers based on several concrete and general intractability assumptions. In Section 4.2, we show even more parallel constructions of pseudo-random functions based on *concrete intractability assumptions* (such as the assumption that factoring Blum-integers is hard). In addition, these constructions are

more efficient and have a simple algebraic structure. The constructions of Section 4.2 are motivated by the constructions of Section 4.1 (and in particular by the notion of k-dimensional synthesizers). We now describe the main results of Chapter 4.

Pseudo-Random Synthesizers and Functions

We introduce (in Section 4.1.3) a new cryptographic primitive which we call a *pseudo-random synthesizer*. A pseudo-random synthesizer is a two variable function, $S(\cdot,\cdot)$, so that if many (but polynomially bounded) random assignments, $\langle x_1,\ldots,x_m\rangle$ and $\langle y_1,\ldots,y_m\rangle$, are chosen to both variables, then the output of S on all the combinations of these assignments, $(f(x_i,y_j))_{i,j=1}^{m}$, is indistinguishable from random to a polynomial-time observer. Our main results are:

1. A construction of pseudo-random functions based on pseudo-random synthesizers (described in Sections 4.1.4-4.1.6). Evaluating such a function involves $\log n$ phases, where each phase consists of several parallel invocations of a synthesizer (with a total of n invocations altogether).

2. Constructions of parallel (NC^1) synthesizers based on standard number-theoretic assumptions such as "factoring is hard", RSA (it is hard to extract roots modulo a composite) and Diffie-Hellman (described in Section 4.1.8). In addition, a very simple construction based on a problem from learning (described in Section 4.1.9). The key-generating algorithm of these constructions is sequential for RSA and factoring, non-uniformly parallel for Diffie-Hellman and parallel for the learning problem.

3. An extremely simple, parallel construction of synthesizers based on what we call a weak pseudo-random function (described in Section 4.1.7). A weak pseudo-random function is indistinguishable from a truly random function to a (polynomial-time bounded) observer who gets to see the value of the function on *uniformly distributed inputs* (instead of inputs of its choice). This construction almost immediately implies constructions of synthesizers based on trapdoor one-way permutations and based on any hard-to-learn problem (under the definition of [23]).

Taking (1) and (2) together we get a pseudo-random function that can be evaluated in NC^2.

Our constructions of pseudo-random functions have additional attractive properties. First, it is possible to obtain from the constructions a sharp time-space tradeoff. Loosely speaking, by keeping m strings as the key we can reduce the amount of work for computing the functions from n invocations of the synthesizer to about $\frac{n}{\log m}$ invocations in $\log n - \log\log m$ phases (thus, also reducing the parallel-time complexity). In addition, the construction obtains a nice incremental property. For any y of Hamming distance one from x, given the computation of $f(x)$ we can compute $f(y)$ with only $\log n$ invocations of the synthesizer (we can also make this property hold for $y = x+1$). We discuss both properties in Section 4.1.6.

Concrete Constructions of Pseudo-Random Functions

This part of the work (described in Section 4.2) studies the efficient construction of several fundamental cryptographic primitives. Our major result are two related constructions of pseudo-random functions based on number-theoretic assumptions. The first construction gives pseudo-random functions iff the *decisional* version of the Diffie-Hellman assumption (**DDH-Assumption**) holds. The second construction is at least as secure as the assumption that *factoring* the so called Blum-integers is hard.[4] Having efficient pseudo-random functions based on factoring is very desirable since this is one of the most established concrete intractability assumption used in cryptography. The construction based on the DDH-Assumption is also attractive since these pseudo-random functions are even more efficient (in that they have a larger output size) and since the construction is more security-preserving as described below. We study the DDH-Assumption in Section 3.1. The properties of our new pseudo-random functions are:

Efficiency: Computing the value of the function at a given point is comparable with two modular exponentiations and is more efficient by an $\Omega(n)$ factor than any previous proposal (that is proven to be as secure as some computational intractability assumption). This is essential for the efficiency of the many applications of pseudo-random functions.

Depth: Given appropriate preprocessing of the key, the value of the functions at any given point can be computed in TC^0, compared with TC^1 for the concrete constructions in Section 4.1. Therefore this construction:

1. Achieves reduced latency for computing the functions in parallel and in hardware implementations.
2. Has profound consequences to computational complexity (i.e., Natural Proofs [122]) and to computational learning-theory.

Simplicity: The simple algebraic structure of the functions implies additional desirable features. To demonstrate this, we show (in Section 4.2.3) a simple zero-knowledge proof for the value of the function and other protocols. We suggest the task of designing additional protocols and improving the current ones as an interesting line for further research.

We note that our constructions (both in Section 4.1 and Section 4.2) do not weaken the security of the underlying assumptions. Take for instance the constructions that are based on factoring. If there is an algorithm for breaking this construction in time t and success α (success α means that the observer has advantage of at least α in distinguishing the pseudo-random function from the random one), then there is an algorithm that works in time $poly(t/\alpha)$ and factors Blum-integers with probability $poly(\alpha/t)$. See [72, 89] for a discussion of security preserving reductions. In their terminology, such a reduction is called

[4] In fact we prove the security of the second construction based on a *generalized* version of the computational DH-Assumption (**GDH-Assumption**). However, breaking the GDH-Assumption modulo a composite would imply an efficient algorithm for factorization.

poly-preserving. In fact, most of our reductions are even more secure than that. In particular, the construction of pseudo-random functions based on the DDH-Assumption (described in Section 4.2.1) is *linear-preserving*: If there is an algorithm for breaking this construction in time t and success α, then there is an algorithm that works in time $t \cdot poly(n)$ (where n is the security parameter) and breaks the DDH-Assumption with probability α. This better security is partly due to the fact that this construction does not rely on the hard-core bits of [4, 65] (that are not known to be linear-preserving).

Some Consequences of the Functions

As mentioned above, the original motivation of our constructions was to overcome the inherent sequentiality of the GGM-Construction and to come up with pseudo-random functions that have *shallow depth*. In Section 4.1 we show parallel constructions of pseudo-random functions based on several (general and concrete) intractability assumptions. The functions constructed under concrete intractability assumptions are computable in NC^2 (or actually in TC^1) given appropriate preprocessing of their key. The functions constructed in Section 4.2 are in TC^0 which is the class of functions computable by constant depth circuits consisting of a polynomial number of threshold gates (and therefore also in NC^1). These functions have the additional advantage of being *efficient* and of having a *simple algebraic structure*. We first discuss the motivation for having parallel pseudo-random functions and then the motivation for the additional properties of our functions:

- For some applications of pseudo-random functions minimizing the latency of computing the functions is essential. Such an application is the encryption of messages on a network, where the latency of computing the function is added to the latency of the network. Having shallow-depth pseudo-random functions implies reduced latency in computing those functions in hardware and parallel implementations (note that pseudo-random functions are likely to be implemented in hardware as is the case for DES).

- Many of the applications of pseudo-random functions preserve the parallel-time complexity of the functions. An important example is the LR-Construction of pseudo-random permutations from pseudo-random functions that is further discussed in Chapter 5. Therefore, our constructions yield parallel strong pseudo-random *permutations* as well.

- There is a deep connection between pseudo-random functions and hardness results for learning. Since a random function cannot be learned, if a concept class is strong enough to contain pseudo-random functions we cannot hope to learn it efficiently [146]: There exists a distribution of concepts, computable in this class, that is hard for *every* efficient learning algorithm, for *every* "non-trivial" distribution on the instances *even* when membership-queries are allowed. Therefore, a typical result that can be obtained from our constructions is that *if factoring is hard then TC^0 cannot be learned*. Notice that the unlearnability result implied by the existence of pseudo-random functions is very strong. Since no construction of pseudo-random functions in NC was known, weaker unlearnability results for NC^1 and TC^0, based on cryptographic assumptions, were obtained in [8, 81, 80] (see Section 4.1.9 for more details). It is also interesting

to compare with the result of Linial, Mansour and Nisan [87] who showed that AC^0 *can* be learned in time slightly super-polynomial under the uniform distribution on the examples.

- Another application of pseudo-random functions in complexity was suggested by Razborov and Rudich [122]. They showed that if a circuit-class contains pseudo-random functions (that are secure against a subexponential-time adversary), then there are no, what they called, *Natural Proofs* (which include all known lower bound techniques) for separating this class from $P/poly$. Therefore, a typical result that can be obtained from our construction is that *if factoring cannot be carried out in subexponential-time, then there are no Natural Proofs for separating* TC^0 *from* $P/poly$.

  We note that one can extract a similar result (assuming the hardness of factoring) from the work of Kharitonov [81], which is based on the pseudo-random generator of Blum, Blum and Shub [24].

Except of being more parallelizable, the constructions of pseudo-random functions that are described in Section 4.2 have two additional advantages over previous ones:

1. It is *efficient*: computing the value of the function at any given point is comparable with two exponentiations. This is the first construction that seems efficient enough to be implemented and indeed these functions were implemented by Langberg in [85]. Given the many applications of pseudo-random functions it is clear that having efficient pseudo-random functions is an important goal.

2. It has a *simple* algebraic structure. To see our main motivation here, consider the Bellare-Goldwasser signature scheme. The public key in this scheme contains a commitment for a key, s, of a pseudo-random function. The signature for a message m is composed of a value y and a non-interactive zero-knowledge proof that $y = f_s(m)$. In order for this scheme to be attractive, we must have a simple non-interactive zero-knowledge proof for claims of the form $y = f_s(m)$. In this and other scenarios we might wish to have additional properties for the functions such as a simple function-sharing scheme in the sense of [47]. It seems that for such properties to be possible we need a simple construction of pseudo-random functions.

   In Section 4.2.3, we consider some desirable features of pseudo-random functions. We also present preliminary results in obtaining these features for our construction of pseudo-random functions: (1) A rather simple zero-knowledge proof for claims of the form $y = f_s(m)$ and $y \neq f_s(m)$. (2) A way to distribute a pseudo-random function among a set of parties such that only an authorized subset can compute the value of the function at any given point. (3) A protocol for "oblivious evaluation" of the value of the function: Assume that a party, $\mathcal{A}$, knows a key, s, of a pseudo-random function. Then $\mathcal{A}$ and a second party, $\mathcal{B}$, can perform a protocol during which $\mathcal{B}$ learns exactly one value $f_s(x)$ of its choice whereas $\mathcal{A}$ does not learn a thing (and, in particular, does not learn x). We consider the task of improving these protocols and designing additional ones to be an interesting line for further research.

Related Work

In addition to introducing pseudo-random functions, Goldreich, Goldwasser and Micali [62] have suggested a construction of such functions from pseudo-random generators that expand the input by a factor of two (like the one in [74]). As mentioned above, the GGM construction is sequential in nature. An idea of Levin [86] is to select some secret hash function h and apply the GGM construction to $h(x)$ instead of x. If $|h(x)| = \log^2 n$, then the depth of the GGM-tree is only $\log^2 n$ and presumably we get a pseudo-random function in NC. The problem with this idea is that we have decreased the security significantly: with probability $1/n^{\log n}$ the function can be broken, irrespective of the security guaranteed by the pseudo-random generator. To put this construction in the "correct" light, suppose that for security parameter k we have some problem whose solution requires time 2^k (on instance of length polynomial in k). If we would like to have security $1/2^k$ for our pseudo-random function, then the Levin construction requires depth k whereas our construction requires depth $\log k$.

Impagliazzo and Naor [74] have provided parallel constructions for several other cryptographic primitives based on the hardness of subset sum (and factoring). The primitives include pseudo-random generators that expand the input by a constant factor[5], universal one-way hash functions and strong bit-commitments.

Blum et. al. [23] proposed a way of constructing in parallel several cryptographic primitives based on problems that are hard to learn. We extend their result by showing that hard-to-learn problems can be used to obtain synthesizers and thus pseudo-random functions.

A different line of work [2, 9, 10, 75, 107, 108, 109, 110, 125], more relevant to derandomization and saving random bits, is to construct *bit*-generators such that their output is indistinguishable from a truly random source to an observer of restricted computational power (e.g. generators against polynomial-size constant-depth circuits). Most of these constructions need no unproven assumptions.

It turns out that there are a number of researchers who observed that the average-case DDH-Assumption yields pseudo-random *generators* with good expansion. One such construction was proposed by Rackoff (unpublished). A different construction is suggested by Gertner and Malkin [56]. This construction is similar to the pseudo-random generator one gets by scaling down our pseudo-random functions.

1.2.2 A Study of Some Number-Theoretical Assumptions

The constructions of pseudo-random functions (described in Section 4.2) are based on two number-theoretical assumptions that are related to the Diffie-Hellman key-exchange protocol. The first is the *decisional* version of the Diffie-Hellman assumption (DDH-Assumption) and the second is the *generalized* version of the (computational) Diffie-Hellman assumption (GDH-Assumption). To better understand the security of our constructions we include (in Chapter 3) a study of these assumptions:

- The DDH-Assumption assumption is relatively new, or more accurately, was *explicitly considered* only recently. In Section 3.1 we survey some of the different applications

[5] They also provided a construction of AC^0 pseudo-random generators with small expansion.

of the assumption and the current knowledge on its security. Furthermore, we show a randomized reduction of the worst-case DDH-Assumption to its average case (based on the random-self-reducibility of the DDH-Problem that was previously used by Stadler [143]). We consider our research of the DDH-Assumption to be of independent importance given that the assumption was recently used in quite a few interesting applications (e.g., [45]).

- The GDH-Assumption was originally considered in the context of a generalization of the Diffie-Hellman key-exchange protocol to $k > 2$ parties. In Section 3.2 we prove that breaking this assumption modulo a so called Blum-integer would imply an efficient algorithm for factoring Blum-integers. Therefore, both the generalized key-exchange protocol and our pseudo-random functions (that are based on the GDH-Assumption) are secure as long as factoring Blum-integers is hard. Our reduction strengthen a previous "worst-case" reduction of Shmuely [139].

1.2.3 A Study of the LR-Construction

As described above, in addition to defining strong pseudo-random permutations, Luby and Rackoff [90] have provided a construction of such permutations, (**LR-Construction**) which was motivated by the structure of DES. The basic building block is the so called Feistel permutation[6] based on a pseudo-random function defined by the key. Their construction consists of four rounds of Feistel permutations (or three rounds, for pseudo-random permutations) each round involves an application of a (different) pseudo-random function (see Figure 5.1.a for an illustration).

The part of our research described in Chapter 5 is a study of the LR-Construction. We reduce somewhat the complexity of the construction and simplify its proof of security by showing that two Feistel permutations are sufficient together with initial and final pair-wise independent permutations. Minimizing the number of invocations of pseudo-random functions makes sense since all known constructions of pseudo-random functions involve non-trivial (though of course polynomial-time) computations. The revised construction and proof provide a framework (described in Section 5.3) in which similar constructions may be designed and their security can be easily proved. We demonstrate this by presenting some additional adjustments of the construction.

Alongside cryptographic pseudo-randomness the last two decades saw the development of the notion of limited independence in various setting and formulations [5, 6, 38, 42, 88, 97, 149]. For a family of functions $\mathcal{F}$ to have some sort of (limited) independence means that if we consider the value of a function f, chosen uniformly at random from $\mathcal{F}$, at each point as a random variable (in the probability space defined by choosing f) then these random variables possess the promised independence property. Thus, a family of permutations on $\{0,1\}^n$ is pair-wise independent if for all $x \neq y$ the values of $f(x)$ and $f(y)$ are uniformly distributed over strings $(a,b) \in \{0,1\}^{2n}$ such that $a \neq b$. Functions of limited independence

[6] A Feistel permutation for a function $f : \{0,1\}^n \mapsto \{0,1\}^n$ is a permutation on $\{0,1\}^{2n}$ defined by $\mathbf{D}_f(L,R) \stackrel{\text{def}}{=} (R, L \oplus f(R))$, where $|L| = |R| = n$. Each of the 16 rounds of DES involves a Feistel permutation of a function determined by the 56 key bits.

are typically much simpler to construct and easier to compute than (cryptographic) pseudo-random functions.

New Results

The goal of this research is to provide a better understanding of the LR-Construction and as a result improve the construction in several respects. Our main observation is that the different rounds of the LR-Construction serve significantly different roles. We show that the first and last rounds can be replaced by pair-wise independent permutations and use this in order to:

1. Simplify the proof of security of the construction (especially in the case of strong pseudo-random permutations) and provide a framework for proving the security of similar constructions.

2. Derive generalizations of the construction that are of practical and theoretical interest. The proof of security for each one of the constructions is practically "free of charge" given the framework.

3. Achieve an improvement in the computational complexity of the pseudo-random permutations – two applications of a pseudo-random function on n bits suffice for computing the value of a pseudo-random permutation on $2n$ bits at a given point (vs. four applications in the original LR-Construction). This implies that the reduction is "optimal".

As discussed in Section 5.4.2, the new construction is in fact a generalization of the original LR-Construction. Thus, the proof of security we give (Theorem 5.2.2) also applies to the original construction. We also show how the main construction can be relaxed by: (1) Using a single pseudo-random function (instead of two) and (2) Using weaker and more efficient permutations (or functions) instead of the pair-wise independent permutations. The main generalizations of our construction (described in Sections 5.5-5.6) can briefly be describe as follows:

1. Using t rounds of (generalized) Feistel permutations (instead of two) the success probability of the distinguisher is reduced from approximately $\frac{m^2}{2^{\ell/2}}$ to approximately $\frac{t}{2} \cdot \frac{m^2}{2^{(1-1/t)\ell}}$, where the permutation is on ℓ bits and the distinguisher makes at most m queries (see Figure 5.3 for an illustration).

2. Instead of applying Feistel permutations on the entire outputs of the first and second rounds, Feistel permutations can be separately applied on each one of their sub-blocks. This is a construction of a strong pseudo-random permutation on *many* blocks using pseudo-random functions on a *single* block (see Figure 5.4 for an illustration).

Finally, we analyzes in Section 5.7 the different constructions of this work as constructions of k-wise δ-dependent permutations.

Related Work

The LR-Construction inspired a considerable amount of research. We try to refer to the more relevant (to this work) part of these directions.

Several alternative proofs of the LR-Construction were presented over the years. Maurer [92] gives a proof of the three-round construction. His proof concentrates on the non-adaptive case, i.e., when the distinguisher has to specify all its queries in advance. A point worth noticing is that indistinguishability under non-adaptive attacks does not necessarily imply indistinguishability under adaptive attacks. For example, a random involution (an involution is a permutation which is the inverse of itself) and a random permutation are indistinguishable under non-adaptive attacks and can be distinguished using a very simple adaptive attack.[7] A different approach toward the proof was described by Patarin [114] (this is the only published proof, we are aware of, for the LR-Construction of *strong* pseudo-random permutations; another proof was given by Koren [83]).

Other papers consider the security of possible variants of the construction. A significant portion of this research deals with the construction of pseudo-random permutations and strong pseudo-random permutations from a *single* pseudo-random function. This line of work is described in Section 5.4.1.

Lucks [91] shows that a hash function can replace the pseudo-random function in the first round of the three-round LR-Construction. His proof is based on [92] and is motivated by his suggestion to use the LR-Construction when the input is divided into two *unequal* parts. Lucks left open the question of the construction of strong pseudo-random permutations.

Somewhat different questions were considered by Even and Mansour [53] and by Kilian and Rogaway [82]. Loosely speaking, the former construct several pseudo-random permutations from a single one, while the latter show how to make exhaustive key-search attacks more difficult. The construction itself amounts, in both cases, to XORing the input of the pseudo-random permutation with a random key and XORing the output of the permutation with a second random key. This construction is essentially DESX which was invented by Ron Rivest (see details and references in [82]).

The background and related work concerning other relevant issues are discussed in the appropriate sections herein: Definitions and constructions of efficient hash functions in Section 5.4.2, reducing the distinguishing probability in Section 5.5 and the construction of pseudo-random permutations (or functions) with large input-length from pseudo-random permutations (or functions) with small input-length in Section 5.6.

[7] An even more striking example is obtained by comparing a random permutation P that satisfies $P(P(0)) = 0$ with a truly random permutation.

Chapter 2

Preliminaries

2.1 Notation

We start by describing the notation used in the subsequent chapters. Additional notation will be described in the relevant chapters.

- $\mathbb{N}$ denotes the set of all natural numbers.
- For any integer $N \in \mathbb{N}$ the multiplicative group modulo N is denoted by $\mathbb{Z}_N^*$ and the additive group modulo N is denoted by $\mathbb{Z}_N$.
- For any integer k, denote by $[k]$ the set of integers — $\{1, 2, \ldots, k\}$. For any two integers $k < m$, denote by $[k..m]$ the set of integers — $\{k, k+1, \ldots, m\}$.
- I^n denotes the set of all n-bit strings, $\{0,1\}^n$.
- U_n denotes the random variable uniformly distributed over I^n.
- Let x be any bit-string, we denote by $|x|$ its length (i.e. the number of bits in x). This should not be confused with the usage of $|\cdot|$ to denote absolute values.
- Let x and y be two bit strings of equal length, then $x \oplus y$ denotes their bit-by-bit exclusive-or and $x \odot y$ denotes their inner product mod 2.
- Let x and y be any two bit strings then $x \circ y$ denotes the string x concatenated with y.
- For any two functions f and g such that the range of g is the domain of f denote by $f \circ g$ their composition (i.e., $f \circ g(x) = f(g(x))$).

2.2 Pseudo-Randomness: Definitions

Pseudo-randomness is the main notion studied by this work. In Section 1.1, we have described the importance of pseudo-random functions and permutations in cryptography, part of their applications and their original constructions. In this section we give the definitions

of pseudo-random functions and permutations. A motivating discussion of these definitions also appears in Section 1.1.

We give all definitions in this sections for some sequence of domains $\{A_n, B_n\}_{n\in\mathbb{N}}$. Let F_n be the set of all $A_n \mapsto B_n$ functions. Let P_n be the set of all permutations over P_n. We will often take $A_n = B_n = I^n$ (in particular, we will often concentrate on length-preserving functions). In some places in this work we make the the domains explicit. For example, in some places we use a phrase of the form "an $A_n \mapsto B_n$ function ensemble".

A **function ensemble** is a sequence $H = \{H_n\}_{n\in\mathbb{N}}$ such that H_n is a distribution over F_n. H is the **uniform function ensemble** if H_n is uniformly distributed over F_n. A **permutation ensemble** is a sequence $H = \{H_n\}_{n\in\mathbb{N}}$ such that H_n is a distribution over P_n. H is the **uniform permutation ensemble** if H_n is uniformly distributed over P_n.

A function ensemble (or a permutation ensemble), $H = \{H_n\}_{n\in\mathbb{N}}$, is **efficiently computable** if the distribution H_n can be sampled efficiently and the functions in H_n can be computed efficiently. That is, there exist probabilistic polynomial-time Turing-machines, $\mathcal{I}$ and $\mathcal{V}$, and a mapping from strings to functions, ϕ, such that (1) $\phi(I(1^n))$ and H_n are identically distributed and (2) $\mathcal{V}(i,x) = (\phi(i))(x)$ (so, in fact, $H_n \equiv \mathcal{V}(\mathcal{I}(1^n), \cdot)$). We denote by f_i the function assigned to i (i.e. $f_i \stackrel{\text{def}}{=} \phi(i)$). We refer to i as the **key** of f_i and to $\mathcal{I}$ as the **key-generating algorithm** of F.

We would like to consider efficiently computable function (or permutation) ensembles that cannot be efficiently distinguished from the uniform ensemble. In our setting, the distinguisher is an oracle machine that on input 1^n can make queries to a function in F_n or a permutation (or permutations) in P_n and outputs a single bit. We assume that on input 1^n the oracle machine makes only queries in A_n. Therefore, n serves as the **security parameter**. An oracle machine has an interpretation both under the uniform complexity model and under the non-uniform model. In the former it is interpreted as a Turing-machine with a special oracle-tape (in this case efficient means probabilistic polynomial-time) and in the latter as a circuit-family with special oracle-gates (in this case efficient means polynomial-size). The discussion of this work is independent of the chosen interpretation.

Let M be an oracle machine, let f be a function in F_n and H_n a distribution over F_n. Denote by $M^f(1^n)$ the distribution of M's output when its queries are answered by f and denote by $M^{H_n}(1^n)$ the distribution $M^f(1^n)$, where f is distributed according to H_n. We would also like to consider oracle machines with access both to a permutation and to its inverse. Let M be such a machine, let f be a permutation in P_n and H_n a distribution over P_n. Denote by $M^{f,f^{-1}}(1^n)$ the distribution of M's output when its queries are answered by f and f^{-1} and denote by $M^{H_n,H_n^{-1}}(1^n)$ the distribution $M^{f,f^{-1}}(1^n)$, where f is distributed according to H_n.

Definition 2.2.1 (advantage) *Let M be an oracle machine and let $H = \{H_n\}_{n\in\mathbb{N}}$ and $\tilde{H} = \{\tilde{H}_n\}_{n\in\mathbb{N}}$ be two function (or permutation) ensembles. We call the function*

$$\left|\Pr[M^{H_n}(1^n) = 1] - \Pr[M^{\tilde{H}_n}(1^n) = 1]\right|$$

the advantage *M achieves in distinguishing between H and $\tilde{H}$.*

Let M be an oracle machine and let $H = \{H_n\}_{n\in\mathbb{N}}$ and $\tilde{H} = \{\tilde{H}_n\}_{n\in\mathbb{N}}$ be two permutation ensembles. We call the function

$$\left|\Pr[M^{H_n, H_n^{-1}}(1^n) = 1] - \Pr[M^{\tilde{H}_n, \tilde{H}_n^{-1}}(1^n) = 1]\right|$$

the advantage *M achieves in distinguishing between $\langle H, H^{-1}\rangle$ and $\langle \tilde{H}, \tilde{H}^{-1}\rangle$.*

Definition 2.2.2 (ε-distinguish) *We say that M ε-distinguishes between H and $\tilde{H}$ (resp. $\langle H, H^{-1}\rangle$ and $\langle \tilde{H}, \tilde{H}^{-1}\rangle$) for $\varepsilon = \varepsilon(n)$ if for infinitely many n, the advantage M achieves in distinguishing between H and $\tilde{H}$ (resp. $\langle H, H^{-1}\rangle$ and $\langle \tilde{H}, \tilde{H}^{-1}\rangle$) is at least $\varepsilon(n)$.*

Definition 2.2.3 (negligible functions) *A function $h : \mathbb{N} \mapsto \mathbb{N}$ is* negligible *if for every constant $c > 0$ and all sufficiently large n,*

$$h(n) < \frac{1}{n^c}.$$

Definition 2.2.4 (PFE) *Let $H = \{H_n\}_{n\in\mathbb{N}}$ be an efficiently computable* function *ensemble and let $R = \{R_n\}_{n\in\mathbb{N}}$ be the uniform function ensemble. H is a* pseudo-random function ensemble *if for every efficient oracle-machine M, the advantage M has in distinguishing between H and R is negligible.*

Definition 2.2.5 (PPE) *Let $H = \{H_n\}_{n\in\mathbb{N}}$ be an efficiently computable* permutation *ensemble and let $R = \{R_n\}_{n\in\mathbb{N}}$ be the uniform permutation ensemble. H is a* pseudo-random permutation ensemble *if for every efficient oracle-machine M, the advantage M has in distinguishing between H and R is negligible.*

Definition 2.2.6 (SPPE) *Let $H = \{H_n\}_{n\in\mathbb{N}}$ be an efficiently computable permutation ensemble and let $R = \{R_n\}_{n\in\mathbb{N}}$ be the uniform permutation ensemble. H is a* strong pseudo-random permutation ensemble *if for every efficient oracle-machine M, the advantage M has in distinguishing between $\langle H, H^{-1}\rangle$ and $\langle R, R^{-1}\rangle$ is negligible.*

Remark 2.2.1 *We use the phrase "f is a pseudo-random function" as an abbreviation for "f is distributed according to a pseudo-random function ensemble" and similarly for "f is a pseudo-random permutation" and "f is a strong pseudo-random permutation".*

Remark 2.2.2 *In the definitions above and in the rest of this work, we interpret "efficient computation" as "probabilistic polynomial-time" and "negligible" as "smaller than $1/poly$". This is a rather standard choice and it significantly simplifies the presentation. However, all the proofs in this work easily imply more quantitative results (see e.g. Remark 4.2.1). Moreover, all the reductions of this work are security-preserving in the sense of [72, 89] (as is also discussed in the introduction).*

2.3 k-Wise Independent Functions and Permutations

The notions of k-wise independent functions and k-wise "almost" independent functions [5, 6, 38, 42, 88, 97, 149] (under several different formulations) play a major role in contemporary computer science. These are distributions of functions such that their value on any given k inputs is uniformly or "almost" uniformly distributed. Several constructions of such functions and a large variety of applications have been suggested over the years.

We briefly review the definitions of k-wise independence (and k-wise δ-dependence). The definitions of pair-wise independence (and pair-wise δ-dependence) can be derived by taking $k = 2$.

Definition 2.3.1 *Let D_1 and D_2 be two distributions defined over Ω, the variation distance (or statistical distance) between D_1 and D_2 is*

$$\|D_1 - D_2\| = \frac{1}{2} \sum_{\omega \in \Omega} |D_1(\omega) - D_2(\omega)| .$$

Definition 2.3.2 *Let A and B be two sets, $0 \leq \delta \leq 1$, k an integer ($2 \leq k \leq |A|$) and F a distribution of $A \mapsto B$ functions.*

Let $x_1, x_2, \ldots, x_k$ be k distinct members of A, consider the following two distributions:

1. *$\langle f(x_1), f(x_2), \ldots, f(x_k) \rangle$ where f is distributed according to F.*
2. *The uniform distribution over B^k.*

F is k-wise independent if for all $x_1, x_2, \ldots, x_k$ the two distributions are identical. F is k-wise δ-dependent if for all $x_1, x_2, \ldots, x_k$ the two distributions are of variation distance at most δ.

These definitions are naturally extended to permutations:

Definition 2.3.3 *Let A be a set, $0 \leq \delta \leq 1$, k an integer ($2 \leq k \leq |A|$) and F a distribution of permutations over A.*

Let $x_1, x_2, \ldots, x_k$ be k distinct members of A, consider the following two distributions:

1. *$\langle f(x_1), f(x_2), \ldots, f(x_k) \rangle$ where f is distributed according to F.*
2. *The uniform distribution over sequences of k* distinct *elements of A.*

F is k-wise independent if for all $x_1, x_2, \ldots, x_k$ the two distributions are identical. F is k-wise δ-dependent if for all $x_1, x_2, \ldots, x_k$ the two distributions are of variation distance at most δ.

The connection of this work to k-wise independence is bidirectional as described in the following two paragraphs.

Pair-wise independent permutations are used in several places in this work (e.g., in Section 4.2.1). Pair-wise independent permutations are used in an especially substantial way in Chapter 5. As shown in Section 5.2, pair-wise independent permutations can replace the

first and fourth rounds of the LR-Construction. Let A be a finite field. The permutation $f_{a,b}(x) \stackrel{\text{def}}{=} a \cdot x + b$, where $a \neq 0, b \in A$ are uniformly distributed, is pair-wise independent. Thus, there are pair-wise independent permutations over I^n (the permutations $f_{a,b}$ with operations over $GF(2^n)$). In Section 5.4.2, it is shown that we can use even more efficient functions and permutations in our construction. In particular, we define and consider the concept of ε-AXU$_2$ functions [38, 128].

In contrast with the case of pair-wise independent permutations, we are not aware of any "good" constructions of k-wise δ-dependent permutations for general k and δ. The LR-Construction offers a partial solution to this problem ("partial" because of the bounded value of δ that can be achieved). Using k-wise δ'-dependent functions on n bits instead of pseudo-random functions in the LR-Construction yields a k-wise δ-dependent permutation on $2n$ bits (for $\delta = O(k^2/2^n + \delta')$). In Section 5.7 we analyze the different constructions of Chapter 5 as constructions of k-wise δ-dependent permutations.

Chapter 3

A Study of Some Number-Theoretical Assumptions

The constructions of pseudo-random functions in Section 4.2 are based on two number-theoretical assumptions that are related to the Diffie-Hellman key-exchange protocol. The first is the *decisional* version of the Diffie-Hellman assumption (**DDH-Assumption**) and the second is the *generalized* version of the (computational) Diffie-Hellman assumption (**GDH-Assumption**). To better understand the security of our constructions we include in this section a study of these assumptions.

3.1 The Decisional Diffie-Hellman Assumption

The construction of pseudo-random functions that is described in Section 4.2.1 is based on the DDH-Assumption. This assumption is relatively new, or more accurately, was *explicitly* considered only recently. We therefore devote this section to a discussion of the DDH-Assumption: we describe and define the assumption, consider some of its different applications and the current knowledge on its security. Furthermore, we show in Section 3.1.3 a randomized reduction of the worst-case DDH-Assumption to its average case.

3.1.1 Background

The DH-Assumption was introduced in the context of the Diffie and Hellman [50] key-exchange protocol. Informally, a key-exchange protocol is a way for two parties, $\mathcal{A}$ and $\mathcal{B}$, to agree on a common key, $K_{\mathcal{A},\mathcal{B}}$, while communicating over an insecure (but authenticated) channel. Such a protocol is secure if any efficient third party, $\mathcal{C}$, with access to the communication between $\mathcal{A}$ and $\mathcal{B}$ (but not to their private random strings) cannot tell apart $K_{\mathcal{A},\mathcal{B}}$ from a random value (i.e., $K_{\mathcal{A},\mathcal{B}}$ is pseudo-random to $\mathcal{C}$). This guarantees that it is computationally infeasible for an eavesdropper to gain "any" partial information on $K_{\mathcal{A},\mathcal{B}}$.

Let P be a large prime publicly known. All exponentiations in the rest of this section (Section 3.1) are in $\mathbb{Z}_P^*$. To simplify the exposition, we omit the expression "mod P" from now on. Given a generator g of $\mathbb{Z}_P^*$, the Diffie-Hellman key-exchange protocol goes as follows: $\mathcal{A}$ chooses an integer a uniformly at random in $[P-2]$ and sends g^a to $\mathcal{B}$. In return $\mathcal{B}$ chooses

an integer b uniformly at random in $[P-2]$ and sends g^b to $\mathcal{A}$. Both $\mathcal{A}$ and $\mathcal{B}$ can now compute $g^{a\cdot b}$ and their common key, $K_{\mathcal{A},\mathcal{B}}$, is defined by $g^{a\cdot b}$ in some publicly known manner. For this protocol to be secure we must have, at the minimum, that the standard *computational* version of the Diffie-Hellman assumption (**DH-Assumption**) holds:

Given $\langle g, g^a, g^b\rangle$, it is hard to compute $g^{a\cdot b}$.

The reason is that if this assumption does not hold, then $\mathcal{C}$ (as above) can also compute $K_{\mathcal{A},\mathcal{B}}$.

One method to produce the key, $K_{\mathcal{A},\mathcal{B}}$, is to apply the Goldreich-Levin [65] hard-core function[1] to $g^{a\cdot b}$ (an important improvement on the security of such an application was made by Shoup [140]). If the DH-Assumption holds, then this method indeed gives a pseudo-random key. However, the proof in [65] only implies the pseudo-randomness of the key in case its length is at most logarithmic in the security parameter. A much more ambitious method is to take $g^{a\cdot b}$ itself as the key. For instance, in the ElGamal cryptosystem, given the public key g^a the encryption of a message m is $\langle g^b, g^{a\cdot b}\cdot m\rangle$. The security of the key-exchange protocol now relies on the DDH-Assumption:

Given $\langle g, g^a, g^b, z\rangle$, it is hard to decide whether or not $z = g^{a\cdot b}$.

However, when g is a generator of $\mathbb{Z}_P^*$, we have that g^a and g^b do give some information on $g^{a\cdot b}$. For example, if either g^a or g^b is a quadratic residue, then so is $g^{a\cdot b}$. A standard solution for this problem is to take g to be a generator of the subgroup of $\mathbb{Z}_P^*$ of order Q, where Q is a large prime divisor of $P-1$. In fact, for most applications, using g of order Q is an advantage since Q may be much smaller than P (say, 160 bits long) which results in a substantial improvement in efficiency. The reason that Q may be as small is that all known *subexponential* algorithms for computing the discrete log are *subexponential* in the length of P (as long as $P-1$ is not too smooth) even when applied to the subgroup of size Q generated by g (see, [96, 111] for surveys on algorithms for the discrete log; the best known algorithm for general groups has time square root of the size of the group).

How Much Confidence Can we Have in the DDH-Assumption?

It is clear that the computational DH-Problem is at most as hard as computing the discrete log (given $\langle g, g^a\rangle$ find a). Recent works by Maurer and Wolf [93] and Boneh and Lipton [28] show that in several settings these two problems are in fact equivalent. For example, Maurer and Wolf showed that given some information which only depends on P and an efficient algorithm for computing the DH-Problem in $\mathbb{Z}_P^*$, one can efficiently compute the discrete log in $\mathbb{Z}_P^*$ (so in some nonuniform sense these problems are equivalent). Shoup [140] showed that there are no efficient "generic"-algorithms for computing the discrete log or the DH-Problem, where a "generic"-algorithm is one that does not "exploit" any special properties of the encoding of group elements. A bit more formally, a generic algorithm is one that works for a "black-box" group (where each element has a random encoding and given the encodings of a and b the algorithm can query for the encodings of $a+b$ and $-a$).

[1] For example, to get a key of one bit, we can define $K_{\mathcal{A},\mathcal{B}}$ to be the inner product mod 2 of $g^{a\cdot b}$ and a random string r (chosen by one of the parties and sent to the other over the insecure channel).

Perhaps, the best evidence for the validity of the DH-Assumption is the fact that it endured extensive research over the last two decades. This research does not seem to undermine the (stronger) decisional version of the DH-Assumption as well. In addition, the DDH-Assumption did appear both explicitly and implicitly in several previous works. However, it seems that, given the many applications of the DDH-Assumption, a more extensive study of its security is in place.

To some extent, the DDH-Assumption is supported by the work of Shoup [140] and the work of Boneh and Venkatesan [29]. Shoup showed that the DDH-Problem is hard for any "generic"-algorithm (as above). Boneh and Venkatesan showed that computing the k ($\approx \sqrt{\log P}$) most significant bits of $g^{a \cdot b}$ (given $\langle g, g^a, g^b \rangle$) is as hard as computing $g^{a \cdot b}$ itself. A recent result with applications to the DDH-Assumption was shown by Canetti, Friedlander and Shparlinski [35].

In Section 3.1.3 we prove an attractive feature of the DDH-Assumption: There is quite a simple randomized reduction between its worst-case and its average-case for fixed P and Q. More specifically:

> *For any primes P and Q (such that Q divides $P-1$), the following statements are equivalent:*
>
> - *Given $\langle P, Q, g, g^a, g^b \rangle$, it is easy to distinguish with non-negligible advantage between $g^{a \cdot b}$ and g^c, where g is a uniformly chosen element of order Q in $\mathbb{Z}_P^*$, and a, b and c are uniformly chosen from $\mathbb{Z}_Q$.*
> - *It is easy to decide with overwhelming success probability for $\langle P, Q, g, g^a, g^b, g^c \rangle$ whether or not $c = a \cdot b \bmod Q$, where a, b and c are* any *three elements in $\mathbb{Z}_Q$ and g is* any *element of order Q in $\mathbb{Z}_P^*$.*

This reduction is based on the random-self-reducibility of the DDH-Problem that was previously used by Stadler [143]. The reduction may strengthen our confidence in the DDH-Assumption and in the security of its applications.

For most applications of the DDH-Assumption (including ours) there is no reason to insist on working in a subgroup of $\mathbb{Z}_P^*$ (where P is a prime). Therefore, a natural question is how valid is this assumption for other groups. Specific groups that were considered in the context of the DH-Assumption are: (1) $\mathbb{Z}_N^*$ where N is a composite. McCurley and Shmuely [95, 139] showed that for many of those groups breaking the DH-Assumption is at least as hard as factoring N. (2) Elliptic-curve groups, for which (in some cases) no subexponential algorithms for the discrete log are currently known. We stress that the randomized reduction mentioned above relies on the primality of the order of g.

The Decisional DH-Assumption is Very Attractive

It turns out that the DDH-Assumption was assumed in several previous works (both explicitly and implicitly). In the following, we briefly refer to some of those works and describe some additional applications.

The most obvious application of the DDH-Assumption is to the Diffie-Hellman key-exchange protocol and to the related public-key cryptosystem, namely the ElGamal cryptosystem — given the public key g^a the encryption of a message m is $\langle g^b, g^{a \cdot b} \cdot m \rangle$. Assume

that the message space is restricted to the subgroup generated by g. In this case, it is easy to see that the semantic security (see [68]) of the cryptosystem is equivalent to the DDH-Assumption. In the general case (without the restriction on the message space), we can use the following related cryptosystem: given the public key $\langle g^a, h\rangle$ the encryption of a message m is $\langle g^b, h(g^{a\cdot b}) \oplus m\rangle$, where h is a pair-wise independent hash function from n-bit strings to strings of approximately the length of Q (see Lemma 4.2.2 for more details on the role of h). Therefore, given the DDH-Assumption, we get a probabilistic encryption of many bits for the price of a single (or two) exponentiation. This is comparable with the Blum-Goldwasser cryptosystem [26].

Other applications that previously appeared are:

- Bellare and Micali [19] showed an efficient non-interactive oblivious transfer of many bits that relies on the DDH-Assumption.
- Brands [30] pointed out that several suggestions for undeniable signatures (as the one in [39] where this concept was introduced) implicitly rely on the DDH-Assumption. If this assumption does not hold then such schemes are in fact *digital* signatures.
- Canetti [34] gave a simple construction based on the DDH-Assumption for a new cryptographic primitive called "Oracle Hashing" (later renamed "perfectly one-way probabilistic hash functions"). Loosely, these are hash functions that "hide all partial information" on their input.
- Franklin and Haber [55] showed a construction of a joint encryption scheme based on the the DDH-Assumption modulo a *composite*. Using this scheme they showed how to get an efficient protocol for secure circuit computation.
- Stadler [143] presents verifiable secret sharing based on the DDH-Assumption.
- Steiner, Tsudik and Waidner [144] showed how to extend the Diffie-Hellman protocol to a key-exchange protocol for a group of parties. They reduced the security of the extended protocol to the DDH-Assumption (by showing that the DDH-Assumption implies the *decisional* GDH-Assumption).

A very attractive application of the DDH-Assumption was recently proposed by Cramer and Shoup [45]. They have presented a new public-key cryptosystem that is secure against adaptive chosen ciphertext attacks. Both encryption and decryption in this cryptosystem only require a few exponentiations (in addition to universal one-way hashing).

To all these applications we can add:

- A pseudo-random generator that practically doubles the input length. Essentially, the generator is defined by $G_{P,Q,g,g^a}(b) = \langle g^b, g^{a\cdot b}\rangle$.[2] As mentioned in the introduction, several unpublished constructions of pseudo-random generators based on the DDH-Assumption were previously suggested.

[2] In fact, the output of G_{P,Q,g,g^a} is a pseudo-random pair of values in the subgroup generated by g. In order to get a pseudo-random value in $\{0,1\}^\ell$, for ℓ of approximately twice the length of Q, one needs to hash the output of the generator (see Lemma 4.2.2). A similar observation holds for the constructions of pseudo-random synthesizers and pseudo-random functions.

- A pseudo-random synthesizer (see definition in Section 4.1.3) whose output length is similar to its arguments length, essentially defined by $S_{P,Q,g}(a,b) = g^{a \cdot b}$.

Both these constructions are overshadowed by the construction of pseudo-random functions introduced in Section 4.2.1.

3.1.2 Formal Definition

To formalize the DDH-Assumption, we first need to specify an efficiently samplable distribution for P, Q and g (where g is an element of order Q in $\mathbb{Z}_P^*$).

Let n be the security parameter, for some function $\ell : \mathbb{N} \mapsto \mathbb{N}$ we want to choose an n-bit prime P with an $\ell(n)$-bit prime Q that divides $P-1$. A natural way to do this is to choose P and Q uniformly at random subject to those constraints. However, it is possible to consider different distributions. For example, it is not inconceivable that the assumption holds when for every n we have a *single* possible choice of P, Q and g. Another common example is letting P and Q satisfy $P = 2 \cdot Q + 1$ (although choosing a smaller Q may increase the efficiency of most applications). In order to keep our results general, we let P, Q and g be generated by *some* polynomial-time algorithm IG (where IG stands for instance generator).

Definition 3.1.1 (IG) *The Diffie-Hellman instance generator, IG, is a probabilistic polynomial-time algorithm such that on input 1^n the output of IG is distributed over triplets $\langle P, Q, g \rangle$, where P is an n-bit prime, Q a (large) prime divisor of $P-1$ and g an element of order Q in $\mathbb{Z}_P^*$.*

For the various applications of the DDH-Assumption we need its average-case version. Namely, when a and b are uniformly chosen and c is either $a \cdot b$ or uniformly chosen. In Section 3.1.3 it is shown that a worst-case choice of a, b and c can be reduced to a uniform choice. Similarly, the assumption is not strengthened if g (generated by IG) is taken to be a uniformly chosen element of order Q in $\mathbb{Z}_P^*$.

Assumption 3.1.1 (Decisional Diffie-Hellman) *For every probabilistic polynomial-time algorithm $\mathcal{A}$, every constant $\alpha > 0$ and all sufficiently large n,*

$$\left| \Pr[\mathcal{A}(P, Q, g, g^a, g^b, g^{a \cdot b}) = 1] - \Pr[\mathcal{A}(P, Q, g, g^a, g^b, g^c) = 1] \right| < \frac{1}{n^\alpha},$$

where the probabilities are taken over the random bits of $\mathcal{A}$, the choice of $\langle P, Q, g \rangle$ according to the distribution $IG(1^n)$ and the choice of a, b and c uniformly *at random in $\mathbb{Z}_Q$.*

3.1.3 A Randomized Reduction

In this subsection we use a simple randomized reduction to show that for every P, Q and g the DDH-Problem is either very hard on the average or very easy in the worst-case. Given the current knowledge of the DDH-Problem, such a result strengthens our belief in the DDH-Assumption. The main part of the reduction (Lemma 3.1.2) was previously used by Stadler [143].

Definition 3.1.2 *For any $\langle P, Q, g\rangle$ such that P is a prime, Q a prime divisor of $P-1$ and g an element of order Q in $\mathbb{Z}_P^*$ the function $DDH_{\langle P,Q,g\rangle}$ is defined by*

$$DDH_{P,Q,g}(g^a, g^b, g^c) = \begin{cases} 1 & \text{if } c = a \cdot b \bmod Q \\ 0 & \text{otherwise} \end{cases}$$

for any three elements a, b, c in $\mathbb{Z}_Q$.

Theorem 3.1.1 *Let $\mathcal{A}$ be any probabilistic algorithm with running time $t = t(n)$ and $\varepsilon = \varepsilon(n)$ any positive function such that $1/\varepsilon$ is efficiently constructible. There exists a polynomial $p = p(n)$ and a probabilistic algorithm $\mathcal{A}'$ with running time $(t(n) \cdot p(n))/(\varepsilon(n))^2$ such that for* any *choice of $\langle P, Q, g\rangle$ as in Definition 3.1.2 if:*

$$\left| \Pr[\mathcal{A}(P, Q, g, g^a, g^b, g^{a\cdot b}) = 1] - \Pr[\mathcal{A}(P, Q, g, g^a, g^b, g^c) = 1] \right| > \varepsilon(n),$$

where the probabilities are taken over the random bits of $\mathcal{A}$ and the choice of a, b and c uniformly *at random in $\mathbb{Z}_Q$, then for* any *a, b and c in $\mathbb{Z}_Q$:*

$$\Pr[\mathcal{A}'(P, Q, g, g^a, g^b, g^c) \neq DDH_{P,Q,g}(g^a, g^b, g^c)]| < 2^{-n},$$

where the probability is only *over the random bits of $\mathcal{A}'$.*

In particular, if $\mathcal{A}$ is probabilistic polynomial-time and $\varepsilon(n) \geq 1/poly(n)$, then $\mathcal{A}'$ is also probabilistic polynomial-time.

Blum and Micali [27] introduced the concept of random-self-reducibility (and randomized reductions). Informally, a problem is random-self-reducible if solving the problem on *any* instance x can be efficiently reduced to solving the problem on a random instance y (or on polynomial number of random instances). I.e., for any instance x, a random instance y can be efficiently sampled using a random string r such that given r and the solution of the problem on y it is easy to compute the solution of the problem on x. A problem that is random-self-reducible can either be efficiently solved for every instance with overwhelming success probability or it cannot be solved for a random instance with non-negligible success probability.

Our randomized reduction is closely related to other known reductions. Blum and Micali [27] showed that for any specific prime P and generator g, the discrete log problem is random-self-reducible: given $\langle P, g, g^a\rangle$ for *any* a it is easy to generate a random instance $\langle P, g, g^{a+r} = g^a \cdot g^r\rangle$ (where r is uniform in $[P-1]$). Given the solution for the random instance (i.e., $a + r$) it is easy to compute the solution for the original instance (i.e., a). A similar property was shown for the DH-Problem (e.g. [93]): given $\langle P, g, g^a, g^b\rangle$ for *any* a and b it is easy to generate a random instance $\langle P, g, g^{a+r}, g^{b+s}\rangle$ (where r and s are uniform in $[P-1]$). Given the solution for the random instance (i.e., $z = g^{(a+r)\cdot(b+s)}$) it is easy to compute the solution for the original instance (i.e., $g^{a\cdot b} = z \cdot (g^a)^{-s} \cdot (g^b)^{-r} \cdot g^{-s\cdot r}$).

However, in order to prove Theorem 3.1.1, we need a somewhat different reduction. In particular, we need to use the fact that g is an element of prime order (Theorem 3.1.1 is not true when g is a generator of $\mathbb{Z}_P^*$).

Lemma 3.1.2 *There exists a probabilistic polynomial-time algorithm, $\mathcal{R}$ such that on* any *input*

$$\langle P, Q, g, g^a, g^b, g^c \rangle,$$

where P is a prime, Q a prime divisor of $P-1$, g an element of order Q in $\mathbb{Z}_P^*$ and a, b, c are three elements in $\mathbb{Z}_Q$ the output of $\mathcal{R}$ is:

$$\langle P, Q, g, g^{a'}, g^{b'}, g^{c'} \rangle,$$

where if $c = a \cdot b \bmod Q$, then a' and b' are uniform in $\mathbb{Z}_Q$ and $c' = a' \cdot b' \bmod Q$ and if $c \neq a \cdot b \bmod Q$, then a', b' and c' are all *uniform in $\mathbb{Z}_Q$.*

Proof. $\mathcal{R}$ chooses s_1, s_2 and r uniformly in $\mathbb{Z}_Q$, computes

$$\begin{aligned} g^{a'} &= (g^a)^r \cdot g^{s_1}, \\ g^{b'} &= g^b \cdot g^{s_2}, \\ g^{c'} &= (g^c)^r \cdot (g^a)^{r \cdot s_2} \cdot (g^b)^{s_1} \cdot g^{s_1 \cdot s_2} \end{aligned}$$

and outputs

$$\langle P, Q, g, g^{a'}, g^{b'}, g^{c'} \rangle.$$

Let $c = a \cdot b + e \bmod Q$ for e in $\mathbb{Z}_Q$ then:

$$a' = r \cdot a + s_1 \bmod Q, \ b' = b + s_2 \bmod Q, \ c' = a'b' + e \cdot r \bmod Q.$$

If $e = 0$ we get that a' and b' are uniformly distributed in $\mathbb{Z}_Q$ and $c' = a' \cdot b' \bmod Q$. If $e \neq 0$ we get that a', b' and c' are all uniformly distributed in $\mathbb{Z}_Q$ (this is the place we use the fact that Q is a prime which implies that $e \cdot r \bmod Q$ is uniformly distributed in $\mathbb{Z}_Q$). Therefore, the output of $\mathcal{R}$ has the desired distribution. □

Proof. (of Theorem 3.1.1) Let $\mathcal{A}$ be any probabilistic algorithm with running time $t = t(n)$, let $\varepsilon = \varepsilon(n)$ be any positive function such that $1/\varepsilon$ is efficiently constructible and let $\langle P, Q, g \rangle$ be as in Definition 3.1.2. Assume that:

$$\left| \Pr[\mathcal{A}(P, Q, g, g^a, g^b, g^{a \cdot b}) = 1] - \Pr[\mathcal{A}(P, Q, g, g^a, g^b, g^c) = 1] \right| > \varepsilon(n),$$

where the probabilities are taken over the random bits of $\mathcal{A}$ and the choice of a, b and c *uniformly* at random in $\mathbb{Z}_Q$.

Let $\mathcal{R}$ be the probabilistic polynomial-time algorithm that is guaranteed to exist by Lemma 3.1.2. By the definition of $\mathcal{R}$ and our assumption, we get that for *any* a, b and $c \neq a \cdot b \bmod Q$ in $\mathbb{Z}_Q$:

$$\left| \Pr[\mathcal{A}(\mathcal{R}(P, Q, g, g^a, g^b, g^{a \cdot b})) = 1] - \Pr[\mathcal{A}(\mathcal{R}(P, Q, g, g^a, g^b, g^c)) = 1] \right| > \varepsilon(n).$$

Now the probabilities are *only* taken over the random bits of $\mathcal{A}$ and $\mathcal{R}$. Therefore, by standard methods of amplification we can define a probabilistic algorithm $\mathcal{A}'$ such that for *any* a, b and $c \neq a \cdot b \bmod Q$ in $\mathbb{Z}_Q$:

$$\Pr[\mathcal{A}'(P, Q, g, g^a, g^b, g^{a \cdot b}) = 1] - \Pr[\mathcal{A}'(P, Q, g, g^a, g^b, g^c) = 1] > 1 - 2^{-n}.$$

On any input $\langle P, Q, g, g^a, g^b, g^c \rangle$, the output of $\mathcal{A}'$ is essentially a threshold function of $O(n/(\varepsilon(n))^2)$ independent values — $\mathcal{A}(\mathcal{R}(P, Q, g, g^a, g^b, g^c))$. It is clear that $\mathcal{A}'$ satisfies the conditions required in Theorem 3.1.1. □

3.2 The GDH-Assumption Modulo a Composite

The Generalized Diffie-Hellman Assumption (**GDH-Assumption**) was originally considered in the context of a key-exchange protocol for $k > 2$ parties (see e.g., [139, 144]). This protocol is an extension of the (extremely influential) Diffie-Hellman key-exchange protocol [50]. Given a group G and an element $g \in G$, the high-level structure of the protocol is as follows: Party $i \in [k] \stackrel{\text{def}}{=} \{1, 2, \ldots, k\}$ chooses a secret value, a_i. The parties exchange messages of the form $g^{\prod_{i \in I} a_i}$ for several proper subsets, $I \subsetneq [k]$. Given these messages, each of the parties can compute $g^{\prod_{i \in [k]} a_i}$ and this value defines their common key (in some publicly known way). Since the parties use an insecure (though authenticated) channel, it is essential that the messages they exchange do not reveal $g^{\prod_{i \in [k]} a_i}$. The GDH-Assumption is even stronger: informally, it states that it is hard to compute $g^{\prod_{i \in [k]} a_i}$ for an algorithm that can query $g^{\prod_{i \in I} a_i}$ for *any* proper subset, $I \subsetneq [k]$ of its choice. The precise statement of the assumption is given in Section 2.1.

In Section 4.2.2, we propose another application to the GDH-Assumption. We show an attractive construction of pseudo-random functions that is secure as long as the GDH-Assumption holds. Motivated by this application, we[3] provide in this section a proof that the GDH-Assumption modulo a Blum-integer follows from the assumption that factoring Blum-integers is hard. Similar reductions were previously described in the context of the standard Diffie-Hellman assumption by McCurley [95] and Shmuely [139]. In fact, Shmuely [139] also provided a related reduction for the GDH-Assumption (modulo a composite) itself. Her reduction works when the algorithm that breaks the GDH-Assumption succeeds in computing $g^{\prod_{i \in [k]} a_i}$ for *every* choice of values $\langle a_1, a_2, \ldots, a_k \rangle$ (which is not sufficient for the applications of the GDH-Assumption). In contrast, our reduction works even when the algorithm breaking the GDH-Assumption only succeeds for some non-negligible fraction of the $\langle a_1, \ldots, a_k \rangle$.

3.2.1 The Assumptions

In this section we define the GDH-Assumption in $\mathbb{Z}_N^*$ (the multiplicative group modulo N), where N is a so called Blum-integer. We also define the assumption that factoring Blum-integers is hard. The restriction to Blum-integers is quite standard and it makes the reduction of factoring to the GDH-Problem much simpler. We therefore start by defining Blum-integers and describing some of their properties

Blum-integers: An integer N is a Blum-integer if $N = P \cdot Q$ where P and Q are two distinct primes of the same length, and both P and Q are congruent to 3 mod 4 (i.e. $P = Q = 3 \bmod 4$). A very helpful property of a Blum-integer N is that squaring is a permutation over the subgroup of *quadratic residues* in $\mathbb{Z}_N^*$ (an element $x \in \mathbb{Z}_N^*$ is a quadratic residue if there exist a $y \in \mathbb{Z}_N^*$ such that $y^2 = x \bmod N$). An important application (along with a proof) of this property was given by Blum, Blum and Shub [24]. One way to verify this property is by considering the Chinese-remainder representation of a quadratic residue x. In $\mathbb{Z}_P^*$ (as well as $\mathbb{Z}_Q^*$), x has exactly two distinct square roots $\pm y$. Since -1 is *not* a quadratic

[3] The work described here is joint with Eli Biham and Dan Boneh.

residue in $\mathbb{Z}_P^*$, exactly one of the square roots $\pm y$ is a quadratic residue in itself. Therefore, x has exactly four our distinct square roots, $\pm y, \pm y'$ in $\mathbb{Z}_N^*$, from which exactly one is a quadratic residue in itself. Another property of a Blum-integer N that we use in our proof is that the order of any quadratic-residue x in $\mathbb{Z}_N^*$ is odd.

In order to keep our result general, we let N (in both assumptions) be generated by *some* polynomial-time algorithm FIG (where FIG stands for factoring-instance-generator):

Definition 3.2.1 (FIG) *The factoring-instance-generator, FIG, is a probabilistic polynomial-time algorithm such that on input 1^n its output, $N = P \cdot Q$, is distributed over $2n-bit$ integers, where P and Q are two $n-bit$ primes and $P = Q = 3 \bmod 4$ (such N is known as a Blum-integer).*

A natural way to define FIG is to let $FIG(1^n)$ be uniformly distributed over $2n-bit$ Blum-integers[4]. However, other choices were previously considered (e.g., letting P and Q obey some "safety" conditions).

The GDH-Assumption

To formalize the GDH-Assumption (which is described in the introduction) we use the following two definitions:

Definition 3.2.2 *Let N be any possible output of $FIG(1^n)$, let g be any quadratic-residue in $\mathbb{Z}_N^*$ and let $\vec{a} = \langle a_1, a_2, \ldots, a_k \rangle$ be any sequence of $k \geq 2$ elements of $[N]$. Define the function $h_{N,g,\vec{a}}$ with domain $\{0,1\}^k$ such that for any k-bit input, $x = x_1 x_2 \cdots x_k$,*

$$h_{N,g,\vec{a}}(x) \stackrel{\text{def}}{=} g^{\prod_{x_i=1} a_i} \bmod N.$$

Define $h^r_{N,g,\vec{a}}$ to be the restriction of $h_{N,g,\vec{a}}$ to the set of all k-bit strings except 1^k (i.e., the restriction of $h_{N,g,\vec{a}}$ to $\{0,1\}^k \setminus \{1^k\}$).

Definition 3.2.3 (ε-solving the GDH$_k$-Problem) *Let $\mathcal{A}$ be a probabilistic oracle machine, $k = k(n)$ an integer-valued function such that $\forall n, k(n) \geq 2$ and $\varepsilon = \varepsilon(n)$ a real-valued function. $\mathcal{A}$ ε-solves the GDH$_k$-Problem if for infinitely many n,*

$$\Pr[\mathcal{A}^{h^r_{N,g,\vec{a}}}(N,g) = h_{N,g,\vec{a}}(1^n)] > \varepsilon(n),$$

where the probability is taken over the random bits of $\mathcal{A}$, the choice of N according to the distribution $FIG(1^n)$, the choice of g uniformly at random in the set of quadratic-residues in $\mathbb{Z}_N^*$ and the choice of each of the values in $\vec{a} = \langle a_1, a_2, \ldots, a_{k(n)} \rangle$ uniformly at random in $[N]$.

Informally, the GDH-Assumption is that there is no "efficient" oracle machine $\mathcal{A}$ that ε-solves the GDH$_k$-Problem for "non-negligible" ε. We formalize this in the standard way of interpreting "efficient" as "probabilistic polynomial-time" and "non-negligible" as "larger than $1/poly$". However, our reduction (Theorem 3.2.1) is more quantitative.

[4]We note that $2n-bit$ Blum-integers are a non-negligible fraction of all $2n-bit$ integers and that it is easy to sample a uniformly distributed $2n-bit$ Blum-integer.

Assumption 3.2.1 (The GDH-Assumption Modulo a Blum-Integer) *Let $\mathcal{A}$ be any probabilistic polynomial-time oracle machine and $k = k(n) \geq 2$ any integer-valued function that is bounded by a polynomial (and is efficiently-constructible). There is no positive constant α such that $\mathcal{A}$ $\frac{1}{n^\alpha}$-solves the GDH_k-Problem.*

The Factoring-Assumption

We formalize the assumption that factoring Blum-integers is hard in an analogous way:

Definition 3.2.4 (ε-factoring) *Let $\mathcal{A}$ be a probabilistic Turing-machine and $\varepsilon = \varepsilon(n)$ a real-valued function. $\mathcal{A}$ ε-factors if for infinitely many n,*

$$\Pr[\mathcal{A}(P \cdot Q) \in \{P, Q\}] > \varepsilon(n),$$

where the distribution of $N = P \cdot Q$ is $FIG(1^n)$.

Assumption 3.2.2 (Factoring Blum-Integers) *Let $\mathcal{A}$ be any probabilistic polynomial-time oracle machine. There is no positive constant α such that $\mathcal{A}$ $\frac{1}{n^\alpha}$-factors.*

Reducing Factoring to the GDH-Problem

Theorem 3.2.1 *Assumption 3.2.1 (the GDH-Assumption) is implied by Assumption 3.2.2 (Factoring). Furthermore, assume that $\mathcal{A}$ is a probabilistic oracle machine with running-time $t = t(n)$ such that $\mathcal{A}$ ε-solves the GDH_k-Problem (where $k = k(n) \geq 2$, is an integer-valued function that is efficiently-constructible and $\varepsilon = \varepsilon(n)$ a real-valued function). Then there exists a probabilistic Turing-machine $\mathcal{A}'$ with running time $t'(n) = poly(n, k(n)) \cdot t(n)$ that ε'-factors, where $\varepsilon'(n) = \varepsilon(n)/2 - O(k(n) \cdot 2^{-n})$.*

As an intuition to the proof, let us first describe it under the assumption that $\mathcal{A}$ computes $h_{N,g,\vec{a}}(1^k)$ for *any* sequence $\vec{a} = \langle a_1, a_2, \ldots, a_{k(n)} \rangle$. The algorithm $\mathcal{A}'$ can use $\mathcal{A}$ to extract square-roots in $\mathbb{Z}_N^*$ and consequently $\mathcal{A}'$ can factor N (as shown in [120]). This is done as follows:

$\mathcal{A}'$ first samples v uniformly at random in $\mathbb{Z}_N^*$ and computes $g = v^{2^k} \bmod N$. Let ℓ be the order of g in $\mathbb{Z}_N^*$ (note that ℓ is not known to $\mathcal{A}'$). Since N is a Blum-integer and g is a quadratic-residue we have that ℓ is odd. This implies that $2 \in \mathbb{Z}_\ell^*$ and therefore $2^{-1} \bmod \ell$ exists (and is simply $\frac{\ell+1}{2}$). For simplicity of exposition, denote by $g^{2^{-1}} \bmod N$ the value $g^{2^{-1} \bmod \ell} \bmod N$. Hence, $g^{2^{-1}} \bmod N$ is not just *any* square-root of g in $\mathbb{Z}_N^*$ but is rather precisely equal to $g^{\frac{\ell+1}{2}} \bmod N$. In the same way denote by $g^{2^{-i}} \bmod N$ the value $g^{2^{-i} \bmod \ell} \bmod N$. Under this notation we have for every $0 < i < k$ that $g^{2^{-i}} = v^{2^{k-i}} \bmod N$.

Let $\vec{a} = \langle a_1, \ldots, a_k \rangle$ be the vector, where for all i, $a_i = 2^{-1} \bmod \ell$. It follows that algorithm $\mathcal{A}'$ can easily compute $h_{N,g,\vec{a}}(x)$ for any $x \neq 1^k$ (if exactly $i < k$ bits of x are 1 we have that $h_{N,g,\vec{a}}(x) = g^{2^{-i}} \bmod N = v^{2^{k-i}} \bmod N$). Hence, $\mathcal{A}'$ can invoke $\mathcal{A}$ with input $\langle N, g \rangle$ and answer every query, q, of $\mathcal{A}$ with $h^r_{N,g,\vec{a}}(q)$. Eventually, $\mathcal{A}$ outputs the value $u = h_{N,g,\vec{a}}(1^n) = g^{2^{-k}} \bmod N$. We now have that $u^2 = v^2 \bmod N$ and that $\Pr[u = \pm v] = 1/2$. This implies that $\Pr[gcd(u - v, N) \in \{P, Q\}] = 1/2$ which enables $\mathcal{A}'$ to factor N. The complete proof follows the same lines along with additional "randomization" of the a_i's (achieved by taking $a_i = 2^{-1} + r_i$) which eliminates the assumption that $\mathcal{A}$ always succeeds.

Proof. Assume that $\mathcal{A}$ is as in Theorem 3.2.1, we define the algorithm $\mathcal{A}'$ that is guaranteed to exist by the theorem. Let $N = P \cdot Q$ be any $2n$-bit Blum-integer. Given N as its input, $\mathcal{A}'$ performs the following basic steps (we later describe how these steps can be carried out in the required running time):

1. Sample v uniformly at random in $\mathbb{Z}_N^*$. Compute $k = k(n)$ and $g = v^{2^k} \bmod N$. Denote by ℓ the order of g in $\mathbb{Z}_N^*$ (note that ℓ is not known to $\mathcal{A}'$).

   As mentioned above, since N is a Blum-integer and g is a quadratic-residue we have that ℓ is odd. This implies that $2 \in \mathbb{Z}_\ell^*$ and therefore $2^{-1} \bmod \ell$ exists (and is simply $\frac{\ell+1}{2}$).

2. Sample each one of the values in $\langle r_1, r_2, \ldots, r_k \rangle$ uniformly at random in $[N]$. For $1 \leq i \leq k$, denote by a_i the value $r_i + 2^{-1} \bmod \ell$ (again, note that a_i is not known to $\mathcal{A}'$). Denote by $\vec{a}$ the sequence $\langle a_1, a_2, \ldots, a_k \rangle$.

3. Invoke $\mathcal{A}$ with input $\langle N, g \rangle$ and answer each query, q, of $\mathcal{A}$ with the value $h^r_{N,g,\vec{a}}(q)$.

4. Given that $\mathcal{A}$ outputs the correct value — $h_{N,g,\vec{a}}(1^n)$, compute $u = g^{2^{-k}} \bmod N$. As noted below, $u^2 = v^2 \bmod N$. If $u \neq \pm v \bmod N$, output $gcd(u - v, N)$ which is indeed in $\{P, Q\}$. Otherwise, output 'failed'.

The Running-Time of $\mathcal{A}'$:
Steps (1) and (2) can easily be carried out in time $poly(n, k(n))$. For steps (3) and (4) to be carried out in time $t'(n) = poly(n, k(n)) \cdot t(n)$ it is enough to show that:

a. For every query $q \in \{0,1\}^k \setminus \{1^k\}$ the value $h_{N,g,\vec{a}}(q)$ can be computed in time $poly(n, k(n))$.

b. Given $h_{N,g,\vec{a}}(1^n)$, the value $g^{2^{-k}} \bmod N$ can be computed in time $poly(n, k(n))$.

Recall that whenever 2^{-1} appears in the exponent it denotes the value $2^{-1} \bmod \ell = \frac{\ell+1}{2}$. Similarly, 2^{-i} in the exponent denotes the value $2^{-i} \bmod \ell = \left(\frac{\ell+1}{2}\right)^i \bmod \ell$. Therefore, under this notation, for every i the value $g^{2^{-i}} \bmod N$ is a quadratic-residue (since g is a quadratic-residue). The key-observation for proving (a) and (b) is that for all $0 < i < k$, we have that $g^{2^{-i}} = v^{2^{k-i}} \bmod N$. For $i = 1$, this is implied by the fact that both $g^{2^{-1}}$ and $v^{2^{k-1}}$ are square roots of g and they are both quadratic-residues. Since squaring is a permutation over the set of quadratic-residues in $\mathbb{Z}_N^*$ (for any Blum-integer, N) we must have that $g^{2^{-1}}$ and $v^{2^{k-1}}$ are equal. By induction on $0 < i < k$, we get that $g^{2^{-i}} = v^{2^{k-i}} \bmod N$ in the same way. Therefore, for every $q = q_1 q_2 \ldots q_k \neq 1^k$:

$$h_{N,g,\vec{a}}(q) = g^{\prod_{q_i=1} a_i} = g^{\prod_{q_i=1}(r_i + 2^{-1})} = g^{\sum_{j=0}^{k-1} \alpha_j 2^{-j}} = v^{\sum_{j=0}^{k-1} \alpha_j 2^{k-j}} \bmod N,$$

where the values $\{\alpha_j\}_{j=0}^{k-1}$ are the integers for which the two polynomials (in the variable x) $\prod_{q_i=1}(r_i + x)$ and $\sum_{j=0}^{k-1} \alpha_j x^j$ are identical over $\mathbb{Z}$. Since these values can easily be computed in time $poly(n, k(n))$, we get that (a) holds. Similarly:

$$h_{N,g,\vec{a}}(1^k) = g^{\prod_{i=1}^{k} a_i} = g^{\prod_{i=1}^{k}(r_i + 2^{-1})} = g^{2^{-k}} \cdot g^{\sum_{j=0}^{k-1} \beta_j 2^{-j}} = g^{2^{-k}} \cdot v^{\sum_{j=0}^{k-1} \beta_j 2^{k-j}} \bmod N,$$

where the values $\{\beta_j\}_{j=0}^{k-1}$ can easily be computed in time $poly(n, k(n))$. We now get that (b) holds since:

$$g^{2^{-k}} = h_{N,g,\vec{a}}(1^k) \cdot \left(v^{\sum_{j=0}^{k-1} \beta_j 2^{k-j}}\right)^{-1} \bmod N.$$

The Success-Probability of $\mathcal{A}'$:
It remains to show that $\mathcal{A}'$ ε'-factors, where $\varepsilon'(n) = \varepsilon(n)/2 - O(k(n) \cdot 2^{-n})$. Recall that u denotes the value $g^{2^{-k}} \bmod N$. As shown above $v^2 = g^{2^{-(k-1)}} (= u^2) \bmod N$. Therefore, it is not hard to verify that:

$$\Pr[\mathcal{A}'(N) \in \{P, Q\}] = \Pr\left[(u \neq \pm v \bmod N) \text{ and } \left(\mathcal{A}^{h^r_{N,g,\vec{a}}}(N, g) = h_{N,g,\vec{a}}(1^n)\right)\right].$$

Note that $\mathcal{A}^{h^r_{N,g,\vec{a}}}(N, g)$ does not depend on v itself but rather on v^2. Therefore, $\mathcal{A}^{h^r_{N,g,\vec{a}}}(N, g)$ is equally distributed for any two assignments, w and $\bar{w}$, of v as long as $w^2 = \bar{w}^2 \bmod N$. This follows from the fact that all the values of $h^r_{N,g,\vec{a}}$ (including $g = h^r_{N,g,\vec{a}}(0^n)$) only depend on v^2: For every $q = q_1 q_2 \ldots q_k \neq 1^k$ the value $h_{N,g,\vec{a}}(q)$ was shown above to be

$$v^{\sum_{j=0}^{k-1} \alpha_j 2^{k-j}} \bmod N = \left(v^2\right)^{\sum_{j=1}^{k} \alpha_{j-1} 2^{k-j}} \bmod N,$$

where the α_j's only depend on q and the r_i's. We therefore get that:

$$\Pr[\mathcal{A}'(N) \in \{P, Q\}] = 1/2 \cdot \Pr[\mathcal{A}^{h^r_{N,g,\vec{a}}}(N, g) = h_{N,g,\vec{a}}(1^n)].$$

Let N be chosen from $FIG(1^n)$. We need to show that for infinitely many n,

$$\Pr[\mathcal{A}'(N) \in \{P, Q\}] > \varepsilon'(n).$$

Which is equivalent to showing that for infinitely many n,

$$\Pr[\mathcal{A}^{h^r_{N,g,\vec{a}}}(N, g) = h_{N,g,\vec{a}}(1^n)] > \varepsilon(n) - O(k(n) \cdot 2^{-n}).$$

To do so, we need a couple of simple facts on the distribution of g and of each $a_i \bmod \ell$. Let us first recall the definition of statistical distance:

Definition 3.2.5 *Let X and Y be two random variables over D. The statistical difference between X and Y is defined to be*

$$\frac{1}{2} \sum_{a \in D} |\Pr[X = a] - \Pr[Y = a]|.$$

We now have that:

Fact 1 g is a uniformly distributed quadratic-residue in $\mathbb{Z}_N^*$.

> **Reason:** v^2 is a uniformly distributed quadratic-residue in $\mathbb{Z}_N^*$ and squaring is a permutation over the set of quadratic-residues in $\mathbb{Z}_N^*$ (since N is a Blum-integer).

Fact 2 Let r and a' be uniformly distributed in $[N]$ and denote by a the value $r+2^{-1} \bmod \ell$. Then a and $a' \bmod \ell$ are of statistical distance $O(2^{-n})$.

Reason: ℓ divides $(Q-1)(P-1)$. Therefore the distribution of a conditioned on the event that $r \in [(Q-1)(P-1)]$ is the same as the distribution of $a' \bmod \ell$ conditioned on the event that $a' \in [(Q-1)(P-1)]$ (and in both cases it is simply the uniform distribution over $\mathbb{Z}_\ell$). It remains to notice that $\Pr[r \in [(Q-1)(P-1)]] = \Pr[a' \in [(Q-1)(P-1)]] = \frac{(Q-1)(P-1)}{N} = 1 - \frac{P+Q}{N} + \frac{1}{N} = 1 - O(2^{-n})$.

Let each value in $\vec{a'} = \langle a'_1, a'_2, \ldots, a'_k \rangle$ be uniformly distributed in $[N]$. Since $\mathcal{A}$ ε-solves the GDH_k-Problem and given Fact 1, we have that:

$$\Pr[\mathcal{A}^{h^r_{N,g,\vec{a'}}}(N,g) = h_{N,g,\vec{a'}}(1^n)] > \varepsilon(n).$$

Given Fact 2, it is easy to verify that the two random variables $h_{N,g,\vec{a}}$ and $h_{N,g,\vec{a'}}$ are of statistical distance $O(k(n) \cdot 2^{-n})$. Therefore, we can conclude that:

$$\Pr[\mathcal{A}^{h^r_{N,g,\vec{a}}}(N,g) = h_{N,g,\vec{a}}(1^n)] > \varepsilon(n) - O(k(n) \cdot 2^{-n}),$$

which completes the proof of the theorem. □

3.2.2 Conclusions

We showed that breaking the generalized Diffie-Hellman assumption modulo a Blum-integer is at least as hard as factoring Blum-integers. This implies that the security of the generalized Diffie-Hellman key-exchange protocol (which is mentioned in the introduction) can be based on the assumption that factoring is hard. In addition, as shown in Section 4.2.2, it implies the existence of efficient pseudo-random functions which are at least as secure as Factoring.

A possible line for further research is the study of the generalized Diffie-Hellman assumption in other groups and the relation between the generalized Diffie-Hellman assumption and the standard Diffie-Hellman assumption. It is interesting to note that the *decisional* version of the generalized Diffie-Hellman assumption is equivalent to the *decisional* version of the standard Diffie-Hellman assumption (as shown in [144] and in Section 4.2.1).

Chapter 4

Constructions of Pseudo-Random Functions

In this chapter we present several constructions of pseudo-random functions (see the introduction for a discussion on the definitions, applications and original-construction of pseudo-random functions and see Section 2.2 for their actual definitions). For our constructions, we introduce and study a new cryptographic primitive which we call a *pseudo-random synthesizer* and a generalization of this primitive which we call a *k-dimensional pseudo-random synthesizer*. These primitives are of independent interest as well. In Section 4.1, we show a *parallel* construction of pseudo-random functions from a pseudo-random synthesizer and parallel constructions of pseudo-random synthesizers based on several concrete and general intractability assumptions. In Section 4.2, we show even more parallel constructions of pseudo-random functions based on *concrete intractability assumptions* (such as the assumption that factoring Blum-integers is hard). In addition, these constructions are more efficient and have a simple algebraic structure. The constructions of Section 4.2 are motivated by the constructions of Section 4.1 (and in particular by the notion of k-dimensional synthesizers). A more detailed description of the results obtained in the two stages of the research and of their applications appears is the introduction (in Section 1.2).

4.1 Pseudo-Random Synthesizers and Functions

4.1.1 Organization

In Section 4.1.3 we define pseudo-random synthesizers and collections of pseudo-random synthesizers and discuss their properties. In Section 4.1.4 we describe our parallel construction of pseudo-random functions from pseudo-random synthesizers and in Section 4.1.5 we prove its security. In Section 4.1.6 we describe a related construction of pseudo-random functions. In addition, we discuss the time-space tradeoff and the incremental property of our constructions. In Section 4.1.7 we discuss the relations between pseudo-random synthesizers and other cryptographic primitives. In Section 4.1.8 we describe constructions of pseudo-random synthesizers based on several number-theoretic assumptions. In Section 4.1.9 we show how to construct pseudo-random synthesizers from hard-to-learn problems and consider a very sim-

ple concrete example. We also discuss the application of parallel pseudo-random functions to learning-theory.

4.1.2 Notation

The notation and definitions used in Section 4.1 are given in Chapter 2. Additional notation that are used in this section include:

- Let X be any random variable, we denote by $X^{k\times\ell}$ the $k\times\ell$ matrix whose entries are independently identically distributed according to X. We denote by X^k the vector $X^{1\times k}$.
- We identify functions of two variables and functions of one variable in the natural way. I.e, by letting $f : I^n \times I^n \mapsto I^k$ be equivalent to $f : I^{2n} \mapsto I^k$ and letting $f(x, y)$ be the same value as $f(x \circ y)$ (where $x \circ y$ stands for x concatenated with y).

4.1.3 Pseudo-random Synthesizers

As mentioned above, we introduce in this work a new cryptographic primitive called a pseudo-random synthesizer. In this section we define pseudo-random synthesizers and describe their properties.

Motivation

Pseudo-random synthesizers are efficiently computable functions of two variables. The significant feature of such a function, S, is that given polynomially-many uniformly distributed assignments, $\langle x_1, \ldots, x_m\rangle$ and $\langle y_1, \ldots, y_m\rangle$, for both variables, the output of S on all the combinations of these assignments, $(f(x_i, y_j))_{i,j=1}^m$, is pseudo-random (i.e, is indistinguishable from random to a polynomial-time observer). This is a strengthening of an important property of pseudo-random *generators* — the indistinguishability of a polynomial sample:

A pseudo-random (bit) generator [27, 150], is a polynomial-time computable function, $G : \{0,1\}^* \mapsto \{0,1\}^*$, such that $\forall x \in I^n, |G(x)| = \ell(n) > n$ and $G(U_n)$ is pseudo-random (i.e. $\{G(U_n)\}_{n\in\mathbb{N}}$ and $\{U_{\ell(n)}\}_{n\in\mathbb{N}}$ are computationally indistinguishable). It turns out that this definition implies that: Given polynomially-many uniformly distributed assignments, $\langle z_1, \ldots, z_m\rangle$, the sequence $\{(G(z_i)\}_{i=1}^m$, is pseudo-random.

The major idea behind the definition of pseudo-random synthesizers is to obtain a function, S, such that $\{(S(z_i)\}_{i=1}^m$ remains pseudo-random even when the z_i's *are not completely independent*. More specifically, pseudo-random synthesizers require that $\{(S(z_i)\}_{i=1}^m$ remains pseudo-random even when the z_i's are of the form $\{x_i \circ y_j\}_{i,j=1}^m$. Our work shows that (under some standard intractability assumptions) it is possible to obtain such a function S and that this property is indeed very powerful. As a demonstration to their strength, we note below that pseudo-random synthesizers are useful even when no restriction is made on their output length (which is very different than what we have for pseudo-random generators).

Remark 4.1.1 *It is important to note that there exist pseudo-random generators that are* not *pseudo-random synthesizers. An immediate example is the generator defined by* $G(x \circ$

$y) \stackrel{\text{def}}{=} G'(x) \circ y$, *where* G' *is also a pseudo-random generator. A more natural example is the subset-sum generator [74],* $G = G_{a_1,a_2,\ldots,a_n}$, *which is defined by* $G(z) = \sum_{z_i=1} a_i$. *This is not a pseudo-random synthesizer (for fixed values* $a_1, a_2, \ldots, a_n$*) since for every four* $n/2$*-bit strings,* x_1, x_2, y_1 *and* y_2, *we have that* $G(x_1 \circ y_1) + G(x_2 \circ y_2) = G(x_1 \circ y_2) + G(x_2 \circ y_1)$.

Formal Definition

We first introduce an additional notation to formalize the phrase "all different combinations":

Notation 4.1.1 *Let* f *be an* $I^{2n} \mapsto I^{\ell}$ *function and let* $X = \{x_1, \ldots, x_k\}$ *and* $Y = \{y_1, \ldots, y_m\}$ *be two sequences of* n*-bit strings. We define* $\mathbf{C}_f(X, Y)$ *to be the* $k \times m$ *matrix* $(f(x_i, y_j))_{i,j}$ *(*$\mathbf{C}$ *stands for combinations).*

We can now define what a pseudo-random synthesizer is:

Definition 4.1.1 (pseudo-random synthesizer) *Let* ℓ *be any* $\mathbb{N} \mapsto \mathbb{N}$ *function and let* $S : \{0,1\}^* \times \{0,1\}^* \mapsto \{0,1\}^*$ *be a polynomial-time computable function such that* $\forall x, y \in I^n, |S(x,y)| = \ell(n)$. *Then* S *is a* pseudo-random synthesizer *if for every probabilistic polynomial-time algorithm,* $\mathcal{D}$, *every two polynomials* $p(\cdot)$ *and* $m(\cdot)$, *and all sufficiently large* n,

$$\left|\Pr\left[\mathcal{D}(\mathbf{C}_S(X,Y)) = 1\right] - \Pr\left[\mathcal{D}((U_{\ell(n)})^{m(n)\times m(n)}) = 1\right]\right| < \frac{1}{p(n)},$$

where X *and* Y *are independently drawn from* $(U_n)^{m(n)}$. *That is, for random* X *and* Y *the matrix* $\mathbf{C}_S(X,Y)$ *cannot be efficiently distinguished from a random matrix.*

Expanding the Output Length

In Definition 4.1.1 no restriction was made on the output-length function, ℓ, of the pseudo-random synthesizer. However, our parallel construction of pseudo-random functions uses (parallel) pseudo-random synthesizers *with linear output length*, $\ell(n) = n$. The following lemma shows that any synthesizer, S, can be used to construct another synthesizer S', with large output-length, such that S and S' have the same parallel time complexity. Therefore, for the construction of pseudo-random functions in NC it is enough to show the existence of synthesizers with *constant output length* in NC.

Lemma 4.1.1 *Let* S *be a pseudo-random synthesizer with arbitrary output-length function,* ℓ, *in* NC^i *(resp.* AC^i*). Then for every constant* $0 < \varepsilon < 2$, *there exists a pseudo-random synthesizer* S' *in* NC^i *(resp.* AC^i*) such that its output-length function,* ℓ', *satisfies* $\ell'(n) = \Omega(n^{2-\varepsilon})$.

Proof. For every constant $c > 0$, define S^c as follows: Let $k_n \stackrel{\text{def}}{=} \max\{k \in \mathbb{Z} : k^{c+1} < n\}$. On input $x, y \in I^n$, regard the first k_n^{c+1} bits of x and y as two length-k_n^c sequences, X and Y, of k_n-bit strings. $S^c(x,y)$ is defined to be $\mathbf{C}_S(X,Y)$ (viewed as a single bit-string rather than a matrix). Notice that the following properties hold for S^c:

1. *S^c is indeed a pseudo-random synthesizer*: For any polynomial $m(\cdot)$, let X' and Y' be independently drawn from $(U_n)^{m(n)}$ and let X and Y be independently drawn from $(U_{k_n})^{m(n)\cdot k_n^c}$. By the definition of S^c, the distributions $\mathbf{C}_{S^c}(X',Y')$ and $\mathbf{C}_S(X,Y)$ are identical. Taking into account the fact that n is polynomial in k_n, we conclude that every polynomial-time distinguisher for S^c is also a polynomial-time distinguisher for S. Since S is a pseudo-random synthesizer so is S^c.

2. *Let ℓ_c denote the output-length function of S^c, then $\ell_c(n) = \Omega(n^{2-\frac{2}{c+1}})$*: Since c is a constant and $n < (k_n+1)^{c+1}$, for every n it holds that

$$\ell_c(n) = (k_n)^{2c} \cdot l(k_n) \geq (k_n)^{2c} = \Omega(n^{\frac{2c}{c+1}}) = \Omega(n^{2-\frac{2}{c+1}}).$$

3. *S^c is in NC^i (resp. AC^i)*: Immediate from the definition of S^c.

Thus, by taking S' to be S^c for some $c > \frac{2}{\varepsilon} - 1$ we obtain the lemma. □

The construction of Lemma 4.1.1 has the advantage that it is very simple and that the parallel time complexity of S and S' is identical. Nevertheless, it has an obvious disadvantage: The security of S' is related to the security of S on a much smaller input length. For example, if $\ell(n) = 1$ and $\ell'(n) = n$ then the security of S' on k^2-bit strings is related to the security S on k-bit strings. This results in a substantial increase in the time and space complexity of any construction that uses S'.

We now show an alternative construction to the one of Lemma 4.1.1 that is more security-preserving. The alternative construction uses a pseudo-random generator G that expands the input by a factor of 2 and relies on the GGM-Construction:

Corollary 4.1.2 *(of [62]) Let G be a pseudo-random generator in NC^i (resp. AC^i) such that $\forall s, |G(s)| = 2|s|$. Then for every polynomial $p(\cdot)$ there exists a pseudo-random generator G' in NC^{i+1} (resp. AC^{i+1}) such that $\forall s, |G'(s)| = p(|s|)\cdot|s|$.*

G' is defined as follows: On input s it computes $G(s) = s_0 \circ s_1$ and recursively generates $\frac{p(|s|)\cdot|s|}{2}$ bits from s_0 and $\frac{p(|s|)\cdot|s|}{2}$ bits from s_1. The number of levels required is $\lceil \log p(|s|) \rceil = O(\log|s|)$. Using Corollary 4.1.2 we get:

Lemma 4.1.3 *Let S be a pseudo-random synthesizer with arbitrary output-length function, ℓ, in NC^i (resp. AC^i). Let G be a pseudo-random generator in NC^j (resp. AC^j) such that $\forall s, |G(s)| = 2|s|$. Let k denote $\max\{i, j+1\}$. Then for every positive constant c, there exists a pseudo-random synthesizer S' in NC^k (resp. AC^k) such that its output-length function, ℓ', satisfies $\ell'(n) = \Omega(n^{2c}\cdot l(n))$.*

Furthermore, the construction of S' is linear-preserving in the sense of [72, 89] (the exact meaning of this claim is described below).

Proof.(sketch) S' is defined as follows: On input $x, y \in I^n$, compute $X = G'(x) = \{x'_1, \ldots, x'_{\lceil n^c \rceil}\}$ and $Y = G'(y) = \{y'_1, \ldots, y'_{\lceil n^c \rceil}\}$, where G' is the pseudo-random generator that is guaranteed to exist by Corollary 4.1.2. $S'(x,y)$ is defined to be $\mathbf{C}_S(X,Y)$.

It is immediate that S' is in NC^k (resp. AC^k) and that $\ell'(n) = \Omega(n^{2c} \cdot l(n))$. It is also not hard to verify that S' is indeed a pseudo-random synthesizer and (from the proof of Corollary 4.1.2) that the construction of S' is linear-preserving in the following sense:

Assume that there exists an algorithm that works in time $t(n)$ and distinguishes $\mathbf{C}_{S'}(X', Y')$ from $(U_{\ell'(n)})^{m'(n)\times m'(n)}$ with bias $\alpha(n)$, where X' and Y' are independently drawn from $(U_n)^{m'(n)}$. Let $m(n) = m'(n) \cdot \lceil n^c \rceil$. Then one of the following holds:

1. The same algorithm distinguishes $\mathbf{C}_S(X, Y)$ from $(U_{\ell(n)})^{m(n)\times m(n)}$ with bias $\alpha(n)/2$, where X and Y are independently drawn from $(U_n)^{m(n)}$.

2. There exists an algorithm that works in time $t(n) + m^2(n) \cdot poly(n)$ and distinguishes $G(U_n)$ from random with bias $\alpha(n)/O(m(n))$.

□

The construction of Lemma 4.1.3 is indeed more security-preserving than the construction of Lemma 4.1.1 (since the security of S' relates to the security of S and G on the same input length). However, the time complexity of S' is still substantially larger than the time complexity of S, and the parallel time complexity of S' might also be larger. Given the drawbacks of both construction, it seems that a direct construction of efficient and parallel synthesizers with linear output length is very desirable.

Collection of Pseudo-Random Synthesizers

A natural way to relax the definition of a pseudo-random synthesizer is to allow a distribution of functions for every input length rather than a single function. To formalize this we use the concept of an efficiently computable function ensemble.

Definition 4.1.2 (collection of pseudo-random synthesizers) *Let ℓ be any $\mathbb{N} \mapsto \mathbb{N}$ function and let $S = \{S_n\}_{n\in\mathbb{N}}$ be an efficiently computable $I^{2n} \mapsto I^{\ell}$ function ensemble. S is* a collection of $I^{2n} \mapsto I^{\ell}$ pseudo-random synthesizers *if for every probabilistic polynomial-time algorithm, $\mathcal{D}$, every two polynomials $p(\cdot)$ and $m(\cdot)$, and all sufficiently large n,*

$$\left|\Pr\left[\mathcal{D}(\mathbf{C}_{S_n}(X, Y)) = 1\right] - \Pr\left[\mathcal{D}((U_{\ell(n)})^{m(n)\times m(n)}) = 1\right]\right| < \frac{1}{p(n)},$$

where X and Y are independently drawn from $(U_n)^{m(n)}$.

As shown below, a collection of pseudo-random synthesizers is sufficient for our construction of pseudo-random functions. Working with a collection of synthesizers (rather than a single synthesizer) enables us to move some of the computation into a preprocessing stage during the key-generation. This is especially useful if all other computations can be done in parallel.

Note that Lemma 4.1.1 and Lemma 4.1.3 easily extend to collections of synthesizers.

4.1.4 A Parallel Construction of Pseudo-Random Functions

This section describes the construction of pseudo-random functions, using pseudo-random synthesizers as building blocks. The intuition of this construction is best explained through the concept of a k-dimensional pseudo-random synthesizer. This is a natural generalization of the "regular" (two-dimensional) synthesizer. Informally, an efficiently computable function of k variables, S^k, is a k-dimensional pseudo-random synthesizer if:

> *Given polynomially-many, uniformly-chosen, assignments for each variable,* $\left\{\{a_{j,i}\}_{i=1}^{m}\right\}_{j=1}^{k}$, *the output of* S^k *on all the combinations* $M = \left(S^k(a_{1,i_1}, a_{2,i_2}, \ldots, a_{k,i_k})\right)_{i_1,i_2,\ldots,i_k=1}^{m}$ *cannot be efficiently distinguished from uniform by an algorithm that can access* M *at points of its choice*

Note that this definition is somewhat different from the two-dimensional case. For any constant k (and in particular for $k = 2$) the matrix M is of polynomial size and we can give it as an input to the distinguisher. In general, M might be too large and therefore we let the distinguisher "access M at points of its choice".

Using this concept, the construction of pseudo-random functions can be described in two steps:

1. A parallel construction of an n-dimensional synthesizer, S^n, from a two-dimensional synthesizer, S, that has output length $\ell(n) = n$. This is a recursive construction, where the $2k$-dimensional synthesizer, S^{2k}, is defined using a k-dimensional synthesizer, S^k:

$$S^{2k}(x_1, x_2, \ldots, x_{2k}) \stackrel{\text{def}}{=} S^k(S(x_1, x_2), S(x_3, x_4), \ldots, S(x_{2k-1}, x_{2k})).$$

2. An immediate construction of the pseudo-random function, f, from S^n:

$$f_{\langle a_{1,0}, a_{1,1}, a_{2,0}, a_{2,1}, \ldots a_{n,0}, a_{n,1}\rangle}(x) \stackrel{\text{def}}{=} S^n(a_{1,x_1}, a_{2,x_2}, \ldots, a_{n,x_n}).$$

In fact, pseudo-random functions can be constructed from a *collection* of synthesizers. In this case, for each level of the recursion a different synthesizer is sampled from the collection. As noted below, for some collections of synthesizers (as those constructed in this work) it is enough to sample a single synthesizer for all levels.

Formal Definition

The following operation on sequences is used in the construction:

Definition 4.1.3 *For every function* $S : I^{2n} \mapsto I^n$ *and every sequence,* $L = \{\ell_1, \ell_2, \ldots, \ell_k\}$, *of* n*-bit strings define* $\mathbf{SQ}_S(L)$ *to be the sequence* $L' = \{\ell'_1, \ldots, \ell'_{\lceil \frac{k}{2} \rceil}\}$, *where* $\ell'_i = S(l_{2i-1}, \ell_{2i})$ *for* $i \leq \lfloor \frac{k}{2} \rfloor$ *and if* k *is odd, then* $\ell'_{\lceil \frac{k}{2} \rceil} = \ell_k$ *(***SQ** *stands for squeeze).*

We now turn to the construction itself:

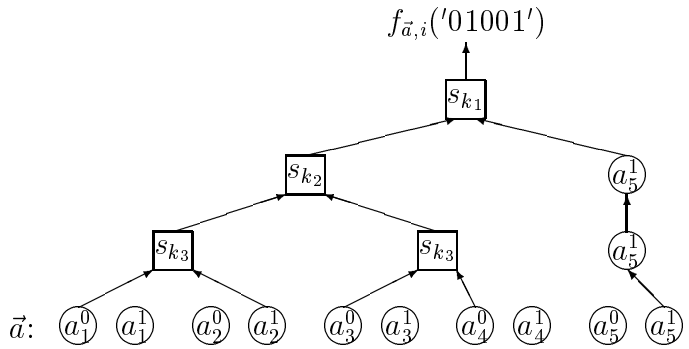

Figure 4.1: Computing the Value of the Pseudo-Random Function for $n = 5$

Construction 4.1.1 (Pseudo-Random Functions) *Let $S = \{S_n\}_{n\in\mathbb{N}}$ be a collection of $I^{2n} \mapsto I^n$ pseudo-random synthesizers and let $\mathcal{I}_S$ be a probabilistic polynomial-time key-generating algorithm for S. For every possible value, k, of $\mathcal{I}_S(1^n)$, denote by s_k the corresponding $I^{2n} \mapsto I^n$ function. The function ensemble $F = \{F_n\}_{n\in\mathbb{N}}$ is defined as follows:*

- *(*key-generation*) On input 1^n, the probabilistic polynomial-time key-generating algorithm $\mathcal{I}_F$ outputs a pair $(\vec{a}, \vec{k})$, where $\vec{a} = \{a_{1,0}, a_{1,1}, a_{2,0}, a_{2,1}, \ldots, a_{n,0}, a_{n,1}\}$ is sampled from $(U_n)^{2n}$ and $\vec{k} = \{k_1, k_2, \ldots, k_{\lceil \log n\rceil}\}$ is generated by $\lceil \log n\rceil$ independent executions of $\mathcal{I}_S$ on input 1^n (i.e. is sampled from $(\mathcal{I}_S(1^n))^{\lceil \log n\rceil}$).*

- *(*evaluation*) For every possible value, $(\vec{a}, \vec{k})$, of $\mathcal{I}_F(1^n)$ the function $f_{\vec{a},\vec{k}} : I^n \mapsto I^n$ is defined as follows: On an n-bit input, $x = x_1 x_2 \ldots x_n$, the function outputs the single value in*

$$\mathbf{SQ}_{s_{k_1}}(\mathbf{SQ}_{s_{k_2}}(\ldots \mathbf{SQ}_{s_{k_{\lceil \log n\rceil}}}(\{a_{1,x_1}, a_{2,x_2}, \ldots, a_{n,x_n}\})\ldots)).$$

Finally, F_n is defined to be the random variable that assumes as values the functions $f_{\vec{a},\vec{k}}$ with the probability space induced by $\mathcal{I}_F(1^n)$.

The evaluation of $f_{\vec{a},\vec{k}}(x)$ can be thought of as a recursive labeling process of a binary tree with n leaves and depth $\lceil \log n\rceil$. The i^{th} leaf has two possible labels, $a_{i,0}$ and $a_{i,1}$. The i^{th} input bit, x_i selects one of these labels a_{i,x_i}. The label of each internal node at depth d is the value of $s_{k_{d+1}}$ on the labels of its children. The value of $f_{\vec{a},\vec{k}}(x)$ is simply the label of the root. (Figure 4.1 illustrates the evaluation of $f_{\vec{a},\vec{k}}$ for $n = 5$.) We note that this labeling process is very different than the one associated with the GGM-Construction [62]. First, the binary tree is of depth $\lceil \log n\rceil$ instead of depth n as in [62]. Secondly, the labeling process is bottom-up instead of top-down as in [62] (i.e. starting at leaves instead of the root). Moreover, here each input defines a different labeling of the tree whereas in [62] the labeling of the tree is fully determined by the key (and the input only determines a leaf such that its label is the value of the function on this input).

Efficiency of the Construction

It is clear that F is efficiently computable (given that S is efficiently computable). Furthermore, the parallel time complexity of functions in F_n is larger by a factor of $O(\log n)$ than

the parallel time complexity of functions in S_n. The parallel time complexity of $\mathcal{I}_S$ and $\mathcal{I}_F$ is identical.

We note that, for simplicity, the parameter n serves a double role. n is both the length of inputs to $f_{\vec{a},\vec{k}} \in F_n$ and the security parameter for such a function (the second role is expressed by the fact that the strings in $\vec{a}$ are n-bit long). In practice, however, these roles would be separated. The security parameter would be determined by the quality of the synthesizers and the length of inputs to the pseudo-random functions would be determined by their application. In fact, one can usually use a pseudo-random function with a reasonably small input-length (say 160-bit long to prevent a "birthday attack"). This is implied by the suggestion of Levin [86] to pair-wise independently hash the input before applying the pseudo-random function (this idea is described with more details in the introduction).

Reducing the Key-Length

An apparent disadvantage of Construction 4.1.1 is the large key-length of a function $f_{\vec{a},\vec{k}} \in F_n$. In particular, the sequence $\vec{a}$ is defined by $2n^2$ bits. However, this is not truly a problem since: (a) In Section 4.1.6 a related construction is described (Construction 4.1.2) where $\vec{a}$ consists of a constant number of strings (and is therefore defined by $O(n)$ bits). (b) The truly random sequence $\vec{a}$ can be replaced by a pseudo-random sequence without increasing the depth of the construction (by more than a constant factor). This is achieved as follows: Let G be a pseudo-random generator that expands the input by a factor of 2. Let G' be the pseudo-random generator that can be constructed from G according to Corollary 4.1.2 for $p(n) = 2n$ (i.e. by using $\lceil \log n + 1 \rceil$ levels of the recursion). Then $\vec{a}$ can be replaced by $G'(\tilde{a})$, where $\tilde{a}$ is an n-bit seed.

In addition to $\vec{a}$, the key of $f_{\vec{a},\vec{k}} \in F_n$ consists of $\lceil \log n \rceil$ keys of functions in S_n. It turns out that for some collections of synthesizers (such as those described in this work) this overhead can be eliminated as well. This is certainly true when using a *single* synthesizer instead of a collection. Moreover, from the proof of security for Construction 4.1.1 one can easily extract the following claim: If the collection of synthesizers remains secure even when it uses a public key (i.e. if $\mathbf{C}_{s_k}(X,Y)$ remains pseudo-random even when the distinguisher sees k), then the $\lceil \log n \rceil$ keys can be replaced with a single one (i.e. the same key can be used at all levels of the recursion).

4.1.5 Security of the Construction

Theorem 4.1.4 *Let S and F be as in Construction 4.1.1 and let $R = \{R_n\}_{n\in\mathbb{N}}$ be the uniform $I^n \mapsto I^n$ function ensemble. Then F is an efficiently computable pseudo-random function ensemble. Furthermore, any efficient distinguisher, $\mathcal{M}$, between F and R yields an efficient distinguisher, $\mathcal{D}$, for S such that the success probability of $\mathcal{D}$ is smaller by a factor of at most $\lceil \log n \rceil$ than the success probability of $\mathcal{M}$.*

To prove Theorem 4.1.4, we use of a hybrid argument (for details about this proof technique, see [60]): We first define a sequence of $\lceil \log n \rceil + 1$ function distributions such that the two extreme distributions are R_n and F_n. We then show that any distinguisher for two

neighboring distributions can be transformed into a distinguisher for the pseudo-random synthesizers. For simplicity, we define those hybrid-distributions in case $n = 2^\ell$. The definition easily extends to a general value of n such that Claim 4.1.1 still holds.

For any $0 \leq j \leq \ell$, denote by H_n^j the j^{th} hybrid-distribution. The computation of functions in H_n^j may be described as a labeling process of a binary tree with n leaves and depth ℓ (an analogous description for F_n appears in Section 4.1.4). Here, the labeling process starts with nodes at depth $\ell - j$. The i^{th} such node has 2^{2^j} possible labels, $\{a_{i,s} : s \in I^{2^j}\}$ (which are part of the key). The i^{th} 2^j-bit substring of the input, x_i, selects one of these labels, a_{i,x_i}. The rest of the labeling process is the same as it was for functions in F_n: The label of each node at depth $d < \ell - j$ is the value of $s_{k_{d+1}}$ on the labels of its children. The value of the function on this input is simply the label of the root.

Another way to think of H_n^j is via the concept of a k-dimensional synthesizer (see Section 4.1.4). As was the case for F^n, the construction of functions in H_n^j can be described in two steps: (1) A recursive construction of a $2^{\ell-j}$-dimensional synthesizer, $S^{2^{\ell-j}}$, from a two-dimensional synthesizer, S. (2) An immediate construction of the pseudo-random function, f, from $S^{2^{\ell-j}}$:

$$f_{\{a_{r,s}:1\leq r\leq 2^{\ell-j}, s\in I^{2^j}\}}(x_1 \circ x_2 \ldots \circ x_{2^{\ell-j}}) \stackrel{\text{def}}{=} S^{2^{\ell-j}}(a_{1,x_1}, a_{2,x_2}, \ldots, a_{2^{\ell-j},x_{2^{\ell-j}}}).$$

We turn to the formal definition of the hybrid-distributions:

Definition 4.1.4 *Let $\mathcal{I}_S$ be the key-generating algorithms of S. Let n, ℓ and j be three integers such that $n = 2^\ell$ and $0 \leq j \leq \ell$. For every sequence, $\vec{k} = \{k_1, k_2, \ldots, k_{\ell-j}\}$ of possible values of $\mathcal{I}_S(1^n)$ and for every length-$2^{2^j}2^{\ell-j}$ sequence of n-bit strings, $\vec{a} = \{a_{r,s} : 1 \leq r \leq 2^{\ell-j}, s \in I^{2^j}\}$ the function $f_{\vec{a},\vec{k}} : I^n \mapsto I^n$ is defined as follows: On input $x = x_1 \circ x_2 \ldots \circ x_{2^{\ell-j}}$, where $\forall 1 \leq i \leq 2^{\ell-j}, x_i \in I^{2^j}$ the function outputs the single value in*

$$\mathbf{SQ}_{s_{k_1}}(\mathbf{SQ}_{s_{k_2}}(\ldots \mathbf{SQ}_{s_{k_{\ell-j}}}(a_{1,x_1}, a_{2,x_2}, \ldots, a_{2^{\ell-j},x_{2^{\ell-j}}}) \ldots)).$$

H_n^j is the random variable that assumes as values the functions $f_{\vec{a},\vec{k}}$ defined above, where the k_i's are independently distributed according to $\mathcal{I}_S(1^n)$ and $\vec{a}$ is independently distributed according to $(U_n)^{2^{2^j}2^{\ell-j}}$.

This definition immediately implies that:

Claim 4.1.1 *H_n^0 and F_n are identically distributed and $H_n^{\lceil \log n \rceil}$ and R_n are identically distributed.*

The proof below shows that for every $0 \leq j < \ell$ the two neighboring ensembles H_n^j and H_n^{j+1} are computationally indistinguishable. As shown below, this implies Theorem 4.1.4 by a standard hybrid argument.

Proof. (of Theorem 4.1.4) As mentioned in Section 4.1.4, it is obvious that F is an efficiently computable function ensemble. Assume that F is not pseudo-random. By the definition of

pseudo-random function ensembles, there exists a polynomial-time oracle machine, $\mathcal{M}$, and a polynomial $p(\cdot)$ so that for infinitely many n,

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

where $R=\{R_n\}_{n\in\mathbb{N}}$ is the uniform $I^n\mapsto I^n$ function ensemble. Let $t(\cdot)$ be a polynomial that bounds the number queries that $\mathcal{M}$ makes on input 1^n.

Given $\mathcal{M}$, we define the probabilistic polynomial-time algorithm $\mathcal{D}$ that distinguishes the output of S_n from random. Let $m=m(n)$ be defined by $m(n)\stackrel{\text{def}}{=}t(n)\cdot n$. For every n, the input of $\mathcal{D}$ is an $m(n)\times m(n)$ matrix, $B=(b_{i,j})$, whose entries are n-bit strings. As part of its algorithm, $\mathcal{D}$ invokes $\mathcal{M}$ on input 1^n. The definition of m allows $\mathcal{D}$ to answer all the queries of $\mathcal{M}$ (which are bounded by $t(n)$). It is shown below that $\mathcal{D}$ distinguishes between the following two distributions of B: (a) $\mathbf{C}_{S_n}(X,Y)$ where X and Y are independently drawn from $(U_n)^{m(n)}$. (b) $(U_n)^{m(n)\times m(n)}$.

For simplicity of presentation, we only define the algorithm that $\mathcal{D}$ performs for $n=2^\ell$. It is easy to extend this definition to a general value of n such that Claim 4.1.2 and Claim 4.1.3 still hold. On input $B=(b_{i,j})_{i,j=1}^{m(n)}$, the algorithm is defined as follows:

1. Choose $0\leq J<\ell$ uniformly at random.
2. Generate $\vec{k}=\{k_1,k_2,\ldots,k_{\ell-J-1}\}$ by $\ell-J-1$ independent executions of $\mathcal{I}_S$ on input 1^n.
3. Extract $2^{\ell-J-1}$ sub-matrices of B: For $1\leq i\leq 2^{\ell-J-1}$, denote by $B^i=\left(b^i_{u,v}\right)_{u,v=1}^{t(n)}$ the $t(n)\times t(n)$ diagonal sub-matrix of B defined by
$$b^i_{u,v}\stackrel{\text{def}}{=}b_{u+((i-1)\cdot t(n)+1),v+((i-1)\cdot t(n)+1)}.$$
4. Invoke $\mathcal{M}$ on input 1^n. Denote by $q^r=q^r_1\circ q^r_2\ldots\circ q^r_{2^{\ell-J}}$ the r^{th} query $\mathcal{M}$ makes, where $q^r_i\in I^{2^J}$ for $1\leq i\leq 2^{\ell-J}$. On each of these queries $\mathcal{D}$ answers as follows:
   For every $1\leq i\leq 2^{\ell-J-1}$, denote by $a_{i,q^r_{2i-1}\circ q^r_{2i}}$ the entry $b^i_{u,v}$ of B^i where
$$u=\min\{1\leq j\leq r: q^j_{2i-1}=q^r_{2i-1}\}\ and\ v=\min\{1\leq j\leq r: q^j_{2i}=q^r_{2i}\}.$$
   Answer the query with the single value in
$$\mathbf{SQ}_{s_{k_1}}(\ldots\mathbf{SQ}_{s_{k_{\ell-J-1}}}(\{a_{1,q^r_1\circ q^r_2},\ldots,a_{2^{\ell-J-1},q^r_{2^{\ell-J}-1}\circ q^r_{2^{\ell-J}}}\})\ldots).$$
5. Output whatever $\mathcal{M}$ outputs.

It is obvious that $\mathcal{D}$ is a polynomial-time algorithm. To show that $\mathcal{D}$ is also a distinguisher for the pseudo-random synthesizers, we first state and prove the following two claims:

Claim 4.1.2 *For every $0\leq J<\ell$,*

$$\Pr\left[\mathcal{D}((U_n)^{m(n)\times m(n)})=1|J=j\right]=\Pr\left[\mathcal{M}^{H_n^{j+1}}(1^n)=1\right].$$

Proof. As part of its algorithm, $\mathcal{D}$ denotes some of B's entries by names of the form "$a_{r,s}$", where $1 \leq r \leq 2^{\ell-J-1}$ and $s \in I^{2^{J+1}}$. Note that $\mathcal{D}$ never denotes an entry of B by two different names. Assume, for the sake of the proof, that any name $a_{r,s}$ that was not used by $\mathcal{D}$ is assigned an independently and uniformly distributed n-bit string. Denote by $\vec{a}$ the sequence $\{a_{r,s} : 1 \leq r \leq 2^{\ell-J-1}, s \in I^{2^{J+1}}\}$. It is easy to verify that:

1. When B is uniformly distributed, the distribution of $\vec{a}$ is identical to $(U_n)^{2^{2^{J+1}}2^{\ell-J-1}}$.
2. $\mathcal{D}$ answers every query, q, of $\mathcal{M}$ with the value $f_{\vec{a},\vec{k}}(q)$, where $f_{\vec{a},\vec{k}}$ is as in the definition of H_n^{J+1}.

The claim immediately follows from (1) and (2) and from the definition of $\vec{k}$. □

Claim 4.1.3 *Let X and Y be independently drawn from $(U_n)^{m(n)}$. Then for every $0 \leq J < \ell$,*

$$\Pr\left[\mathcal{D}(\mathbf{C}_{S_n}(X,Y)) = 1 | J = j\right] = \Pr\left[\mathcal{M}^{H_n^j}(1^n) = 1\right].$$

Proof. Let $X = \{x_1, x_2, \ldots, x_{m(n)}\}$ and $Y = \{y_1, y_2, \ldots, y_{m(n)}\}$ be independently drawn from $(U_n)^{m(n)}$ and let s_k be drawn from S_n. Assume that the input of $\mathcal{D}$ is $B = \mathbf{C}_{s_k}(X,Y)$. For the sake of the proof, define the vector $\vec{a'} = \{a'_{i,s} : 1 \leq i \leq 2^{\ell-J}, s \in I^{2^J}\}$ as follows:

- If $\mathcal{D}$ denoted by $a_{i,q_{2i-1}^r \circ q_{2i}^r}$ the entry $b_{u,v}^i$ of B^i, then define a'_{2i-1,q_{2i-1}^r} to be $x_{(i-1)\cdot t(n)+u}$ and a'_{2i,q_{2i}^r} to be $y_{(i-1)\cdot t(n)+v}$. Note that $a_{i,q_{2i-1}^r \circ q_{2i}^r} = s_k(a'_{2i-1,q_{2i-1}^r}, a'_{2i,q_{2i}^r})$.
- For all other values in $\vec{a'}$ assign an independently and uniformly distributed n-bit string.

It is easy to verify that the distribution of $\vec{a'}$ is identical to $(U_n)^{2^{2^J}2^{\ell-J}}$. Let $\vec{k'}$ be the sequence $\{k_1, k_2, \ldots, k_{\ell-J-1}, k\}$ and let $f_{\vec{a'},\vec{k'}}$ be as in the definition of H_n^J. We now have that the answer $\mathcal{D}$ gives to the r^{th} query, q^r, of $\mathcal{M}$ is:

$$\begin{aligned}
& \mathbf{SQ}_{s_{k_1}}(\ldots \mathbf{SQ}_{s_{k_{\ell-J-1}}}(\{a_{1,q_1^r \circ q_2^r}, \ldots, a_{2^{\ell-J-1},q_{2^{\ell-J-1}}^r \circ q_{2^{\ell-J}}^r}\}) \ldots) \\
= \; & \mathbf{SQ}_{s_{k_1}}(\ldots \mathbf{SQ}_{s_{k_{\ell-J-1}}}(\mathbf{SQ}_{s_k}(\{a'_{1,q_1^r}, a'_{2,q_2^r}, \ldots, a'_{2^{\ell-J},q_{2^{\ell-J}}^r}\})) \ldots) \\
= \; & f_{\vec{a'},\vec{k'}}(q^r).
\end{aligned}$$

From this fact and from the definition of $\vec{a'}$ and $\vec{k'}$, we immediately get the claim. □

By Claims 4.1.1, 4.1.2 and 4.1.3, we can now conclude the following: Let X and Y be independently drawn from $(U_n)^{m(n)}$, then for infinitely many n,

$$\begin{aligned}
& \left|\Pr\left[\mathcal{D}(\mathbf{C}_{S_n}(X,Y)) = 1\right] - \Pr\left[\mathcal{D}((U_n)^{m(n)\times m(n)}) = 1\right]\right| \\
= \; & \frac{1}{\lceil \log n \rceil} \cdot \left| \sum_{j=0}^{\lceil \log n \rceil - 1} \Pr\left[\mathcal{D}(\mathbf{C}_{S_n}(X,Y)) = 1 | J = j\right] - \sum_{j=0}^{\lceil \log n \rceil - 1} \Pr\left[\mathcal{D}((U_n)^{m(n)\times m(n)}) = 1 | J = j\right] \right| \\
= \; & \frac{1}{\lceil \log n \rceil} \cdot \left| \sum_{j=0}^{\lceil \log n \rceil - 1} \Pr\left[\mathcal{M}^{H_n^j}(1^n) = 1\right] - \sum_{j=0}^{\lceil \log n \rceil - 1} \Pr\left[\mathcal{M}^{H_n^{j+1}}(1^n) = 1\right] \right|
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{\lceil \log n \rceil} \cdot \left| \Pr\left[\mathcal{M}^{H^0_n}(1^n) = 1\right] - \Pr\left[\mathcal{M}^{H^{\lceil \log n \rceil}_n}(1^n) = 1\right] \right| \\
&= \frac{1}{\lceil \log n \rceil} \cdot \left| \Pr\left[\mathcal{M}^{F_n}(1^n) = 1\right] - \Pr\left[\mathcal{M}^{R_n}(1^n) = 1\right] \right| \\
&> \frac{1}{p(n) \cdot \lceil \log n \rceil}.
\end{aligned}
$$

This contradicts the assumption that S is a collection of pseudo-random synthesizers and completes the proof of Theorem 4.1.4. □

Corollary 4.1.5 *For any collection of pseudo-random synthesizers, S, such that its functions are computable in NC^i there exists an efficiently computable pseudo-random function ensemble, F, such that its functions are computable in NC^{i+1}. Furthermore, the corresponding key-generating algorithms, $\mathcal{I}_S$ and $\mathcal{I}_F$, have the same parallel time complexity.*

Proof. By Lemma 4.1.1, we can construct from, S, a new collection of $I^{2n} \mapsto I^n$ pseudo-random synthesizers, S', such that its functions are computable in NC^i. By Theorem 4.1.4, we can construct from S' an efficiently computable pseudo-random function ensemble, F, such that its functions are computable in NC^{i+1}. Both constructions preserve the parallel time complexity of the key-generating algorithms. □

4.1.6 A Related Construction and Additional Properties

Though designed to enable efficient computation in parallel, Construction 4.1.1 obtains some additional useful properties. In this section we describe two such properties: a rather sharp time-space tradeoff and an incremental property. We also show how to adjust the construction in order to improve upon these properties.

Time-Space Tradeoff

Construction 4.1.1 has the advantage of a sharp time-space tradeoff. In order to get an even sharper tradeoff, we describe an alternative construction of pseudo-random functions. The best way to understand the revised construction is by viewing the computation process backwards: Every function on n-bits is defined by the length-2^n sequence of all its values. Assume that we could sample and store two length-$\lceil \sqrt{2^n} \rceil$ sequences, X and Y, of random strings as the key of a pseudo-random function. In this case, given a pseudo-random synthesizer, S, we can define the 2^n values of the pseudo-random function to be the entries of the matrix $\mathbf{C}_S(X,Y)$. In order to reduce the key size, we can replace the random sequences, X and Y, with pseudo-random sequences. Such sequences X and Y can be obtained *together* from $\mathbf{C}_S(X',Y')$, where X' and Y' are two shorter random sequences (of length approximately $\sqrt{2 \cdot 2^{n/2}}$). By continuing this process of reducing the key size $\log n$ times, we get a key with constant number of strings (see Figure 4.2 for an illustration of the construction).

In order to understand where the original construction is "wasteful" in the size of the key we can describe it in similar terms: The 2^n values of the function are still the values of $\mathbf{C}_S(X,Y)$ for two sequences X and Y (in the description of the computation as a tree

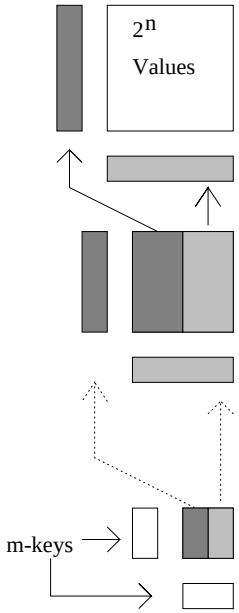

Figure 4.2: Illustration of the Alternative Construction

labeling process these are all the possible labels of the root's children) but then we get X and Y *separately* as $\mathbf{C}_S(X', Y')$ and $\mathbf{C}_S(X'', Y'')$. By the time the sequences have constant-length, there are $O(n)$ of those.

Returning to the new construction, note that if we allow a key of m strings we only need $t \approx \log n - \log\log m$ of the steps described above. Computing such functions requires t phases and in each phase several parallel invocations of S. The total number of invocations of S is $2^t - 1 \approx \frac{n}{\log m}$. This seems to be a relatively sharp time-space tradeoff and, to the best of our knowledge, one that cannot be obtained by the GGM-construction.

For some applications, like the protection of the data on a disk, we need pseudo-random functions with reasonably small amount of entries. In this case, by storing relatively few strings, we can achieve a very easy-to-compute function. For example, 512 random ℓ-bit strings define a pseudo-random $I^{30} \mapsto I^{\ell}$ function. Computing this function requires only 3 invocations of a pseudo-random synthesizer in 2 phases.

We formalize the definition of the alternative construction:

Construction 4.1.2 (Alternative Construction of Pseudo-Random Functions) *Let $S = \{S_n\}_{n\in\mathbb{N}}$ be a collection of $I^{2n} \mapsto I^n$ pseudo-random synthesizers and let $\mathcal{I}_S$ be a probabilistic polynomial-time key-generating algorithm for S. For every possible value, k, of $\mathcal{I}_S(1^n)$, denote by s_k the corresponding $I^{2n} \mapsto I^n$ function. The function ensemble $F = \{F_n\}_{n\in\mathbb{N}}$ is defined as follows:*

- (key-generation) *Let m_j denote the value $2^j + 2$ and let t_n denote the smallest integer t such that $m_t \geq n$. On input 1^n, the probabilistic polynomial-time key-generating algorithm $\mathcal{I}_F$ outputs a pair $(\vec{a}, \vec{k})$, where $\vec{a} = \{a_0, a_1, \ldots, a_{2^{m_0}-1}\}$ is generated according to $(U_n)^{2^{m_0}}$ and $\vec{k} = \{k_1, k_2, \ldots, k_{t_n}\}$ is generated by t_n independent executions of $\mathcal{I}_S$ on input 1^n (i.e. is sampled from $(\mathcal{I}_S(1^n))^{t_n}$).*

- (evaluation) *For every possible value,* $(\vec{a},\vec{k})$*, of* $\mathcal{I}_F(1^n)$ *and every* j *such that* $0 \leq j \leq t_n$*, define the function* $f^j_{\vec{a},\vec{k}} : I^{m_j} \mapsto I^n$ *in recursion on* j*: For* $x \in I^{m_0}$*, define* $f^0_{\vec{a},\vec{k}}(x)$ *to be* a_x*. For any* $j > 0$ *and* $x = x_1 \circ x_2 \in I^{m_j}$ *(*x_1 *and* x_2 *are* $(2^{j-1}+1)$*-bit strings) define* $f^j_{\vec{a},\vec{k}}(x)$ *to be* $s_{k_j}(f^{j-1}_{\vec{a},\vec{k}}(0,x_1), f^{j-1}_{\vec{a},\vec{k}}(1,x_2))$*.*

  For every $x \in I^n$ *the value of the function* $f_{\vec{a},\vec{k}} : I^n \mapsto I^n$ *on* x *is defined to be* $f^{t_n}_{\vec{a},\vec{k}}(x')$*, where* x' *is obtained by padding* x *with* $m_{t_n} - n$ *zeros.*

Finally, F_n *is defined to be the random variable that assumes as values the functions* $f_{\vec{a},\vec{k}}$ *with the probability space induced by* $\mathcal{I}_F(1^n)$*.*

The proof of security for Construction 4.1.2 is omitted since it is almost identical to the proof of security for Construction 4.1.1.

We can now state the exact form of the time-space tradeoff under the notation of Construction 4.1.2. If $\vec{a}$ contains 2^{m_i} strings instead of 2^{m_0}, then we can define $f^i_{\vec{a},\vec{k}}(x)$ to be a distinct value in $\vec{a}$ for every $x \in I^{m_i}$ and keep the recursive definition of $f^j_{\vec{a},\vec{k}}$ as before for $j > i$. In this case, computing $f_{\vec{a},\vec{k}}(x)$ can be done in $t_n - i$ phases with a total of $2^{t_n - i} - 1$ invocations of the synthesizers. The next lemma follows (for simplicity this lemma is stated in terms of a synthesizer instead of a collection of synthesizers):

Lemma 4.1.6 *Let* S *be a pseudo-random synthesizer with output-length function* $\ell(n) = n$*. Assume that* S *can be computed in parallel-time* $D(n)$ *and work* $W(n)$ *(on* n*-bit inputs). Then for every* $m = m(n)$ *such that* $2^{m_0} \leq m(n) < 2^n$ *there exists an efficiently computable pseudo-random function ensemble* $F = \{F_n\}_{n\in\mathbb{N}}$ *such that the key of a function in* F_n *is a sequence of at most* $m(n)$ *random* n*-bit strings and this function can be computed in parallel-time* $(\log n - \log\log m(n) + O(1))D(n)$ *and using work of* $O(\frac{n}{\log m(n)})W(n)$*.*

Incremental Property

We now describe an observation of Mihir Bellare that gives rise to an interesting incremental property of our construction. (For the formulation and treatment of incremental cryptography, see the work of Bellare, Goldreich and Goldwasser [15, 16].)

Let f be any function in F_n, where $F = \{F_n\}_{n\in\mathbb{N}}$ is the pseudo-random function ensemble defined in Construction 4.1.1. Let $x, y \in I^n$ be of Hamming distance one (x and y differ on exactly one bit). Then given the computation of $f(x)$ (including all intermediate values), we only need additional $\log n$ invocations of the pseudo-random synthesizers (instead of n) in order to evaluate $f(y)$. The easiest way to see the correctness of this observation is to recall the description of the computation of $f(x)$ as a labeling process on a depth-$\log n$ binary tree. The only labels that change as a result of flipping one bit of x are those of the nodes on a path from one leaf to the root (i.e. $\log n + 1$ labels).

If a Gray-code representation[1] of numbers is used, we get a similar observation for the computation of $f(x)$ and $f(x+1)$: Given the computation of one of these values, computing

[1] A permutation, P, on I^n is called a Gray-code representation if for every $0 \leq x < 2^n$ the Hamming distance between $P(x)$ and $P(x+1 \bmod 2^n)$ is one. Such a P defines a Hamiltonian-cycle on the n-dimensional cube. It is not hard to see that an easy-to-compute P can be defined.

the other requires only additional $\log n$ invocations of the pseudo-random synthesizers. It is not hard to imagine situations where one of these incremental properties is useful.

The observation regarding the computation of $f(x)$ and $f(y)$, for x and y of Hamming distance one, also holds for the functions of Construction 4.1.2. The observation regarding the computation of $f(x)$ and $f(x+1)$ holds if we use a different representation of numbers (this representation is similar to a Gray-code, though a bit more complicated).

4.1.7 Synthesizers Based on General Cryptographic Primitives

Sections 4.1.7-4.1.9 are mostly devoted to showing parallel constructions of pseudo-random synthesizers. In this section we provide a simple construction of pseudo-random synthesizers based on what we call weak pseudo-random functions. This construction immediately implies a construction of pseudo-random synthesizers based on trapdoor one-way permutations (and an additional construction, based on any hard-to-learn problem, which is considered in Section 4.1.9). An interesting line for further research is the parallel construction of pseudo-random synthesizers from other cryptographic primitives. In particular, we do not know of such a construction from pseudo-random generators or directly from one-way functions.

Weak Pseudo-Random Functions

The reason pseudo-random functions are hard to construct is that they must endure a very powerful kind of an attack. The adversary (the distinguisher) may query their values at every point and may adapt its queries based on the answers it gets. We can weaken the opponent by letting the only access it has to the function be a polynomial sample of random points and the value of the function at these points (more on definitions of function families that are weaker than pseudo-random functions can be found in [104]). We call functions that look random to such an adversary *weak* pseudo-random functions. In this section it is shown that weak pseudo-random functions yield pseudo-random synthesizers in a straightforward manner. We therefore get a parallel construction of (standard) pseudo-random functions from weak pseudo-random functions.

For simplicity, we define weak pseudo-random functions as length-preserving. In their definition we use the following notation:

Notation 4.1.2 *For every function f and every sequence $X = \{x_1 \dots x_k\}$ of values in the domain of f, denote by $\mathbf{V}(X, f)$ the sequence $\{x_1, f(x_1), x_2, f(x_2) \dots x_k, f(x_k)\}$ ($\mathbf{V}$ stands for values).*

Definition 4.1.5 (collection of weak pseudo-random functions) *An efficiently computable $I^n \mapsto I^n$ function ensemble $F = \{F_n\}_{n\in\mathbb{N}}$, is a* collection of weak pseudo-random functions *if for every probabilistic polynomial-time algorithm, $\mathcal{D}$, every two polynomials $p(\cdot)$ and $m(\cdot)$, and all sufficiently large n,*

$$\left|\Pr\left[\mathcal{D}(\mathbf{V}((U_n)^{m(n)}, F_n)) = 1\right] - \Pr\left[\mathcal{D}((U_n)^{2m(n)}) = 1\right]\right| < \frac{1}{p(n)}.$$

Let F be a collection of weak pseudo-random functions and let $\mathcal{I}$ be the polynomial-time key-generating algorithm for F. Lemma 4.1.7 shows how to construct a pseudo-random synthesizer from F and $\mathcal{I}$. Since the random bits of $\mathcal{I}$ can be replaced by pseudo-random bits, we can assume that $\mathcal{I}$ only uses n truly random bits on input 1^n. In fact, this is only a simplifying assumption which is not really required for the construction of pseudo-random synthesizers. For every $r \in I^n$, denote by $\mathcal{I}_r(1^n)$ the value of $\mathcal{I}(1^n)$ when $\mathcal{I}$ uses r as its random bits.

Lemma 4.1.7 *Let F and $\mathcal{I}$ be as above and define $S : \{0,1\}^* \times \{0,1\}^* \mapsto \{0,1\}^*$ such that $\forall x, y \in I^n, S(x,y) = f_{\mathcal{I}_y(1^n)}(x)$. Then S is a pseudo-random synthesizer.*

Proof. It is obvious that S is efficiently computable. Assume, in contradiction to the lemma, that S is not a pseudo-random synthesizer. Then there exists a probabilistic polynomial-time algorithm, $\mathcal{D}$, and polynomials $p(\cdot)$ and $m(\cdot)$, such that for infinitely many n,

$$\left|\Pr\left[\mathcal{D}(\mathbf{C}_S(X,Y)) = 1\right] - \Pr\left[\mathcal{D}((U_n)^{m(n)\times m(n)}) = 1\right]\right| > \frac{1}{p(n)},$$

where X and Y are independently drawn from $(U_n)^{m(n)}$.

For every n and every $0 \leq i \leq m(n)$, define the i^{th} hybrid distribution H_n^i over $m(n)\times m(n)$ matrices as follows: The first i columns are distributed according to $\mathbf{C}_S(X,Y)$, where X is drawn from $(U_n)^{m(n)}$ and Y is independently drawn from $(U_n)^i$. The last $m(n)-i$ columns are independently distributed according to $(U_n)^{m(n)\times(m(n)-i)}$. It is immediate that for infinitely many n,

$$\left|\Pr\left[\mathcal{D}(H_n^{m(n)}) = 1\right] - \Pr\left[\mathcal{D}(H_n^0) = 1\right]\right| > \frac{1}{p(n)}.$$

We now define a distinguisher $\mathcal{D}'$ for F. Given $\{x_1, z_1, x_2, z_2, \ldots, x_{m(n)}, z_{m(n)}\}$ as its input, $\mathcal{D}'$ performs the following algorithm:

1. Define $X = \{x_1, \ldots, x_{m(n)}\}$ and $Z = \{z_1, \ldots, z_{m(n)}\}$.
2. Uniformly choose $0 < J \leq m(n)$.
3. Sample Y from $(U_n)^{J-1}$ and generate an $m(n)\times m(n)$ matrix B whose first $J-1$ columns are $\mathbf{C}_S(X,Y)$, its J^{th} column is Z^t and the last $m(n) - J$ columns are independently distributed according to $(U_n)^{m(n)\times(m(n)-J)}$.
4. Output $\mathcal{D}(B)$.

It is obvious that $\mathcal{D}'$ is efficiently computable. It is also easy to verify that

$$\Pr\left[\mathcal{D}'(\mathbf{V}((U_n)^{m(n)}, F_n)) = 1 | J = j\right] = \Pr\left[\mathcal{D}(H_n^j) = 1\right]$$

and that

$$\Pr\left[\mathcal{D}'((U_n)^{2m(n)}) = 1 | J = j\right] = \Pr\left[\mathcal{D}(H_n^{j-1}) = 1\right].$$

Thus, by a standard hybrid argument, we get that for infinitely many n,

$$\left|\Pr\left[\mathcal{D}'(\mathbf{V}((U_n)^{m(n)}, F_n)) = 1\right] - \Pr\left[\mathcal{D}'((U_n)^{2m(n)}) = 1\right]\right| > \frac{1}{p(n)m(n)},$$

in contradiction to the assumption that F is a collection of weak pseudo-random functions. We can therefore conclude the lemma. □

Notice that S (as in Lemma 4.1.7) obeys an even more powerful requirement than is needed by the definition of a pseudo-random synthesizer: For random X and Y the matrix $\mathbf{C}_S(X,Y)$ cannot be efficiently distinguished from a random matrix *even* if we allow the distinguisher access to X.

Corollary 4.1.8 *(of Lemma 4.1.7) If there exist weak pseudo-random functions that can be sampled and evaluated in NC, then there also exist a pseudo-random synthesizer in NC and (standard) pseudo-random functions that can be sampled and evaluated in NC.*

Trapdoor One-Way Permutations

We now describe a rather simple construction of weak pseudo-random functions from a collection of trapdoor permutations. Therefore, given Lemma 4.1.7, we get a construction of a pseudo-random synthesizer out of a collection of trapdoor permutations. This pseudo-random synthesizer is in NC if the trapdoor permutations can be sampled and inverted in NC (in fact, there is an additional requirement of a hard-core predicate in NC but this is already satisfied by [65]). Since we have no concrete example of this sort, we only give a brief and informal description of the construction (for formal definitions of trapdoor one-way permutations and hard-core bits, see e.g. [60, 89]).

Let $F = \{F_n\}_{n\in\mathbb{N}}$ be a permutation ensemble such that every $f_i \in F_n$ is a permutation over a domain D_n. Informally, F is a collection of trapdoor one-way permutations if the key generating algorithm $\mathcal{I}_F$ of F outputs both a public-key, i, and a trapdoor-key, $t(i)$, and we have that:

- Given i, the function f_i is easy to compute everywhere but hard to invert on the average.
- Given $t(i)$ the function f_i is easy to compute and to invert everywhere.

Let $F = \{F_n : D_n \mapsto D_n\}_{n\in\mathbb{N}}$ be a collection of trapdoor one-way permutations. Assume that the collection is one-way for the uniform distribution over the inputs (i.e. it is hard to compute x given $F_n(x)$, where x is uniformly distributed in D_n). Let the sequence of functions $\{b_n : D_n \mapsto I^1\}_{n\in\mathbb{N}}$ be a hard-core predicate for F. Informally, this means that given $F_n(x)$ for a uniformly distributed x, it is hard to guess $b_n(x)$ with probability which is non-negligibly better than half. We can now define a collection of weak pseudo-random functions $G = \{G_n\}_{n\in\mathbb{N}}$ in the following way:

For every $x \in I^n$, denote by $D_n(x)$ the element in D_n sampled using x as the random bits. For every key i of $f_i \in F_n$, define $g_i : I^n \mapsto I^1$ as follows:

$$\forall x \in I^n, g_i(x) \stackrel{\text{def}}{=} b_n(f_i^{-1}(D_n(x)))$$

(Note that computing $g_i(x)$ requires knowledge of the trapdoor-key $t(i)$.) Let G_n be the random variable that assumes as values the functions g_i with the probability space induced by the distribution over the keys in F_n.

Claim 4.1.4 *(without proof) If F is a collection of trapdoor one-way permutations and $\{b_n\}_{n\in\mathbb{N}}$ is a hard-core predicate for F, then the function ensemble G which is defined above is a collection of weak pseudo-random functions.*

4.1.8 Number-Theoretic Constructions of Pseudo-Random Synthesizers

In this section we present several NC^1 (or actually, TC^0) constructions of pseudo-random synthesizers based on concrete, frequently-used, intractability assumptions. The first construction is at least as secure as the *computational* Diffie-Hellman [50] assumption. Since the Diffie-Hellman assumption modulo a composite is not stronger than factoring [95, 139] (see also Section 3.2) this implies a construction that is at least as secure as Factoring. Finally, we show two constructions that are at least as secure as the RSA assumption [126]. Although the RSA assumption is not weaker than factoring, the constructions based on RSA might have other advantages. For example, under the assumption that $\Omega(n)$ least-significant bits are simultaneously hard for RSA, we get pseudo-random synthesizers with linear output length. In addition, the constructions based on RSA and their proof of security use several interesting ideas that might be useful elsewhere.

The constructions of NC^1 pseudo-random synthesizers imply constructions of NC^2 pseudo-random functions (in fact we get TC^1 functions). In Section 4.2 we present more efficient direct constructions of pseudo-random functions (that are in particular in TC^0). These constructions are either based on the *decisional* Diffie-Hellman assumption or on factoring. Some of the ideas and tools used in Section 4.2 are similar to those used here.

We first address some issues that are common to all constructions presented here.

The evaluation of our pseudo-random synthesizers in NC^1 relies on a preprocessing stage. This stage can be performed as part of the (sequential) key-generating algorithm. In this idea we follow the work of Kearns and Valiant, [80]. In their context, the additional data is "forced" into the input whereas in our context it is added to the key.

The analysis of the parallel-time complexity of the synthesizers uses previous results on the parallel-time complexity of arithmetic operations (see Karp and Ramachandran [79] for a survey). In particular, we use the result of Beame, Cook and Hoover, [12]. They showed that iterated multiplication (multiplying n numbers of length n) and additional related operations can be performed by log-depth circuits (these circuits can be constructed efficiently, though sequentially). The results of [12] enable the computation of modular exponentiation in NC^1 given preprocessing that only depends on the base. This follows from the fact that

computing $b^e \bmod N$ is reduced to an iterated multiplication (and an additional modular reduction) given the values $b^i \bmod N$ (for $0 \leq i \leq$ the length of e).

The pseudo-random synthesizers constructed in this section are Boolean functions[2]. Section 4.1.3 showed two methods for expanding the output-length of pseudo-random synthesizers. The method of Lemma 4.1.1 requires a pseudo-random generator that expands the input by a factor of two. A natural choice for this purpose (in the case of the synthesizers which are described in this section) is the pseudo-random generators of Blum, Blum and Shub [24] or the one of Hastad, Schrift and Shamir [70]. Given appropriate preprocessing, both generators can be computed in NC^1 and their security is based on the assumption that factoring integers (Blum-integers in [70]) is hard.

We note that all the constructions of this section give *collections* of pseudo-random synthesizers. However, the security of theses synthesizers does not rely on keeping their key private. As discussed in Section 4.1.4, this allows us to use a single synthesizer at all the levels of Construction 4.1.1 (and of Construction 4.1.2).

Common Tools

In our constructions, we use the result of Goldreich and Levin [65] which gives a hard-core predicate for "any" one-way function:

Theorem 4.1.9 *([65]) Let f be any one-way function. For every probabilistic polynomial-time algorithm, $\mathcal{A}$, for every polynomial, $p(\cdot)$ and all sufficiently large n,*

$$\Pr\left[\mathcal{A}(f(x), r) = r \odot x\right] < \frac{1}{2} + \frac{1}{p(n)},$$

where x and r are independently drawn from U_n (recall that $r \odot x$ denotes the inner product mod 2 *of r and x).*

In fact, we use their result in a slightly different context. Loosely speaking, if given $f(x)$ it is hard to compute $g(x)$ with non-negligible probability, then given $f(x)$ it is also hard to guess $g(x) \odot r$ with non-negligible advantage over $1/2$. To verify that the Goldreich-Levin result applies in this context, let us state a main technical lemma regarding the Goldreich-Levin-Rackoff reconstruction algorithm:[3]

Lemma 4.1.10 *There exists a probabilistic oracle machine $\mathcal{M}$ such that for any $x \in \{0,1\}^n$, any $\varepsilon > 0$, and any probabilistic machine $\mathbf{B}_x$, if*

$$\Pr_{r \in U_n}\left[\mathbf{B}_x(r) = r \odot x\right] \geq 1/2 + \varepsilon,$$

then $\mathcal{M}^{\mathbf{B}_x}(1^n, \varepsilon)$:

- *runs in time $(1/\varepsilon)^2 \cdot poly(n)$,*

[2] In fact, all these synthesizers can be made to output a logarithmic number of bits. Furthermore, given stronger assumptions they may output an even larger number of bits. See Remark 4.1.3 for an example

[3] The Goldreich-Levin Theorem is a constructive one that enables reconstruction of x given an algorithm for guessing $x \odot r$. See [60] for details; the algorithm there is due to Rackoff.

- *makes* $O((n/\varepsilon)^2)$ *queries to* $\mathbf{B}_x$, *and*
- *outputs a list* L *of* $O((1/\varepsilon)^2 \cdot n)$ *values such that*

$$\Pr[x \in L] \geq 1/2.$$

Given this lemma, it is not hard to verify that if there exists an efficient machine that on input $f(x)$ guesses $g(x) \odot r$ with non-negligible advantage over $1/2$, then there also exists an efficient machine $\mathcal{M}$ that on input $f(x)$ computes $g(x)$ with non-negligible probability (by first producing polynomial many candidates for $g(x)$ and then choosing among them at random). In general, $\mathcal{M}$ may not be able to verify whether it got the right value $g(x)$ (unlike the case where $g(x) = x$, i.e. that f is a one-way function). However, in the contexts of the Diffie-Hellman assumption, Shoup [140] has shown how to increase the certainty of a machine such as $\mathcal{M}$ using the random-self reducibility of the problem.

In addition to the Goldreich-Levin result, the proof of security for all the constructions uses the next-bit prediction tests of Blum and Micali [27]. The equivalence between pseudo-random ensembles and ensembles that pass all polynomial-time next-bit tests was shown by Yao [150].

The Diffie-Hellman Assumption

We now define a collection of pseudo-random synthesizers that are at least as secure as the *computational* Diffie-Hellman assumption (**DH-Assumption**). This assumption was introduced in the seminal paper of Diffie and Hellman [50] (as a requirement for the security of their key-exchange protocol). The validity of the DH-Assumption was studied quite extensively over the last two decades. A few notable representatives of this research are [28, 93, 140]. Maurer and Wolf [93] and Boneh and Lipton [28] have shown that in several settings the DH-Assumption is equivalent to the assumption that computing the discrete log is hard. In particular, for any specific prime P there is an efficient reduction (given some information that only depends on P) of the discrete log problem in $\mathbb{Z}_P^*$ to the DH-Problem in $\mathbb{Z}_P^*$. Shoup [140] has shown that the DH-Assumption holds against what he calls "generic"-algorithms. See Section 3.1 for more details on these results and on the DH-Assumption in general.

For concreteness, we state the DH-Assumption in the group $\mathbb{Z}_P^*$, where P is a prime. However, our construction works just as well given the DH-Assumption in other groups. We use this fact in Section 4.1.8 to get pseudo-random synthesizers which are at least as secure as Factoring. In order to formalize the DH-Assumption in $\mathbb{Z}_P^*$, we need to specify the distribution of P. One possible choice is to let P be a uniformly distributed prime of a given length. However, there are other possible choices. For example, it is not inconceivable that P can be fixed for any given length. As in Chapter 3, we keep our results general by letting P be generated by *some* polynomial-time algorithm IG_{DH} (where IG stands for instance generator):

Definition 4.1.6 (IG_{DH}) *The Diffie-Hellman instance generator,* IG_{DH}, *is a probabilistic polynomial-time algorithm such that on input* 1^n *the output of* IG_{DH} *is distributed over* n*-bit primes.*

In addition, we need to specify the distribution of a generator, g, of $\mathbb{Z}_P^*$. It can be shown that if the DH-Assumption holds for some distribution of g, then it also holds if we let g be a uniformly distributed generator of $\mathbb{Z}_P^*$ (since there exists a simple randomized-reduction of the DH-Problem for any g to the DH-Problem with a uniformly distributed g).

All exponentiations in the rest of this subsection are in $\mathbb{Z}_P^*$ (the definition of P will be clear by the context). To simplify the notation, we omit the expression "mod P" from now on. We can now formally state the DH-Assumption (for the instance generator given IG_{DH}):

Assumption 4.1.1 *(Diffie-Hellman [50]) For every probabilistic polynomial-time algorithm, $\mathcal{A}$, for every polynomial, $q(\cdot)$ and all sufficiently large n,*

$$\Pr\left[\mathcal{A}(P, g, g^a, g^b) = g^{ab}\right] < \frac{1}{q(n)},$$

where the distribution of P is $IG_{DH}(1^n)$, the distribution of g is uniform over the set of generators of $\mathbb{Z}_P^*$ and the distribution of $\langle a, b\rangle$ is $(U_n)^2$.

Based on this assumption we define a collection of $I^{2n} \mapsto I^1$ pseudo-random synthesizers, S_{DH}:

Definition 4.1.7 *For every n-bit prime, P, every generator, g, of $\mathbb{Z}_P^*$ and every $r \in I^n$, define $s_{P,g,r} : I^{2n} \mapsto I^1$ by:*

$$\forall x, y \in I^n, s_{P,g,r}(x, y) \stackrel{\text{def}}{=} g^{xy} \odot r.$$

Let S_n to be the random variable that assumes as values the functions $s_{P,g,r}$, where the distribution of P is $IG_{DH}(1^n)$, the distribution of g is uniform over the set of generators of $\mathbb{Z}_P^*$ and the distribution of r is U_n. The function ensemble S_{DH} is defined to be $\{S_n\}_{n\in\mathbb{N}}$.

Note that in Section 4.2.1 we show a direct and efficient construction of n-dimensional pseudo-random synthesizers based on the (stronger) *decisional* version of the DH-Assumption (which gives very efficient pseudo-random functions).

Theorem 4.1.11 *If the DH-Assumption (Assumption 4.1.1) holds, then S_{DH} is a collection of $I^{2n} \mapsto I^1$ pseudo-random synthesizers.*

Proof. It is obvious that $S_{DH} = \{S_n\}_{n\in\mathbb{N}}$ is efficiently computable. Assume that S_{DH} is not a collection of pseudo-random synthesizers. Then there exists a polynomial $m(\cdot)$ such that the ensemble $E = \{E_n\}$ is not pseudo-random, where $E_n = \mathbf{C}_{S_n}(X, Y)$ for X and Y that are independently drawn from $(U_n)^{m(n)}$. Therefore, there exists an efficient next-bit prediction test, $\mathcal{T}$, and a polynomial $q(\cdot)$ such that for infinitely many n it holds that:

> *Given a prefix of E_n of uniformly chosen length, $\mathcal{T}$ succeeds to predict the next bit with probability greater than $\frac{1}{2} + \frac{1}{q(n)}$.*

We now show how to use $\mathcal{T}$ in order to define an efficient algorithm $\mathcal{A}$ such that for infinitely many n,

$$\Pr\left[\mathcal{A}(P, g, g^a, g^b, r) = g^{ab} \odot r\right] > \frac{1}{2} + \frac{1}{q(n)},$$

where the distribution of P, g, a and b is as in the DH-Assumption and r is drawn from U_n. By Theorem 4.1.9, this means that g^{ab} can also be efficiently computed which contradicts the DH-Assumption and completes the proof of the lemma.

In the definition of $\mathcal{A}$ we use the fact that in order to compute $g^{xy} = (g^x)^y = (g^y)^x$ it is enough to either know g^x and y or g^y and x (i.e. it is not required to know both x and y). This enables $\mathcal{A}$ to define a matrix which is distributed according to E_n such that one of its entries is $g^{ab} \odot r$ (the value $\mathcal{A}$ tries to guess) and all other entries can be computed by $\mathcal{A}$. It is now possible for $\mathcal{A}$ to guess $g^{ab} \odot r$ by invoking $\mathcal{T}$ on the appropriate prefix of this matrix.

In more details, on input $\langle P, g, g^a, g^b, r\rangle$ the algorithm $\mathcal{A}$ is defined as follows:

1. Uniformly choose $1 \leq i, j \leq m(n)$.
2. Define $X = \{x_1, \ldots, x_{m(n)}\}$ and $Y = \{y_1, \ldots, y_{m(n)}\}$ by setting $x_i = a$, $y_j = b$ and independently drawing all other values from U_n.
3. Define $B = (b_{u,v})_{u,v=1}^{m(n)}$ to be $\mathbf{C}_{s_{P,g,r}}(X, Y)$.

   Note that $\mathcal{A}$ knows all the values of X and Y *except* x_i and y_j. Therefore, $\mathcal{A}$ can compute all the entries of B *except* $b_{i,j} = g^{ab} \odot r$.
4. Invoke $\mathcal{T}$ and feed it with all the entries of B up to $b_{i,j}$ (i.e. the first $i-1$ rows and the first $j-1$ entries of the i^{th} row).
5. Output $\mathcal{T}$'s prediction of $b_{i,j}$.

It is obvious that $\mathcal{A}$ is efficient. Furthermore, since the distribution of B is exactly E_n, it is immediate that for infinitely many n, $\Pr\left[\mathcal{A}(P, g, g^a, g^b, r) = g^{ab} \odot r\right] > \frac{1}{2} + \frac{1}{q(n)}$, where the distribution of P, g, a, b and r is as above. This contradicts the DH-Assumption and proves the lemma. □

Corollary 4.1.12 *If the DH assumption (Assumption 4.1.1) holds, then there exist pseudo-random functions that are computable in NC^2 (given a sequential precomputation which is part of the key-generating algorithm).*

Proof. By Theorem 4.1.11, given that the DH-Assumption holds, S_{DH} is a collection of pseudo-random synthesizers. If the key-generating algorithm precomputes $g^{2^i} \bmod P$ for $1 \leq i \leq n$, then the functions of S_{DH} can be evaluated in NC^1. This precomputation reduces any modular exponentiation (with g as the base) to an iterated multiplication and an additional modular reduction (see also the discussion at the beginning of this section). By Corollary 4.1.5, there exist pseudo-random functions in NC^2 (the key-generating algorithm in both cases is sequential). □

Remark 4.1.2 *Assume that $IG_{DH}(1^n)$ has a single possible value, P, for every n. Then S_{DH} can be transformed into a synthesizer rather than a collection of synthesizers. In this case, the key-generating algorithm of the pseudo-random functions we get is in "non-uniform" NC.*

Composite Diffie-Hellman Assumption and Factoring

The collection of pseudo-random synthesizers, S_{DH}, is at least as secure as the DH-Assumption *modulo a prime*. As mentioned above, the DH-Assumption in any other group gives a corresponding construction of pseudo-random synthesizers with practically the same proof of security. We now consider the DH-Assumption *modulo a Blum-integer* (**composite DH-Assumption**). McCurley [95] and Shmuely [139] have shown that the composite DH-Assumption is implied by the assumption that factoring Blum-integers is hard. See also Section 3.2, for definitions and proof that are more consistent with our setting. We therefore get a simple construction of pseudo-random synthesizers which is at least as secure as Factoring. In the subsequent, we give the relevant definitions and claims (for better readability we repeat some of the definitions from Section 3.2). We omit the proofs (since they are practically the same as in Section 4.1.8).

To formalize the composite DH-Assumption we let this composite be generated by *some* polynomial-time algorithm FIG. We restrict the output of FIG to the set of Blum-integers. This restriction is quite standard and it is meant to simplify the reduction of the composite DH-Assumption to factoring.

Definition 4.1.8 (FIG) *The factoring instance generator, FIG, is a probabilistic polynomial-time algorithm such that on input 1^n its output, N, is distributed over $2n$-bit integers, where $N = P \cdot Q$ for two n-bit primes, P and Q, such that $P \equiv Q \equiv 3 \bmod 4$ (such an integer is known as a Blum-integer).*

We note that it is essential for $FIG(1^n)$ to have super-polynomial number of possible values since otherwise factoring would be non-uniformly easy (in this respect it is very different from the case of the Diffie-Hellman instance generator, IG_{DH}). One natural distribution of $FIG(1^n)$ that has this property is the uniform distribution over $2n$-bit Blum-integers.

All exponentiations in the rest of this subsection are in $\mathbb{Z}_N^*$ (the definition of N will be clear by the context). To simplify the notation, we omit the expression "mod N" from now on. We can now define both the composite DH-Assumption and the assumption that factoring Blum-integers is hard (for the instance generator FIG):

Assumption 4.1.2 *(Composite Diffie-Hellman) For every probabilistic polynomial-time algorithm, $\mathcal{A}$, for every polynomial, $q(\cdot)$ and all sufficiently large n,*

$$\Pr\left[\mathcal{A}(N, g, g^a, g^b) = g^{ab}\right] < \frac{1}{q(n)},$$

where the distribution of N is $FIG(1^n)$, the distribution of g is uniform over the set of quadratic-residues in $\mathbb{Z}_N^*$ and the distribution of $\langle a, b\rangle$ is $(U_{2n})^2$.

Assumption 4.1.3 *(Factoring) For every probabilistic polynomial-time algorithm, $\mathcal{A}$, for every polynomial, $q(\cdot)$ and all sufficiently large n,*

$$\Pr[\mathcal{A}(P \cdot Q) \in \{P, Q\}] < \frac{1}{q(n)},$$

where the distribution of $N = P \cdot Q$ is $FIG(1^n)$.

We define a collection of $I^{4n} \mapsto I^1$ pseudo-random synthesizers, S_F (in analogy to the definition of S_{DH}):

Definition 4.1.9 *For every $2n$-bit Blum-integer, N, every quadratic-residue, g, in $\mathbb{Z}_N^*$ and every $r \in I^{2n}$, define $s_{N,g,r} : I^{2n} \mapsto I^1$ by:*

$$\forall x, y \in I^{2n}, s_{N,g,r}(x, y) \stackrel{\text{def}}{=} g^{xy} \odot r.$$

Let S_n to be the random variable that assumes as values the functions $s_{N,g,r}$, where the distribution of N is $FIG(1^n)$, the distribution of g is uniform over the set of quadratic-residues in $\mathbb{Z}_N^*$ and the distribution of r is U_{2n}. The function ensemble S_{DH} is defined to be $\{S_n\}_{n\in\mathbb{N}}$.

In the same way Theorem 4.1.11 was proven we get that:

Theorem 4.1.13 *If the composite DH-Assumption (Assumption 4.1.2) holds, then S_F is a collection of $I^{4n} \mapsto I^1$ pseudo-random synthesizers.*

As shown in [95, 139] and in Theorem 3.2.1, the composite DH-Assumption (Assumption 4.1.2) is implied by the factoring assumption (Assumption 4.1.3). We therefore get that:

Corollary 4.1.14 *(of Theorem 4.1.13) If factoring Blum-integers is hard (Assumption 4.1.3), then S_F is a collection of $I^{4n} \mapsto I^1$ pseudo-random synthesizers.*

Finally, we can conclude that:

Corollary 4.1.15 *If factoring Blum-integers is hard (Assumption 4.1.3), then there exist pseudo-random functions that are computable in NC^2 (given a sequential precomputation which is part of the key-generating algorithm).*

The RSA Assumption

We now define two collections of pseudo-random synthesizers under the assumption that the RSA-permutations of Rivest, Shamir and Adleman [126] are indeed one-way (i.e. under the assumption that it is hard to extract roots modulo a composite). This assumption is not weaker than the factoring assumption. However, the constructions based on RSA might have other advantages. For example, the second RSA-construction gives pseudo-random synthesizers with linear output length under the assumption that $\Omega(n)$ least-significant bits are simultaneously hard for RSA. Another reason to include these constructions is that they use several interesting techniques that might be useful elsewhere.

As was the case with the previous assumptions, we keep the definition of the RSA-Assumption general by using *some* polynomial-time instance generator, IG_{RSA} (our constructions of synthesizers however will impose further conditions on IG_{RSA}):

Definition 4.1.10 (IG_{RSA}) *The RSA instance generator, IG_{RSA}, is a probabilistic polynomial-time algorithm such that on input 1^n its output is distributed over pairs $\langle N, e\rangle$. Where $N = P \cdot Q$ is a $2n$-bit integer, P and Q are two n-bit primes and $e \in \mathbb{Z}_{\varphi(N)}^*$ (i.e. e is relatively prime to the order of $\mathbb{Z}_N^*$ which is denoted by $\varphi(N)$).*

All exponentiations in the rest of this subsection are in $\mathbb{Z}_N^*$ (the definition of N will be clear by the context). To simplify the notation, we omit the expression "mod N" from now on. We can now define the RSA-Assumption (for the instance generator given IG_{RSA}):

Assumption 4.1.4 *(RSA [126]) For every probabilistic polynomial-time algorithm, $\mathcal{A}$, for every polynomial, $q(\cdot)$ and all sufficiently large n,*

$$\Pr\left[\mathcal{A}(N, e, m^e) = m\right] < \frac{1}{q(n)},$$

where the distribution of $\langle N, e\rangle$ is $IG_{RSA}(1^n)$ and m is uniformly distributed in $\mathbb{Z}_N^*$.

The RSA-Assumption gives a collection of trapdoor one-way permutations: the public key is $\langle N, e\rangle$, the function $f_{N,e}$ is defined by $f_{N,e}(m) = m^e$ and the trapdoor-key is $\langle N, e, d\rangle$, where $d = e^{-1} \bmod \mathbb{Z}^*_{\varphi(N)}$ (which enables efficient inversion by the formula $m = (m^e)^d$). In Section 4.1.7 we showed a general construction of pseudo-random synthesizers out of trapdoor one-way permutations. However, a straightforward application of this construction to the RSA collection gives very inefficient synthesizers. In the following few paragraphs we describe this construction, the reasons it is inefficient and some of the ideas and tools that allow us to get more efficient synthesizers (which are also computable in NC^1).

Applying the construction of Section 4.1.7 to the RSA collection (using the Goldreich-Levin [65] hard-core predicate) gives the following collection of synthesizers: The key of each synthesizer is a uniformly distributed string r and for every $x, y \in I^n$, $s_r(x, y) \stackrel{\text{def}}{=} m^d \odot r$, where x samples the trapdoor-key $\langle N, e, d\rangle$ and y samples a uniformly chosen element $m \in \mathbb{Z}_N^*$. The most obvious drawback of this definition is that computing the value $s_r(x, y)$ consists of sampling an RSA trapdoor-key. In particular, computing $s_r(x, y)$ consists of sampling a Blum-integer, N. This (rather heavy) operation might be acceptable as part of the key-generating algorithm of the pseudo-random synthesizers (or functions) but is extremely undesirable as part of their evaluation.

In the direct constructions of pseudo-random synthesizers based on RSA, we manage to "push" the composite N into the key of the synthesizers (thus overcoming the drawback described in the previous paragraph). Nevertheless, we are still left with the following problem: Computing m^d in NC^1 requires precomputation that depends on m. To enable this precomputation, it seems that m needs to be part of the key as well. However, in the construction which is described above, m depends on the input and is uniformly distributed for a random input. In order to overcome this problem, we show a method of sampling m almost uniformly at random in a way that facilitates the necessary preprocessing. This method uses the *subset product* functions. We first define these functions and then describe the way they are used in our context.

Definition 4.1.11 *Let G be a finite group and let $\vec{y} = \{y_1, \ldots, y_n\}$ be an n-tuple of elements in G. For any n-bit string, $x = x_1 \ldots x_n$, define the subset product $SP_{G,\vec{y}}(x)$ to be the product in G of the elements y_i such that $x_i = 1$.*

The following lemma was shown by Impagliazzo and Naor [74] and is based on the leftover hash lemma of [71, 78]:

Lemma 4.1.16 *([74]) Let G be a finite group, $n > c \log |G|$ and $c > 1$. Then for all but an exponentially small (in n) fraction of the choices of $\vec{y} \in (G)^n$, the distribution $SP_{G,\vec{y}}(U_n)$ is statistically indistinguishable within an exponentially small amount from the uniform distribution over G.*

Let N be a $2n$-bit integer, Lemma 4.1.16 gives a way of defining a collection of functions $\{f_k : I^{3n} \mapsto \mathbb{Z}_N^*\}_k$ which solves the problem of sampling an almost uniformly distributed element $m \in \mathbb{Z}_N^*$ for which m^d can be computed in NC^1. This collection is $\{f_{\vec{g}}\}_{\vec{g}} = \{SP_{\mathbb{Z}_N^*,\vec{g}}\}_{\vec{g}}$, where $\vec{g} = \{g_1, \ldots, g_{3n}\}$ is a sequence of $3n$ elements in $\mathbb{Z}_N^*$. The functions $\{f_{\vec{g}}\}_{\vec{g}}$ have the following properties:

1. For almost all choices of the key $\vec{g}$ we have that $f_{\vec{g}}(U_{3n})$ is almost uniformly distributed in $\mathbb{Z}_N^*$.
2. Following preprocessing that depends only on the key, $\vec{g}$, each value $(f_{\vec{g}}(x))^y$ can be computed in NC^1. The values that need to be precomputed are $g_i^{2^j}$, where $1 \leq i \leq 3n$ and $0 \leq j \leq$ the length of y. With these values, the computation of $(f_{\vec{g}}(x))^y$ is reduced to a single iterated multiplication (and an additional modular reduction).

The First RSA Construction

For our first RSA construction we need to assume that it is hard to extract the e^{th} root modulo a composite *when e is a large prime*. To formalize this, we assume that for every possible value $\langle N, e\rangle$ of $IG_{RSA}(1^n)$ we have that e is a $2n$-bit prime (which in particular means that $e \in \mathbb{Z}_{\varphi(N)}^*$). Based on this version of the RSA-Assumption we define a collection of $I^{6n} \mapsto I^1$ pseudo-random synthesizers, S_{RSA1}:

Definition 4.1.12 *Let N be a $2n$-bit integer, let $\vec{g} = \{g_1, \ldots, g_{3n}\}$ be a sequence of $3n$ elements in $\mathbb{Z}_N^*$ and let r be a $2n$-bit string. Define the function $s_{N,\vec{g},r} : I^{6n} \mapsto I^1$ by:*

$$\forall x, y \in I^{3n}, s_{N,\vec{g},r}(x, y) \stackrel{\text{def}}{=} \left(SP_{\mathbb{Z}_N^*,\vec{g}}(x)\right)^y \odot r.$$

Let S_n to be the random variable that assumes as values the functions $s_{N,\vec{g},r}$, where the distribution of N is induced by $IG_{RSA}(1^n)$ and $\vec{g}$ and r are uniformly distributed in their range. The function ensemble S_{RSA1} is defined to be $\{S_n\}_{n\in\mathbb{N}}$.

Note that the only reason we let y be a $3n$-bit number (instead of a $2n$-bit number) is to make both inputs of $s_{N,\vec{g},r}$ be of the same length (which not really necessary for our constructions).

Theorem 4.1.17 *If the RSA-Assumption (Assumption 4.1.4) holds when for every possible value $\langle N, e\rangle$ of $IG_{RSA}(1^n)$ we have that e is a $2n$-bit prime. Then S_{RSA1} is a collection of $I^{6n} \mapsto I^1$ pseudo-random synthesizers.*

Proof. It is obvious that $S_{RSA1} = \{S_n\}_{n\in\mathbb{N}}$ is efficiently computable. Assume that S_{RSA1} is not a collection of pseudo-random synthesizers. Then there exists a polynomial $m(\cdot)$ such that the ensemble $E = \{E_n\}$ is not pseudo-random, where $E_n = \mathbf{C}_{S_n}(X, Y)$ for X and Y that are independently drawn from $(U_{3n})^{m(n)}$. Therefore, there exists an efficient next-bit prediction test, $\mathcal{T}$, and a polynomial $q(\cdot)$ such that for infinitely many n it holds that:

Given a prefix of E_n of uniformly chosen length, $\mathcal{T}$ succeeds to predict the next bit with probability greater than $\frac{1}{2} + \frac{1}{q(n)}$.

We now show how to use $\mathcal{T}$ in order to define an efficient algorithm $\mathcal{A}$ such that for infinitely many n,

$$\Pr\left[\mathcal{A}(N, e, m^e, z, r) = m^z \odot r\right] > \frac{1}{2} + \frac{1}{2q(n)},$$

where the distribution of N, e and m is as in the RSA-Assumption (with the restriction that e is a $2n$-bit prime), r is drawn from U_{2n} and z is uniformly distributed over the set of $3n$-bit integers that are relatively prime to e. By Theorem 4.1.9, this means that m^z can also be efficiently computed with non-negligible probability. Following Shamir [137], we note that given any z such that $\gcd(e, z) = 1$ and given m^z it is easy to compute m. The reason is that if $gcd(e, z) = 1$ then m can be computed by the formula $m = (m^e)^a (m^z)^b$ where $a, b \in \mathbb{Z}$ satisfy that $ae + bz = 1$ (and can be efficiently computed as well). Thus, the existence of such an algorithm $\mathcal{A}$ contradicts the RSA-Assumption and completes the proof of the lemma.

The algorithm $\mathcal{A}$ defines a matrix B which is *almost* identically distributed as E_n. One of the entries of B is $m^z \odot r$ (the value $\mathcal{A}$ tries to guess) and all other entries can be computed by $\mathcal{A}$. It is now possible for $\mathcal{A}$ to guess $m^z \odot r$ by invoking $\mathcal{T}$ on the appropriate prefix of this matrix. In more details, on input $\langle N, e, m^e, z, r\rangle$ the algorithm $\mathcal{A}$ is defined as follows:

1. Uniformly choose $1 \leq i, j \leq m(n)$.

2. Define the values $\{h_1, \ldots, h_{m(n)}\}$ and $\{d_1, \ldots, d_{m(n)}\}$ by setting $h_i = m$, uniformly drawing all other h_u's from $\mathbb{Z}_N^*$, setting $d_j = z \cdot e^{-1} \bmod \varphi(N)$ and drawing all other d_v's from U_{3n}.

3. Define $B = (b_{u,v})_{u,v=1}^{m(n)}$ by setting $b_{u,v} = \left(((h_u)^e)^{d_v}\right) \odot r$.

   Note that $\mathcal{A}$ can compute any entry $b_{u,v}$ *except* $b_{i,j} = m^z \odot r$. The reason is that if $v \neq j$ then $\mathcal{A}$ knows both d_v and $(h_u)^e$ and if $u \neq i$ then $\mathcal{A}$ can compute $b_{u,j} = \left(((h_u)^e)^{z \cdot e^{-1}}\right) \odot r = (h_u)^z \odot r$ since it knows both h_u and z.

4. Invoke $\mathcal{T}$ and feed it with all the entries of B up to $b_{i,j}$ (i.e. the first $i-1$ rows and the first $j-1$ entries of the i^{th} row).

5. Output $\mathcal{T}$'s prediction of $b_{i,j}$.

It is obvious that $\mathcal{A}$ is efficient. In order to complete the proof, we need to show that if $\langle N, e, m^e, z, r\rangle$ are distributed as above, then B and E_n are of exponentially small statistical distance. This would imply that (for infinitely many n) if we feed $\mathcal{T}$ with the bits of B up to $b_{i,j}$ it predicts $b_{i,j} = m^z \odot r$ with probability greater than, say, $\frac{1}{2} + \frac{1}{2q(n)}$. As argued above, this would contradict the RSA-Assumption and would complete the proof.

To see that B and E_n are indeed statistically close notice that:

- Since $e \in \mathbb{Z}_{\varphi(N)}^*$ and $\forall 1 \leq u \leq m(n)$ the value m_u is uniformly distributed in $\mathbb{Z}_N^*$, we have that $\forall 1 \leq u \leq m(n)$ the value $(m_u)^e$ is also uniformly distributed in $\mathbb{Z}_N^*$.

By Lemma 4.1.16, we therefore have that the distribution of $\{(m_u)^e\}_{1\leq u\leq m(n)}$ is statistically close to the distribution of $\{SP_{\mathbb{Z}_N^*,\vec{g}}(x_u)\}_{1\leq u\leq m(n)}$ for uniformly distributed values $\{x_1,\ldots,x_{m(n)}\}\in(I^{3n})^{m(n)}$ and $\vec{g}=\{g_1,\ldots,g_{3n}\}\in(\mathbb{Z}_N^*)^{3n}$.

- For z that is chosen from U_{3n} the distribution of $z\cdot e^{-1} \bmod \varphi(N)$ and $U_{3n} \bmod \varphi(N)$ are statistically close. Since e is a large prime, even given the restriction that z is relatively prime to e these distributions are statistically close.

Given these two observations it is easy to verify that B and E_n are indeed of exponentially small statistical distance. □

Claim 4.1.5 *The functions in S_{RSA1} can be evaluated in NC^1 (given a sequential precomputation which is part of the key-generating algorithm).*

Proof. Given that the key-generating algorithm precomputes $(g_i)^{2^j}$ for $1\leq i,j\leq 3n$, the evaluation of functions in S_{RSA1} is reduced to an iterated multiplication and an additional modular reduction. □

The Second RSA Construction

The security of S_{RSA1} depends on the assumption that it is hard to extract the e^{th} root modulo a composite, where e is a large prime. Here, we define another collection of synthesizers under the assumption that it is hard to extract the e^{th} root modulo a composite, N, without any restriction on the distribution of $e\in\mathbb{Z}^*_{\varphi(N)}$. However, we introduce a new restriction on the possible values of the composite N:

Definition 4.1.13 *Let G_n be the set of $2n$-bit integers $N=P\cdot Q$ such that P and Q are two n-bit primes and $\varphi(N)$ has no odd factor smaller than n^2.*

It is easy to verify that if $N\in G_n$ then a sequence of $3n$ uniformly-chosen odd-values, $\vec{d}=\{d_1,\ldots,d_{3n}\}\in\mathbb{Z}_N$, have a constant probability to be in $\left(\mathbb{Z}^*_{\varphi(N)}\right)^{3n}$ (indeed this probability is $1-o(1)$). By Lemma 4.1.16, given such a sequence, it is easy to almost uniformly sample any polynomial number of values in $\mathbb{Z}^*_{\varphi(N)}$ *even without knowledge of* $\varphi(N)$.[4] This can be done by using the subset product function $SP_{\mathbb{Z}^*_{\varphi(N)},\vec{d}}$. Notice that here the subset product function serves an additional role to the one already described above.

Sieve theory shows that G_n is not too sparse. For example, denote by $B(x)$ the number of primes p smaller than x such that $(p-1)/2$ is the product of two primes each of which is larger than $p^{1/4}$. Then there exists a positive constant c such that $B(x)\geq\frac{cx}{\log^2 x}$. See [121] for several results of this sort (which are more than sufficient for our purpose). As a result we get that: (a) If the RSA-assumption holds for a uniformly distributed value of N, then it also holds under the restriction $N\in G_n$. (b) The uniform distribution over G_n can

[4] Provided that we allow the representation of values in $\mathbb{Z}^*_{\varphi(N)}$ as large integers. That is, we let any integer v represent $u=v \bmod \varphi(N)$. We note that such a value v is just as good as u for our proof *as long as v is bounded by poly(N)*. In particular, using such a representation it is possible to compute $SP_{\mathbb{Z}^*_{\varphi(N)},\vec{d}}$ *even without knowledge of* $\varphi(N)$.

be efficiently sampled (using Bach's algorithm [11]). Given (a) and (b) it seems that this restriction is rather reasonable.

Based on the RSA-Assumption with the restriction that $N \in G_n$, we define a collection of $I^{6n} \mapsto I^1$ pseudo-random synthesizers, S_{RSA2}. In the definition of S_{RSA2}, we use the least-significant bit (LSB) instead of the Goldreich-Levin hard-core bit. Alexi et. al. [4] showed that LSB is a hard-core bit for RSA. Fischlin and Schnorr [54] have recently provided a stronger reduction for this bit.

Definition 4.1.14 *Let N be a $2n$-bit integer, let $\vec{g} = \{g_1, \ldots, g_{3n}\}$ be a sequence of $3n$ elements in $\mathbb{Z}_N^*$ and let $\vec{d} = \{d_1, \ldots, d_{3n}\}$ be a sequence of $3n$ elements in $\mathbb{Z}_{\varphi(N)}^*$. Define the function $s_{N,\vec{g},\vec{d}} : I^{6n} \mapsto I^1$ by:*

$$\forall x, y \in I^{3n},\ s_{N,\vec{g},\vec{d}}(x,y) \stackrel{\text{def}}{=} LSB\left(\left(SP_{\mathbb{Z}_N^*,\vec{g}}(x)\right)^{\left(SP_{\mathbb{Z}_{\varphi(N)}^*,\vec{d}}(y)\right)}\right).$$

Let S_n to be the random variable that assumes as values the functions $s_{N,\vec{g},\vec{d}}$, where the distribution of N is induced by $IG_{RSA}(1^n)$ and $\vec{g}$ and $\vec{d}$ are uniformly distributed in their range. The function ensemble S_{RSA2} is defined to be $\{S_n\}_{n\in\mathbb{N}}$.

Theorem 4.1.18 *If the RSA-Assumption (Assumption 4.1.4) holds when for every possible value $\langle N, e\rangle$ of $IG_{RSA}(1^n)$ we have that $N \in G_n$. Then S_{RSA2} is a collection of $I^{6n} \mapsto I^1$ pseudo-random synthesizers.*

Proof. It is obvious that $S_{RSA2} = \{S_n\}_{n\in\mathbb{N}}$ is efficiently computable. Assume that S_{RSA2} is not a collection of pseudo-random synthesizers. Then there exists a polynomial $m(\cdot)$ such that the ensemble $E = \{E_n\}$ is not pseudo-random, where $E_n = \mathbf{C}_{S_n}(X, Y)$ for X and Y that are independently drawn from $(U_{3n})^{m(n)}$. Therefore, there exists an efficient next-bit prediction test, $\mathcal{T}$, and a polynomial $q(\cdot)$ such that for infinitely many n it holds that:

> *Given a prefix of E_n of uniformly chosen length, $\mathcal{T}$ succeeds to predict the next bit with probability greater than $\frac{1}{2} + \frac{1}{q(n)}$.*

We now show how to use $\mathcal{T}$ in order to define an efficient algorithm $\mathcal{A}$ such that for infinitely many n,

$$\Pr\left[\mathcal{A}(N, e, m^e) = LSB(m)\right] > \frac{1}{2} + \frac{1}{2q(n)},$$

where the distribution of N, e and m is as in the RSA-Assumption (with the restriction that $N \in G_n$). By [4], this contradicts the RSA-Assumption and completes the proof of the lemma.

The basic idea in the definition of the algorithm $\mathcal{A}$ is similar to the proof of Theorem 4.1.17: the algorithm $\mathcal{A}$ defines a matrix B which is *almost* identically distributed as E_n. One of the entries of B is $LSB(m)$ (the value $\mathcal{A}$ tries to guess) and all other entries can be computed by $\mathcal{A}$. It is now possible for $\mathcal{A}$ to guess $LSB(m)$ by invoking $\mathcal{T}$ on the appropriate prefix of this matrix. In more details, on input $\langle N, e, m^e\rangle$ as above, the algorithm $\mathcal{A}$ is defined as follows:

1. Uniformly choose $1 \leq i, j \leq m(n)$.
2. Define $\hat{e}$ to be $e \cdot d$, where d is almost uniformly distributed in $\mathbb{Z}^*_{\varphi(N)}$ (such a value d can be sampled because $N \in G_n$).

   Note that $\hat{e}$ is almost uniformly distributed in $\mathbb{Z}^*_{\varphi(N)}$
3. Define the values $\{h_1, \ldots, h_{m(n)}\}$ and $\{d_1, \ldots, d_{m(n)}\}$ by setting $h_i = m$, uniformly drawing all other h_u's from $\mathbb{Z}^*_N$, setting $d_j = \hat{e}^{-1} \bmod \varphi(N)$ and sampling all other d_v's almost uniformly from $\mathbb{Z}^*_{\varphi(N)}$.
4. Define $B = (b_{u,v})_{u,v=1}^{m(n)}$ by setting $b_{u,v} = LSB\left(\left((h_u)^{\hat{e}}\right)^{d_v}\right)$.

   Note that $\mathcal{A}$ can compute any entry $b_{u,v}$ *except* $b_{i,j} = LSB(m)$. The reason is that if $v \neq j$ then $\mathcal{A}$ knows both d_v and $(h_u)^{\hat{e}}$ and if $u \neq i$ then $\mathcal{A}$ can compute $b_{u,j} = LSB\left(\left((h_u)^{\hat{e}}\right)^{\hat{e}^{-1}}\right) = LSB((h_u))$ since it knows h_u.
5. Invoke $\mathcal{T}$ and feed it with all the entries of B up to $b_{i,j}$ (i.e. the first $i-1$ rows and the first $j-1$ entries of the i^{th} row).
6. Output $\mathcal{T}$'s prediction of $b_{i,j}$.

It is obvious that $\mathcal{A}$ is efficient. It is also easy to verify that if $\langle N, e, m^e \rangle$ is distributed as above, then B and E_n are of exponentially small statistical distance. Therefore, for infinitely many n, if we feed $\mathcal{T}$ with the bits of B up to $b_{i,j}$ it predicts $b_{i,j} = LSB(m)$ with probability greater than, say, $\frac{1}{2} + \frac{1}{2q(n)}$. As argued above, this contradicts the RSA-Assumption and completes the proof of the lemma. □

Claim 4.1.6 *The functions in S_{RSA2} can be evaluated in NC^1 (given a sequential precomputation which is part of the key-generating algorithm).*

Proof. Given that the key-generating algorithm precomputes $(g_i)^{2^j}$ for $1 \leq i, j \leq 3n$, the evaluation of functions in S_{RSA2} is reduced to two iterated multiplication and two modular reductions. □

Remark 4.1.3 *Since Alexi et. al. [4] showed that the $\log n$ least-significant bits are simultaneously hard for RSA we can adjust the functions in S_{RSA2} to output $\log n$ bits. If we make a stronger assumption, that $\Omega(n)$ bits are simultaneously hard for RSA, we get a direct construction of pseudo-random synthesizers with linear output size. Although the stronger assumption is not known to be equivalent to the RSA-Assumption it is still quite standard.*

4.1.9 Pseudo-Randomness and Learning-Theory

Synthesizers Based on Hard-to-Learn Problems

The "traditional" connection between cryptography and learning theory is using cryptographic assumptions to deduce computational non-learnability results. Blum, Furst, Kearns

and Lipton [23] have suggested that the other direction is interesting as well. They have shown how to construct several cryptographic primitives out of *hard-to-learn* functions, in a way that preserves the degree of parallelism of the functions. A major motivation for presenting such constructions is the simplicity of function classes that are believed to be hard for efficient learning.

We show that, under the definitions of [23], pseudo-random synthesizers can easily be constructed from distributions of functions that are hard to learn. Thus, by our constructions, two additional cryptographic primitives can be constructed in parallel out of hard-to-learn functions: (1) pseudo-random generators with large expansion ratio (without assuming, as in [23], that the functions are hard to learn with membership queries) and (2) pseudo-random functions.

There is a difference between standard learning-theory definitions and standard cryptographic definitions. Loosely speaking, a collection of concepts is hard to learn if for every efficient algorithm there exists a distribution over the concepts that is hard to learn *for this specific algorithm*. In cryptographic settings the order of quantifiers is reversed: the hard distribution should be hard for *every* efficient algorithm. In order for hard-to-learn problems to be useful in cryptographic settings an average-case learning model is introduced in [23].

Informally describing one of the definitions in [23], we can say that a distribution ensemble of functions, $F = \{F_n\}_{\in \mathbb{N}}$, is not *weakly predictable on the average* with respect to a distribution D on the inputs, if the following holds: There is no efficient algorithm that can predict $f(\tilde{x})$ with probability $\frac{1}{2} + \frac{1}{poly(n)}$, given $\tilde{x}$ and a polynomial sequence $\{\langle x_i, f(x_i)\rangle\}$, where f is distributed according to F_n and all the inputs are independently distributed according to D.

It is easy to verify that a distribution ensemble of functions, F, is not weakly predictable on the average with respect to the uniform distribution if and only if it is a collection of weak pseudo-random functions. Thus, by Lemma 4.1.7, such a distribution defines a pseudo-random synthesizer S, where $S(x, y)$ is simply $f_{\mathcal{I}_y(1^n)}(x)$ (recall that $f_{\mathcal{I}_y(1^n)}$ denotes the function that is sampled from F_n using y as random bits). Using S we can construct pseudo-random generators and pseudo-random functions. Moreover, by Lemma 4.1.1. the pseudo-random generator we construct may have a large expansion ratio ($n^{1-\varepsilon}$ for every $\varepsilon > 0$). The pseudo-random generator constructed in [23] under the same assumption has expansion ratio bounded above by $1 + 1/n$.

A Concrete Hard-to-Learn Problem

Consider the following distribution on functions with parameters k and n. Each function is defined by a uniformly distributed pair of disjoint sets $A, B \subset \{1, \ldots, n\}$, each of size k. Given an n-bit input, the output of the function is the exclusive-or of two values: the parity of the bits indexed by A and the majority of the bits indexed by B. In [23], it is estimated that these functions (for $k = \log n$) cannot be weakly predictable without using "profoundly" new ideas. If indeed this distribution of functions is not weakly predictable on the average (for any k), then it defines an extremely efficient synthesizer. Therefore, using our constructions, we get rather efficient parallel pseudo-random functions.

The Application of Pseudo-Random Functions to Learning Theory

As observed by Valiant [146], if a concept class contains pseudo-random functions, then we can deduce a very strong unlearnability result for this class. Informally, it means that there exists a distribution of concepts in this class that is hard for *every* learning algorithm, for *every* "non-trivial" distribution on inputs *even* when membership queries are allowed. Since no parallel pseudo-random functions were known before the current work, this observation could not have been applied to NC.

Nevertheless, other techniques based on cryptographic assumptions were used in [8, 80, 81] to show hardness results for NC^1 and TC^0. For example, Kharitonov [81] used the following fact: after preprocessing, a polynomial-length pseudo-random bit-string (based on [24]) can be produced in TC^0 (the length of the string can stay undetermined at the preprocessing stage). The existence of pseudo-random functions in NC (as shown in this chapter under several assumptions) is still of interest to computational learning theory because the result it implies is stronger than previous results. To briefly state the difference, we note that the results of [8, 80] use a very specific distribution on the inputs that is hard-to-learn and the results of [81] strongly rely on the order of quantifiers in learning-theory models which was mentioned above (e.g. for any given learning algorithm [81] shows a different hard concept which can still be easily learned by an algorithm which has a somewhat larger running-time).

4.2 Concrete Constructions of Pseudo-Random Functions

In this section we describe two related constructions of pseudo-random functions based on number-theoretic assumptions. The first construction gives pseudo-random functions iff the *decisional* version of the Diffie-Hellman assumption (**DDH-Assumption**) holds. The second construction is at least as secure as the assumption that factoring the so called Blum-integers is hard.[5] Having efficient pseudo-random functions based on factoring is very desirable since this is one of the most established concrete intractability assumption used in cryptography. The construction based on the DDH-Assumption is also attractive since these pseudo-random functions are even more efficient (in that they have a larger output size) and since the construction is *linear-preserving* (see Remark 4.2.1). To better understand the security of our constructions we include in Chapter 3 a study of the underlying assumptions. As discussed in Section 4.2.2, the constructions of this section are motivated by the construction of Section 4.1 (and in particular by the notion of k-dimensional synthesizers).

The pseudo-random functions that are constructed in this section are *efficient* (computing the value of the function at a given point is comparable with two modular exponentiations which is much more efficient than previous proposals), *have shallow depth* (given appropriate preprocessing of the key, the value of the functions at any given point can be computed in TC^0) and have *a simple algebraic structure*. The properties of our pseudo-random functions

[5] In fact we prove the security of the second construction based on a *generalized* version of the computational DH-Assumption (**GDH-Assumption**). However, breaking the GDH-Assumption modulo a composite would imply an efficient algorithm for factorization (as shown in Section 3.2 and [139]).

and their consequences are discussed at length in the introduction.

The notation and definitions used in this section are given in Chapter 2 (in particular we use definitions of pseudo-randomness from there). In addition, we rely on the definitions of the relevant assumptions from Chapter 3 (mainly on the definition of the DDH-Assumption).

Organization

In Section 4.2.1 we describe a construction of pseudo-random functions based on the DDH-Assumption, prove its security and consider its complexity. In Section 4.2.2 we define the GDH-Assumption and show a related construction of pseudo-random functions based on this assumption. In Section 4.2.3 we consider some of the possible features of our pseudo-random functions.

4.2.1 Construction of Pseudo-Random Functions

In this section we describe a construction of pseudo-random functions based on the DDH-Assumption, prove its security and consider its complexity. A related construction (based on a weaker assumption) is described in Section 4.2.2.

Construction and Main Result

Construction 4.2.1 *We define the function ensemble $F = \{F_n\}_{n\in\mathbb{N}}$. For every n, a key of a function in F_n is a tuple, $\langle P, Q, g, \vec{a}\rangle$, where P is an n-bit prime, Q a prime divisor of $P-1$, g an element of order Q in $\mathbb{Z}_P^*$ and $\vec{a} = \langle a_0, a_1, \ldots a_n\rangle$ a sequence of $n+1$ elements of $\mathbb{Z}_Q$. For any n-bit input, $x = x_1x_2\cdots x_n$, the function $f_{P,Q,g,\vec{a}}$ is defined by:*

$$f_{P,Q,g,\vec{a}}(x) \stackrel{\text{def}}{=} (g^{a_0})^{\prod_{x_i=1} a_i}.$$

The distribution of functions in F_n is induced by the following distribution on their keys: $\vec{a}$ is uniform in its range and the distribution of $\langle P, Q, g\rangle$ is $IG(1^n)$ (see Definition 3.1.1).

It is clear that F is efficiently computable (since IG is efficient). The pseudo-randomness property of F is the following:

Theorem 4.2.1 *Let $F = \{F_n\}_{n\in\mathbb{N}}$ be as in Construction 4.2.1. If the DDH-Assumption (Assumption 3.1.1) holds, then for every probabilistic polynomial-time oracle machine $\mathcal{M}$, every constant $\alpha > 0$, and all sufficiently large n,*

$$\left|\Pr[\mathcal{M}^{f_{P,Q,g,\vec{a}}}(P,Q,g) = 1] - \Pr[\mathcal{M}^{R_{P,Q,g}}(P,Q,g) = 1]\right| < \frac{1}{n^\alpha},$$

where in the first probability $f_{P,Q,g,\vec{a}}$ is distributed according to F_n and in the second probability, the distribution of $\langle P, Q, g\rangle$ is $IG(1^n)$ and $R_{P,Q,g}$ is uniformly chosen in the set of functions with domain $\{0,1\}^n$ and range $\langle g\rangle$ (the subgroup of $\mathbb{Z}_P^*$ generated by g).

Remark 4.2.1 *It is easy to verify from the proof of Theorem 4.2.1 that the following, more quantitative, version of the theorem also holds:*

Assume that there exists a probabilistic oracle machine with running time $t = t(n)$ that distinguishes $f_{P,Q,g,\vec{a}}$ from $R_{P,Q,g}$ (as in Theorem 4.2.1) with advantage $\varepsilon = \varepsilon(n)$. Then there exists a probabilistic algorithm with running time $poly(n) \cdot t(n)$ that breaks the DDH-Assumption with advantage $\varepsilon(n)/n$. Therefore, the reduction is linear-preserving *(see [89]). This reduction is rather unique in that the security of the functions does not significantly decrease when the* number of queries *the distinguisher makes increases.*

Given Theorem 4.2.1, we have that F is "almost" an efficiently computable pseudo-random function ensemble. There is one difference: A function $f_{P,Q,g,\vec{a}}$ in F_n has domain $\{0,1\}^n$ and range $\langle g \rangle$. Therefore, different functions in F_n have different ranges which deviates from the standard definition of pseudo-random functions. However, for many applications of pseudo-random functions this deviation does not present a problem (e.g., the applications of pseudo-random functions to private-key authentication and identification and their applications to digital signatures [17]). In addition, it is rather easy to construct from F pseudo-random functions under the standard definition. In order to show this, we need the following lemma which is a simple corollary of the leftover hash lemma [71, 78]:

Lemma 4.2.2 *Let n, ℓ and e be three positive integers such that $3e < \ell < n$. Let $X \subseteq \{0,1\}^n$ be a set of at least 2^ℓ elements and x uniformly distributed in X. Let H be a family of pair-wise independent, $\{0,1\}^n \mapsto \{0,1\}^{\ell-3e}$, hash functions. Then for all but a 2^{-e} fraction of $h \in H$ the uniform distribution over $\{0,1\}^{\ell-3e}$ and $h(x)$ are of statistical distance of at most 2^{-e}.*

Lemma 4.2.2 suggests the following construction:

Construction 4.2.2 *Let $\ell = \ell(n)$ be an integer-valued function such that for any output, $\langle P, Q, g \rangle$, of $IG(1^n)$ we have that $Q > 2^{\ell(n)}$. Let $F = \{F_n\}_{n \in \mathbb{N}}$ be as in Construction 4.2.1 and $\forall n$, let H_n be a family of pair-wise independent, $\{0,1\}^n \mapsto \{0,1\}^{\ell(n)/2}$, hash functions. We define the function ensemble $\tilde{F} = \{\tilde{F}_n\}_{n \in \mathbb{N}}$. For every n, a key of a function in $\tilde{F}_n$ is a pair, $\langle k, h \rangle$, where k is a key of a function in F_n and $h \in H_n$. For any n-bit input, x, the function $\tilde{f}_{k,h}$ is defined by:*

$$\tilde{f}_{k,h}(x) \stackrel{\text{def}}{=} h(f_k(x)).$$

The distribution of functions in $\tilde{F}_n$ is induced by the following distribution on their keys: h is uniform in H_n and the distribution of k is the same as the distribution of keys in F_n.

Note that choosing the range of the hash functions to be $\{0,1\}^{\ell(n)/2}$ is arbitrary. One can choose the range to be $\{0,1\}^{\ell(n)-e(n)}$ for any function $e(n)$ such that $2^{-e(n)/3}$ is negligible. Using Theorem 4.2.1 and Lemma 4.2.2 we can easily conclude:

Theorem 4.2.3 *If the DDH-Assumption (Assumption 3.1.1) holds, then $\tilde{F} = \{\tilde{F}_n\}_{n \in \mathbb{N}}$ (as in Construction 4.2.2) is an efficiently computable pseudo-random function ensemble.*

Remark 4.2.2 *From Theorem 4.2.1 and Lemma 4.2.2 it follows that $\tilde{F} = \{\tilde{F}_n\}_{n \in \mathbb{N}}$ remains indistinguishable from the uniform function-ensemble even when the distinguisher has access to $\langle P, Q, g \rangle$ and to h (as in the definition of functions in $\tilde{F}_n$).*

Proof of Security

There are a few possible approaches to proving Theorem 4.2.1. One approach is related to Construction 4.1.1 (and in particular to the concept of an n-dimensional synthesizer). Indeed, Construction 4.1.1 has motivated the constructions of Section 4.2 (the connection is described in Section 4.2.2). However, the proof we give here for Theorem 4.2.1 follows an analogous line to the proof of security for the GGM-Construction of pseudo-random functions [62]. This may seem surprising since the two constructions look very different. Nevertheless, in some sense, one may view our construction as a careful application (or a generalization) of the GGM-Construction. In the following few paragraphs we describe the similarities and differences between the two constructions.

Let G be a pseudo-random generator that doubles its input. Define G^0 and G^1 such that for any n-bit string x, both $G^0(x)$ and $G^1(x)$ are n-bit strings and $G(x) = \langle G^0(x), G^1(x)\rangle$. Under the GGM-Construction, the key of a pseudo-random function $f_s : \{0,1\}^n \mapsto \{0,1\}^n$ is a uniformly chosen n-bit string, s. For any n-bit input, $x = x_1 x_2 \cdots x_n$, the function f_s is defined by:

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

The definition of f_s can be thought of as a recursive labeling process of a depth-n binary tree. The key s is the label of the root and it induces a labeling of all the nodes in the tree. The labels of the 2^n leaves correspond to the 2^n different outputs of the function. In contrast, in our construction no tree appears in the design and no particular order is attached to the input bits. Nevertheless, we were able to relate the proof of security of the two constructions.

The DDH-Assumption implies a simple pseudo-random generator that practically doubles its input: $G_{P,Q,g,g^a}(b) \stackrel{\text{def}}{=} \langle g^b, g^{a\cdot b}\rangle$ (whose output is a pseudo-random pair of values in the subgroup generated by g) . It is tempting to use this generator for the GGM-Construction. However, a straightforward application of the GGM-Construction would give a rather inefficient function. We therefore suggest a slight change to the definition of the generator:

$$\tilde{G}_{P,Q,g,g^a}(g^b) = \langle \tilde{G}^0_{P,Q,g,g^a}(g^b), \tilde{G}^1_{P,Q,g,g^a}(g^b)\rangle \stackrel{\text{def}}{=} \langle g^b, g^{a\cdot b}\rangle.$$

At a first look this seems absurd: $\tilde{G}_{P,Q,g,g^a}$ is not efficiently computable unless the DH-Problem is easy. Therefore, if $\tilde{G}_{P,Q,g,g^a}$ is efficiently computable, then it is not pseudo-random. However, $\tilde{G}_{P,Q,g,g^a}$ has the following property that allows us to use a generalization of the GGM-Construction: $\tilde{G}_{P,Q,g,g^a}(g^b)$ is efficiently computable if either a or b are known. A more general way to state this is:

1. $\tilde{G}_{P,Q,g,g^a}$ is efficiently computable (on any input), given the random bits that were used to sample it (in particular, given a).

2. For any $\tilde{G}_{P,Q,g,g^a}$, it is easy to generate the distribution of its output, $\tilde{G}_{P,Q,g,g^a}(g^b)$, on a uniformly chosen input (this fact implies Lemma 4.2.4).

We now obtain the pseudo-random functions of Construction 4.2.1 using the GGM-Construction where at each level of the construction we use a different value, g^a, for the generator:

$$f_{P,Q,g,a_0,a_1,\ldots,a_n}(x) \stackrel{\text{def}}{=} \tilde{G}^{x_n}_{P,Q,g,g^{a_n}}(\cdots(\tilde{G}^{x_2}_{P,Q,g,g^{a_2}}(\tilde{G}^{x_1}_{P,Q,g,g^{a_1}}(g^{a_0}))\cdots).$$

We turn to the formal proof of Theorem 4.2.1. First we show (in Lemma 4.2.4) that a polynomial sample, $\langle \tilde{G}_{P,Q,g,g^a}(g^{b_1}), \ldots \tilde{G}_{P,Q,g,g^a}(g^{b_t})\rangle$ is pseudo-random iff a single sample, $\tilde{G}_{P,Q,g,g^a}(g^b)$, is pseudo-random. In preliminary versions of this work the proof of Lemma 4.2.4 used a hybrid-argument based on property (2) above (which is similar to the corresponding argument in [62]). However, Victor Shoup has pointed out that one can use the randomized-reduction of the DDH-Problem (see Section 3.1.3) for an alternative proof of the lemma. The new proof is both simpler and more security-preserving. Given a distinguisher for the polynomial-sample we get a distinguisher for the single sample that achieves *the same advantage*. Based on this property, the security of the functions in our proof of Theorem 4.2.1 does not significantly decrease when the *number of queries* the distinguisher makes increases (which is very different from the proofs of security for the functions in [62] and in Section 4.1).

Definition 4.2.1 *Let n and t be any pair of positive integers. Define the two distributions $I^{n,t}_{\mathbf{R}}$ and $I^{n,t}_{\mathbf{PR}}$ as follows:*

$$I^{n,t}_{\mathbf{R}} \stackrel{\text{def}}{=} \langle P, Q, g, g^a, g^{b_1}, g^{c_1}, \ldots, g^{b_t}, g^{c_t}\rangle$$

and

$$I^{n,t}_{\mathbf{PR}} \stackrel{\text{def}}{=} \langle P, Q, g, g^a, g^{b_1}, g^{a\cdot b_1}, \ldots, g^{b_t}, g^{a\cdot b_t}\rangle,$$

where $\langle P, Q, g\rangle$ is distributed according to $IG(1^n)$ and all the values in $\langle a, b_1, \ldots, b_t, c_1, \ldots, c_t\rangle$ are uniform in $\mathbb{Z}_Q$.

Lemma 4.2.4 *(Indistinguishability of a Polynomial Sample) If the DDH-Assumption (Assumption 3.1.1) holds, then for every probabilistic polynomial-time algorithm $\mathcal{D}$, every polynomial $t(\cdot)$, every constant $\alpha > 0$ and all sufficiently large n,*

$$\left|\Pr[\mathcal{D}(I^{n,t(n)}_{t(n)})) = 1] - \Pr[\mathcal{D}(I^{n,t(n)}_{\mathbf{R}})) = 1]\right| < \frac{1}{n^\alpha}.$$

Proof. Let $\varepsilon = \varepsilon(n)$ be any positive real-valued function. Assume that there exists a probabilistic polynomial-time algorithm $\mathcal{D}$ and a polynomial $t(\cdot)$ such that for infinitely many n,

$$\left|\Pr[\mathcal{D}(I^{n,t(n)}_{\mathbf{PR}})) = 1] - \Pr[\mathcal{D}(I^{n,t(n)}_{\mathbf{R}})) = 1]\right| > \varepsilon(n).$$

We define a probabilistic polynomial-time algorithm $\mathcal{A}$ such that for infinitely many n,

$$\left|\Pr[\mathcal{A}(P, Q, g, g^a, g^b, g^{a\cdot b}) = 1] - \Pr[\mathcal{A}(P, Q, g, g^a, g^b, g^c) = 1]\right| > \varepsilon(n),$$

where the probabilities are taken over the random bits of $\mathcal{A}$, the choice of $\langle P, Q, g\rangle$ according to the distribution $IG(1^n)$ and the choice of a, b and c *uniformly* at random in $\mathbb{Z}_Q$. For $\varepsilon(n) = \frac{1}{n^\alpha}$ this contradicts the DDH-Assumption and completes the proof of the lemma.

Let the input of $\mathcal{A}$ be $\langle P, Q, g, g^a, g^b, g^{\tilde{c}}\rangle$, where P is n-bit long and $\tilde{c}$ is either $a \cdot b \bmod Q$ or uniform in $\mathbb{Z}_Q$. Using a randomized reduction similar to that in the proof of Lemma 3.1.2, $\mathcal{A}$ generates $t(n)$ random pairs $g^{b_i}, g^{\tilde{c}_i}$ such that $\forall i, \tilde{c}_i = a \cdot b_i \bmod Q$ iff $\tilde{c} = a \cdot b \bmod Q$. $\mathcal{A}$ now invokes $\mathcal{D}$ on these values to distinguish between the two possible distributions of its own input. More formally, $\mathcal{A}$ executes the following algorithm:

1. Define $t = t(n)$ and sample each one of the values in $\langle d_1, \ldots, d_t, e_1, \ldots, e_t \rangle$ uniformly at random in $\mathbb{Z}_Q$.

2. Define the sequence I to be

$$\langle P, Q, g, g^a, \tilde{\mathcal{R}}_{d_1,e_1}(g^a, g^b, g^{\tilde{c}}), \ldots, \tilde{\mathcal{R}}_{d_t,e_t}(g^a, g^b, g^{\tilde{c}}) \rangle,$$

   where

$$\forall i,\ \tilde{\mathcal{R}}_{d_i,e_i}(g^a, g^b, g^{\tilde{c}}) \stackrel{\text{def}}{=} \left(g^b\right)^{d_i} \cdot g^{e_i},\ \left(g^{\tilde{c}}\right)^{d_i} \cdot (g^a)^{e_i}\ .$$

3. Output $\mathcal{D}(I)$

Denote by $g^{b_i}, g^{\tilde{c}_i}$ the value $\tilde{\mathcal{R}}_{d_i,e_i}(g^a, g^b, g^{\tilde{c}})$. By the same arguments used in the proof of Lemma 3.1.2 we have that:

- If $\tilde{c} = a \cdot b \bmod Q$, then $b_1, \ldots, b_t$ are uniform in $\mathbb{Z}_Q$ (and independent of each other and of a) and $\forall i,\ \tilde{c}_i = a \cdot b_i \bmod Q$.
- If $\tilde{c} \neq a \cdot b \bmod Q$, then $b_1, \ldots, b_t, \tilde{c}_1, \ldots, b_t, \tilde{c}_t$ are *all* uniform in $\mathbb{Z}_Q$ (and independent of each other and of a).

Therefore, by the definitions of $\mathcal{A}$, $I^{n,t}_{\mathbf{PR}}$ and $I^{n,t}_{\mathbf{R}}$ it easily follows that:

$$\Pr[\mathcal{A}(P, Q, g, g^a, g^b, g^{a \cdot b}) = 1] = \Pr[\mathcal{D}(I^{n,t}_{\mathbf{PR}}) = 1]$$

and

$$\Pr[\mathcal{A}(P, Q, g, g^a, g^b, g^c) = 1] = \Pr[\mathcal{D}(I^{n,t}_{\mathbf{R}}) = 1].$$

It is now immediate that infinitely many n,

$$\left|\Pr[\mathcal{A}(P, Q, g, g^a, g^b, g^{a \cdot b}) = 1] - \Pr[\mathcal{A}(P, Q, g, g^a, g^b, g^c) = 1]\right| > \varepsilon(n),$$

where the probabilities are as above. □

The proof of Theorem 4.2.1 given Lemma 4.2.4 uses a hybrid-arguments which is a proof-technique for showing that two distributions are indistinguishable. See [60] for details on hybrid-arguments. Loosely, the method for showing that D and D' are indistinguishable is to (1) Define a polynomial-length sequence of efficient distributions $D_0, D_1, \ldots, D_m$ with $D_0 = D$ and $D_m = D'$. (2) Show that any two neighboring distributions D_{j-1} and D_j are indistinguishable. In fact, in the uniform version of this argument (e.g. in the proof of Theorem 4.2.1) we usually show that it is hard to distinguish D_{J-1} and D_J where J is *uniformly chosen* in $[m]$. Furthermore, in the proof of Theorem 4.2.1 (as well as in the corresponding proofs in [62] and in Section 4.1) the $n + 1$ distributions that are (implicitly) defined are of functions and they are *not efficiently samplable*. For example, one of the two extreme distributions is the uniform distribution (which is certainly not efficiently samplable). Nevertheless, a uniform function can be efficiently "simulated" by an algorithm that answers each query at random (under the restriction of keeping consistency of its answers for repeating queries). Since all other intermediate function distributions can be "simulated" in the same sense we can still apply the hybrid-argument. We now turn to the formal proof (where the arguments described above are implicit).

Proof. (of Theorem 4.2.1) Let $\varepsilon = \varepsilon(n)$ be any positive real-valued function. Assume that there exists a probabilistic polynomial-time oracle machine $\mathcal{M}$ such that for infinitely many n,

$$\left|\Pr[\mathcal{M}^{f_{P,Q,g,\vec{a}}}(P,Q,g)=1] - \Pr[\mathcal{M}^{R_{P,Q,g}}(P,Q,g)=1]\right| > \varepsilon(n),$$

where the probabilities are as in Theorem 4.2.1. Let $t(\cdot)$ be a polynomial that bounds the running time of $\mathcal{M}$. We define a probabilistic polynomial-time algorithm $\mathcal{D}$, such that for infinitely many n,

$$\left|\Pr[\mathcal{D}(I^{n,t(n)}_{\mathbf{PR}})=1] - \Pr[\mathcal{D}(I^{n,t(n)}_{\mathbf{R}})=1]\right| > \frac{1}{n}\cdot\varepsilon(n).$$

By Lemma 4.2.4, for $\varepsilon(n) = \frac{1}{n^\alpha}$ this contradicts the DDH-Assumption and completes the proof of the theorem.

On any input $\langle P,Q,g,g^a,g^{b_1},g^{\tilde{c}_1},g^{b_2},g^{\tilde{c}_2},\ldots,g^{b_t},g^{\tilde{c}_t}\rangle$, where P is n bits long (and either each $\tilde{c}_i$ is $a\cdot b_i \bmod Q$ or each $\tilde{c}_i$ is uniform in $\mathbb{Z}_Q$), $\mathcal{D}$ executes the following algorithm:

1. Sample J uniformly at random in $[n]$.
2. Sample each one of the values in $\langle a_{J+1}, a_{J+2}, \ldots, a_n\rangle$ uniformly at random in $\mathbb{Z}_Q$.
3. Invoke $\mathcal{M}$ on input $\langle P,Q,g\rangle$ and answer its queries in the following way: Let the queries asked by $\mathcal{M}$ be $\langle x^1, x^2, \ldots x^m\rangle$. The i^{th} query x^i is an n-bit string. Denote $x^i = \bar{x}^i x^i_J x^i_{J+1}\cdots x^i_n$, where $\bar{x}^i$ is a $(J-1)$-bit string and $x^i_J, x^i_{J+1}, \ldots, x^i_n$ are single bits. To answer the i^{th} query define $\ell = \ell(i) = \min\{i' | \bar{x}^{i'} = \bar{x}^i\}$ and answer the query by
$$\begin{cases} (g^{\tilde{c}_\ell})^{\prod_{x^i_k=1, k>J} a_k} & \text{if } x^i_J = 1 \\ (g^{b_\ell})^{\prod_{x^i_k=1, k>J} a_k} & \text{if } x^i_J = 0 \end{cases}$$
These answers are well defined since $m \le t$.
4. Output whatever $\mathcal{M}$ outputs.

From the definition of $\mathcal{D}$ we have that for $f_{P,Q,g,\vec{a}}$ and $R_{P,Q,g}$ as in Theorem 4.2.1,

$$\Pr[\mathcal{D}(I^{n,t}_{\mathbf{PR}})=1 \mid J=1] = \Pr[\mathcal{M}^{f_{P,Q,g,\vec{a}}}(P,Q,g)=1],$$

$$\Pr[\mathcal{D}(I^{n,t}_{\mathbf{R}})=1 \mid J=n] = \Pr[\mathcal{M}^{R_{P,Q,g}}(P,Q,g)=1]$$

and for any $0 < j < n$

$$\Pr[\mathcal{D}(I^{n,t}_{\mathbf{R}})=1 \mid J=j] = \Pr[\mathcal{D}(I^{n,t}_{\mathbf{PR}})=1 \mid J=j+1].$$

By the assumption we get that for infinitely many n,

$$\begin{aligned} & \left|\Pr[\mathcal{D}(I^{n,t}_{\mathbf{PR}})=1] - \Pr[\mathcal{D}(I^{n,t}_{\mathbf{R}})=1]\right| \\ = & \left|\frac{1}{n}\cdot\sum_{j=1}^{n}\Pr[\mathcal{D}(I^{n,t}_{\mathbf{PR}})=1 \mid J=j] - \frac{1}{n}\cdot\sum_{j=1}^{n}\Pr[\mathcal{D}(I^{n,t}_{\mathbf{R}})=1 \mid J=j]\right| \end{aligned}$$

$$\begin{aligned} &= \frac{1}{n} \cdot \left| \Pr[\mathcal{D}(I^{n,t}_{\mathbf{PR}}) = 1 \mid J = 1] - \Pr[\mathcal{D}(I^{n,t}_{\mathbf{R}}) = 1 \mid J = n] \right| \\ &= \frac{1}{n} \cdot \left| \Pr[\mathcal{M}^{f_{P,Q,g,\vec{a}}}(P,Q,g) = 1] - \Pr[\mathcal{M}^{R_{P,Q,g}}(P,Q,g) = 1] \right| \\ &> \frac{1}{n} \cdot \varepsilon(n). \end{aligned}$$

This completes the proof of the theorem. □

Efficiency of the Pseudo-Random Functions

Consider a function $f_{P,Q,g,\vec{a}} \in F_n$ (where $\vec{a} = \langle a_0, a_1, \ldots a_n \rangle$) as in Construction 4.2.1. Computing the value of this function at any given point, x, involves one multiple product (a product of polynomially many numbers), $y = a_0 \cdot \prod_{x_i=1} a_i$ (which can be performed modulo Q), and one modular exponentiation, g^y. This gives a pseudo-random function which is much more efficient than previous constructions. Furthermore, one can use preprocessing in order to get improved efficiency. The most obvious preprocessing is computing the values g^{2^i} (for every positive integer i up to the length of Q). Now computing the value of the function requires two multiple products modulo a prime[6]. Additional preprocessing can reduce the work by a factor of $O(\log n)$ (see Brickell et. al. [32]). Actually, to compute the value of the pseudo-random function of Construction 4.2.2, we also need one application of a pairwise independent hash function but this operation is very cheap compared with a multiple product or a modular exponentiation.

As described in the introduction (in Section 1.2.1), we are also interested in finding the parallel-time complexity of the pseudo-random functions. In order to do so, let us first recall the result of Beame, Cook and Hoover [12] who showed that devision and related operations *including multiple product* are computable in NC^1. Based on this result Reif and Tate [123, 124] showed that these operations are also computable in TC^0. The exact depth required for these operations was considered in [141, 142] where it was shown that multiple sum is in TC^0_2, multiplication and division in TC^0_3 and multiple product in TC^0_4 (recall that for every integer i the class of functions computable by depth i circuits consisting of a polynomial number of threshold gates is denoted by TC^0_i).

By the results above, we get that after preprocessing (i.e., computing the values g^{2^i}), it is possible to evaluate the function $f_{P,Q,g,\vec{a}}$ in TC^0 (since all the necessary operations can be performed in TC^0):

Theorem 4.2.5 *Let $F = \{F_n\}_{n\in\mathbb{N}}$ be as in Construction 4.2.1. Then there exists a polynomial, $p(\cdot)$, and an integer i such that for every $n \in \mathbb{N}$ and every function $f_k \in F_n$ there exists a depth i threshold circuit of size bounded by $p(n)$ that computes f_k.*

The exact depth of the functions: Theorem 4.2.5 can be obtained by a naive application of the results in [141, 142]. A more detailed analysis of the function $f_{P,Q,g,\vec{a}}$ reveals that some further optimizations in the depth are possible. Using additional preprocessing, this

[6] In the case that Q is much smaller than P we have that the first multiple product is much cheaper than the second

function can in fact be evaluated in TC^0_5. Since the exact depth of the construction is peripheral to this work (and since a formal proof of this claim would repeat quite a lot of the analysis in [141, 142]) we only comment on a few facts regarding $f_{P,Q,g,\vec{a}}$ that allow us to reduce the depth. First, note that in both multiple products we can assume any preprocessing of the values in the multiplication (since these values are taken from the sequence $\langle a_0, a_1, \ldots a_n \rangle$ or from the set $\{g^{2^i}\}$). Second, we don't need the actual value of the first multiple product, $y = \prod_{x_i=1} a_i$: Computing values r_i (obtained by the CRT-representation) for which $y = \sum m_i \cdot r_i$ (where the values m_i are known in advance and can be preprocessed) is just as good. Finally, the value P is also known in advance. Therefore, the depth of the final modular reduction can be reduced by precomputing the values $2^i \bmod P$.

Remark 4.2.3 *The same analysis hold for efficiency and depth of the pseudo-random functions of Construction 4.2.3.*

4.2.2 Construction Based on Factoring or the GDH-Assumption

In this section we show an additional construction of pseudo-random functions – Construction 4.2.3, that is very similar to Construction 4.2.2. The security of Construction 4.2.3 is reduced to the GDH-Assumption which is a generalization of the *computational* DH-Assumption. This construction is interesting for two main reasons:

1. The GDH-Assumption is implied by the DDH-Assumption but they are not known to be equivalent. Therefore, Construction 4.2.3 may still be valid even if the DDH-Assumption does not hold. In addition, the GDH-Assumption modulo a so called Blum-integer is not stronger than the assumption that *factoring* Blum-integers is hard. This gives an attractive construction of pseudo-random functions that is at least as secure as Factoring (which was recently improved in [105]).

2. Construction 4.2.3 is based on a somewhat different methodology than Construction 4.2.2. It may be easier to apply this methodology in order to construct pseudo-random functions based on additional assumptions (in fact, Construction 4.2.2 was obtained as a modification of Construction 4.2.3).

The GDH-Assumption

The GDH-Assumption was previously considered in the context of a key-exchange protocol for a group of parties (see e.g., [139, 144]). In this protocol, party $i \in [n]$ chooses a secret value, a_i. After executing the protocol, each of these parties can compute $g^{\prod_{i\in[n]} a_i}$ and this value defines their common key. While executing the protocol, an eavesdropper may learn values of the form $g^{\prod_{i\in I} a_i}$ for several proper subsets, $I \subsetneq [n]$. It is essential to assume that even with this knowledge it is hard to compute $g^{\prod_{i\in[n]} a_i}$. The GDH-Assumption is even stronger: Informally, this assumption says that it is hard to compute $g^{\prod_{i\in[n]} a_i}$ for an algorithm that can query $g^{\prod_{i\in I} a_i}$ for *any* proper subset, $I \subsetneq [n]$ of its choice.

To remain consistent with the DDH-Assumption, we state the GDH-Assumption (Assumption 4.2.1) in a subgroup of $\mathbb{Z}^*_P$ of order Q (where P and Q are primes). In fact, the

corresponding assumption in any other group implies a corresponding construction of pseudo-random functions. For example, since breaking the GDH-Assumption modulo a composite is at least as hard as factoring (as shown in Section 3.2 and [139]), we obtain in Section 4.2.2 a construction of pseudo-random functions which is at least as secure as Factoring. Furthermore, in contrast with the DDH-Assumption, one can consider the GDH-Assumption in $\mathbb{Z}_P^*$ itself (i.e., when g is a generator of $\mathbb{Z}_P^*$).

In order to formalize the GDH-Assumption, we use the following definition:

Definition 4.2.2 *Let $\langle P, Q, g\rangle$ be any possible output of $IG(1^n)$ and let $\tilde{a} = \langle \tilde{a}_1, \tilde{a}_2, \ldots \tilde{a}_n\rangle$ be any sequence of n elements of $\mathbb{Z}_Q$. Define the function $h_{P,Q,g,\tilde{a}}$ with domain $\{0,1\}^n$ such that for any n-bit input, $x = x_1 x_2 \cdots x_n$,*

$$h_{P,Q,g,\tilde{a}}(x) \stackrel{\text{def}}{=} g^{\prod_{x_i=1} \tilde{a}_i}.$$

Define $h^r_{P,Q,g,\tilde{a}}$ to be the restriction of $h_{P,Q,g,\tilde{a}}$ to inputs $\{0,1\}^n \setminus \{1^n\}$.

Assumption 4.2.1 (Generalized Diffie-Hellman) *For every probabilistic polynomial-time oracle machine $\mathcal{A}$, every constant $\alpha > 0$ and all sufficiently large n,*

$$\Pr[\mathcal{A}^{h^r_{P,Q,g,\tilde{a}}}(P,Q,g) = h_{P,Q,g,\tilde{a}}(1^n)] < \frac{1}{n^\alpha},$$

where the probability is taken over the random bits of $\mathcal{A}$, the choice of $\langle P,Q,g\rangle$ according to the distribution $IG(1^n)$ and the choice of each of the values in $\tilde{a} = \langle \tilde{a}_1, \tilde{a}_2, \ldots \tilde{a}_n\rangle$ uniformly at random in $\mathbb{Z}_Q$.

As a corollary of Theorem 4.2.1 we have that if the DDH-Assumption holds, then so does the GDH-Assumption. In fact, we get that the DDH-Assumption implies the *decisional* GDH-Assumption (this was also previously shown in [144]):

Corollary 4.2.6 *If the DDH-Assumption (Assumption 3.1.1) holds, then for every probabilistic polynomial-time oracle machine $\mathcal{A}$, every constant $\alpha > 0$ and all sufficiently large n,*

$$\left|\Pr[\mathcal{A}^{h^r_{P,Q,g,\tilde{a}}}(P,Q,g,h_{P,Q,g,\tilde{a}}(1^n)) = 1] - \Pr[\mathcal{A}^{h^r_{P,Q,g,\tilde{a}}}(P,Q,g,g^c) = 1]\right| < \frac{1}{n^\alpha},$$

where the probabilities are taken over the random bits of $\mathcal{A}$, the choice of $\langle P,Q,g\rangle$ according to the distribution $IG(1^n)$, the choice of each of the values in $\tilde{a} = \langle \tilde{a}_1, \tilde{a}_2, \ldots \tilde{a}_n\rangle$ uniformly at random in $\mathbb{Z}_Q$ and the choice of c uniformly at random in $\mathbb{Z}_Q$.

Motivation to the construction

Construction 4.2.3 is motivated by the concept of pseudo-random synthesizers and by Construction 4.1.1 of pseudo-random functions using pseudo-random synthesizers as building blocks. To see the connection, let us recall some of the main ideas of Construction 4.1.1. Informally, a pseudo-random synthesizer, S, is:

An efficiently computable function of two arguments such that given polynomially-many, uniformly-chosen, inputs for each argument, $\{x_i\}_{i=1}^m$ and $\{y_i\}_{i=1}^m$, the output of S on all the combinations, $(S(x_i, y_j))_{i,j=1}^m$, cannot be efficiently distinguished from uniform.

A natural generalization is a k-dimensional pseudo-random synthesizer. Informally, a k-dimensional pseudo-random synthesizer, S, may be defined to be:

An efficiently computable function of k arguments such that given polynomially-many, uniformly-chosen, inputs for each argument, $\left\{\left\{x_i^j\right\}_{i=1}^m\right\}_{j=1}^k$, the output of S on all the combinations, $M = \left(S(x_{i_1}^1, x_{i_2}^2, \ldots, x_{i_k}^k)\right)_{i_1,i_2,\ldots,i_k=1}^m$, cannot be efficiently distinguished from uniform by an algorithm that can access M at points of its choice.

Construction 4.1.1 can be viewed as first recursively applying a 2-dimensional synthesizer to get an n-dimensional synthesizer, S, and then defining the pseudo-random function, f, by:

$$f_{\langle a_{1,0},a_{1,1},a_{2,0},a_{2,1},\ldots a_{n,0},a_{n,1}\rangle}(\sigma_1\sigma_2\ldots\sigma_n) \stackrel{\text{def}}{=} S(a_{1,\sigma_1}, a_{2,\sigma_2}, \ldots, a_{n,\sigma_n}).$$

However, using this construction, the depth of the n-dimensional synthesizer (and the pseudo-random functions) is larger by a logarithmic factor than the depth of the 2-dimensional synthesizer. Therefore, a natural problem is to come up with a direct construction of an n-dimensional synthesizer.

In this section it is shown that under the GDH-Assumption the function, $S_{P,Q,g,r}$, defined by $S_{P,Q,g,r}(a_1, a_2, \ldots, a_n) \stackrel{\text{def}}{=} \left(g^{\prod_{i=1}^n a_i}\right) \odot r$, is an n-dimensional synthesizer. Construction 4.2.3 is then obtained as described above.

The Construction

We turn to the construction of pseudo-random functions:

Construction 4.2.3 *We define the function ensemble $F = \{F_n\}_{n\in\mathbb{N}}$. For every n, a key of a function in F_n is a tuple, $\langle P, Q, g, \vec{a}, r\rangle$, where P is an n-bit prime, Q a prime divisor of $P-1$, g an element of order Q in $\mathbb{Z}_P^*$, $\vec{a} = \langle a_{1,0}, a_{1,1}, a_{2,0}, a_{2,1}, \ldots a_{n,0}, a_{n,1}\rangle$ a sequence of $2n$ elements of $\mathbb{Z}_Q$ and r an n-bit string. For any n-bit input, $x = x_1x_2\cdots x_n$, the Binary-function, $f_{P,Q,g,\vec{a},r}$, is defined by:*

$$f_{P,Q,g,\vec{a},r}(x) \stackrel{\text{def}}{=} \left(g^{\prod_{i=1}^n a_{i,x_i}}\right) \odot r,$$

(where $\odot$ denotes the inner product mod 2*). The distribution of functions in F_n is induced by the following distribution on their keys: $\vec{a}$ and r are uniform in their range and the distribution of $\langle P, Q, g\rangle$ is $IG(1^n)$.*

Theorem 4.2.7 *If the GDH-Assumption (Assumption 4.2.1) holds, then $F = \{F_n\}_{n\in\mathbb{N}}$ (as in Construction 4.2.3) is an efficiently computable pseudo-random function ensemble.*

In order to prove Theorem 4.2.7 we need the following corollary of the Goldreich-Levin hard-core-bit theorem [65] (more precisely, the setting of this corollary is somewhat different than the one considered in [65] but their result still applies):

Corollary 4.2.8 *If the GDH-Assumption (Assumption 4.2.1) holds, then for every probabilistic polynomial-time oracle machine $\mathcal{A}$, every constant $\alpha > 0$ and all sufficiently large n,*

$$\left|\Pr[\mathcal{A}^{h^r_{P,Q,g,\tilde{a}}}(P,Q,g,r,(h_{P,Q,g,\tilde{a}}(1^n)) \odot r) = 1] - \Pr[\mathcal{A}^{h^r_{P,Q,g,\tilde{a}}}(P,Q,g,r,\sigma) = 1]\right| < \frac{1}{n^\alpha},$$

where the probabilities are taken over the random bits of $\mathcal{A}$, the choice of $\langle P,Q,g\rangle$ according to the distribution $IG(1^n)$, the choice of each of the values in $\tilde{a} = \langle \tilde{a}_1, \tilde{a}_2, \ldots \tilde{a}_n\rangle$ uniformly at random in $\mathbb{Z}_Q$, the choice of r uniformly at random in $\{0,1\}^n$ and the choice of σ uniformly at random in $\{0,1\}$.

Proof.(of Theorem 4.2.7) Let $F = \{F_n\}_{n\in\mathbb{N}}$ be as in Construction 4.2.3. It is clear that F is efficiently computable. Assume that F is not pseudo-random, then there exists a probabilistic polynomial-time oracle machine $\mathcal{M}$ and a constant $\alpha > 0$ such that for infinitely many n,

$$\left|\Pr[\mathcal{M}^{f_{P,Q,g,\vec{a},r}}(P,Q,g,r) = 1] - \Pr[\mathcal{M}^{R_n}(P,Q,g,r) = 1]\right| > \frac{1}{n^\alpha},$$

where in the first probability $f_{P,Q,g,\vec{a},r}$ is distributed according to F_n and in the second probability R_n is uniformly distributed over the set of $\{0,1\}^n \mapsto \{0,1\}$ functions, $\langle P,Q,g\rangle$ is distributed according to $IG(1^n)$ and r is a uniformly chosen n bit string.

Let $t(\cdot)$ be a polynomial that bounds the running time of $\mathcal{M}$. We define a probabilistic polynomial-time oracle machine $\mathcal{A}$ such that for infinitely many n,

$$\left|\Pr[\mathcal{A}^{h^r_{P,Q,g,\tilde{a}}}(P,Q,g,r,(h_{P,Q,g,\tilde{a}}(1^n)) \odot r) = 1] - \Pr[\mathcal{A}^{h^r_{P,Q,g,\tilde{a}}}(P,Q,g,r,\sigma) = 1]\right| > \frac{1}{n^\alpha \cdot t(n)},$$

where the probabilities are as in Corollary 4.2.8. By Corollary 4.2.8 this would contradict the GDH-Assumption and would complete the proof of the theorem.

Given access to $h^r_{P,Q,g,\tilde{a}}$ and on input $\langle P,Q,g,r,\tilde{\sigma}\rangle$ (where we expect $\tilde{\sigma}$ to either be uniformly chosen or to be $(h_{P,Q,g,\tilde{a}}(1^n)) \odot r$), $\mathcal{A}$ executes the following algorithm:

1. Define $t = t(n)$ and sample J uniformly at random in $[t]$.
2. Sample each one of $\langle b_1, b_2, \ldots, b_n\rangle$ uniformly at random in $\mathbb{Z}_Q$.
3. Invoke $\mathcal{M}$ on input $\langle P,Q,g,r\rangle$ and answer its queries in the following way: Let the queries asked by $\mathcal{M}$ be $\langle x^1, x^2, \ldots x^m\rangle$ and assume without loss of generality that all those queries are distinct.
   - Answer each one of the first $J-1$ queries with a uniformly chosen bit.
   - Answer the J^{th} query with $\tilde{\sigma}$.

- Let x^i be the i^{th} query for $i > J$ and define the n-bit string $z = z_1 z_2 \ldots z_n$ such that z_k is 1 if the k^{th} bit of x^i and the k^{th} bit of x^J are equal and 0 otherwise. Since $x^i \neq x^J$ we have that $z \neq 1^n$. Finally, answer the i^{th} query with

$$\left(\left(h^r_{P,Q,g,\tilde{a}}(z)\right)^{\prod_{z_k=0} b_k}\right) \odot r.$$

4. Output whatever $\mathcal{M}$ outputs.

Denote $\tilde{a} = \langle \tilde{a}_1, \tilde{a}_2, \ldots \tilde{a}_n \rangle$. From the definition of $\mathcal{A}$ we have that all its answers to queries x^i for $i > J$ are $f_{P,Q,g,\vec{a},r}(x^i)$ where $\vec{a} = \langle a_{1,0}, a_{1,1}, a_{2,0}, a_{2,1}, \ldots a_{n,0}, a_{n,1} \rangle$ depends on the J^{th} query $x^J = x^J_1 x^J_2 \ldots x^J_n$ as follows: For every $1 \leq k \leq n$ if $x^J_k = 0$ then $a_{k,0} = \tilde{a}_k$ and $a_{k,1} = b_k$ and if $x^J_k = 1$ then $a_{k,1} = \tilde{a}_k$ and $a_{k,0} = b_k$. The first $J-1$ queries are answered by $\mathcal{A}$ uniformly at random. The only answer that depends on $\tilde{\sigma}$ is the J^{th} answer itself. This answer is of course uniformly distributed in case $\tilde{\sigma}$ is uniform. It is also not hard to verify that the J^{th} answer is $f_{P,Q,g,\tilde{a},r}(x^J)$ in case $\tilde{\sigma} = (h_{P,Q,g,\tilde{a}}(1^n)) \odot r$. We can therefore conclude that:

$$\Pr[\mathcal{A}^{h^r_{P,Q,g,\tilde{a}}}(P,Q,g,r,(h_{P,Q,g,\tilde{a}}(1^n)) \odot r) = 1 \mid J = 1] = \Pr[\mathcal{M}^{f_{P,Q,g,\vec{a},r}}(P,Q,g,r) = 1],$$

$$\Pr[\mathcal{A}^{h^r_{P,Q,g,\tilde{a}}}(P,Q,g,r,\sigma) = 1 \mid J = t(n)] = \Pr[\mathcal{M}^{R_n}(P,Q,g,r) = 1],$$

and

$$\Pr[\mathcal{A}^{h^r_{P,Q,g,\tilde{a}}}(P,Q,g,r,\sigma) = 1 \mid J = j] = \Pr[\mathcal{A}^{h^r_{P,Q,g,\tilde{a}}}(P,Q,g,r,(h_{P,Q,g,\tilde{a}}(1^n)) \odot r) = 1 \mid J = j{+}1],$$

where the probabilities are as above. Therefore, by the standard hybrid argument we get from the assumption that for infinitely many n,

$$\left|\Pr[\mathcal{A}^{h^r_{P,Q,g,\tilde{a}}}(P,Q,g,r,(h_{P,Q,g,\tilde{a}}(1^n)) \odot r) = 1] - \Pr[\mathcal{A}^{h^r_{P,Q,g,\tilde{a}}}(P,Q,g,r,\sigma) = 1]\right| > \frac{1}{n^\alpha \cdot t(n)}.$$

□

Remark 4.2.4 *From the proof of Theorem 4.2.7 we get that F is pseudo-random even if the distinguisher (denoted by $\mathcal{M}$ in the proof) has access to P, Q, g and r.*

Pseudo-Random Functions at Least as Secure as Factoring

The proof of Theorem 4.2.7 does not rely on the specific group for which the GDH-Assumption is defined. Therefore, the corresponding assumption in any other group implies a corresponding construction of pseudo-random functions. An especially interesting example is taking the GDH-Assumption modulo a composite. Since breaking this assumption is at least as hard as factoring (as shown in Section 3.2 and [139]), we obtain an attractive construction of pseudo-random functions which is at least as secure as Factoring. This construction was recently improved by Naor, Reingold and Rosen [105], where an efficient method is provided for expanding the one bit output of our functions while paying only a small overhead in the complexity of the evaluation (i.e. one modular multiplication for each additional output bit).

In particular, this implies a length-preserving pseudo-random function that is at least as secure as Factoring whose evaluation requires only a constant number of modular multiplications per output bit. In this subsection, we repeat the definition of the GDH-Assumption and the construction of pseudo-random functions with the group set to $\mathbb{Z}_N^*$, where N is a Blum-integer (for better readability we also repeat some of the definitions from Section 3.2). The proof of security is practically the same as the proof of Theorem 4.2.7 (and is therefore omitted).

Similarly to the case of the DDH-Assumption, we keep our results general by letting the composite N be generated by *some* polynomial-time algorithm FIG (where FIG stands for factoring-instance-generator):

Definition 4.2.3 (FIG) *The factoring-instance-generator, FIG, is a probabilistic polynomial-time algorithm such that on input 1^n of FIG its output, N, is distributed over $2n-$bit integers, where $N = P \cdot Q$ for two $n-$bit primes, P and Q, such that $P \equiv Q \equiv 3 \bmod 4$ (such an integer is known as a Blum-integer).*

The GDH-Assumption Modulo a Composite:

Definition 4.2.4 *Let N be any possible output of $FIG(1^n)$, let g be any quadratic-residue in $\mathbb{Z}_N^*$ and let $\tilde{a} = \langle \tilde{a}_1, \tilde{a}_2, \ldots \tilde{a}_n \rangle$ be any sequence of n elements of $[N]$. Define the function $h_{N,g,\tilde{a}}$ with domain $\{0,1\}^n$ such that for any n-bit input, $x = x_1 x_2 \cdots x_n$,*

$$h_{N,g,\tilde{a}}(x) \stackrel{\text{def}}{=} g^{\prod_{x_i=1} \tilde{a}_i} \bmod N.$$

Define $h^r_{N,g,\tilde{a}}$ to be the restriction of $h_{N,g,\tilde{a}}$ to inputs $\{0,1\}^n \setminus \{1^n\}$.

Assumption 4.2.2 (Generalized Diffie-Hellman in $\mathbb{Z}_N^*$) *For every probabilistic polynomial-time oracle machine $\mathcal{A}$, every constant $\alpha > 0$ and all sufficiently large n,*

$$\Pr[\mathcal{A}^{h^r_{N,g,\tilde{a}}}(N,g) = h_{N,g,\tilde{a}}(1^n)] < \frac{1}{n^\alpha},$$

where the probability is taken over the random bits of $\mathcal{A}$, the choice of N according to the distribution $FIG(1^n)$, the choice of g uniformly at random in the set of quadratic-residues in $\mathbb{Z}_N^*$ and the choice of each of the values in $\tilde{a} = \langle \tilde{a}_1, \tilde{a}_2, \ldots \tilde{a}_n \rangle$ uniformly at random in $[N]$.

The Construction and its Security:

Construction 4.2.4 *We define the function ensemble $F = \{F_n\}_{n \in \mathbb{N}}$. For every n, a key of a function in F_n is a tuple, $\langle N, g, \vec{a}, r \rangle$, where N is a $2n$-bit Blum-integer, g is a quadratic-residue in $\mathbb{Z}_N^*$, $\vec{a} = \langle a_{1,0}, a_{1,1}, a_{2,0}, a_{2,1}, \ldots a_{n,0}, a_{n,1} \rangle$ is a sequence of $2n$ values in $[N]$ and r is a $2n$-bit string. For any n-bit input, $x = x_1 x_2 \cdots x_n$, the Binary-function, $f_{N,g,\vec{a},r}$, is defined by:*

$$f_{N,g,\vec{a},r}(x) \stackrel{\text{def}}{=} \left(g^{\prod_{i=1}^n a_{i,x_i}} \bmod N \right) \odot r.$$

The distribution of functions in F_n is induced by the following distribution on their keys: $g, \vec{a}$ and r are uniform in their range and the distribution of N is $FIG(1^n)$.

In the same way Theorem 4.2.7 is proven, we get that:

Theorem 4.2.9 *If the GDH-Assumption in $\mathbb{Z}_N^*$ (Assumption 4.2.1) holds, then $F = \{F_n\}_{n\in\mathbb{N}}$ (as in Construction 4.2.4) is an efficiently computable pseudo-random function ensemble.*

However, breaking the GDH-Assumption in $\mathbb{Z}_N^*$ is at least as hard as factoring N (Theorem 3.2.1). We can therefore deduce that:

Corollary 4.2.10 *(of Theorem 4.2.9 and Theorem 3.2.1) Let $F = \{F_n\}_{n\in\mathbb{N}}$ be as in Construction 4.2.4 and assume that F is* not *an efficiently computable pseudo-random function ensemble. Then there exists a probabilistic polynomial-time algorithm $\mathcal{A}$ and a constant $\alpha > 0$ such that for infinitely many n:*

$$\Pr[\mathcal{A}(P \cdot Q) = \langle P, Q\rangle] > \frac{1}{n^{\alpha}},$$

where the distribution of $N = P \cdot Q$ is $FIG(1^n)$.

4.2.3 Additional Properties of the Pseudo-Random Functions

The pseudo-random functions of Constructions 4.2.2 and 4.2.3 have a simple algebraic structure. We consider this to be an important advantage over all previous constructions, mainly since several attractive features seem more likely to exist for a simple construction of pseudo-random functions. An interesting example arises by the work of Bellare and Goldwasser [17]. They suggest a way to design a digital-signature scheme that is very attractive given efficient pseudo-random functions and an efficient *non-interactive zero-knowledge proof* for claims of the form $y = f_s(m)$ (when a commitment to a key, s, of a pseudo-random function, f_s, is available as part of the public-key). Another very attractive scheme one may desire is a *function-sharing* scheme for pseudo-random functions (in an analogous meaning to function-sharing schemes for trapdoor one-way permutations as defined in [47]). Two examples for applications of such schemes are efficient metering of web usage [99] and the distribution of KDCs (key-distribution centers) [100].

In this section, we suggest preliminary designs for several protocols. Though there is much room for improving these designs, they are still a significant improvement over the protocols that are available for all previous constructions of pseudo-random functions (including commonly used block-ciphers such as DES) and they serve as a demonstration to the potential of our construction. The main purpose of this section is to stimulate further research both in improving our designs and in suggesting designs for other protocols (as the non-interactive zero-knowledge proof and the function-sharing scheme mentioned above). We therefore do not insist on formal definitions and proofs for our protocols. In addition, we focus on the construction of Section 4.2.1 (using similar designs for the construction of Section 4.2.2 may be problematic due to the way this construction uses the Goldreich-Levin hard-core bit).

A point worth noticing is that we describe our designs for the functions of Construction 4.2.1. That is, we ignore the pair-wise independent hashing introduced in Construction 4.2.2. However, as mentioned in Section 4.2.1, for many applications (as the undeniable

signatures suggested below) the extra hashing in Construction 4.2.2 is unnecessary. In addition, recall Remark 4.2.2 that the pseudo-random functions of Construction 4.2.2 remain pseudo-random even if the hash function is public. Therefore, the protocols we design here for Construction 4.2.1 imply similar protocols for Construction 4.2.2. For example, when distributing a pseudo-random function to a set of parties, we can distribute the function obtained by ignoring the hash function and supply each party with the value of this hash function.

Zero-Knowledge Proof for the Value of the Function

As mentioned above, a non-interactive zero-knowledge proof for claims of the form $y = f_s(m)$ is required by the Bellare-Goldwasser digital-signature scheme. Similarly a simple (interactive) zero-knowledge proof for claims of the form $y = f_s(m)$ and $y \neq f_s(m)$ implies a simple construction of undeniable signatures. Informally, undeniable signatures, which were introduced by Chaum and Antwerpen [39], are public-key schemes that allow a party to sign messages such that his participation is required in order to verify or deny a signature. We can let the public key for an undeniable signature be a commitment to a key, s, of a pseudo-random function, f_s. A signature for a message m can simply be $f_s(m)$ (or $f_s(H(m))$ for a collision-intractable hash function, H). Now the confirmation protocol for a message m and a signature y is simply a zero-knowledge proof that $y = f_s(m)$ whereas the denial protocol is a zero-knowledge proof that $y \neq f_s(m)$.

In this section we describe such zero-knowledge proofs for the values of the functions of Construction 4.2.1. To be a bit more accurate, both the commitment for a function and the zero-knowledge proofs *do reveal some of the values of the function.* Therefore, these are zero-knowledge proofs for a *restriction* of the function to a subset of its inputs (those with their last two bits set to 1), when the value of the function on the rest of the inputs is publicly available (this can be formalized by allowing the zero-knowledge simulator access to all other values of its choice).

The protocol for $y = f_s(x)$ is essentially several parallel applications of a zero-knowledge proof for the result of the Diffie-Hellman protocol. These proofs are strongly related to Schnorr's identification protocol [135]. In order to make his proof into a zero-knowledge proof we use a strong-receiver commitment scheme[7] in a rather standard way (using a *strong-sender* commitment scheme gives perfect zero-knowledge *arguments*). We denote the commit (resp. reveal) phase of this protocol by COMMIT (resp. REVEAL).

Definition 4.2.5 *(a commitment for f_s) Let $F = \{F_n\}_{n\in\mathbb{N}}$ be as in Construction 4.2.1 and*

[7]Loosely speaking, a commitment scheme is a protocol between a sender and a receiver that has two phases: The commit phase in which the sender "commits" to a value p, and the reveal phase in which the sender opens this commitment. Such a protocol should have two properties: (1) It is binding: a computationally bounded sender should not be able to open two distinct values $p \neq p'$ in the reveal phase. (2) It is (semantically) secure: for any $p \neq p'$, at the end of the commit phase the receiver should not be able to distinguish a commitment to p from a commitment to p'. (See e.g. [98] for formal definitions and constructions.) Two stronger variants of commitment schemes are strong-sender and strong-receiver. In strong-sender commitments even a computationally unbounded sender should not be able to open two distinct values $p \neq p'$. In strong-receiver commitments (also known as information-theoretic commitments), the commit phase leaks no information on p in an information-theoretically sense.

let $f_{P,Q,g,\vec{a}}$ be some function in F_n, where $\vec{a} = \langle a_0, a_1, \ldots a_n \rangle$. A commitment for $f_{P,Q,g,\vec{a}}$ is

$$\langle P, Q, g, g^{a_0}, g^{a_0 \cdot a_1}, g^{a_0 \cdot a_2}, \ldots, g^{a_0 \cdot a_n} \rangle.$$

Protocol 4.2.1 *(ZK-Proof for $y = f_s(x)$)*
The prover, $\mathcal{P}$, knows the key $s = \langle P, Q, g, \vec{a} \rangle$ and the verifier $\mathcal{V}$ knows the commitment for f_s. The common input is a pair $\langle x, y \rangle$, where $x = x_1 x_2 \cdots x_n \in \{0,1\}^n$ satisfies that $x_{n-1} = x_n = 1$ and y is in the subgroup of $\mathbb{Z}_P^*$ generated by g. Denote by $\tilde{g}$ the value g^{a_0} and assume wlog that for some k, $x_1 = x_2 = \cdots = x_k = 0$ whereas $x_{k+1} = \cdots = x_n = 1$. The protocol for proving $y = f_s(x)$ is defined as follows:

1. *$\mathcal{V}$ chooses a value e uniformly from $\mathbb{Z}_Q^*$ and sends COMMIT(e) to $\mathcal{P}$.*
2. *For each $i \in [(k+1)..n]$, $\mathcal{P}$ chooses r_i uniformly from $\mathbb{Z}_Q$, computes $c_i = \left(\prod_{j=1}^{i-1} a_j\right) \bmod Q$ and sends $\langle y_i = \tilde{g}^{c_i}, \tilde{g}^{r_i}, (y_i)^{r_i} \rangle$ to $\mathcal{V}$. Denote by y_{n+1} the value y.*
3. *$\mathcal{V}$ sends REVEAL(e) to $\mathcal{P}$.*
4. *For each $i \in [(k+1)..n]$, $\mathcal{P}$ sends $d_i = e \cdot a_i + r_i \bmod Q$ to $\mathcal{V}$.*
5. *$\mathcal{V}$ accepts if the following conditions hold: (1) $y_{k+1} = \tilde{g}$. (2) $\forall i \in [(k+1)..n]$, $(y_i)^{r_i} \cdot (y_{i+1})^e = (y_i)^{d_i}$ and $(\tilde{g}^{a_i})^e \cdot \tilde{g}^{r_i} = \tilde{g}^{d_i}$.*

In addition to the ideas used by the protocol for $y = f_s(x)$, the protocol for $y \neq f_s(x)$ uses the randomized-reduction of Section 3.1.3 in order to prove that a value is *not* the result of the Diffie-Hellman protocol.

Protocol 4.2.2 *(ZK-Proof for $y \neq f_s(x)$)*
Let the setting and notation be as in Protocol 4.2.1. The protocol for proving $y \neq f_s(x)$ is defined as follows:

1. *$\mathcal{V}$ chooses a value e uniformly from $\mathbb{Z}_Q^*$ and a uniform subset, J, of $[n]$. $\mathcal{V}$ sends COMMIT(e, J) to $\mathcal{P}$.*
2. *For each $i \in [(k+1)..n]$, $\mathcal{P}$ chooses r_i uniformly from $\mathbb{Z}_Q$, computes $c_i = \left(\prod_{j=1}^{i-1} a_j\right) \bmod Q$ and sends $\langle y_i = \tilde{g}^{c_i}, \tilde{g}^{r_i}, (y_i)^{r_i} \rangle$ to $\mathcal{V}$.*

   Also, for every $i \in [n]$, $\mathcal{P}$ chooses u_i and v_i uniformly from $\mathbb{Z}_Q$ and sends $\langle \ell_i^1 = \tilde{g}^{u_i \cdot a_n + v_i}, \ell_i^2 = y^{u_i} \cdot \tilde{g}^{v_i} \rangle$ to $\mathcal{V}$.
3. *$\mathcal{V}$ sends REVEAL(e, J) to $\mathcal{P}$.*
4. *For each $i \in [(k+1)..(n-1)]$, $\mathcal{P}$ sends $d_i = e \cdot a_i + r_i \bmod Q$ to $\mathcal{V}$.*

   Also, for every $i \in J$, $\mathcal{P}$ sends u_i and v_i to $\mathcal{V}$ and for every $i \in [n] \setminus J$, $\mathcal{P}$ sends $u_i \cdot a_n + v_i$ to $\mathcal{V}$.
5. *$\mathcal{V}$ accepts if the following conditions hold: (1) $y_{k+1} = \tilde{g}$. (2) $\forall i \in [(k+1)..(n-1)]$, $(y_i)^{r_i} \cdot (y_{i+1})^e = (y_i)^{d_i}$ and $(\tilde{g}^{a_i})^e \cdot \tilde{g}^{r_i} = \tilde{g}^{d_i}$. (3) $\forall i \in J$, $\ell_i^1 = (\tilde{g}^{a_n})^{u_i} \cdot \tilde{g}^{v_i}$ and $\ell_i^2 = y^{u_i} \cdot \tilde{g}^{v_i}$ (i.e., $\mathcal{P}$ sent the correct values). (4) $\forall i \in [n] \setminus J$, we have that $\ell_i^1 = \tilde{g}^{u_i \cdot a_n + v_i}$ (i.e., $\mathcal{P}$ sent the correct value) and $(y_n)^{u_i \cdot a_n + v_i} \neq \ell_i^2$.*

Properties: As mentioned above, we do not give formal proofs for the properties of Protocols 4.2.1 and 4.2.2. Nevertheless, we now informally describe these properties:

1. Completeness: A prover that knows the key s can always convince the legal verifier of the correct statement $y = f_s(x)$ or $y \neq f_s(x)$.

2. Soundness: No prover can convince the legal verifier of an incorrect statement $y = f_s(x)$ or $y \neq f_s(x)$ with non-negligible probability. In case the commitment scheme is not a strong-receiver commitment scheme, soundness only holds against a computationally bounded (possibly cheating) prover.

3. Zero-Knowledge: The conversation of any (possibly cheating) $\mathcal{V}^*$ with $\mathcal{P}$ (that is trying to prove a correct statement $y = f_s(x)$ or $y \neq f_s(x)$) can be efficiently simulated by an oracle machine that has access to any value $f_s(x')$ where at least one of the bits x'_{n-1} and x'_n is zero.

Function Sharing

For many applications (as the undeniable signature-scheme described above and the applications described in [99, 100]), one would like a simple function-sharing scheme for pseudo-random functions. In recent years there has been considerable work on threshold public-key cryptography and in particular on function-sharing schemes for trapdoor-permutations (see [47, 48, 49] for some of the early works on this subject). However, threshold cryptography is very desirable in the setting of private key as well. In particular, it is possible to define function-sharing schemes for pseudo-random functions in an analogous way to definitions of De-Santis et. al. [47]. Informally, given some (monotone) access structure for ℓ parties and a key s of a pseudo-random function f_s, we want a way to give party i a function Sh_i such that the following two conditions hold: (1) For any value x and any authorized subset of the parties $J \subset [\ell]$, it is easy to compute $f_s(x)$ given $\{Sh_j(x)\}_{j \in J}$. (2) Let $J \subset [\ell]$ be any unauthorized subset of the parties and $\mathcal{A}$ any efficient algorithm that is given the shares $\{Sh_j\}_{j \in J}$ as input. Then even after $\mathcal{A}$ adaptively queries all the shares $\{Sh_j\}_{j \notin J}$ at points $\langle x^1 \ldots x^m \rangle$ of its choice, $\mathcal{A}$ cannot tell apart the restriction of f_s to all other inputs (apart of $\langle x^1 \ldots x^m \rangle$) from a random function.

Unfortunately, we do not know of an efficient function-sharing scheme for the pseudo-random functions that are constructed in this section (as well as for any other pseudo-random function; see [100] and references therein for various approximations of function-sharing scheme for a pseudo-random functions). We do however know how to distribute these functions in a weaker sense. The main difference is that now an authorized subset of the parties is required to engage in a protocol in order to compute $f_s(x)$. Such a scheme has several disadvantages compared with a function-sharing scheme (see [47] for a discussion). In particular, the definition of security is more delicate (since it should consider executions of the protocol when an unauthorized subset is cheating). However, such a scheme might still be useful.

Consider a function $f_s = f_{P,Q,g,\vec{a}} \in F_n$, where $F = \{F_n\}_{n \in \mathbb{N}}$ as in Construction 4.2.1, $\vec{a} = \langle a_0, a_1, \ldots a_n \rangle$. In order to distribute f_s we can simply share its key s among the parties (in fact, since P, Q and g can be public, it is enough to share $\vec{a}$). Whenever a legal subset

wants to compute the function at a point x it can engage in a secure function evaluation of $f_s(x)$. This idea works for every function f_s using secret sharing schemes as in [136] and universal techniques for secure function evaluation [21, 46, 66]. However, the specific protocols that are possible in our case are much more efficient given the simplicity of our functions. As an example, assume that the only authorized subset is $[\ell]$ itself: Party i gets the key of the function $Sh_i = f_{P,Q,g,\vec{a}^i} \in F_n$, where $\vec{a}^i = \langle a_0^i, a_1^i, \ldots a_n^i \rangle$ and the values $\{\vec{a}^i\}_{i\in[\ell]}$ are uniformly chosen, subject to the condition that $\forall 0 \leq j \leq n,\ a_j = \prod_{i\in[\ell]} a_j^i \bmod Q$. Now, for every input x if $\forall i, Sh_i(x) = g^{c_i}$ and $f_s(x) = g^c$ then $c = \prod_{i\in[\ell]} c_i \bmod Q$. Therefore, the computation of $f_s(x)$, can be done in n steps. At step 1 the first party publishes g^{c_1}. At step i party i can compute and publish $g^{\prod_{i=1}^{i} c_i}$ (and perhaps also prove this value). Additional access structures for which we can distribute the function $f_{P,Q,g,\vec{a}}$ are (1) "t out of ℓ": the authorized subsets are the ones with at least t elements (2) Access structures that can be described by a small monotone formula.

Oblivious Evaluation

We also suggest a new and attractive feature for a pseudo-random function — a protocol for "oblivious evaluation" of its value: Assume that a party, $\mathcal{A}$, knows a key, s, of a pseudo-random function. Then $\mathcal{A}$ and a second party, $\mathcal{B}$, can perform a protocol during which $\mathcal{B}$ learns exactly one value $f_s(x)$ of its choice whereas $\mathcal{A}$ does not learn a thing (and, in particular, does not learn x). One possible application of such a protocol is for "blind-authentication". It is also interesting to compare with a protocol for oblivious-transfer, during which $\mathcal{B}$ learns one of *two* possible values (whereas in oblivious evaluation $\mathcal{B}$ learns one of exponential many possible values).

We now describe a preliminary design for oblivious-evaluation of the functions of Construction 4.2.1. This is a rather inefficient protocol since it requires $\Theta(n)$ rounds, still it is much more efficient than what we can show for previous constructions of pseudo-random functions. The main idea of the protocol is the following: Let $u_j = (g^{a_0})^{\prod_{x_i=1, i\leq j} a_i}$ and $v_j = (g^{a_0})^{\prod_{x_i=o, i\leq j} a_i}$ The output of step j is $(u_j)^r$ and $(v_j)^{r'}$ for two values r and r', known to $\mathcal{B}$. Now $\mathcal{B}$ chooses t and t' uniformly at random, computes $(u_j)^{r\cdot t}$ and $(v_j)^{r'\cdot t'}$ and send this pair to $\mathcal{A}$ in a uniformly chosen order (in fact, $\mathcal{B}$ also has to prove that he knows such values t and t'). If $x_{j+1} = 1$ $\mathcal{B}$ asks for $((u_j)^{r\cdot t})^{a_{j+1}}$ and otherwise for $\left((v_j)^{r'\cdot t'}\right)^{a_{j+1}}$.

We note that subsequently to our work, two rather efficient oblivious-evaluation schemes were designed for our functions by Benny Pinkas and Ronald Cramer and by Benny Pinkas and Moni Naor [119].

4.3 Further Research

In Sections 4.1.7-4.1.9 we discussed the existence of pseudo-random synthesizers in NC. Additional work should be done in this area. The most obvious question is what are the general assumptions (in cryptography or in other fields) that imply the existence of pseudo-random synthesizers in NC. In particular, whether there exist parallel constructions of pseudo-random synthesizers out of pseudo-random generators or directly from one-way functions.

It is also of interest to find parallel constructions of pseudo-random synthesizers based on other concrete intractability assumptions. A task of practical importance is to derive more efficient concrete constructions of pseudo-random synthesizers in order to get efficient constructions of pseudo-random functions. As described in Section 4.1.3, an important contribution to the efficiency of the pseudo-random functions would be a direct construction of synthesizers with linear output length.

An extensive research field deals with pseudo-random generators that "fool" algorithms performing space-bounded computations. This kind of generators can be constructed without any (unproven) assumptions; see [10, 107, 108, 110] for some definitions, constructions and applications. It is possible that the concept of pseudo-random synthesizers and the idea of our construction can be applied to the "world" of space-bounded computations. As a motivation remark, note that the construction in [107] bares some resemblance to the GGM construction.

In some sense we can think of the inner product function as a pseudo-random synthesizer for space bounded computation. Let $IP(x,y)$ be the inner product of x and y (mod 2) and let X and Y be random length-m sequences of n-bit strings. For some constant $0 < \alpha < 1$ and $s = \alpha n$ it can be shown that $\mathbf{C}_{IP}(X,Y)$ is a pseudo-random generator for $SPACE(s)$ with parameter $\varepsilon = 2^{-\Omega(s)}m^2$ (when $\mathbf{C}_{IP}(X,Y)$ is given row by row). The only fact we use is that approximating IP is "hard" in the communication complexity model (see [41, 148]).

One might also try to apply the concept of pseudo-random synthesizers for other classes of algorithms. For example [2, 109] construct pseudo-random generators for polynomial-size constant-depth circuits, and in general for any class for which hard problems are known.

Our primary motivation for introducing pseudo-random synthesizers is the parallel construction of pseudo-random functions. The special characteristics of pseudo-random synthesizers lead us to believe that other desired applications may exist. For instance, pseudo-random synthesizers easily define a pseudo-random generator with large output length and the ability to directly compute subsequences of the output. This and the properties discussed in Section 4.1.6 suggests that pseudo-random synthesizers may be useful for software implementations of pseudo-random generators or functions. Another possible application of the idea of Construction 4.1.1 that should be examined is to convert encryption methods that are not immune to chosen plain-text attacks into ones that are immune.

Section 4.2 gives two very efficient constructions of pseudo-random functions. The first construction is based on the *decisional* DH-Assumption (Assumption 3.1.1) and the second construction is based on a generalization of the *computational* DH-Assumption (Assumption 4.2.1). We study these assumptions in Chapter 3. A natural line for further research is the additional study of the validity of these assumptions and the relations between these assumptions and the standard *computational* DH-Assumption. Since our constructions can be based on the corresponding assumptions for other groups (e.g., in elliptic-curve groups), it is interesting to study the validity of these assumptions as well.

As discussed in Section 4.2.2, the pseudo-random functions constructed in Section 4.2 are motivated by the constructions of Section 4.1. In fact, Section 4.2 can be described as direct and efficient constructions of n-dimensional pseudo-random synthesizers (see Section 4.1.4). An interesting problem is to construct efficient n-dimensional synthesizers based on different intractability assumptions.

Another interesting line for further research is improving the protocols described in Section 4.2.3 and designing additional protocols (as a non-interactive zero-knowledge proof for the value of a pseudo-random function and a function-sharing scheme in the sense of [47]).

An alternative direction for constructing parallel pseudo-random functions is to try and generalize the philosophy behind the Data Encryption Standard (DES) while maintaining its apparent efficiency. Some interesting ideas and results on the generalization of DES can be found in Cleve's work [43, 44].

Chapter 5

Constructions of Pseudo-Random Permutations

The work described in this chapter is a study of the LR-Construction of pseudo-random permutations from pseudo-random function (see the introduction for a discussion on pseudo-random permutations and their applications as well as a description of the LR-Construction). Our goal is to provide a better understanding of the LR-Construction and as a result improve the construction in several respects. Our main observation is that the different rounds of the LR-Construction serve significantly different roles. We show that the first and last rounds can be replaced by pair-wise independent permutations and use this in order to :

1. Simplify the proof of security of the construction (especially in the case of strong pseudo-random permutations) and provide a framework for proving the security of similar constructions.

2. Derive generalizations of the construction that are of practical and theoretical interest. The proof of security for each one of the constructions is practically "free of charge" given the framework.

3. Achieve an improvement in the computational complexity of the pseudo-random permutations – two applications of a pseudo-random function on n bits suffice for computing the value of a pseudo-random permutation on $2n$ bits at a given point (vs. four applications in the original LR-Construction). This implies that the reduction is "optimal".

As discussed in Section 5.4.2, the new construction is in fact a generalization of the original LR-Construction. Thus, the proof of security (Theorem 5.2.2) also applies to the original construction. The following is a brief and informal description of the main results and the organization of this chapter:

Section 5.2 Presents the main construction and proves its security: pair-wise independent permutations can replace the first and fourth rounds of the LR-Construction (see Figure 5.1 for an illustration).

Section 5.3 Highlights the high-level structure of the proof of security which provides a framework that enables us to relax and generalize the main construction.

Section 5.4 Shows how the main construction can be relaxed by:

5.4.1 Using a single pseudo-random function (instead of two) and

5.4.2 Using weaker and more efficient permutations (or functions) instead of the pairwise independent permutations.

Section 5.5 Provides a simple generalization of the main construction: using t rounds of (generalized) Feistel permutations (instead of two) the success probability of the distinguisher is reduced from approximately $\frac{m^2}{2^{\ell/2}}$ to approximately $\frac{t}{2} \cdot \frac{m^2}{2^{(1-1/t)\ell}}$, where the permutation is on ℓ bits and the distinguisher makes at most m queries (see Figure 5.3 for an illustration).

Section 5.6 Provides a second generalization of the main construction. Instead of applying Feistel permutations on the entire outputs of the first and second rounds, Feistel permutations can be separately applied on each one of their sub-blocks. This is a construction of a strong pseudo-random permutation on *many* blocks using pseudo-random functions on a *single* block (see Figure 5.4 for an illustration).

Section 5.7 Analyzes the different constructions of the chapter as constructions of k-wise δ-dependent permutations.

Section 5.8 Suggests directions for further research.

5.1 Notation

We use in this chapter notation and definitions given in Chapter 2 (in particular we use definitions of pseudo-randomness and k-wise independence that appear there). Additional notation that is used in this chapter include:

- I^n denotes the set of all n-bit strings, $\{0,1\}^n$.
- F_n denotes the set of all $I^n \mapsto I^n$ functions and P_n denotes the set of all such permutations ($P_n \subset F_n$). In this chapter we concentrate on pseudo-random functions in F_n and pseudo-random permutations in P_n.
- For $x \in I^{2n}$, denote by $x_{|_L}$ the first (left) n bits of x and by $x_{|_R}$ the last (right) n bits of x.

Definition 5.1.1 (Feistel Permutations) [1] *For any function $f \in F_n$, let $\mathbf{D}_f \in P_{2n}$ be the permutation defined by $\mathbf{D}_f(L,R) \stackrel{\text{def}}{=} (R, L \oplus f(R))$, where $|L| = |R| = n$.*

[1] **D** stands for DES-like, another common term for these permutations.

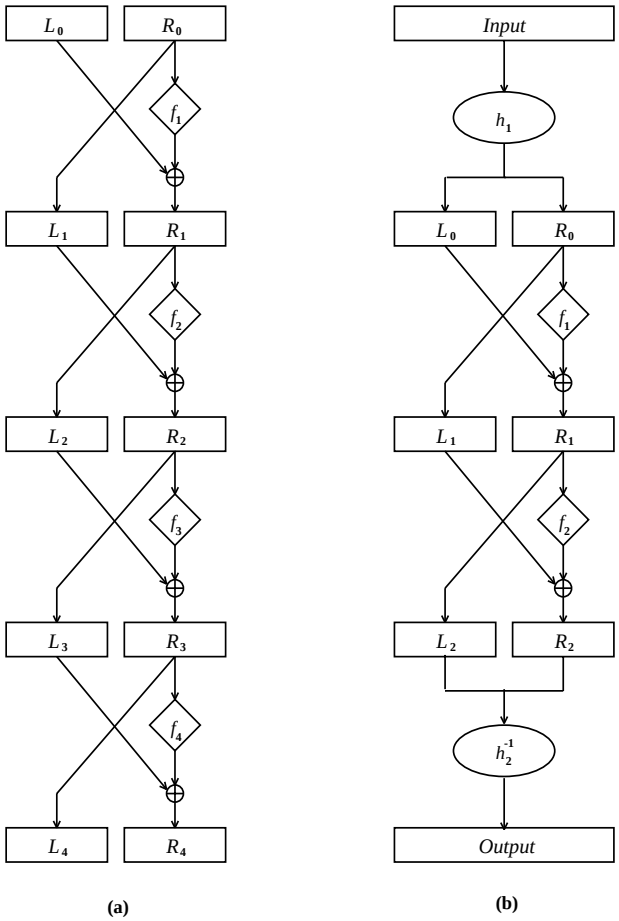

Figure 5.1: Constructions of SPPE: (a) The original LR-Construction (b) The revised Construction. In (a) and (b): $\forall i \geq 1$, $L_i = R_{i-1}$ and $R_i = L_{i-1} \oplus f_i(R_{i-1})$. In (b): $\langle L_0, R_0 \rangle = h_1(Input)$ and $Output = h_2^{-1}(\langle L_2, R_2 \rangle)$.

Notice that Feistel permutations are as easy to invert as they are to compute (since the inverse permutation satisfies $\mathbf{D}_f^{-1}(L,R) = (R \oplus f(L), L)$; that is, $\mathbf{D}_f^{-1}(L,R) \equiv \rho \circ \mathbf{D}_f \circ \rho$ for $\rho(L,R) \stackrel{\text{def}}{=} (R,L)$). Therefore, the LR-Construction (and its different variants which are introduced in Sections 5.5 & 5.6) are easy to invert.

Recall that the Luby and Rackoff design of PPE (resp. SPPE) is $\mathbf{D}_{f_3} \circ \mathbf{D}_{f_2} \circ \mathbf{D}_{f_1}$ (resp. $\mathbf{D}_{f_4} \circ \mathbf{D}_{f_3} \circ \mathbf{D}_{f_2} \circ \mathbf{D}_{f_1}$) where all f_is are independent (length-preserving) pseudo-random functions and $\mathbf{D}_{f_i}$ as in Definition 5.1.1 (see Figure 5.1.a for an illustration).

5.2 Construction of PPE and SPPE

5.2.1 Intuition

As mentioned above, a principle observation of this work is that the different rounds of the LR-Construction serve significantly different roles. To illustrate this point, consider two rounds of the construction. Namely, $E = \mathbf{D}_{f_2} \circ \mathbf{D}_{f_1}$, where $f_1, f_2 \in F_n$ are two indepen-

dently chosen pseudo-random functions. It is not hard to verify that E is computationally indistinguishable from a random permutation to any efficient algorithm that has access to pairs $\{\langle x_i, E(x_i)\rangle\}_{i=1}^m$, where the sequence $\{x_i\}_{i=1}^m$ is uniformly distributed. The intuition is as follows: Note that it is enough to prove the pseudo-randomness of E when f_1 and f_2 are *truly random functions* (instead of pseudo-random). Let $(L_i^0, R_i^0) = x_i$ and $(L_i^2, R_i^2) = E(x_i)$, by the definition of E we get that $L_i^2 = L_i^0 \oplus f_1(R_i^0)$ and $R_i^2 = R_i^0 \oplus f_2(L_i^2)$. Since the sequence $\{x_i\}_{i=1}^m$ is uniformly distributed, we have that with good probability (better than $(1 - \frac{m^2}{2^{n+1}})$) $R_i^0 \neq R_j^0$ for all $i \neq j$. Conditioned on this event, the sequence $\{L_i^2\}_{i=1}^m$ is uniformly distributed and independent of the sequence $\{x_i\}_{i=1}^m$ (since f_1 is random). We now have that with good probability $L_i^2 \neq L_j^2$ for all $i \neq j$. Conditioned on this event, the sequence $\{R_i^2\}_{i=1}^m$ is uniformly distributed and independent of both $\{L_i^2\}_{i=1}^m$ and $\{x_i\}_{i=1}^m$. Notice that this argument still works if the sequence $\{x_i\}_{i=1}^m$ is only pair-wise independent.

Nevertheless, as Luby and Rackoff showed, E can be easily distinguished from a random permutation by an algorithm that gets to see the value of E or E^{-1} on inputs of its choice. The reason is that for any values L_1, L_2 and R such that $L_1 \neq L_2$ we have that $E(L_1, R)_{|_L} \oplus E(L_2, R)_{|_L} = L_1 \oplus L_2$. In contrast, for a truly random permutation, the probability of this event is 2^{-n}. This is the reason that the LR-Construction includes three or four rounds.

If we think of the second and third rounds of the LR-Construction as the permutation E, then the discussion above implies that the role of the first and fourth rounds is to prevent the distinguisher from directly choosing the inputs of E and E^{-1}. We show that this goal can also be achieved with "combinatorial" constructions (e.g., pair-wise independent permutations), rather than "cryptographic" (i.e., pseudo-random functions). In particular, the LR-Construction remains secure when the first and fourth rounds are replaced with pair-wise independent permutations (see Figure 5.1 for an illustration).

5.2.2 Construction and Main Result

Definition 5.2.1 *For any $f_1, f_2 \in F_n$ and $h_1, h_2 \in P_{2n}$, define*

$$W(h_1, f_1, f_2) \stackrel{\text{def}}{=} \mathbf{D}_{f_2} \circ \mathbf{D}_{f_1} \circ h_1$$

and

$$S(h_1, f_1, f_2, h_2) \stackrel{\text{def}}{=} h_2^{-1} \circ \mathbf{D}_{f_2} \circ \mathbf{D}_{f_1} \circ h_1.$$

Theorem 5.2.1 *Let $h_1, h_2 \in P_{2n}$ be pair-wise independent permutations (similarly to Remark 2.2.1 this is an abbreviation for "distributed according to a pair-wise independent permutation ensemble") and let $f_1, f_2 \in F_n$ be pseudo-random functions; h_1, h_2, f_1 and f_2 are independently chosen. Then $W = W(h_1, f_1, f_2)$ is a pseudo-random permutation and $S = S(h_1, f_1, f_2, h_2)$ is a strong pseudo-random permutation (W and S as in Definition 5.2.1).*

Furthermore, assume that no efficient oracle-machine that makes at most $m = m(n)$ queries, ε-distinguishes between the pseudo-random functions and random functions for $\varepsilon = \varepsilon(n)$ (see Definition 2.2.2). Then no efficient oracle-machine that makes at most m queries to W (resp. S and S^{-1}) ε'-distinguishes W (resp. S) from a random permutation for $\varepsilon' = 2\varepsilon + \frac{m^2}{2^n} + \frac{m^2}{2^{2n}}$.

Remark 5.2.1 *The conditions of Theorem 5.2.1 are meant to simplify the exposition of the theorem and of its proof. These conditions can be relaxed, as discussed in Section 5.4. The main points are the following:*

1. *A single pseudo-random function f can replace both f_1 and f_2*
2. *h_1 and h_2 may obey weaker requirements than pair-wise independence. For example, it is enough that for every $x \neq y$:*

$$\Pr[h_1(x)_{|R} = h_1(y)_{|R}] \leq 2^{-n} \text{ and } \Pr[h_2(x)_{|L} = h_2(y)_{|L}] \leq 2^{-n}.$$

5.2.3 Proof of Security

We now prove the security of the SPPE-construction; the proof of security for the PPE-construction is very similar (and, in fact, a bit simpler). As with the original LR-Construction, the main task is to prove that the permutations are pseudo-random when f_1 and f_2 are truly random (instead of pseudo-random).

Theorem 5.2.2 *Let $h_1, h_2 \in P_{2n}$ be pair-wise independent permutations and let $f_1, f_2 \in F_n$ be random functions. Define $S = S(h_1, f_1, f_2, h_2)$ (as in Definition 5.2.1) and let $R \in P_{2n}$ be a random permutation. Then for any oracle machine M (not necessarily an efficient one) that makes at most m queries,*

$$\left|\Pr[M^{S,S^{-1}}(1^{2n}) = 1] - \Pr[M^{R,R^{-1}}(1^{2n}) = 1]\right| \leq \frac{m^2}{2^n} + \frac{m^2}{2^{2n}}.$$

Theorem 5.2.1 follows easily from Theorem 5.2.2 (see a proof-sketch in the sequel). In order to prove Theorem 5.2.2, we introduce additional notation.

Let G denote the permutation that is accessible to the machine M (G is either S or R). There are two types of queries M can make: either $(+, x)$ which denotes the query "what is $G(x)$?" or $(-, y)$ which denotes the query "what is $G^{-1}(y)$?". For the i^{th} query M makes, define the query-answer pair $\langle x_i, y_i \rangle \in I^{2n} \times I^{2n}$, where either M's query was $(+, x_i)$ and the answer it got was y_i or M's query was $(-, y_i)$ and the answer it got was x_i. We assume that M makes exactly m queries and refer to the sequence $\{\langle x_1, y_1 \rangle, ..., \langle x_m, y_m \rangle\}$ of all these pairs as the *transcript* (of M's computation).

Notice that no limitations were imposed on the computational power of M. Therefore, M can be assumed to be deterministic (we can always fix the random tape that maximizes the advantage M achieves). This assumption implies that for every $1 \leq i \leq m$ the i^{th} query of M is fully determined by the first $i-1$ query-answer pairs. Thus, for every i it can be determined from the transcript whether the i^{th} query was $(+, x_i)$ or $(-, y_i)$. We also get that M's output is a (deterministic) function of its transcript. Denote by $C_M[\{\langle x_1, y_1 \rangle, ..., \langle x_{i-1}, y_{i-1} \rangle\}]$ the i^{th} query of M as a function of the previous query-answer pairs and denote by $C_M[\{\langle x_1, y_1 \rangle, ..., \langle x_m, y_m \rangle\}]$ the output of M as a function of its transcript.

Definition 5.2.2 *Let σ be a sequence $\{\langle x_1, y_1 \rangle, ..., \langle x_m, y_m \rangle\}$, where for $1 \leq i \leq m$ we have that $\langle x_i, y_i \rangle \in I^{2n} \times I^{2n}$. Then σ is a* possible M-transcript *if for every $1 \leq i \leq m$*

$$C_M[\{\langle x_1, y_1 \rangle, ..., \langle x_{i-1}, y_{i-1} \rangle\}] \in \{(+, x_i), (-, y_i)\}.$$

Let us consider yet another distribution on the answers to M's queries (which, in turn, induces another distribution on the possible M-transcripts). Consider a random process $\tilde{R}$ that on the ith query of M answers as follows:

1. If M's query is $(+, x)$ and for some $1 \leq j < i$ the j^{th} query-answer pair is $\langle x, y \rangle$, then $\tilde{R}$'s answer is y (for an arbitrary such query-answer pair, $\langle x, y \rangle$).

2. If M's query is $(-, y)$ and for some $1 \leq j < i$ the j^{th} query-answer pair is $\langle x, y \rangle$, then $\tilde{R}$'s answer is x (for an arbitrary such query-answer pair, $\langle x, y \rangle$).

3. If neither 1 nor 2 holds, then $\tilde{R}$'s answer is a uniformly chosen $2n$-bit string.

It is possible that $\tilde{R}$ provides answers that are not consistent with *any* permutation:

Definition 5.2.3 *Let $\sigma = \{\langle x_1, y_1 \rangle, ..., \langle x_m, y_m \rangle\}$ be any possible M-transcript. σ is* inconsistent *if for some $1 \leq j < i \leq m$ the corresponding query-answer pairs satisfy $x_i = x_j$ and $y_i \neq y_j$ or $y_i = y_j$ and $x_i \neq x_j$. Otherwise, σ is* consistent.

We first show (in Proposition 5.2.3) that the advantage M might have in distinguishing between the process $\tilde{R}$ and the random permutation R is small. The reason is that as long as $\tilde{R}$ answers consistently (which happens with good probability) it "behaves" exactly as a random permutation. In order to formalize this, we consider the different distributions on the transcript of M (induced by the different distributions on the answers it gets).

Definition 5.2.4 *Let T_S, T_R and $T_{\tilde{R}}$ be the random variables such that T_S is the transcript of M when its queries are answered by S, T_R is the transcript of M when its queries are answered by R and $T_{\tilde{R}}$ is the transcript of M when its queries are answered by $\tilde{R}$.*

Notice that by these definitions (and by our assumptions) $M^{S,S^{-1}}(1^{2n}) = C_M(T_S)$ (are the same random variables) and $M^{R,R^{-1}}(1^{2n}) = C_M(T_R)$.

Proposition 5.2.3

$$\left| \Pr_{\tilde{R}}[C_M(T_{\tilde{R}}) = 1] - \Pr_{R}[C_M(T_R) = 1] \right| \leq \frac{m^2}{2^{2n+1}}.$$

Proof. For any possible and consistent M-transcript σ we have that

$$\Pr_{R}[T_R = \sigma] = \frac{2^{2n}!}{(2^{2n} - m)!} = \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma \mid T_{\tilde{R}} \textit{ is consistent}].$$

Therefore, the distribution of $T_{\tilde{R}}$ *conditioned* on $T_{\tilde{R}}$ being consistent is exactly the distribution of T_R. Furthermore, the probability that $T_{\tilde{R}}$ is inconsistent is small: $T_{\tilde{R}}$ is inconsistent if for some $1 \leq j < i \leq m$ the corresponding query-answer pairs satisfy $x_i = x_j$ and $y_i \neq y_j$ or $y_i = y_j$ and $x_i \neq x_j$. For a given i and j this event happens with probability at most 2^{-2n}. Hence,

$$\Pr_{\tilde{R}}[T_{\tilde{R}} \textit{ is inconsistent}] \leq \binom{m}{2} \cdot 2^{-2n} < \frac{m^2}{2^{2n+1}}.$$

The proposition follows:

$$\begin{aligned}
& \left|\Pr_{\tilde{R}}[C_M(T_{\tilde{R}})=1]-\Pr_{R}[C_M(T_R)=1]\right| \\
\leq\ & \left|\Pr_{\tilde{R}}[C_M(T_{\tilde{R}})=1 \mid T_{\tilde{R}} \textit{ is consistent}]-\Pr_{R}[C_M(T_R)=1]\right|\cdot \Pr_{\tilde{R}}[T_{\tilde{R}} \textit{ is consistent}] \\
& + \left|\Pr_{\tilde{R}}[C_M(T_{\tilde{R}})=1 \mid T_{\tilde{R}} \textit{ is inconsistent}]-\Pr_{R}[C_M(T_R)=1]\right|\cdot \Pr_{\tilde{R}}[T_{\tilde{R}} \textit{ is inconsistent}] \\
\leq\ & \Pr_{\tilde{R}}[T_{\tilde{R}} \textit{ is inconsistent}] \\
<\ & \frac{m^2}{2^{2n+1}}.
\end{aligned}$$

□

It remains to bound the advantage M might have in distinguishing between $T_{\tilde{R}}$ and T_S. The intuition is that for every possible and consistent M-transcript σ unless some "bad" and "rare" event on the choice of h_1 and h_2 (as in the definition of S) happens, the probability that $T_S=\sigma$ is exactly the same as the probability that $T_{\tilde{R}}=\sigma$. We now formally define this event (Definition 5.2.5) and bound its probability (Proposition 5.2.4).

Convention 5.2.1 *For any possible M-transcript $\sigma=\{\langle x_1,y_1\rangle,...,\langle x_m,y_m\rangle\}$ we can assume hereafter that if σ is consistent then for $i\neq j$ both $x_i\neq x_j$ and $y_i\neq y_j$ (this means that M never asks a query if its answer is determined by a previous query-answer pair).*

Definition 5.2.5 *For every specific choice of pair-wise independent permutations $h_1,h_2\in P_{2n}$ (in the definition of S) define $BAD(h_1,h_2)$ to be the set of all possible and consistent M-transcripts, $\sigma=\{\langle x_1,y_1\rangle,\ldots,\langle x_m,y_m\rangle\}$, satisfying:*

$$\exists 1\leq i<j\leq m \textit{ such that } h_1(x_i)_{|_R}=h_1(x_j)_{|_R} \textit{ or } h_2(y_i)_{|_L}=h_2(y_j)_{|_L}.$$

Proposition 5.2.4 *Let $h_1,h_2\in P_{2n}$ be pair-wise independent permutations then for any possible and consistent M-transcript $\sigma=\{\langle x_1,y_1\rangle,...,\langle x_m,y_m\rangle\}$ we have that:*

$$\Pr_{h_1,h_2}[\sigma\in BAD(h_1,h_2)]<\frac{m^2}{2^n}.$$

Proof. By definition, $\sigma\in \mathrm{BAD}(h_1,h_2)$ if there exist $1\leq i<j\leq m$ such that either $h_1(x_i)_{|_R}=h_1(x_j)_{|_R}$ or $h_2(y_i)_{|_L}=h_2(y_j)_{|_L}$. For any given i and j both $\Pr_{h_1}[h_1(x_i)_{|_R}=h_1(x_j)_{|_R}]$ and $\Pr_{h_2}[h_2(y_i)_{|_L}=h_2(y_j)_{|_L}]$ are smaller than 2^{-n} (since h_1 and h_2 are pair-wise independent). Therefore,

$$\Pr_{h_1,h_2}[\sigma\in \mathrm{BAD}(h_1,h_2)]<\binom{m}{2}\cdot 2\cdot 2^{-n}<\frac{m^2}{2^n}.$$

□

The key lemma for proving Theorem 5.2.2 is:

Lemma 5.2.5 *Let $\sigma = \{\langle x_1, y_1\rangle, ..., \langle x_m, y_m\rangle\}$ be any possible and consistent M-transcript, then*

$$\Pr_S[T_S = \sigma \mid \sigma \notin BAD(h_1, h_2)] = \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma].$$

Proof. Since σ is a possible M-transcript we have that for all $1 \leq i \leq m$,

$$C_M[\{\langle x_1, y_1\rangle, ..., \langle x_{i-1}, y_{i-1}\rangle\}] \in \{(+, x_i), (-, y_i)\}.$$

Therefore, $T_{\tilde{R}} = \sigma$ iff for all $1 \leq i \leq m$, the i^{th} answer $\tilde{R}$ gives is y_i in the case that $C_M[\{\langle x_1, y_1\rangle, ..., \langle x_{i-1}, y_{i-1}\rangle\}] = (+, x_i)$ and otherwise its i^{th} answer is x_i. Assume that $\tilde{R}$ answered "correctly" (i.e., y_i or x_i as above) for each one of the first $i-1$ queries. Then, by Convention 5.2.1 and the definition of $\tilde{R}$, its i^{th} answer is an independent and uniform $2n$-bit string. Therefore,

$$\Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma] = 2^{-2nm}.$$

Since σ is a possible M-transcript we have that $T_S = \sigma$ iff for all $1 \leq i \leq m$, $y_i = S(x_i)$. Consider any specific choice of permutations h_1 and h_2 (for which $S = S(h_1, f_1, f_2, h_2)$) such that $\sigma \notin \mathrm{BAD}(h_1, h_2)$. Let $(L_i^0, R_i^0) = h_1(x_i)$ and $(L_i^2, R_i^2) = h_2(y_i)$. By the definition of S, we get that:

$$y_i = S(x_i) \iff f_1(R_i^0) = L_i^0 \oplus L_i^2 \text{ and } f_2(L_i^2) = R_i^0 \oplus R_i^2.$$

For every $1 \leq i < j \leq m$ both $R_i^0 \neq R_j^0$ and $L_i^2 \neq L_j^2$ (otherwise $\sigma \in \mathrm{BAD}(h_1, h_2)$). Therefore, since f_1 and f_2 are random, we have that for every choice of h_1 and h_2 such that $\sigma \notin \mathrm{BAD}(h_1, h_2)$ the probability that $T_S = \sigma$ is exactly 2^{-2nm}. We can conclude:

$$\Pr_S[T_S = \sigma \mid \sigma \notin \mathrm{BAD}(h_1, h_2)] = 2^{-2nm},$$

which complete the proof of the lemma. □

Proof. **(of Theorem 5.2.2)** Let Γ be the set of all possible and consistent M-transcripts σ such that $M(\sigma) = 1$.

$$\begin{aligned}
& \left|\Pr_S[C_M(T_S) = 1] - \Pr_{\tilde{R}}[C_M(T_{\tilde{R}}) = 1]\right| \\
\leq\ & \left|\sum_{\sigma\in\Gamma}\left(\Pr_S[T_S = \sigma] - \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma]\right)\right| + \Pr_{\tilde{R}}[T_{\tilde{R}} \textit{ is inconsistent}] \\
\leq\ & \sum_{\sigma\in\Gamma}\left|\Pr_S[T_S = \sigma | \sigma \notin \mathrm{BAD}(h_1, h_2)] - \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma]\right| \cdot \Pr_{h_1,h_2}[\sigma \notin \mathrm{BAD}(h_1, h_2)] \qquad (5.1) \\
& + \left|\sum_{\sigma\in\Gamma}\left(\Pr_S[T_S = \sigma | \sigma \in \mathrm{BAD}(h_1, h_2)] - \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma]\right) \cdot \Pr_{h_1,h_2}[\sigma \in \mathrm{BAD}(h_1, h_2)]\right| \qquad (5.2) \\
& + \Pr_{\tilde{R}}[T_{\tilde{R}} \textit{ is inconsistent}]. \qquad (5.3)
\end{aligned}$$

We already showed in the proof of Proposition 5.2.3 that the value of Expression (3) is smaller than $\frac{m^2}{2^{2n+1}}$, by Lemma 5.2.5 we get that the value of Expression (1) is 0. Therefore, it remains to bound the value of Expression (2): Assume without loss of generality that

$$\begin{aligned} & \sum_{\sigma\in\Gamma} \Pr_S[T_S = \sigma \mid \sigma \in \mathrm{BAD}(h_1,h_2)] \cdot \Pr_{h_1,h_2}[\sigma \in \mathrm{BAD}(h_1,h_2)] \\ \leq & \sum_{\sigma\in\Gamma} \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma] \cdot \Pr_{h_1,h_2}[\sigma \in \mathrm{BAD}(h_1,h_2)], \end{aligned}$$

then using Proposition 5.2.4 we get that

$$\begin{aligned} & \left| \sum_{\sigma\in\Gamma} \left(\Pr_S[T_S = \sigma \mid \sigma \in \mathrm{BAD}(h_1,h_2)] - \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma] \right) \cdot \Pr_{h_1,h_2}[\sigma \in \mathrm{BAD}(h_1,h_2)] \right| \\ \leq & \sum_{\sigma\in\Gamma} \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma] \cdot \Pr_{h_1,h_2}[\sigma \in \mathrm{BAD}(h_1,h_2)] \\ \leq & \max_{\sigma\in\Gamma} \Pr_{h_1,h_2}[\sigma \in \mathrm{BAD}(h_1,h_2)] \\ < & \frac{m^2}{2^n}. \end{aligned}$$

Thus, we can conclude that:

$$\left| \Pr_S[C_M(T_S) = 1] - \Pr_{\tilde{R}}[C_M(T_{\tilde{R}}) = 1] \right| < \frac{m^2}{2^n} + \frac{m^2}{2^{2n+1}}.$$

Using Proposition 5.2.3 we complete the proof:

$$\begin{aligned} & \left| \Pr_S[M^{S,S^{-1}}(1^{2n}) = 1] - \Pr_R[M^{R,R^{-1}}(1^{2n}) = 1] \right| \\ = & \left| \Pr_S[C_M(T_S) = 1] - \Pr_R[C_M(T_R) = 1] \right| \\ \leq & \left| \Pr_S[C_M(T_S) = 1] - \Pr_{\tilde{R}}[C_M(T_{\tilde{R}}) = 1] \right| + \left| \Pr_{\tilde{R}}[C_M(T_{\tilde{R}}) = 1] - \Pr_R[C_M(T_R) = 1] \right| \\ < & \frac{m^2}{2^n} + \frac{m^2}{2^{2n}}. \end{aligned}$$

□

Given Theorem 5.2.2, the proof of Theorem 5.2.1 is essentially the same as the corresponding proof of the original LR-Construction (the proof of Theorem 1 of [90], given their main Lemma). The proof-idea is the following: Define three distributions:

- $S_1 = S(h_1, f_1, f_2, h_2)$, where $h_1, h_2 \in P_{2n}$ are pair-wise independent and $f_1, f_2 \in F_n$ are pseudo-random functions.
- $S_2 = S(h_1, g_1, f_2, h_2)$, where $h_1, h_2 \in P_{2n}$ are pair-wise independent, $f_2 \in F_n$ a pseudo-random function and $g_1 \in F_n$ a random function.
- $S_3 = S(h_1, g_1, g_2, h_2)$, where $h_1, h_2 \in P_{2n}$ are pair-wise independent and $g_1, g_2 \in F_n$ are random functions.

It is enough to show that for every oracle machine, for all but finite number of n:

1. $\left|\Pr[M^{S_1,S_1^{-1}}(1^{2n})=1]-\Pr[M^{S_2,S_2^{-1}}(1^{2n})=1]\right|\leq\varepsilon(n)$
2. $\left|\Pr[M^{S_2,S_2^{-1}}(1^{2n})=1]-\Pr[M^{S_3,S_3^{-1}}(1^{2n})=1]\right|\leq\varepsilon(n)$

If (1) or (2) do not hold, then we can construct an efficient oracle-machine M' that ε-distinguishes the pseudo-random functions from the random functions in contradiction to the assumption. Assume for example that for infinitely many n:

$$\left|\Pr[M^{S_1,S_1^{-1}}(1^{2n})=1]-\Pr[M^{S_2,S_2^{-1}}(1^{2n})=1]\right|>\varepsilon(n).$$

The oracle-machine M' on input 1^n and with access to a function $O\in F_n$ first samples pair-wise independent permutations, $h_1,h_2\in P_{2n}$, and a pseudo-random function $f_2\in F_n$. M' then invokes M with input 1^{2n} and answers its queries with the values of S and S^{-1}, for $S=S(h_1,O,f_2,h_2)$. When M halts so does M', and M' outputs whatever M outputs. Notice that if O is a pseudo-random function then the distribution of S is S_1, whereas if O is a truly random function then the distribution of S is S_2. This is the reason that M' distinguishes a pseudo-random function from a random one with advantage greater than $\varepsilon(n)$. Similar hybrid-arguments apply to all the other constructions of this chapter.

5.3 The Framework

As we shall see in Sections 5.4–5.6, the construction of Section 5.2 can be relaxed and generalized in several ways. The different pseudo-random permutations obtained share a similar structure and almost identical proof of security. In this section we examine the proof of Theorem 5.2.2 in a more abstract manner. Our goal is to establish a framework for proving (almost) all the constructions of this chapter and to suggest a way for designing and proving additional constructions.

Our framework deals with constructions of a pseudo-random permutation S on ℓ bits which is the composition of three permutations: $S\equiv h_2^{-1}\circ E\circ h_1$. (see Figure 5.2 for an illustration). In general, h_1 and h_2^{-1} are "lightweight" and E is where most of the work is done. E is constructed from pseudo-random functions and for the purpose of the analysis we assume (as in Theorem 5.2.2) that these functions are truly random. In Section 5.2, for example, $\ell=2n$, h_1 and h_2 are chosen as pair-wise independent permutations and $E\equiv\mathbf{D}_{f_2}\circ\mathbf{D}_{f_1}$ for random $f_1,f_2\in F_n$.

The framework starts with E which may be easily distinguished from a truly random permutation and transforms it via h_1 and h_2 into a pseudo-random permutation. The property E should have is that for almost every sequence, $\{\langle x_1,y_1\rangle,\ldots,\langle x_m,y_m\rangle\}$, the probability that $\forall i,\ y_i=E(x_i)$ is "close" to what we have for a truly random permutation:

Definition 5.3.1 *A sequence,* $\{\langle x_1,y_1\rangle,\ldots,\langle x_m,y_m\rangle\}$, *is* E-Good *if* $\Pr_E[\forall i,\ y_i=E(x_i)]=2^{-\ell\cdot m}$.

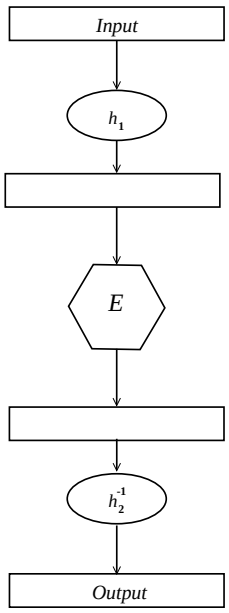

Figure 5.2: The high-level structure of the different constructions of SPPE.

We assume that apart from some "rare" sequences all others are E-Good. Loosely speaking, the role of h_1 and h_2 is to ensure that under any (adaptive chosen plaintext and ciphertext) attack on S the inputs and outputs of E form an E-Good sequence with a very high probability.

For the exact properties needed from the distributions on h_1, h_2 and E, we shall try to follow the statement and proof of Theorem 5.2.2. The goal is to show that S is indistinguishable from a truly random permutation R on ℓ bits. Specifically, that for some small ε (whose choice is explained hereafter), for any oracle machine M (not necessarily an efficient one) that makes at most m queries,

$$\left|\Pr[M^{S,S^{-1}}(1^\ell) = 1] - \Pr[M^{R,R^{-1}}(1^\ell) = 1]\right| \leq \varepsilon + \frac{m^2}{2^\ell}.$$

Let the notions of query-answer pair, a transcript, the function C_M, a possible M-transcript, the random process $\tilde{R}$, a consistent transcript and the random variables T_S, T_R and $T_{\tilde{R}}$ be as in the proof of Theorem 5.2.2. Proposition 5.2.3 (saying that the distance between T_R and $T_{\tilde{R}}$ is bounded by the probability that $T_{\tilde{R}}$ is inconsistent and that this probability is bounded by $\frac{m^2}{2^{\ell+1}}$) still holds. The heart of applying the framework is in specifying the "bad" M-transcripts for given h_1 and h_2. This set $\mathrm{BAD}_E(h_1, h_2)$ replaces $\mathrm{BAD}(h_1, h_2)$ in Definition 5.2.5 and in the rest of the proof. It contains possible and consistent M-transcripts and should have the property that any $\{\langle x_1, y_1\rangle, \ldots, \langle x_m, y_m\rangle\}$ *not in* $\mathrm{BAD}_E(h_1, h_2)$ satisfies that $\{\langle h_1(x_1), h_2(y_1)\rangle, \ldots, \langle h_1(x_m), h_2(y_m)\rangle\}$ is E-Good. Note that Definition 5.2.5 is indeed a special case of the above and also that, by this property,

$$\Pr_S[T_S = \sigma \mid \sigma \notin \mathrm{BAD}_E(h_1, h_2)] = 2^{-\ell \cdot m}.$$

This implies that Lemma 5.2.5 where $\mathrm{BAD}(h_1, h_2)$ is replaced with $\mathrm{BAD}_E(h_1, h_2)$ is true:

Lemma 5.3.1 *Let $\sigma = \{\langle x_1, y_1\rangle, ..., \langle x_m, y_m\rangle\}$ be any possible and consistent M-transcript, then*

$$\Pr_S[T_S = \sigma \mid \sigma \notin BAD_E(h_1, h_2)] = \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma].$$

For $\mathrm{BAD}_E(h_1, h_2)$ to be useful we must have that

$$\Pr_{h_1,h_2}[\sigma \in \mathrm{BAD}_E(h_1, h_2)] \leq \varepsilon \tag{5.1}$$

and this substitutes Proposition 5.2.4. This is the only place in the proof where we use the definition of ε and the definition of the distributions of h_1 and h_2. As demonstrated in Sections 5.4.2 & 5.6.1, there is actually a tradeoff between reducing the requirements from h_1 and h_2 and having a somewhat larger value of ε. Applying (5.1) and Lemma 5.3.1 as in the proof of Theorem 5.2.2 we conclude:

Theorem 5.3.2 *Let h_1, h_2, E be distributed over permutations in P_ℓ, let $S \equiv h_2^{-1} \circ E \circ h_1$ and let $R \in P_\ell$ be a random permutation. Suppose that $BAD_E(h_1, h_2)$ is as above and ε satisfies (5.1). Then for any oracle machine M (not necessarily an efficient one) that makes at most m queries,*

$$\left|\Pr[M^{S,S^{-1}}(1^\ell) = 1] - \Pr[M^{R,R^{-1}}(1^\ell) = 1]\right| \leq \varepsilon + \frac{m^2}{2^\ell}.$$

To summarize, the major point in proving the security of the different constructions is to define the set $\mathrm{BAD}_E(h_1, h_2)$ such that for any possible and consistent M-transcript, σ, both $\Pr_S[T_S = \sigma \mid \sigma \notin \mathrm{BAD}_E(h_1, h_2)] = 2^{-\ell \cdot m}$ and $\Pr_{h_1,h_2}[\sigma \in \mathrm{BAD}_E(h_1, h_2)] \leq \varepsilon$ (for the specific ε in the claim we are proving). This suggests that the critical step for designing a pseudo-random permutation, using the framework described in this section, is to come up with a permutation E such that the set of E-Good sequences is "large enough" and "nice enough". Note that to meet this end one can use different or more general definitions of an E-Good sequence with only minor changes to the proof (as is the case for the permutation $\hat{S}$ in Section 5.6).

5.4 Relaxing the Construction

5.4.1 PPE and SPPE with a Single Pseudo-Random Function

Since Luby and Rackoff introduced their construction a considerable amount of research [112, 114, 115, 116, 118, 130, 131, 132, 134, 151] has focused on the following question: Can we obtain a similar construction of PPE or SPPE such that every permutation will be constructed from a *single* pseudo-random function?

Apparently, this line of research originated in the work of Schnorr [134]. Schnorr considered the LR-Construction, where the functions used are truly random, as a pseudo-random generator that is secure if not too many bits are accessible. The security of Schnorr's generator does not depend on any unproven assumption. This notion of local-randomness is further treated in [92, 94]. Since the key of a random function is huge it makes sense to minimize

the number of functions and, indeed, Schnorr suggested $\mathbf{D}_f \circ \mathbf{D}_f \circ \mathbf{D}_f$ as pseudo-random (the suggested permutation was later shown to be distinguishable from random [130]).

Following is an informal description of some of these results. Let $f \in F_n$ be a random function. Then:

- For all $i, j, k \geq 1$ the permutation $\mathbf{D}_{f^i} \circ \mathbf{D}_{f^j} \circ \mathbf{D}_{f^k}$ is not pseudo-random [151].
- For all $i, j, k, \ell \geq 1$ the permutation $\mathbf{D}_{f^i} \circ \mathbf{D}_{f^j} \circ \mathbf{D}_{f^k} \circ \mathbf{D}_{f^\ell}$ is not strongly pseudo-random [131].
- $\mathbf{D}_{f^2} \circ \mathbf{D}_f \circ \mathbf{D}_f \circ \mathbf{D}_f$ is pseudo-random [118].
- $\mathbf{D}_f \circ \mathbf{D}_I \circ \mathbf{D}_{f^2} \circ \mathbf{D}_f \circ \mathbf{D}_I \circ \mathbf{D}_{f^2}$ is strongly pseudo-random, where $I \in F_n$ is the identity function [132].
- $\mathbf{D}_{f \circ \xi \circ f} \circ \mathbf{D}_f \circ \mathbf{D}_f$ is pseudo-random and $\mathbf{D}_{f \circ \xi \circ f} \circ \mathbf{D}_f \circ \mathbf{D}_f \circ \mathbf{D}_f$ is strongly pseudo-random, where ξ is, for example, a rotation of one bit [116].

A critique that has been voiced often is that using only one pseudo-random function does not seem too significant: A pseudo-random function on $n+2$ bits can replace 4 pseudo-random functions on n bits or, alternatively, a small key can be used to pseudo-randomly generate a larger key. It should also be noticed that the new constructions require additional invocations of the pseudo-random functions which imply an increase in the computation time. Furthermore, these results involve detailed and non-trivial proofs (to a point, where some papers claim to find inaccuracies in others).

The adjustment of the LR-Construction we suggest in Section 5.2 can easily be converted into a construction of PPE and SPPE from a single pseudo-random function. Simply replace both (pseudo-random) functions, f_1 and f_2, with a single (pseudo-random) function f. This solution does not suffer from the drawbacks of the previous ones. The construction and the proof remain as simple as before and the pseudo-random function is only invoked twice at each computation of the permutation. The additional key-length for the pair-wise independent functions (h_1 and h_2) is not substantial (especially compared to the length of a truly random function). Consider, for example, the construction of SPPE when we use a truly random function f:

Theorem 5.4.1 *Let $h_1, h_2 \in P_{2n}$ be pair-wise independent permutations and let $f \in F_n$ be a random function. Define $S = S(h_1, f, f, h_2)$ (as in Definition 5.2.1) and let $R \in P_{2n}$ be a random permutation. Then for any oracle machine M (not necessarily an efficient one) that makes at most m queries,*

$$\left|\Pr[M^{S,S^{-1}}(1^{2n}) = 1] - \Pr[M^{R,R^{-1}}(1^{2n}) = 1]\right| \leq \frac{2m^2}{2^n} + \frac{m^2}{2^{2n}}.$$

The proof follows the framework described in Section 5.3. The set $\mathrm{BAD}(h_1, h_2)$ (Definition 5.2.5) is replaced with the set $\mathrm{BAD}_1(h_1, h_2)$ defined to be:

The set of all possible and consistent M-transcripts, $\sigma = \{\langle x_1, y_1\rangle, \ldots, \langle x_m, y_m\rangle\}$, satisfying that there exist $1 \leq i < j \leq m$ such that either $h_1(x_i)_{|R} = h_1(x_j)_{|R}$ or $h_2(y_i)_{|L} = h_2(y_j)_{|L}$ (as before), or *there exist $1 \leq i, j \leq m$ such that $h_1(x_i)_{|R} = h_2(y_j)_{|L}$.*

In order to apply Theorem 5.3.2, it is enough to note that by this definition we get that for any possible and consistent M-transcripts, σ, both $\Pr_S[T_S = \sigma \mid \sigma \notin \mathrm{BAD}_1(h_1, h_2)] = 2^{-2nm}$ (hence, it is a proper definition according to the framework) and $\Pr_{h_1,h_2}[\sigma \in \mathrm{BAD}_1(h_1, h_2)] < \frac{2m^2}{2^n}$.

5.4.2 Relaxing the Pair-Wise Independence Requirement

One might interpret the construction of Section 5.2 in the following way: given the task of constructing *efficient* pseudo-random permutations it is enough to concentrate on the efficient construction of pseudo-random *functions*. The assumption that supports such a claim is that the computation of pseudo-random functions is much more expensive than the computation of pair-wise independent permutations. Therefore, computing the value of the pseudo-random permutation (that is constructed in Section 5.2) on any input of $2n$ bits is essentially equivalent to two invocations of a pseudo-random function with n-bit inputs. In this section we show that we can use even weaker permutations instead of the pair-wise independent ones – resulting in an even more efficient construction of pseudo-random permutations.

As mentioned in Section 5.3, the only place in Section 5.2 we use the fact that h_1 and h_2 are pair-wise independent permutations is in the proof of Proposition 5.2.4. In fact, the exact requirement on h_1 and h_2 we use is that for every $x \neq y$:

$$\Pr_{h_1}[h_1(x)_{|R} = h_1(y)_{|R}] \leq 2^{-n} \text{ and } \Pr_{h_2}[h_2(x)_{|L} = h_2(y)_{|L}] \leq 2^{-n}.$$

Furthermore, we can replace 2^{-n} with any $\varepsilon \geq 2^{-n}$ and still get a construction of pseudo-random permutations (with somewhat larger distinguishing probability). Consider, for example, the revised statement of Theorem 5.2.2:

Theorem 5.4.2 *Let H^1 and H^2 be distributions of permutations in P_{2n} such that for every pair of $2n$-bit strings $x \neq y$:*

$$\Pr_{h_1 \in H^1}[h_1(x)_{|R} = h_1(y)_{|R}] \leq \varepsilon \text{ and } \Pr_{h_2 \in H^2}[h_2(x)_{|L} = h_2(y)_{|L}] \leq \varepsilon.$$

Let h_1 be distributed according to H^1, h_2 distributed according to H^2 and let $f_1, f_2 \in F_n$ be random functions. Define $S = S(h_1, f_1, f_2, h_2)$ (as in Definition 5.2.1) and let $R \in P_{2n}$ be a random permutation. Then for any oracle machine M (not necessarily an efficient one) that makes at most m queries,

$$\left|\Pr[M^{S,S^{-1}}(1^{2n}) = 1] - \Pr[M^{R,R^{-1}}(1^{2n}) = 1]\right| < m^2 \cdot \varepsilon + \frac{m^2}{2^{2n}}.$$

The proof follows the framework described in Section 5.3. This time the definition of $\mathrm{BAD}(h_1, h_2)$ stays unchanged and, in order to apply Theorem 5.3.2, we only need to note that for any possible and consistent M-transcript σ, $\Pr_{h_1,h_2}[\sigma \in \mathrm{BAD}(h_1, h_2)] < m^2 \cdot \varepsilon$.

The conditions on H^1 and H^2 in Theorem 5.4.2 are somewhat nonstandard (since the requirements are on half the bits of the output). Nevertheless, these conditions are satisfied by more traditional requirements on function-families. In particular, one can use the concept of ε-AXU_2 functions:

Definition 5.4.1 *A distribution on $I^n \mapsto I^n$ functions (or permutations), H, is ε-AXU_2 if for every $x \neq y$ and every z ($x, y, z \in I^n$),*

$$\Pr_{h \in H}[h(x) \oplus h(y) = z] \leq \varepsilon.$$

This concept was originally defined by Carter and Wegman [38]; We use the terminology of Rogaway [128].

It is easy to verify that the conditions on H^1 and H^2 in Theorem 5.4.2 are satisfied if both H^1 and H^2 are $((2^n - 1)^{-1} \cdot \varepsilon)$-$\mathrm{AXU}_2$. Such a distribution of permutations over I^{2n}, for $\varepsilon = (2^n + 1)^{-1}$, is $h_a(x) \stackrel{\mathrm{def}}{=} a \cdot x$ where a is uniform in $I^{2n} \setminus \{0\}$ and the multiplication is in $GF(2^{2n})$.

Another way to construct H^1 and H^2 is by *using Feistel permutations* with ε-AXU_2 *functions*. Let H be a distribution of ε-AXU_2 functions on n bits strings, then we can define H^1 to be $\{\mathbf{D}_h\}_{h \in H}$ and H^2 to be $\{\mathbf{D}_h^{-1}\}_{h \in H}$. The reason is that for every two *distinct* $2n$-bit strings $x = (L^1, R^1)$ and $y = (L^2, R^2)$ and every function $h \in F_n$ we have by definition that:

$$\mathbf{D}_h(x)_{|R} = \mathbf{D}_h(y)_{|R} \iff h(R^1) \oplus h(R^2) = L^1 \oplus L^2.$$

If $R^1 = R^2$ then $L^1 \neq L^2$ and therefore $\mathbf{D}_h(x)_{|R} \neq \mathbf{D}_h(y)_{|R}$; otherwise, by the definition of ε-AXU_2 functions:

$$\Pr_{h \in H}[\mathbf{D}_h(x)_{|R} = \mathbf{D}_h(y)_{|R}] = \Pr_{h \in H}[h(R^1) \oplus h(R^2) = L^1 \oplus L^2] \leq \varepsilon.$$

Thus, H^1 satisfies its requirement and similarly for H^2.

By using Feistel permutations to construct H^1 and H^2 we get the original LR-Construction as a special case (since a random function is in particular 2^{-n}-AXU_2). Thus, the proof of security in Section 5.2 also holds for the original LR-Construction. The idea of using ε-AXU_2 functions instead of pseudo-random functions for the first round of the LR-Construction was previously suggested by Lucks [91].

Another advantage of this approach is that it allows us to use many efficient constructions of function families. An example of efficient 2^{-n}-AXU_2 functions are Vazirani's "shift"-family [147]. A key of such a function is a uniformly chosen string $a \in I^{2n-1}$ and the j^{th} bit of $f_a(x)$ $(1 \leq j \leq n)$ is defined to be $\sum_{i=1}^{n} x_i a_{j+i-1} \bmod 2$.

A substantial amount of research [38, 69, 84, 128, 145, 149] deals with the construction of *efficient* hash functions. This line of work contains constructions that obey weaker definitions on function families than pair-wise independence and in particular contains constructions of ε-AXU_2 functions. Unfortunately, these functions were designed to be especially efficient

when their output is substantially smaller than their input (since, they were mainly brought up in the context of authentication) which is not true in our case (but is relevant in Section 5.6). An additional objective is to reduce the size of the family of hash functions (e.g., [67, 84]). In our setting the purpose of this is to reduce the key-length of the pseudo-random permutations.

5.5 Reducing the Distinguishing Probability

There are various circumstances where it is desirable to have a pseudo-random permutation on relatively *few* bits (say 128). This is especially true when we want to minimize the size of the hardware-circuit that implements the permutation or the communication bandwidth with the (hardware or software) component that computes the permutation.

Let F be a pseudo-random permutation on ℓ bits (note that $n = \ell/2$ in Section 5.2) constructed from truly random functions (on $\ell/2$ bits) using the LR-Construction. As shown by Patarin [115], F can be distinguished (with constant probability) from a random permutation using $O(2^{\ell/4})$ queries (which means that the analysis of the LR-Construction, where the distinguishing probability for m queries is $O(\frac{m^2}{2^{\ell/2}})$, is tight). Therefore, the LR-Construction on ℓ bits can only be used if $2^{\ell/4}$ is large enough to bound the number of queries in the attack on the block cipher.

In this section, a simple generalization of the construction of Section 5.2 is presented. Using this construction, the adversary's probability of distinguishing between the pseudo-random and random permutations can be reduced to roughly $\frac{t}{2} \cdot \frac{m^2}{2^{(1-1/t)\ell}}$ for every integer $2 \leq t \leq \ell$ (for $t = 2$ we get the original construction). To achieve this security $t + 2$ permutations are composed. The initial and final are pair-wise independent permutations, the rest are (generalized) Feistel permutations defined by $I^{(1-1/t)\ell} \mapsto I^{\ell/t}$ random (or pseudo-random) functions (see Figure 5.3 for an illustration).

Patarin [117] shows that if we take six rounds of the LR-Construction (instead of three or four), then the resulting permutation cannot be distinguished from a random permutation with advantage better than $\frac{5m^3}{2^\ell}$ (improving [115]). This means that distinguishing the six-round construction from a truly random permutation (with constant probability) requires at least $\Omega(2^{\ell/3})$ queries. The bound we achieve in this section ($\Omega(2^{(1-1/t)\cdot\ell/2})$) is better (for any $t \geq 4$). Note that our construction uses pseudo-random functions with larger input-length, which might be a disadvantage for some applications.

In order to describe our generalized constructions we first extend Feistel permutations to deal with the case where the underlying functions have arbitrary input and output lengths (instead of length-preserving functions as in Definition 5.1.1). We note that using such "unbalanced" Feistel permutations was previously suggested in [7, 91, 133].

Definition 5.5.1 (Generalized Feistel Permutations) *For any two positive integers, s and ℓ', and any function $f : I^{\ell'} \mapsto I^s$ let $\ell = \ell' + s$ and let $\mathbf{D}_f \in P_\ell$ be the permutation defined by $\mathbf{D}_f(L, R) \stackrel{\text{def}}{=} (R, L \oplus f(R))$, where $|L| = s$ and $|R| = \ell'$.*

We can now define the revised construction and consider its security. These are simple generalizations of the construction in Section 5.2 and of its proof of security.

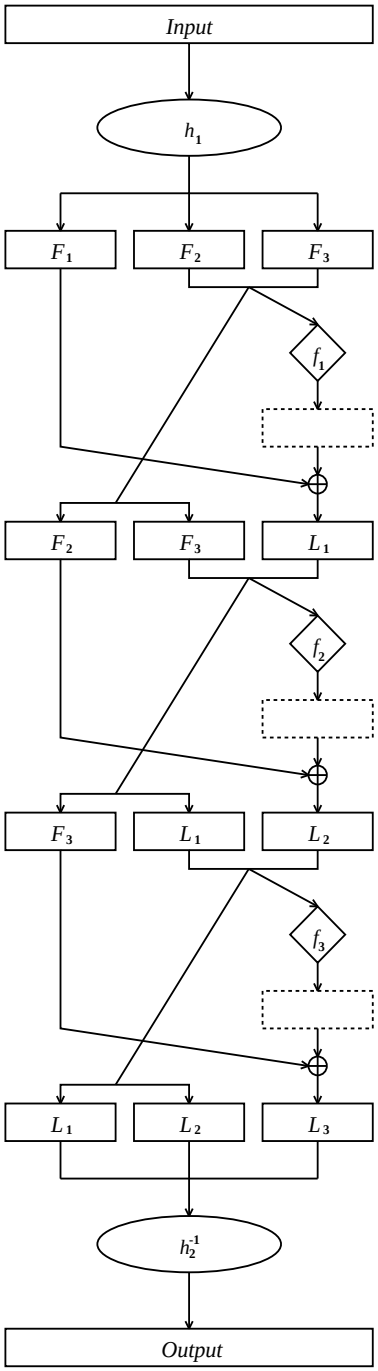

Figure 5.3: Construction of strong pseudo-random permutations with reduced distinguishing probability using $t+2$ rounds (here $t = 3$). Recall, $f_i : I^{(1-1/t)\ell} \mapsto I^{\ell/t}$ (here $f_i : I^{2\ell/3} \mapsto I^{\ell/3}$).

Definition 5.5.2 ($t+2$-Round Construction) *For any integers $2 \leq t \leq \ell$, let s and r be integers such that $\ell = s \cdot t + r$ (where $r < t$). For any $h_1, h_2 \in P_\ell$, $f_1, f_2, \ldots, f_r : I^{\ell-s-1} \mapsto I^{s+1}$ and $f_{r+1}, \ldots, f_t : I^{\ell-s} \mapsto I^s$ define*

$$W(h_1, f_1, f_2 \ldots, f_t) \stackrel{\text{def}}{=} \mathbf{D}_{f_t} \circ \mathbf{D}_{f_{t-1}} \circ \ldots \circ \mathbf{D}_{f_1} \circ h_1$$

and

$$S(h_1, f_1, f_2, \ldots, f_t, h_2) \stackrel{\text{def}}{=} h_2^{-1} \circ \mathbf{D}_{f_t} \circ \mathbf{D}_{f_{t-1}} \circ \ldots \circ \mathbf{D}_{f_1} \circ h_1.$$

(We get the construction of Definition 5.2.1 by choosing $t = 2$, $s = \ell/2$ and $r = 0$.)

Theorem 5.5.1 *Let W and S be as in Definition 5.5.2, where h_1 and h_2 are pair-wise independent permutations and $f_1, f_2, \ldots, f_t$ are pseudo-random functions (t is allowed to be a function of ℓ); h_1, h_2 and $f_1, f_2, \ldots, f_t$ are independently chosen. Then W is a pseudo-random permutation and S a strong pseudo-random permutation.*

Furthermore, assume that no efficient oracle-machine that makes at most $m = m(\ell)$ queries, ε-distinguishes between the pseudo-random functions and random functions for $\varepsilon = \varepsilon(n)$. Then no efficient oracle-machine that makes at most m queries to W (resp. S and S^{-1}) ε'-distinguishes W (resp. S) from a random permutation, for $\varepsilon' = t \cdot \varepsilon + \frac{t}{2} \cdot \frac{m^2}{2^{\ell - \lceil \ell/t \rceil}} + \frac{m^2}{2^\ell}$.

In case the middle functions are truly random this reduces to:

Theorem 5.5.2 *Let S be as in Definition 5.5.2, where h_1 and h_2 are pair-wise independent permutations and $f_1, f_2, \ldots, f_t$ are random functions and let $R \in P_\ell$ be a random permutation. Then for any oracle machine M (not necessarily an efficient one) that makes at most m queries,*

$$\left| \Pr[M^{S, S^{-1}}(1^\ell) = 1] - \Pr[M^{R, R^{-1}}(1^\ell) = 1] \right| \leq \frac{t}{2} \cdot \frac{m^2}{2^{\ell - \lceil \ell/t \rceil}} + \frac{m^2}{2^\ell}.$$

The proof of Theorem 5.5.2 follows the framework described in Section 5.3. Assume for simplicity that $\ell = s \cdot t$, the set $\mathrm{BAD}(h_1, h_2)$ (Definition 5.2.5) is replaced with the set $\mathrm{BAD}_2(h_1, h_2)$ defined to be:

> *The set of all possible and consistent M-transcripts, $\sigma = \{\langle x_1, y_1 \rangle, \ldots, \langle x_m, y_m \rangle\}$, satisfying that there exist $1 \leq i < j \leq m$ and $1 \leq k \leq t$ such that*
>
> $$(F_i^{k+1}, \ldots, F_i^t, L_i^1, \ldots, L_i^{k-1}) = (F_j^{k+1}, \ldots, F_j^t, L_j^1, \ldots, L_j^{k-1}),$$
>
> *where $(F_i^1, F_i^2, \ldots, F_i^t) = h_1(x_i)$ and $(L_i^1, L_i^2, \ldots, L_i^t) = h_2(y_i)$ ($|F_i^1| = |F_i^2| = \ldots = |F_i^t| = |L_i^1| = |L_i^2| = \ldots = |L_i^t| = s$).*

This guarantees that for any possible and consistent M-transcript σ we have that $\Pr_S[T_S = \sigma \mid \sigma \notin \mathrm{BAD}_2(h_1, h_2)] = 2^{-\ell m}$ (and hence, it is a proper definition according to the framework). The reason is that, under the notation above,

$$\forall i, y_i = S(x_i) \Longleftrightarrow \forall 1 \leq i \leq m,\ \forall 1 \leq k \leq t,\ f_k(F_i^{k+1}, \ldots, F_i^t, L_i^1, \ldots, L_i^{k-1}) = F_i^k \oplus \mathrm{L}_i^k.$$

Therefore, given any specific choice of h_1 and h_2 (in the definition of S) such that $\sigma \notin \mathrm{BAD}_2(h_1, h_2)$ the event $T_S = \sigma$ is composed of $m \cdot t$ independent events each of which has probability 2^{-s} to happen. In order to apply Theorem 5.3.2, it remains to note that for any such σ we have that

$$\Pr_{h_1,h_2}[\sigma \in \mathrm{BAD}_2(h_1, h_2)] < t \cdot \binom{m}{2} \cdot 2^{-(\ell - \lceil \ell/t \rceil)} < \frac{t}{2} \cdot \frac{m^2}{2^{\ell - \lceil \ell/t \rceil}}.$$

Remark 5.5.1 *The construction of this section achieves a substantial improvement in security over the construction in Section 5.2 even for a small constant $t > 2$ (that is, with a few additional applications of the pseudo-random functions). Nevertheless, it might be useful for some applications to take a larger value of t. Choosing $t = \ell$ reduces the advantage the distinguisher may achieve to roughly $\frac{\ell \cdot m^2}{2^\ell}$.*

5.6 SPPE on Many Blocks Using PFE or PPE on a Single Block

Consider the application of pseudo-random permutations to encryption, i.e., using $f(M)$ in order to encrypt a message M, where f is a pseudo-random permutation. Assume also that we want to use DES for this purpose. We now have the following problem: while DES works on fixed and relatively small length strings, we need a permutation on $|M|$-bit long strings, where the length of the message, $|M|$, may be large and may vary between different messages.

This problem is not restricted to the usage of DES (though the fact that DES was designed for hardware implementation contributes to it). Usually, a direct construction of pseudo-random permutations or pseudo-random functions (if we want to employ the LR-Construction) with large input-length is expensive. Therefore, we would like a way to construct pseudo-random permutations (or functions) on *many blocks* from pseudo-random permutations (or functions) on *a single block*.

Several such constructions were suggested in the context of DES (see e.g. [31] for the different modes of operation for DES). The simplest, known as the electronic codebook mode (ECB-mode), is to divide the input into sub-blocks and to apply the pseudo-random permutation on each sub-block separately. This solution suffers from the obvious drawback that every sub-block of output solely depends on a single sub-block of input (and, in particular, the permutation on the complete input is not pseudo-random). This may leak information about the message being encrypted. See Section 5.6.2 for further discussion and additional related work.

In this section we consider a generalization of the construction of Section 5.2 that uses pseudo-random functions (or permutations) on a *single* block to construct strong pseudo-random permutations on *many* blocks. The idea is as follows: apply a pair-wise independent permutation on the entire input, divide the value you get into sub-blocks and apply two rounds of Feistel-permutations (or one round of a pseudo-random permutation) on each sub-block separately, finally, apply a second pair-wise independent permutation on the entire value you get (see Figure 5.4 for an illustration).

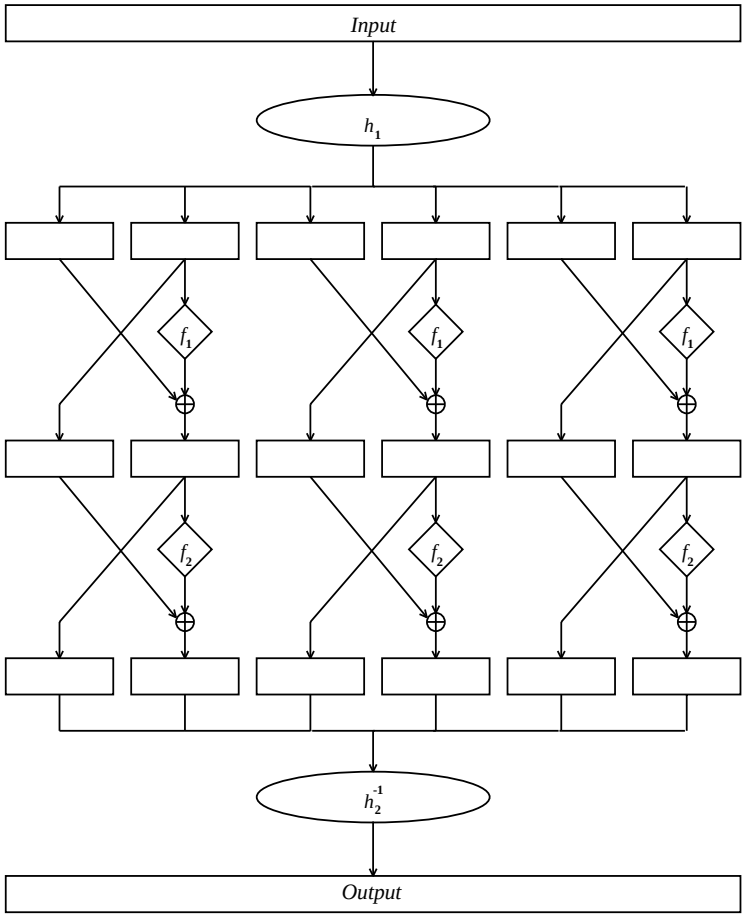

Figure 5.4: Construction of a strong pseudo-random permutation on many (six in this case) blocks from a pseudo-random function on a single block.

This solution resembles the ECB-mode, it is almost as simple and it is highly suitable for parallel implementation. Contrary to the ECB-mode, this construction does give a pseudo-random permutation on *the entire message* (though the security parameter is still relative to the length of a sub-block).

For simplicity, we only describe the construction using truly random functions (or a truly random permutation). The analysis of the construction when pseudo-random functions are used follows easily. In addition, we restrict our attention to the construction of *strong* pseudo-random permutations.

Definition 5.6.1 *For any two integers b and s, for any function $g \in F_s$ let $g^{\times b} \in F_{b \cdot s}$ be the function defined by:*

$$g^{\times b}(x_1, x_2, \ldots, x_b) \stackrel{\text{def}}{=} (g(x_1), g(x_2), \ldots, g(x_b)).$$

For any $f_1, f_2 \in F_n$ and $h_1, h_2 \in P_{2nb}$, define:

$$S(h_1, f_1, f_2, h_2) \stackrel{\text{def}}{=} h_2^{-1} \circ \mathbf{D}_{f_2}^{\times b} \circ \mathbf{D}_{f_1}^{\times b} \circ h_1.$$

For any $p \in P_{2n}$ and $h_1, h_2 \in P_{2nb}$, define:

$$\hat{S}(h_1, p, h_2) \stackrel{\text{def}}{=} h_2^{-1} \circ p^{\times b} \circ h_1.$$

Theorem 5.6.1 *Let $h_1, h_2 \in P_{2nb}$ be pair-wise independent permutations, let $f_1, f_2 \in F_n$ be random functions and $p \in P_{2n}$ a random permutation. Define $S = S(h_1, f_1, f_2, h_2)$ and $\hat{S} = \hat{S}(h_1, p, h_2)$ (as in Definition 5.6.1) and let $R \in P_{2nb}$ be a random permutation. Then for any oracle machine M (not necessarily an efficient one) that makes at most m queries,*

$$\left|\Pr[M^{S,S^{-1}}(1^{2nb}) = 1] - \Pr[M^{R,R^{-1}}(1^{2nb}) = 1]\right| \leq \frac{m^2 \cdot b^2}{2^n} + \frac{m^2}{2^{2nb}}$$

and

$$\left|\Pr[M^{\hat{S},\hat{S}^{-1}}(1^{2nb}) = 1] - \Pr[M^{R,R^{-1}}(1^{2nb}) = 1]\right| \leq \frac{m^2 \cdot b^2}{2^{2n-1}}.$$

The proof of Theorem 5.6.1 for S follows the framework described in Section 5.3. The set $\mathrm{BAD}(h_1, h_2)$ (Definition 5.2.5) is replaced with the set $\mathrm{BAD}_3(h_1, h_2)$ defined to be:

> *The set of all possible and consistent M-transcripts, $\sigma = \{\langle x_1, y_1\rangle, \ldots, \langle x_m, y_m\rangle\}$, such that either there are two equal values in $\{F_i^{2j}\}_{1 \leq i \leq m,\ 1 \leq j \leq b}$ or there are two equal values in $\{L_i^{2j-1}\}_{1 \leq i \leq m,\ 1 \leq j \leq b}$, where $(F_i^1, F_i^2, \ldots, F_i^{2b}) = h_1(x_i)$ and $(L_i^1, L_i^2, \ldots, L_i^{2b}) = h_2(y_i)$ ($|F_i^1| = |F_i^2| = \ldots = \left|F_i^{2b}\right| = |L_i^1| = |L_i^2| = \ldots = \left|L_i^{2b}\right| = n$).*

This guarantees that for any possible and consistent M-transcript σ we have that

$$\Pr_S[T_S = \sigma \mid \sigma \notin \mathrm{BAD}_3(h_1, h_2)] = 2^{-2n \cdot b \cdot m}$$

(and hence, it is a proper definition according to the framework). The reason is that, under the notation above,

$$\forall i, y_i = S(x_i) \Longleftrightarrow \forall 1 \leq i \leq m,\ \forall 1 \leq j \leq b,\ f_1(F_i^{2j}) = F_i^{2j-1} \oplus L_i^{2j-1} \text{ and } f_2(L_i^{2j-1}) = F_i^{2j} \oplus L_i^{2j}.$$

Therefore, given any specific choice of h_1 and h_2 (in the definition of S) such that $\sigma \notin \mathrm{BAD}_3(h_1, h_2)$ the event $T_S = \sigma$ is composed of $2m \cdot b$ independent events each of which has probability 2^{-n} to happen. In order to apply Theorem 5.3.2, it remains to note that for any such σ we have that

$$\Pr_{h_1,h_2}[\sigma \in \mathrm{BAD}_3(h_1, h_2)] \leq 2 \cdot \binom{m \cdot b}{2} \cdot 2^{-n} < \frac{m^2 \cdot b^2}{2^n}.$$

The proof of Theorem 5.6.1 for $\hat{S}$ slightly deviates from the framework described in Section 5.3 (providing yet another evidence to the claim that "nobody is perfect"). The set $\mathrm{BAD}(h_1, h_2)$ (Definition 5.2.5) is replaced with the set $\mathrm{BAD}_4(h_1, h_2)$ defined to be:

> *The set of all possible and consistent M-transcripts, $\sigma = \{\langle x_1, y_1\rangle, \ldots, \langle x_m, y_m\rangle\}$, such that either there are two equal values in $\{F_i^j\}_{1 \leq i \leq m,\ 1 \leq j \leq b}$ or there are two equal values in $\{L_i^j\}_{1 \leq i \leq m,\ 1 \leq j \leq b}$, where $(F_i^1, F_i^2, \ldots, F_i^b) = h_1(x_i)$ and $(L_i^1, L_i^2, \ldots, L_i^b) = h_2(y_i)$ ($|F_i^1| = |F_i^2| = \ldots = \left|F_i^b\right| = |L_i^1| = |L_i^2| = \ldots = \left|L_i^b\right| = 2n$).*

Now we have that for any possible and consistent M-transcript σ,

$$\Pr_{h_1,h_2}[\sigma \in \mathrm{BAD}_4(h_1, h_2)] \leq 2 \cdot \binom{m \cdot b}{2} \cdot 2^{-2n} < \frac{m^2 \cdot b^2}{2^{2n}}$$

but now for any such σ,

$$\Pr_{\hat{S}}[T_{\hat{S}} = \sigma \mid \sigma \notin \mathrm{BAD}_4(h_1, h_2)] = \frac{2^{2n}!}{(2^{2n} - m \cdot b)!}$$

instead of $2^{-2n \cdot b \cdot m}$ as "required" by the framework. However, the difference in probabilities is rather small which result in only a minor deviation from the proof of Theorem 5.2.2.

5.6.1 Relaxing the Construction

As in Section 5.4.2 we would like to reduce the requirements from h_1 and h_2 in Theorem 5.6.1. Our main motivation in doing so is to *decrease the key-length* of the pseudo-random permutations. We would like the key-length to be of order n – the length of the small sub-blocks and *not* of order $2nb$ – the length of the complete input (in some cases we may allow a small dependence on b).

We sketch a way to redefine the distributions on h_1 and h_2 in the definition of $\hat{S}$ (almost the same ideas apply to the definition of S). The requirement these distributions have to obey is that for any possible and consistent M-transcript σ we have that $\Pr_{h_1,h_2}[\sigma \in \mathrm{BAD}_4(h_1, h_2)]$ is "small". We use the following notation: For any $2n \cdot b$-bit string $z = (z_1, z_2, \ldots, z_b)$ (such that $\forall j,\ |z_j| = 2n$) and for all $1 \leq i \leq b$, denote by $z_{|_i}$ the substring z_i (the i^{th} substring of z). The requirement above can be achieved by sampling h_1 and h_2 according to a permutation distribution H such that for some small $\varepsilon \geq 2^{-2n}$ we have that:

1. For any $2n \cdot b$-bit string x, $\forall 1 \leq i < j \leq b$, $\Pr_{h \in H}[h(x)_{|_i} = h(x)_{|_j}] \leq \varepsilon$ and
2. For any $2n \cdot b$-bit strings $x \neq x'$, $\forall 1 \leq i, j \leq b$, $\Pr_{h \in H}[h(x)_{|_i} = h(x')_{|_j}] \leq \varepsilon$.

We start by defining a permutation distribution H' that almost achieves this: A permutation $h' = h'_{u_1,u_2}$ sampled from H' is defined by two ε'-AXU$_2$ functions, $u_1 : I^{2n} \mapsto I^{2n}$ and $u_2 : I^{\lceil \log b \rceil} \mapsto I^{2n}$ (see definition of ε-AXU$_2$ functions in Section 5.4.2). For any $z = (z_1, z_2, \ldots, z_b)$ (such that $\forall j,\ |z_j| = 2n$),

$$h'_{u_1,u_2}(z) \stackrel{\text{def}}{=} (z_1 \oplus u_1(z_b) \oplus u_2(1), z_2 \oplus u_1(z_b) \oplus u_2(2), \ldots, z_{b-1} \oplus u_1(z_b) \oplus u_2(b-1), z_b \oplus u_2(b)).$$

It is not hard to verify that:

1′ For any $2n \cdot b$-bit string x, $\forall 1 \leq i < j \leq b$, $\Pr_{h' \in H'}[h'(x)_{|_i} = h'(x)_{|_j}] \leq \varepsilon'$ and

2′ For any $2n \cdot b$-bit strings $x \neq x'$ *such that* $x_{|_b} \neq x'_{|_b}$ and for all $1 \leq i, j \leq b$, $\Pr_{h' \in H'}[h'(x)_{|_i} = h'(x')_{|_j}] \leq \varepsilon'$.

In order to eliminate the additional requirement in (2′) that $x_{|_b} \neq x'_{|_b}$, we define the permutation distribution H such that a permutation h sampled from H is defined to be $h' \circ \mathbf{D}_g$ (see Definition 5.5.1), where h' is sampled according to H' and $g : I^{2n \cdot (b-1)} \mapsto I^{2n}$ is a ε'-AXU$_2$ function (see Figure 5.5 for an illustration). Using (1′) and (2′) and the fact that for any $2n \cdot b$-bit strings $x \neq x'$:

$$\Pr_g[\mathbf{D}_g(x)_{|_b} = \mathbf{D}_g(x')_{|_b}] \leq \varepsilon',$$

we get that H satisfies (1) and (2) for $\varepsilon = 2\varepsilon'$.

Notice that the computation of a function $h \in H$ is essentially equivalent to one computation of an ε-AXU$_2$ function, $g : I^{2n \cdot (b-1)} \mapsto I^{2n}$, and a few additional XOR operations per block. Using efficient constructions of ε-AXU$_2$ functions [38, 69, 84, 128, 145, 149] we get an efficient function h. Krawczyk [84] shows a construction of $\frac{m+\ell}{2^{\ell-1}}$-AXU$_2$ functions from m bits to ℓ bits with ℓ key-bits. Using these functions we can achieve the desired goal of reducing the key-length of h to O(n) bits.

5.6.2 Related Work

The construction presented in this section is certainly not the only solution to the problem at hand. We refer in brief to some additional solutions:

As mentioned above, DES modes of operation were suggested as a way of encrypting long messages. However, none of these modes constitutes a construction of a pseudo-random permutation[2]. Note that when the encryption of a message M is $f(M)$, for a pseudo-random permutation f, then the only information that is leaked on M is whether or not M is equal to a previously encrypted message. This is not true for DES modes of operation. For instance, when using the cipher block chaining mode (CBC-mode), the encryptions of two

[2]However, as shown by Bellare et. al. [18], the CBC-mode does define a construction of a pseudo-random *function* with small output length. A somewhat related solution to this problem is the so called *cascade* construction that is considered by Bellare et. al. [14].

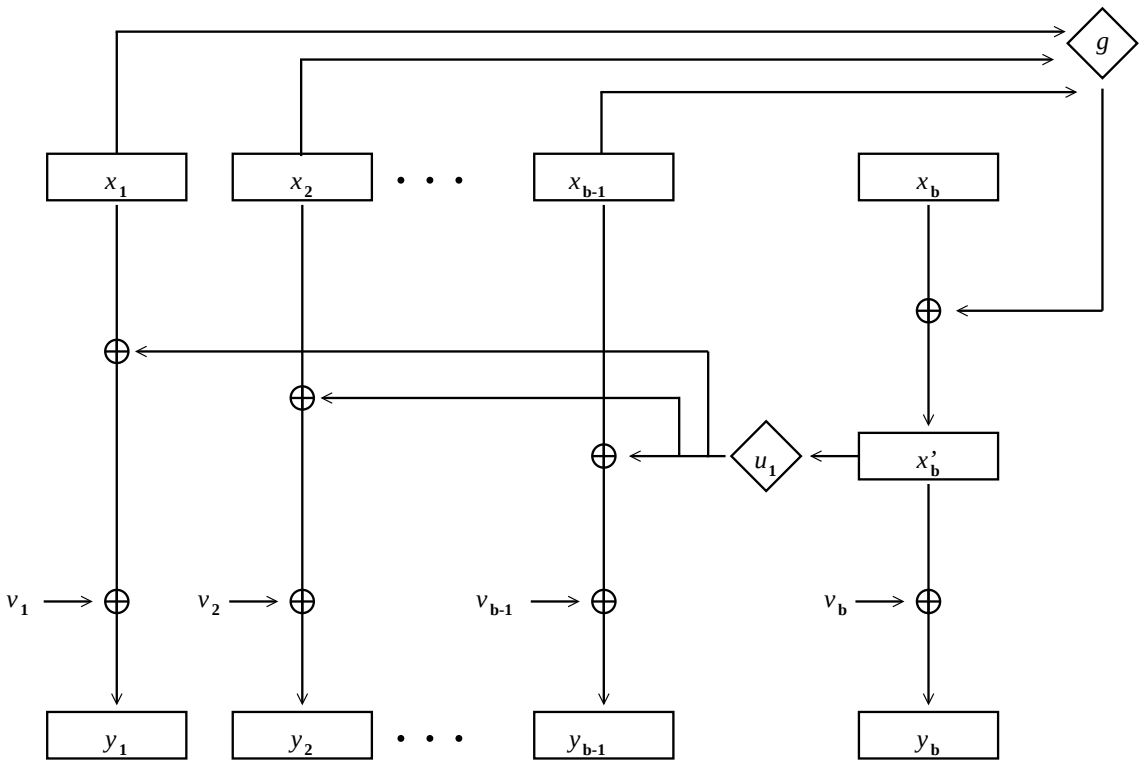

Figure 5.5: The Construction of $h \in H$. Each v_i denotes the string $u_2(i)$

messages with identical prefix will also have an identical prefix. The ECB-mode leaks even more information – the existence of two identical sub-blocks (in two different encrypted messages or in a single message). The reason that the ECB-mode leaks so much information is that every ciphertext-block solely depends on a single plaintext-block. Our construction implies that only very little and "non-cryptographic" diffusion (the permutations h_1 and h_2) is required in order to overcome this flaw of the ECB-mode.

Bellare and Rogaway [20] show how to convert the CBC-mode in order to construct a pseudo-random permutation with large input-length (this is the only place we are aware of that explicitly refers to the problem). The amount of work in their construction is comparable with two applications of the original CBC-mode (approximately twice the work of our construction, assuming that h_1 and h_2 are relatively efficient). The security of this construction is of similar order to the security of our construction. In contrast to our construction, [20] (as well as [14, 18]) is sequential in nature.

A different approach is to define a length-preserving pseudo-random function $\tilde{F}$ on $\tilde{\ell}$ bits using a length-preserving pseudo-random function F on ℓ bits (where $\ell < \tilde{\ell}$) and then to apply our version of the LR-Construction using $\tilde{F}$ in order to get a pseudo-random *permutation* on $2 \cdot \tilde{\ell}$ bits. The function $\tilde{F}$ can be defined to be $G \circ F \circ h$ where, h is a pair-wise independent hash function from $\tilde{\ell}$ bits to ℓ bits and G a pseudo-random (bit) generator from ℓ bits to $\tilde{\ell}$ bits. This idea may be attributed in part to Carter and Wegman [149]. Anderson and Biham [7] and Lucks [91] show how to directly apply similar ideas into the LR-Construction. A comparison between this approach and our construction relies on the specific parameters

of the different primitives that are used. In particular, the parameters of the pseudo-random function F vs. the pseudo-random generator G. For instance, for this approach to be more efficient than our construction we need that one application of G would be more efficient than $\lceil \tilde{\ell}/\ell \rceil$ applications of F.

Reducing the Distinguishing Probability

All the constructions of a pseudo-random permutation on many blocks from a pseudo-random function (or permutation) on a single block that are described in this subsection (including ours) have the following weakness: If the length of a single block is too small (e.g., 64-bits), then the pseudo-random permutation on many blocks is very weak even when the original pseudo-random function (or permutation) is very secure (e.g., completely random). In the following few paragraphs we discuss this problem and a way to overcome it.

Consider the permutation $S = S(h_1, f_1, f_2, h_2)$ (as in Definition 5.6.1), where $h_1, h_2 \in P_{2nb}$ are pair-wise independent permutations and $f_1, f_2 \in F_n$ are random functions. Our analysis of the security of S (Theorem 5.6.1) fails when the number of queries that the adversary makes is $\Omega(2^{n/2}/b)$ (in fact this analysis is tight). Having $2^{n/2}/b$ large enough forces a significant restriction on n. Therefore, a natural question is whether we can improve the security of the construction. A simple Information-Theoretic argument implies that all such constructions can be distinguished from random using $O(2^n/b)$ queries. This follows from the fact that with $O(2^n/b)$ queries the adversary gets much more bits than the length of the permutation's secret-key. Hence, the distribution of the answers to these queries is statistically very different from uniform (which allows an all-powerful adversary to distinguish the permutation from random).

In order to match this bound we first note that the somewhat high distinguishing probability of S is due to its vulnerability to a birthday-attack on the length of a single block. An adversary that makes $\Omega(2^{n/2}/b)$ uniformly chosen queries to S will force a collision in the inputs to f_1 (or f_2) with a constant probability. Such a collision foils our analysis (and can indeed be used to distinguish S from uniform). The solution lies in the following observation: The problem of foiling birthday-attacks when constructing a pseudo-random permutation on *many* blocks can be reduced to the problem of foiling birthday-attacks when constructing a pseudo-random function (or permutation) on *two* blocks. We demonstrate this using the Aiello and Venkatesan [1] construction of pseudo-random *functions*.

Let $\tilde{f}_1$ and $\tilde{f}_2$ be two independent copies of the pseudo-random *functions* on $2n$ bits we get when using truly random functions on n bits in the construction of Aiello and Venkatesan. By [1] distinguishing each $\tilde{f}_i$ from a truly random function (with constant probability) requires $\Omega(2^n)$ queries. Let h_1 and $h_2 \in P_{2nb}$ be pair-wise independent permutations and let the permutation $\tilde{S} = S(h_1, \tilde{f}_1, \tilde{f}_2, h_2)$ be as in Theorem 5.6.1 (for the parameters $n' = 2n$ and $b' = b/2$). We now get that distinguishing $\tilde{S}$ from random (with constant probability) requires $O(2^n/b)$ queries which is optimal.

5.7 Constructions of k-Wise δ-Dependent Permutations

In this section, we summarize the connection between the various constructions of this chapter and the task of obtaining k-wise δ-dependent permutations. As mentioned in Section 5.4.1, Schnorr [134] suggested using the LR-Construction with truly random functions in order to get a pseudo-random generator that is secure as long as not too many bits of its output are accessible to the adversary. This idea is further treated by Maurer and Massey [94]. Maurer [92] suggested to replace the truly random functions with what he calls locally random (or almost random) functions. In the terminology of k-wise independence these ideas can be interpreted as a way of using the LR-Construction in order to obtain k-wise δ-dependent permutations from k-wise δ'-dependent functions (as long as k is not too large). Theorem 1 in [92] implies that

> *when k-wise δ'-dependent functions are used instead of pseudo-random functions in the LR-Construction the result is a k-wise δ-dependent permutations for $\delta = O(k^2/2^n + \delta')$.*

Similar observations apply to the different constructions of this chapter as discussed in this section.

Corollary 5.7.1 *(to Theorem 5.2.2) Let $h_1, h_2 \in P_{2n}$ be pair-wise independent permutations and let $f_1, f_2 \in F_n$ be k-wise δ'-dependent functions. Then $S = S(h_1, f_1, f_2, h_2)$ (as in Definition 5.2.1) is a k-wise δ-dependent permutation for*

$$\delta \stackrel{\text{def}}{=} \frac{k^2}{2^n} + \frac{k^2}{2^{2n}} + 2\delta'.$$

Proof. Let $S_1, S_2 \in P_{2n}$ have the following distributions:

- $S_1 = S(h_1, g_1, f_2, h_2)$, where $h_1, h_2 \in P_{2n}$ are pair-wise independent, $f_2 \in F_n$ a k-wise δ'-dependent function and $g_1 \in F_n$ a truly random function.
- $S_2 = S(h_1, g_1, g_2, h_2)$, where $h_1, h_2 \in P_{2n}$ are pair-wise independent and $g_1, g_2 \in F_n$ are truly random functions.

and let $R \in P_{2n}$ be a truly random permutation. It is enough to show that for every k strings of $2n$-bits, $x_1, x_2, \ldots, x_k$, we have:

1. $\| \langle S(x_1), S(x_2), \ldots, S(x_k)\rangle - \langle S_1(x_1), S_1(x_2), \ldots, S_1(x_k)\rangle \| \leq \delta'$
2. $\| \langle S_1(x_1), S_1(x_2), \ldots, S_1(x_k)\rangle - \langle S_2(x_1), S_2(x_2), \ldots, S_2(x_k)\rangle \| \leq \delta'$
3. $\| \langle S_2(x_1), S_2(x_2), \ldots, S_2(x_k)\rangle - \langle R(x_1), R(x_2), \ldots, R(x_k)\rangle \| \leq \frac{k^2}{2^n} + \frac{k^2}{2^{2n}}$

The reason (3) holds is that if we define an oracle machine M such that its i^{th} query is always $(+, x_i)$ and such that

$$\begin{aligned} & C_M(\{\langle x_1, y_1\rangle, \langle x_2, y_2\rangle, \ldots, \langle x_k, y_k\rangle\}) = 1 \\ \iff\ & \Pr[\langle S_2(x_1), \ldots, S_2(x_k)\rangle = \langle y_1, \ldots, y_k\rangle] < \Pr[\langle R(x_1), \ldots, R(x_k)\rangle = \langle y_1, \ldots, y_k\rangle], \end{aligned}$$

we get by the definition of variation distance and from Theorem 5.2.2 that

$$\begin{aligned} & \| \langle S_2(x_1), S_2(x_2), \ldots, S_2(x_k)\rangle - \langle R(x_1), R(x_2), \ldots, R(x_k)\rangle \| \\ = & \left|\Pr[M^{S_2, S_2^{-1}}(1^{2n}) = 1] - \Pr[M^{R, R^{-1}}(1^{2n}) = 1]\right| \\ \leq & \frac{k^2}{2^n} + \frac{k^2}{2^{2n}}. \end{aligned}$$

(1) and (2) hold by the definition of k-wise δ'-dependent functions. For example, if

$$\| \langle S(x_1), S(x_2), \ldots, S(x_k)\rangle - \langle S_1(x_1), S_1(x_2), \ldots, S_1(x_k)\rangle \| > \delta',$$

then we can fix $h_1, h_2 \in P_{2n}$ and $f_2 \in F_n$ in the definition of both S and S_1 such that the inequality still holds. This defines k strings of n-bits, $z_1, z_2, \ldots, z_k$, (not necessarily all different) and a function V for which:

$$\langle S(x_1), S(x_2), \ldots, S(x_k)\rangle = V(\langle f_1(z_1), f_1(z_2), \ldots, f_1(z_k)\rangle) \text{ and}$$

$$\langle S_1(x_1), S_1(x_2), \ldots, S_1(x_k)\rangle = V(\langle g_1(z_1), g_1(z_2), \ldots, g_1(z_k)\rangle).$$

We get a contradiction since for any function V:

$$\begin{aligned} & \|V(\langle f_1(z_1), f_1(z_2), \ldots, f_1(z_k)\rangle) - V(\langle g_1(z_1), g_1(z_2), \ldots, g_1(z_k)\rangle)\| \\ \leq & \|\langle f_1(z_1), f_1(z_2), \ldots, f_1(z_k)\rangle - \langle g_1(z_1), g_1(z_2), \ldots, g_1(z_k)\rangle\| \\ \leq & \delta'. \end{aligned}$$

□

In a similar way we get the following two Corollaries from the constructions of Sections 5.5 & 5.6:

Corollary 5.7.2 *(to Theorem 5.5.2) Let S be as in Definition 5.5.2, where h_1 and h_2 are pair-wise independent permutations and $f_1, f_2, \ldots, f_t$ are k-wise δ'-dependent functions. Then S is a k-wise δ-dependent permutation for*

$$\delta \stackrel{\text{def}}{=} \frac{t}{2} \cdot \frac{k^2}{2^{\ell - \lceil \ell/t \rceil}} + \frac{k^2}{2^\ell} + t \cdot \delta'.$$

Corollary 5.7.3 *(to Theorem 5.6.1) Let $h_1, h_2 \in P_{2nb}$ be pair-wise independent permutations, let $f_1, f_2 \in F_n$ be $b \cdot k$-wise δ'-dependent functions and let $p \in P_{2n}$ be a $b \cdot k$-wise δ'-dependent permutation. Define $S = S(h_1, f_1, f_2, h_2)$ and $\hat{S} = \hat{S}(h_1, p, h_2)$ (as in Definition 5.6.1). Then S is a k-wise δ-dependent permutation for*

$$\delta \stackrel{\text{def}}{=} \frac{k^2 \cdot b^2}{2^n} + \frac{k^2}{2^{2nb}} + 2\delta'$$

and $\hat{S}$ is a k-wise $\hat{\delta}$-dependent permutation for

$$\hat{\delta} \stackrel{\text{def}}{=} \frac{k^2 \cdot b^2}{2^{2n-1}} + \delta'.$$

By taking $t = \ell$ in Corollary 5.7.2 we get a simple construction of a k-wise δ-dependent permutation on ℓ bits for δ as close to $\frac{(\ell+1)\cdot k^2}{2^\ell}$ as we wish. This construction requires ℓ applications of k-wise δ'-dependent functions from $\ell - 1$ bits to a single bit. An interesting question is to find a simple construction of k-wise δ-dependent permutations for an *arbitrarily small* δ and an arbitrary k.

An "old" proposal by Moni Naor (see [129, page 17]) is to apply a card shuffling procedure that requires only few rounds and is oblivious in the sense that the location of a card after each round depends on a few random decisions. The specific card shuffling for which this idea is described in [129] was suggested by Aldous and Diaconis [3]. Unfortunately, to the best of our knowledge, this procedure was never proven to give (with few rounds) an almost uniform ordering of the cards. Nevertheless, we briefly describe it in order to demonstrate the concept of an oblivious card shuffling and the way that such a procedure can be used to construct a k-wise δ-dependent permutation. Finally we describe the main idea in the definition of another oblivious card shuffling for which we can prove that only few rounds are needed.

Each round (shuffle) in a card shuffling procedure is a permutation on the locations of the N cards of a deck (i.e., a permutation on the set $[N] \stackrel{\text{def}}{=} \{1, 2, \ldots, N\}$). In the case of the Aldous and Diaconis [3] card shuffling each such permutation is defined by a uniformly chosen $N/2$-bit string, $r = r_1 r_2 \ldots r_{N/2}$. Denote this permutation by Π_r then:

$$\forall 1 \leq i \leq N/2, \qquad \begin{cases} \Pi_r(i) = 2i - 1 \ \ \& \ \ \Pi_r(i + N/2) = 2i & \text{if } r_i = 1 \\ \Pi_r(i) = 2i \ \ \& \ \ \Pi_r(i + N/2) = 2i - 1 & \text{otherwise} \end{cases}$$

That is, the cards at locations i and $i+N/2$ move to locations $2i-1$ and $2i$ and their internal order is uniformly-chosen independently of all other choices. Note that $\forall x$, evaluating $\Pi_r(x)$ or $\Pi_r^{-1}(x)$ requires the knowledge of a *single* bit of r and therefore this card shuffling is indeed oblivious.

Consider s rounds of the card shuffling described above, $\Pi^s = \Pi_{r^1,\ldots,r^s} \stackrel{\text{def}}{=} \Pi_{r^s} \circ \Pi_{r^{s-1}} \circ \ldots \circ \Pi_{r^1}$, where $\{r^1, \ldots, r^s\}$ are uniformly-distributed and independent of each other. If Π^s is of statistical distance at most δ' from a uniform permutation then we can construct a k-wise δ-dependent permutation, $\tilde{\Pi}^s$, for $\delta = \delta' + \delta''$ as follows: simply take the permutation $\tilde{\Pi}^s$ to be s rounds $\Pi_{r^s} \circ \Pi_{r^{s-1}} \circ \ldots \circ \Pi_{r^1}$ where the $s \cdot N/2$ bits of $\{r^1, \ldots, r^s\}$ are *the outputs of a $(k \cdot s)$-wise δ''-dependent Binary-function, f.* Evaluating $\tilde{\Pi}^s$ (or its inverse) at a given point consists of s invocations of f. Therefore, an interesting problem is to show that Π^s is of exponentially-small statistical distance from a uniform permutation for a small value of s. In [3] it is conjectured that this can be shown for $s = O(\log^2 N)$. While this conjecture is, to the best of our knowledge, still open we can show a different card shuffling procedure for which it can be proven that $O((\log^2 N)$ rounds are sufficient. This card shuffling is defined in a recursive manner: Split the deck into two halves (locations $\{1, \ldots, N/2\}$ and locations $\{N/2, \ldots, N\}$), apply the card shuffling (recursively) on each half of the deck and merge the two (now shuffled) halves in an almost uniform way. A permutation, M, on $[N]$ is a merge if for every i and j such that $1 \leq i < j \leq N/2$ or $N/2 + 1 \leq i < j \leq N$ we have that $M(i) < M(j)$. An oblivious (in the same meaning as above) merging procedure can also be defined recursively but since the construction is rather cumbersome we omit its description. This direction may become attractive given an efficient and simple merging procedure.

A different direction to solving the problem of constructing k-wise δ-dependent permutations is to try and generalize the algebraic construction of pair-wise independent permutations. Leonard Schulman (private communication) suggested such a generalization that yields 3-wise independent permutations. His suggestion is to use sharply 3-transitive permutation groups. A permutation group over the set $[n] = \{1, 2, \ldots, n\}$ is a subgroup of the symmetric group S_n. A permutation group G over $[n]$ is k-transitive if for every two k-tuples $\{a_1, \ldots, a_k\}$ and $\{b_1, \ldots, b_k\}$ of distinct elements of $[n]$ there exist a permutation $\pi \in G$ such that $\forall 1 \leq i \leq k,\ \pi(a_i) = b_i$. A permutation group G over $[n]$ is sharply k-transitive if for every two such tuples there exists exactly one permutation $\pi \in G$ such that $\forall 1 \leq i \leq k,\ \pi(a_i) = b_i$. A sharply k-transitive permutation group is in particular k-wise independent and indeed the algebraic construction of pair-wise independent permutations use a sharply 2-transitive permutation group (containing all the linear permutations). Schulman suggested to use the fact that there are known constructions of sharply 3-transitive permutation groups. However, this approach cannot be generalized to larger values of k: from the classification of finite simple groups it follows that for $k \geq 6$ there are no k-transitive groups over $[n]$ other than the symmetric group S_n and the alternating group A_n and there are only few such groups for $k = 4$ and $k = 5$ (see [33, 127]). One should be careful not to interpret this as implying that for $k \geq 4$ there are no efficient algebraic constructions of k-wise independent permutations. It is however justified to deduce that for $k \geq 4$ any small family of k-wise independent permutations is not a permutation group (i.e. is not closed under composition and inverse).

5.8 Conclusion and Further Work

The constructions described in Sections 5.2 & 5.6 are optimal in their cryptographic work in the sense that the total number of bits on which the cryptographic functions are applied on is exactly the number of bits in the input. Therefore, it seems that in order to achieve the goal of constructing efficient block-ciphers it is sufficient to concentrate on the construction of efficient pseudo-random functions. The depth of the constructions, on the other hand, is twice the depth of the cryptographic functions. It is an interesting question whether there can be a construction of similar depth. The goal of reducing the depth is even more significant in the case of the $t + 2$-round construction in Section 5.5. A different question is finding a simple construction of k-wise δ-dependent permutations for an *arbitrarily small* δ and an arbitrary k. This question is discussed in Section 5.7.

Bibliography

[1] W. Aiello and R. Venkatesan, Foiling Birthday Attacks in Length-Doubling Transformations, *Advances in Cryptology - EUROCRYPT '96*, Lecture Notes in Computer Science, vol. 1070, Springer-Verlag, 1996, pp. 307-320.

[2] M. Ajtai and A. Wigderson, Deterministic simulations of probabilistic constant depth circuits, *Proc. 26th Symp. on Foundations of Computer Science* (1985) 11-19.

[3] D. Aldous and P. Diaconis, Strong uniform times and finite random walks, *Advances in Applied Mathematics*, vol. 8, 1987, pp. 69-97.

[4] W. B. Alexi, B. Chor, O. Goldreich and C. P. Schnorr, RSA and Rabin functions: certain parts are as hard as the whole, *SIAM J. Comput.*, vol. 17(2), 1988, pp. 194-209.

[5] N. Alon, L. Babai and A. Itai, A fast and simple randomized parallel algorithm for the maximal independent set problem, *J. Algorithms*, vol. 7(4), 1986, pp. 567-583.

[6] N. Alon, O. Goldreich, J. Hastad and R. Peralta, Simple constructions for almost k-wise independent random variables, *Random Structures and Algorithms*, vol. 3, 1992, pp. 289-304.

[7] R. Anderson and E. Biham, Two practical and provably secure block ciphers: BEAR and LION, *Proc. Fast Software Encryption*, Lecture Notes in Computer Science, vol. 1039, Springer-Verlag, 1996, pp. 113-120.

[8] D. Angluin and M. Kharitonov, When won't membership queries help?, *J. Comput. System Sci.*, vol. 50, 1995, pp. 336-355.

[9] R. Armoni, On the de-randomization of space-bounded computations *RANDOM: International Workshop on Randomization and Approximation Techniques in Computer Science*, LNCS, 1998.

[10] L. Babai, N. Nisan and M. Szegedy, Multiparty protocols, pseudorandom generators for logspace, and time-space tradeoffs, *J. Comput. System Sci.*, vol. 45(2), 1992, pp. 204-232.

[11] E. Bach, How to generate factored random numbers, *SIAM J. Comput.*, vol. 17(2), 1988, pp. 179-193.

[12] P. W. Beame, S. A. Cook and H. J. Hoover, Log depth circuits for division and related problems, *SIAM J. Comput.*, vol. 15, 1986, pp. 994-1003.

[13] M. Bellare, A. Desai, D. Pointcheval and P. Rogaway, Relations among notions of security for public-key encryption schemes, *Advances in Cryptology- CRYPTO '98*, Lecture Notes in Computer Science, vol. 1462, , 1998.

[14] M. Bellare, R. Canetti and H. Krawczyk, Pseudorandom functions revisited: the cascade construction, *Proc. 37th IEEE Symp. on Foundations of Computer Science*, 1996, pp. 514-523.

[15] M. Bellare, O. Goldreich and S. Goldwasser, Incremental cryptography: the case of hashing and signing, *Advances in Cryptology - CRYPTO '94*, Lecture Notes in Computer Science, vol. 839, Springer-Verlag, 1994, pp. 216-233.

[16] M. Bellare, O. Goldreich and S. Goldwasser, Incremental Cryptography with Application to Virus Protection, *Proc. 27th Ann. ACM Symp. on Theory of Computing*, 1995, pp. 45-56.

[17] M. Bellare and S. Goldwasser, New paradigms for digital signatures and message authentication based on non-interactive zero knowledge proofs, *Advances in Cryptology - CRYPTO '89*, Lecture Notes in Computer Science, vol. 435, Springer-Verlag, 1990, pp. 194-211.

[18] M. Bellare, J. Kilian and P. Rogaway, The security of cipher block chaining, *Advances in Cryptology - CRYPTO '94*, Lecture Notes in Computer Science, vol. 839, Springer-Verlag, 1994, pp. 341-358.

[19] M. Bellare and S. Micali, Non-interactive oblivious transfer and applications, *Proc. Advances in Cryptology - CRYPTO '89*, LNCS, Springer, 1990, pp. 547-557.

[20] M. Bellare and P. Rogaway, Block cipher mode of operation for secure, length-preserving encryption, manuscript in preparation.

[21] Ben-Or M., Goldwasser S. and Wigderson A., Completeness theorems for non-cryptographic fault tolerant distributed computation, *Proc. 20th Ann. ACM Symp. on Theory of Computing*, 1988, pp. 1-9.

[22] E. Biham, D. Boneh and O. Reingold, Breaking generalized Diffie-Hellman modulo a composite is no easier than factoring, *Information Processing Letters*, vol. 70, 1999, pp. 83-87.

[23] A. Blum, M. Furst, M. Kearns and R. J. Lipton, Cryptographic primitives based on hard learning problems, *Advances in Cryptology - CRYPTO '93*, Lecture Notes in Computer Science, vol. 773, Springer-Verlag, 1994, pp. 278-291.

[24] L. Blum, M. Blum and M. Shub, A simple secure unpredictable pseudo-random number generator, *SIAM J. Comput.*, vol. 15, 1986, pp. 364-383.

[25] M. Blum, W. Evans, P. Gemmell, S. Kannan, M. Naor, Checking the correctness of memories, *Algorithmica*, 1994, pp. 225-244. Preliminary version: *Proc. 31st Symp. on Foundations of Computer Science*, 1990.

[26] M. Blum and S. Goldwasser, An efficient probabilistic public-key encryption scheme which hides all partial information, *Advances in Cryptology - CRYPTO '84*, LNCS, vol. 196, Springer, 1984, pp. 289-302.

[27] M. Blum and S. Micali, How to generate cryptographically strong sequence of pseudo-random bits, *SIAM J. Comput.*, vol. 13, 1984, pp. 850-864.

[28] D. Boneh and R. Lipton, Algorithms for Black-Box fields and their application to cryptography, *Advances in Cryptology - CRYPTO '96*, LNCS, vol. 1109, Springer, 1996, pp. 283-297.

[29] D. Boneh and R. Venkatesan, Hardness of computing most significant bits in secret keys in Diffie-Hellman and related schemes, *Advances in Cryptology - CRYPTO '96*, LNCS, vol. 1109, Springer, 1996, pp. 129-142.

[30] S. Brands, An efficient off-line electronic cash system based on the representation problem, CWI Technical Report, CS-R9323, 1993.

[31] G. Brassard, **Modern cryptology**, Lecture Notes in Computer Science, vol. 325, Springer-Verlag, 1988.

[32] E. F. Brickell, D. M. Gordon, K. S. McCurley and D. B. Wilson, Fast exponentiation with precomputation, *Proc. Advances in Cryptology - EUROCRYPT '92*, LNCS, Springer, 1992, pp. 200-207.

[33] P. J. Cameron, Finite permutation groups and finite simple groups, *Bull. London Math. Soc.*, vol. 13, 1981, pp. 1-22.

[34] R. Canetti, Towards realizing random oracles: hash functions that hide all partial information, *Proc. Advances in Cryptology - CRYPTO '97*, LNCS, Springer, 1997 pp. 455-469.

[35] R. Canetti, J. Friedlander and I. Shparlinski, On certain exponential sums and the distribution of Diffie-Hellman triples, *Research report, IBM T. J. Watson Research Center, Number RC 20915 (92645)*, July 1997.

[36] R. Canetti, O. Goldreich and S. Halevi, The random oracle model, revisited, to appear in: *Proc. 30th Ann. ACM Symp. on Theory of Computing*, 1998.

[37] R. Canetti, D. Micciancio and O. Reingold, Perfectly one-way probabilistic hash functions, *Proc. 30th Ann. ACM Symp. on Theory of Computing*, 1998, pp. 131-140.

[38] L. Carter and M. Wegman, Universal hash functions, *J. of Computer and System Sciences*, vol. 18, 1979, pp. 143-154.

[39] D. Chaum and H. van Antwerpen, Undeniable signatures, *Proc. Advances in Cryptology - CRYPTO '89*, LNCS, Springer, 1990, pp. 212-216.

[40] B. Chor, A. Fiat and M. Naor, Tracing traitors, *Advances in Cryptology - CRYPTO '94*, Springer-Verlag, 1994, pp. 257-270.

[41] B. Chor and O. Goldreich, Unbiased bits from sources of weak randomness and probabilistic communication complexity, *SIAM J. Comput.*, vol 17, 1988, pp. 230-261. Preliminary version in: *Proc. 26th IEEE Symp. on Foundations of Computer Science*, 1985, pp. 429-442.

[42] B. Chor and O. Goldreich, On the power of two-point based sampling, *J. Complexity*, vol. 5, 1989, pp. 96-106.

[43] R. Cleve, *Methodologies for designing block ciphers and cryptographic protocols*, Part I, Ph.D. Thesis, University of Toronto, 1989.

[44] R. Cleve, Complexity theoretic issues concerning block ciphers related to D.E.S., *Advances in Cryptology - CRYPTO '90*, Lecture Notes in Computer Science, vol. 537, Springer-Verlag, 1991, pp. 530-544.

[45] R. Cramer and V. Shoup, A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack, *Proc. Advances in Cryptology - CRYPTO '98*, LNCS, vol. 1462, Springer, 1998, pp. 13-25.

[46] D. Chaum, C. Crepeau and I. Damgard, Multiparty unconditionally secure protocols, *Proc. 20th Ann. ACM Symp. on Theory of Computing*, 1988, pp. 11-19.

[47] A. De Santis, Y Desmedt, Y. Frankel and M. Yung, How to share a function securely, *Proc. 26th ACM Symp. on Theory of Computing*, 1994, pp. 522-533.

[48] Y. Desmedt, Society and group oriented cryptography: A new concept *Proc. Advances in Cryptology - CRYPTO '87*, LNCS, Springer, 1987.

[49] Y. Desmedt and Y. Frankel, Threshold cryptosystems, *Advances in Cryptology – CRYPTO '89*, Springer-Verlag, LNCS, vol. 435, 1989, pp. 307–315.

[50] W. Diffie and M. Hellman, New directions in cryptography, *IEEE Trans. Inform. Theory*, vol. 22(6), 1976, pp. 644-654.

[51] D. Dolev, C. Dwork and M. Naor, Non-malleable cryptography, *Proc. 23rd Ann. ACM Symp. on Theory of Computing*, 1991, pp. 542-552. Full version available at: `http://www.wisdom.weizmann.ac.il/~naor`.

[52] T. ElGamal, A public-key cryptosystem and a signature scheme based on discrete logarithms, *Advances in Cryptology - CRYPTO '84*, LNCS, vol. 196, Springer, 1985, pp. 10-18.

[53] S. Even and Y. Mansour, A construction of a cipher from a single pseudorandom permutation, *J. of Cryptology*, vol. 10, 1997, pp. 151-161. Preliminary version in *Advances in Cryptology - ASIACRYPT '91*, Lecture Notes in Computer Science, Springer-Verlag, 1991.

[54] R. Fischlin and C. P. Schnorr, Stronger security proofs for RSA and Rabin bits, *Advances in Cryptology - EUROCRYPT '97*, Lecture Notes in Computer Science, vol. 1233, Springer-Verlag, 1997, pp. 267-279.

[55] M. Franklin and S. Haber, Joint encryption and message-efficient secure computation, *J. of Cryptology*, vol 9(4), 1996, pp. 217-232.

[56] Y. Gertner and T. Malkin, A PSRG based on the decision Diffie-Hellman assumption, preprint, 1997.

[57] O. Goldreich, Two remarks concerning the Goldwasser-Micali-Rivest signature scheme, *Advances in Cryptology - CRYPTO '86*, Lecture Notes in Computer Science, vol. 263, Springer-Verlag, 1987, pp. 104-110.

[58] O. Goldreich, Towards a theory of software protection, *Proc. 19th Ann. ACM Symp. on Theory of Computing*, 1987, pp. 182-194.

[59] O. Goldreich, A note on computational indistinguishability, *Information Processing Letters*, vol. 34, 1990, pp. 277-281.

[60] O. Goldreich, **Foundations of cryptography (fragments of a book)**, 1995. Electronic publication: http://www.eccc.uni-trier.de/eccc/info/ECCC-Books/eccc-books.html (Electronic Colloquium on Computational Complexity).

[61] O. Goldreich, **Modern cryptography, probabilistic proofs and pseudorandomness**, *Algorithms and Combinatorics*, vol. 17, Springer-Verlag, 1998.

[62] O. Goldreich, S. Goldwasser and S. Micali, How to construct random functions, *J. of the ACM.*, vol. 33, 1986, pp. 792-807.

[63] O. Goldreich, S. Goldwasser and S. Micali, On the cryptographic applications of random functions, *Advances in Cryptology - CRYPTO '84*, Lecture Notes in Computer Science, vol. 196, Springer-Verlag, 1985, pp. 276-288.

[64] O. Goldreich and H. Krawczyk, On sparse pseudorandom ensembles, *Random Structures and Algorithms*, vol. 3(2), 1992, pp. 163-174.

[65] O. Goldreich and L. Levin, A hard-core predicate for all one-way functions, *Proc. 21st Ann. ACM Symp. on Theory of Computing*, 1989, pp. 25-32.

[66] O. Goldreich O., S. Micali and A. Wigderson, How to play any mental game, *Proc. 19th Ann. ACM Symp. on Theory of Computing*, 1987, pp. 218-229.

[67] O. Goldreich and A. Wigderson, Tiny families of functions with random properties: a quality-size trade-off for hashing, *Proc. 26th ACM Symp. on Theory of Computing*, 1994, pp. 574-583.

[68] S. Goldwasser and S. Micali, Probabilistic encryption, *J. Comput. System Sci.*, vol. 28(2), 1984, pp. 270-299.

[69] S. Halevi and H. Krawczyk, MMH: message authentication in software in the Gbit/second rates, *Proc. Fast Software Encryption*, Lecture Notes in Computer Science, vol. 1267, Springer-Verlag, 1997, pp. 172-189.

[70] J. Hastad, A. W. Schrift and A. Shamir, The discrete logarithm modulo a composite hides $O(n)$ bits, *J. of Computer and System Sciences*, vol. 47, 1993, pp. 376-404.

[71] J. Hastad, R. Impagliazzo, L. A. Levin and M. Luby, Construction of a pseudo-random generator from any one-way function, *SIAM Journal on Computing*, vol 28(4), 1999, pp. 1364-1396. Preliminary versions by Impagliazzo et. al. in *21st STOC*, 1989 and Hastad in *22nd STOC*, 1990.

[72] A. Herzberg and M. Luby, Public randomness in cryptography, *Advances in Cryptology - CRYPTO '92*, Lecture Notes in Computer Science, vol. 740, Springer-Verlag, 1992, pp. 421-432.

[73] R. Impagliazzo and L. Levin, No better ways to generate hard NP instances than picking uniformly at random, *Proc. 31st IEEE Symp. on Foundations of Computer Science*, 1990 pp. 812-821.

[74] R. Impagliazzo and M. Naor, Efficient Cryptographic schemes provably secure as subset sum, *J. of Cryptology*, vol 9, 1996, pp. 199-216.

[75] R. Impagliazzo, N. Nisan and A. Wigderson, Pseudorandomness for network algorithms, *Proc. 26th ACM Symp. on Theory of Computing* (1994)

[76] R. Impagliazzo and A. Wigderson, P = BPP if E requires exponential circuits: derandomizing the XOR lemma, *Proc. 29th Ann. ACM Symp. on Theory of Computing*, 1997. pp. 220-229.

[77] R. Impagliazzo and A. Wigderson, Randomness vs. time: de-randomization under a uniform assumption. To appear in: *Proc. 39th IEEE Symp. on Foundations of Computer Science*, 1998.

[78] R. Impagliazzo and D. Zuckerman, Recycling random bits, *Proc. 30th IEEE Symposium on Foundations of Computer Science*, 1989, pp. 248-253.

[79] R. M. Karp and V. Ramachandran, Parallel algorithms for shared-memory machines, in: J. van Leeuwen, ed., **Handbook of Theoretical Computer Science, vol. A** MIT Press, 1990, pp. 869-941.

[80] M. Kearns and L. Valiant, Cryptographic limitations on learning Boolean formulae and finite automata, *J. of the ACM.*, vol. 41(1), 1994, pp. 67-95.

[81] M. Kharitonov, Cryptographic hardness of distribution-specific learning, *Proc. 25th ACM Symp. on Theory of Computing*, 1993, pp. 372-381.

[82] J. Kilian and P. Rogaway, How to protect DES against exhaustive key search, *Advances in Cryptology - CRYPTO '96*, Lecture Notes in Computer Science, vol. 1109, Springer-Verlag, 1996, pp. 252-267.

[83] T. Koren, *On the construction of pseudorandom block ciphers*, M.Sc. Thesis (in Hebrew), CS Dept., Technion, Israel, May 1989.

[84] H. Krawczyk, LFSR-based hashing and authentication, *Advances in Cryptology - CRYPTO '94*, Lecture Notes in Computer Science, vol. 839, Springer-Verlag, 1994, pp. 129-139.

[85] M. Langberg, An implementation of efficient pseudo-random functions, 1998. At: `http://www.wisdom.weizmann.ac.il/~naor/p_r_func/abs/abs.html`

[86] L. A. Levin, One-way function and pseudorandom generators, *Proc. 17th Ann. ACM Symp. on Theory of Computing*, 1985, pp. 363-365.

[87] N. Linial, Y. Mansour and N. Nisan, Constant depth circuits, Fourier transform, and learnability, *J. of the ACM.*, vol 40(3), 1993, pp. 607-620.

[88] M. Luby, A simple parallel algorithm for the maximal independent set problem, *SIAM J. Comput.*, vol 15(4), 1986, pp. 1036-1053.

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

[90] M. Luby and C. Rackoff, How to construct pseudorandom permutations and pseudorandom functions, *SIAM J. Comput.*, vol. 17, 1988, pp. 373-386.

[91] S. Lucks, Faster Luby-Rackoff ciphers, *Proc. Fast Software Encryption*, Lecture Notes in Computer Science, vol. 1039, Springer-Verlag, 1996, pp. 189-203.

[92] U. M. Maurer, A simplified and generalized treatment of Luby-Rackoff pseudorandom permutation generators, *Advances in Cryptology - EUROCRYPT '92*, Lecture Notes in Computer Science, vol. 658, Springer-Verlag, 1992, pp. 239-255.

[93] U. Maurer and S. Wolf, Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms, *SIAM Journal on Computing*, vol 28(5), 1999, pp. 1689-1721. Preliminary version by U. Maurer in *Advances in Cryptology - CRYPTO '94*, LNCS, vol. 740, Springer, 1994, pp. 271-281.

[94] U. M. Maurer and J. L. Massey, Local randomness in pseudorandom sequences, *J. of Cryptology*, vol. 4(2), Springer-Verlag, 1991, pp. 135-149.

[95] K. McCurley, A key distribution system equivalent to factoring, *J. of Cryptology*, vol 1, 1988, pp. 95-105.

[96] K. McCurley, The discrete logarithm problem,*Cryptography and Computational Number Theory, Proc. Symp. Appl. Math.*, AMS Lecture Notes, vol. 42, 1990, pp. 49-74.

[97] J. Naor and M. Naor, Small-bias probability spaces: efficient constructions and applications, *SIAM J. Comput.*, vol. 22(4), 1993, pp. 838-856.

[98] M. Naor, Bit commitment using pseudorandomness, *Journal of Cryptology* 4 (1991)

[99] M. Naor and B. Pinkas, Secure and efficient metering, *Advances in Cryptology - EUROCRYPT '98*, Lecture Notes in Computer Science, Springer-Verlag, 1998.

[100] M. Naor, B. Pinkas and O. Reingold, *Distributed Pseudo-Random Functions and KDCs*, Advances in Cryptology - Eurocrypt '99 Proceedings, LNCS vol. 1592, Springer-Verlag, pp. 327-346, 1999.

[101] M. Naor and O. Reingold, Synthesizers and their application to the parallel construction of pseudo-random functions, *Proc. 36th IEEE Symp. on Foundations of Computer Science*, 1995, pp. 170-181.

[102] M. Naor and O. Reingold, On the construction of pseudo-random permutations: Luby-Rackoff revisited, To appear in: *J. of Cryptology*. Preliminary version in: *Proc. 29th Ann. ACM Symp. on Theory of Computing*, 1997. pp. 189-199.

[103] M. Naor and O. Reingold, Number-Theoretic constructions of efficient pseudo-random functions, *Proc. 38th IEEE Symp. on Foundations of Computer Science*, 1997.

[104] M. Naor and O. Reingold, From unpredictability to indistinguishability: A simple construction of pseudo-random functions from MACs, *Advances in Cryptology - CRYPTO '98*, Lecture Notes in Computer Science, vol. 1462, Springer-Verlag, 1998, pp. 267-282.

[105] M. Naor, O. Reingold and A. Rosen, Pseudo-random functions and Factoring, manuscript, 1999.

[106] National Bureau of Standards, Data encryption standard, *Federal Information Processing Standard*, U.S. Department of Commerce, FIPS PUB 46, Washington, DC, 1977.

[107] N. Nisan, Pseudorandom generators for space-bounded computation, *Combinatorica*, vol. 12(4), 1992, pp. 449-461.

[108] N. Nisan, *RL* $\subseteq$ *SC*, *Proc. 24th Ann. ACM Symp. on Theory of Computing*, 1992, pp. 619-623.

[109] N. Nisan and A. Wigderson, Hardness vs. randomness, *J. Comput. System Sci.*, vol. 49(2), 1994, pp. 149-167. Preliminary version: *Proc. 29th IEEE Symp. on Foundations of Computer Science*, 1988, pp. 2-12.

[110] N. Nisan and D. Zuckerman, Randomness is linear in space, *J. Comput. System Sci.*, vol. 52, 1996, pp. 43–52. Preliminary version: More deterministic simulations in logspace *Proc. 25th Ann. ACM Symp. on Theory of Computing*, 1993, pp. 235-244.

[111] A. M. Odlyzko, Discrete logarithms and smooth polynomials, *Contemporary Mathematics, AMS* 1993.

[112] Y. Ohnishi, *A study on data security*, Master Thesis (in Japanese), Tohuku University, Japan, 1988.

[113] R. Ostrovsky, An efficient software protection scheme, *Proc. 22nd Ann. ACM Symp. on Theory of Computing*, 1990, pp. 514-523.

[114] J. Patarin, Pseudorandom permutations based on the DES scheme, *Proc. of EUROCODE '90*, Lecture Notes in Computer Science, vol. 514, Springer-Verlag, 1991, pp. 193-204.

[115] J. Patarin, New results on pseudorandom permutation generators based on the DES scheme, *Advances in Cryptology - CRYPTO '91*, Lecture Notes in Computer Science, vol. 576, Springer-Verlag, 1991. pp. 301-312

[116] J. Patarin, How to construct pseudorandom and super pseudorandom permutations from one single pseudorandom function, *Advances in Cryptology - EUROCRYPT '92*, Lecture Notes in Computer Science, vol. 658, Springer-Verlag, 1992, pp. 256-266.

[117] J. Patarin, Improved security bounds for pseudorandom permutations, *4th ACM Conference on Computer and Communications Security*, 1997, 142-150.

[118] J. Pieprzyk, How to construct pseudorandom permutations from single pseudorandom functions, *Advances in Cryptology - EUROCRYPT '90*, Lecture Notes in Computer Science, vol. 473, Springer-Verlag, 1991, pp. 140-150.

[119] B. Pinkas, private communication.

[120] M. O. Rabin, Digitalized signatures and public-key functions as intractable as factorization, Technical Report, TR-212, MIT Laboratory for Computer Science, 1979.

[121] M. Ram Murty, Artin's conjecture for primitive roots, *The Mathematical Intelligencer*, vol. 10(4), Springer-Verlag, 1988 pp. 59-67.

[122] A. Razborov and S. Rudich, Natural proofs, *J. of Computer and System Sciences*, vol. 55(1), 1997, pp. 24-35. Preliminary version: *Proc. 26th Ann. ACM Symp. on Theory of Computing*, 1994, pp. 204-213.

[123] J. Reif, On threshold circuits and polynomial computation, *Proc. of the 2nd Conference on Structure in Complexity Theory* 1987, pp. 118-123.

[124] J. Reif and S. Tate, On threshold circuits and polynomial computation, *SIAM J. Comput.*, vol. 5, 1992, pp. 896-908.

[125] J. H. Reif and J. D. Tygar, Efficient parallel pseudorandom number generation, *SIAM J. Comput.*, vol. 17(2), 1988, pp. 404-411.

[126] R. L. Rivest, A. Shamir, and L. M. Adleman, A method for obtaining digital signature and public key cryptosystems, *Comma. ACM*, vol. 21, 1978, pp. 120-126.

[127] D. J. S. Robinson, **A course in the theory of groups – 2nd ed.**, New York : Springer-Verlag, 1996.

[128] P. Rogaway, Bucket hashing and its application to fast message authentication, *Advances in Cryptology - CRYPTO '95*, Lecture Notes in Computer Science, vol. 963, Springer-Verlag, 1995, pp. 74-85.

[129] S. Rudich, *Limits on the provable consequences of one-way functions*, PhD Thesis, U. C. Berkeley.

[130] R. A. Rueppel, On the security of Schnorr's pseudo random generator, *Advances in Cryptology - EUROCRYPT '89*, Lecture Notes in Computer Science, vol. 434, Springer-Verlag, 1989, pp. 423-428.

[131] B. Sadeghiyan and J. Pieprzyk, On necessary and sufficient conditions for the construction of super pseudorandom permutations, *Abstracts of ASIACRYPT '91*, Lecture Notes in Computer Science, vol. 739, Springer-Verlag, 1991, pp. 194-209.

[132] B. Sadeghiyan and J. Pieprzyk, A construction for super pseudorandom permutations from a single pseudorandom function, *Advances in Cryptology - EUROCRYPT '92*, Lecture Notes in Computer Science, vol. 658, Springer-Verlag, 1992, pp. 267-284

[133] B. Schneier and J. Kelsey, Unbalanced Feistel networks and block cipher design, *Proc. Fast Software Encryption*, Lecture Notes in Computer Science, vol. 1039, Springer-Verlag, 1996, pp. 121-144.

[134] C. P. Schnorr, On the construction of random number generators and random function generators, *Advances in Cryptology - EUROCRYPT '88*, Lecture Notes in Computer Science, vol. 330, Springer-Verlag, 1988, pp. 225-232.

[135] C. P. Schnorr, Efficient identification and signatures for smart cards, *Proc. Advances in Cryptology - CRYPTO '89*, LNCS, Springer, 1990, pp. 239-252.

[136] A. Shamir, How to share a secret, *Comm. ACM* vol. 22(11), 612-613.

[137] A. Shamir, On the generation of cryptographically strong pseudo-random number sequences, *ACM Trans. Comput. Sys.*, 1983, pp. 38-44.

[138] C. E. Shannon, Communication theory of secrecy systems, *Bell System Technical Journal*, vol. 28(4), 1949.

[139] Z. Shmuely, Composite Diffie-Hellman public-key generating systems are hard to break, Technical Report No. 356, Computer Science Department, Technion, Israel, 1985.

[140] V. Shoup, Lower bounds for discrete logarithms and related problems, *Proc. Advances in Cryptology - EUROCRYPT '97*, Lecture Notes in Computer Science, Springer-Verlag, 1997, pp. 256-266.

[141] K.-Y. Siu, J. Bruck, T. Kailath and T. Hofmeister, Depth efficient neural network for division and related problems, *IEEE Trans. Inform. Theory*, vol. 39, 1993, pp. 946-956.

[142] K.-Y. Siu and V. P. Roychowdhury, On optimal depth threshold circuits for multiplication and related problems, *SIAM J. Disc. Math.*, vol. 7(2), 1994, pp. 284-292.

[143] M. Stadler, Publicly verifiable secret sharing, *Proc. Advances in Cryptology - EUROCRYPT '96*, LNCS, vol. 1070, Springer, 1996, pp. 190-199.

[144] M. Steiner, G. Tsudik and M. Waidner, Diffie-Hellman key distribution extended to group communication, *Proceedings 3rd ACM Conference on Computer and Communications Security*, 1996, pp. 31-37.

[145] D. Stinson, Universal hashing and authentication codes, *Designs, Codes and Cryptography*, vol. 4 (1994), pp. 369-380.

[146] L. G. Valiant, A theory of the learnable, *Comm. ACM*, vol. 27, 1984, pp. 1134-1142.

[147] U. V. Vazirani, *Randomness, adversaries and computation*, PhD Thesis, U. C. Berkeley, 1986.

[148] U. V. Vazirani, Strong communication complexity or generating quasi-random sequences from two communicating semi-random sources, *Combinatorica*, vol. 7, 1987. Preliminary version in: *Proc. 17th Ann. ACM Symp. on Theory of Computing*, 1985, pp. 366-378.

[149] M. Wegman and L. Carter, New hash functions and their use in authentication and set equality, *J. of Computer and System Sciences,* vol. 22, 1981, pp. 265-279.

[150] A. C. Yao, Theory and applications of trapdoor functions, *Proc. 23rd IEEE Symp. on Foundations of Computer Science*, 1982, pp. 80-91.

[151] Y. Zheng, T. Matsumoto and H. Imai, Impossibility and optimality results on constructing pseudorandom permutations, *Advances in Cryptology - EUROCRYPT '89*, Lecture Notes in Computer Science, vol. 434, Springer-Verlag, 1990, pp. 412-422.