A Study of Luby-Rackoff Ciphers

by

Zulfikar Amin Ramzan

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 2001
[February 2001]

Author ....
Department of Electrical Engineering and Computer Science
January 10, 2001

Certified by ....
Ronald L. Rivest
Andrew and Erna Viterbi Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by ....
Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

A Study of Luby-Rackoff Ciphers

by

Zulfikar Amin Ramzan

Submitted to the Department of Electrical Engineering and Computer Science
on January 10, 2001, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

In their ground-breaking paper, Luby and Rackoff [88] formally model the notion of a secure block cipher, and realize this notion under standard cryptographic assumptions. Ciphers based on their original techniques are termed Luby-Rackoff ciphers. This dissertation engages in a deeper study of such ciphers in order to better understand their security properties and make them more practical.

First, we present a new, more practical, Luby-Rackoff cipher. The construction is efficient in terms of computation and key length. We also consider an alternate security analysis of this construction where we make a weaker but more practical underlying assumption, and arrive at a weaker security claim.

Second, we consider modifying the algebraic structure over which some of the cipher operations work. This paradigm shift offers new lines of research, and leads to several interesting results. In particular, we construct a provably-secure cipher, operating over various finite groups, that has better time/space complexity, and uses less key material than previously considered Luby-Rackoff ciphers. Surprisingly, the cipher is insecure when these operations are performed over additive groups attached to finite fields of characteristic two. We also discuss the security of other Luby-Rackoff ciphers in this more general setting.

Third, we propose a more refined notion of security for symmetric-key cryptographic primitives. The model uses the fact that many such primitives involve iterating simpler constructs for some number of rounds, and may be used to gain insight into the security of these constructions. We completely characterize the security of various Luby-Rackoff ciphers under this new model. We surprisingly show that even if an adversary is allowed black-box access to some internal components of the cipher, it still remains secure.

Fourth, we show how to efficiently construct Δ-universal hash functions, which are used in many of the ciphers we consider. We present the square hash function which involves two novel ideas in the construction of such functions. Square hash performs quite well on modern microprocessors. We substantiate our claims via a hand-optimized assembly language implementation. Beyond their use in block cipher design, Δ-universal hash functions are useful for message authentication.

Thesis Supervisor: Ronald L. Rivest
Title: Andrew and Erna Viterbi Professor of Electrical Engineering and Computer Science

Acknowledgments

It is with a great sense of anticipation and trepidation that I begin to write, what I consider to be, one of the most important pieces of my dissertation. There are great number of people who helped me over the years, and I know that it is impossible to thank all of them. But I will try my best to thank as many as I can, and sincerely apologize to those I missed.

First and foremost I would like to thank my advisor, Professor Ronald L. Rivest. Ron has been an invaluable source of help and support over the years. I have greatly benefited from his vast knowledge and perspicacity. Ron was always quick to suggest interesting and relevant research directions and references, and encouraged me to think independently about interesting areas to work on. He has helped me not only with understanding technical matters, but with both the proper written and oral communication of these ideas. For all of his help, I will always be grateful.

Next I would like thank the other faculty with the Cryptography and Information security group at MIT: Professors Shafi Goldwasser, Silvio Micali, and Madhu Sudan. I especially thank Silvio and Madhu for agreeing to serve as thesis readers on my committee. I had the pleasure of taking the introductory graduate course in cryptography taught by Silvio. His incremental approach to helping us carefully understand cryptographic primitives, and why concepts are defined as they are, has been invaluable. Shafi Goldwasser taught an advanced graduate cryptography class which I also had the pleasure of taking. Her ability to capture the essence of intricate ideas, and pick out the gaps in understanding that need to be filled in, is truly phenomenal. In addition, I took a class on Algebra and Computation from Madhu. His ability to truly engage the class, and create an interactive environment, are remarkable.

The research I present in this thesis has been the result of collaborative work with a number of people. In particular, I would like to thank Mark Etzel, Sarvar Patel, Leonid Reyzin, and Ganesh Sundaram for our fruitful collaborations. Sarvar and Ganesh have worked with me extensively from the time I was in the nascent stages of my career as a cryptographer. We learned many of these concepts together. Chapters 3 and 4 were the results of collaborations with Sarvar and Ganesh. I thank Mark Etzel for a number of fruitful discussions on issues related to assembly language implementation, and chapter 6 is the result of collaboration with him and Sarvar. Leo and I started together as graduate students at MIT. We first met while eating "food truck" lunches in the third floor lounge during our first week as graduate students. Interestingly enough, during our third year as graduate students, a brief lunch discussion at nearly the same spot eventually led to the results of chapter 5. Moreover, working with Leo impacted my understanding of the results in many other parts of this thesis.

Beyond the results presented in this dissertation, I had the opportunity to engage in fruitful research collaborations with Anna Lysyanskaya and Matthias Ruhl, both of whom are graduate students at MIT.

My research was financially supported by a National Science Foundation fellowship, by Darpa grant DABT-63-96-C-0018, and by a grant from Merrill Lynch.

There are also a number of people I would like to thank for fun and interesting research discussions that enhanced my understanding of cryptography. Some of these discussions were based on my papers, talks, and thesis drafts, whereas others were discussions of various random research problems. In particular, I would like to thank John Black, Daniel Bleichenbacher, Anand Desai, Rosario Gennaro, Shai Halevi, Markus Jakobsson, Ari Juels, Stefan Lucks, Kazuo Ohta, Rohit Prasankumar, Omer Reingold, Phil Rogaway, Serge Vaudenay, David Wagner, and Peter Winkler. I'm sure I must be missing a number of people on this list – for that I truly apologize. I also thank the various anonymous referees, who've read my papers, for their many insightful comments.

Beyond the people who've helped me learn about cryptography, I would like to thank the people who got me excited about computer science. To start with, I thank Bob Saenger, for a very well-taught computer science class at Bronx Science. It helped spark my decision to pursue an undergraduate CS degree. Next, I thank Professor S.S. Ravi for being a great teacher and mentor. He has always been a fountain of wonderful advice on various aspects of research and learning. I'll always be thankful to Ronitt Rubinfeld for introducing me to formal computer science research. Her enthusiasm was incredibly infectious, and she always seemed to believe in me – even when I didn't. She helped make my undergraduate experience thoroughly enjoyable and rewarding; for that I can't thank her enough.

While at MIT, I had the opportunity to interact with a number of truly brilliant and talented students. It almost felt as if I could always find someone to help me out with a particular question or technical point (or in most cases, give me a million other questions to think about). In particular, I want to thank Ben Adida, Victor Boyko, Dwaine Clarke, Yevgeniy Dodis, John Dunagan, Jon Feldman, Kevin Fu, Venkatesan Guruswami, Helen Hong, Raj Iyer, Stas Jarecki, Adam Klivans, Eric Lehman, David Liben-Nowell, Moses Liskov, Anna Lysyanskaya, Tal Malkin, Danielle Micciancio, Alantha Newman, Sofya Raskhodnikova, Leonid Reyzin, Matthias Ruhl, Amit Sahai, Sudipta Sengupta, Abhi Shelat, Adam Smith, Nati Srebro, Nitin Thaper, Salil Vadhan, and Yoav Yerushalmi. They made the theory group, and the Lab for Computer Science, in general, an exciting and intellectually stimulating place to be.

Another important member of the theory group also deserves special thanks: Be Blackburn. Her seemingly infinite supply of chocolate, hot popcorn, and other goodies helped get me through many long days and nights on the third floor. One of Be's many talents is knowing how to make red tape disappear, which definitely made life much easier in a number of circumstances.

Besides those who helped make MIT a wonderful place to learn computer science, I want to thank the many, many wonderful life-long friends I made while I was here. They have made these past few years incredibly fun. I would like to mention all of them by name, but I know I'd most likely forget someone.

I do, however, want to mention my closest and best friend, Suma Dutta. I thank Suma for all of her love and support as well as for uncountably many wonderful and fun times. I thank her for studying alongside with me, in my office or W20, even when the weather was wonderful outside. Suma's surprise deliveries of warm brownies and other goodies to my office helped me get through the long nights before conference deadlines. She truly helped me focus and get through the many hurdles I faced, especially during this last semester.

Last, but certainly not least, I'd like to thank my family for all of their love and support over the years. And in particular, I want to give my most overwhelming thanks to my wonderful and loving parents: Amin and Nassim Ramzan. They have been there for me through thick and thin, and never hesitated to do what they could to help whenever I asked. I thank them for taking the adventurous risk of coming to this country. They came here solely for the purpose of making sure that I'd have the educational opportunities that they never had. I will forever be indebted to them for giving up their own dreams so that I could achieve mine.

Contents

Chapter 1

Introduction

1.1 Overview

In the late 1970's three ground-breaking papers on cryptography were published. These were the results of Diffie and Hellman [45], Merkle and Hellman [96], and Rivest, Shamir and Adleman [125]. Together these seminal papers pioneered the field of public-key cryptography, and brought cryptography research to the main stream. In each case, the problem of compromising the security of these proposed schemes is reduced to some well-defined mathematical problem. The general concept of reducing one problem to another had already been seen in the theory of NP-completeness whose development, not surprisingly, began just a few years earlier [37, 74].

The idea of designing cryptographic primitives whose security provably relies on the hardness of a well-defined mathematical problem has been the focus of much academic research in cryptography and a great deal of effort has been made to lay down the theoretical foundations of cryptography. The field of cryptography itself, however, is significantly older, and the theoretical foundations that paved the way for the future of public-key cryptography were conspicuously lacking when it came to the conventional field of private-key cryptography. In particular, very little research of this sort had been done in the area of block cipher design. Recall that a *block cipher* is a family of permutations on a message space indexed by a secret key. Each permutation in the family deterministically maps *plaintext* blocks of some fixed length to *ciphertext* blocks of the same length; both the permutation and its inverse are efficiently computable given the key.

Block ciphers have been popular in private-key cryptography for some time. The best known practical example of a block cipher is the U.S. Data Encryption Standard (DES) [108], which has been used to encrypt a large number of transactions. Much of the work surrounding block ciphers has been heuristic in nature, and very little has been understood about why certain constructions are secure while others are not.

In their ground-breaking paper, Luby and Rackoff [88] formally model the notion of a secure block cipher, and show how to realize this notion under standard cryptographic assumptions. Their work was motivated by the design of DES. They consider a block cipher to be secure ("super pseudorandom," or secure under both "chosen plaintext" and "chosen ciphertext" attacks) if, without knowing the key, a polynomial-time adversary with oracle access to both directions of the permutation is unable to distinguish it from a truly random permutation on the same message space. This definition extends the concept of a pseudorandom function generator which is due to Goldreich, Goldwasser, and Micali [62], where the adversary has oracle access only to the forward direction of the function.[1] The Luby-Rackoff construction of a super pseudorandom permutation was a theoretical breakthrough and stimulated a great deal of research. We use the term Luby-Rackoff cipher to describe constructions based on these principles.

In this dissertation, we engage in a deeper study of Luby-Rackoff ciphers with the aim of better understanding their security properties and making them more practical. First, we start by trying to optimize the original Luby-Rackoff construction, both in terms of computation time and key length. Next, we consider what happens when we modify some of the operations in the cipher and examine the security of these variants. Third, we propose a more refined notion of security for block ciphers, in general, and completely characterize the security of various Luby-Rackoff style ciphers under this new model. Finally, we look at how to efficiently construct Δ-universal hash functions, which appear as a component in many of the Luby-Rackoff cipher constructions we consider.

We discuss these areas in more detail in the subsequent chapters. The work in the thesis offers a cohesive presentation of the results in a paper by Reyzin and Ramzan [121]; two papers by Patel, Ramzan, and Sundaram [114, 115]; a paper by Etzel, Patel, and Ramzan [49]; and two standards contributions due to Patel, Ramzan, and Sundaram [116,

[1]The original Luby-Rackoff paper [88] also considers block ciphers that are just *pseudorandom*, or secure against chosen plaintext attack only, where the adversary has access only to the forward direction of the permutation.

117].

The rest of this chapter discusses block ciphers and message authentication codes, which are central to this thesis, and we finish the chapter by summarizing the results we present in the remaining portion of this dissertation.

1.2 Block Ciphers

1.2.1 Overview

Block ciphers are one of the most frequently seen and used symmetric-key cryptographic primitives. A block cipher is a family of permutations on a particular message space, indexed by a secret key. A block cipher, instantiated on a particular key, can be used to encrypt a message. The message space typically consists of fixed-length bit strings. The number of bits needed to represent an element of the message space is called the *block length.* The secret key is also typically a string of bits, but known only to select parties. For example, the U.S. Data Encryption Standard (DES) [108] employs a 56-bit key, and operates on 64-bit message blocks. To encrypt longer messages, one can break the message up into 64-bit blocks (perhaps padding the last block if necessary) and apply the block cipher transformation to each block individually. There are, however, a variety of ways in which one can utilize a block cipher to encrypt data. These methods are typically referred to as *modes of operation.*

1.2.2 Modes of Operation

We now describe various methods for encrypting a large amount of data using a block cipher. We start with a block cipher B, and secret key k. Suppose we want to encrypt the plaintext P. We break it up into b blocks: $P_1, \ldots, P_b$, where the length of these individual plaintext blocks is the block length of B. If the true length of the plaintext message is not a multiple of the block length, we can pad the message until it is.

The simplest mode of operation for a block cipher is the electronic codebook (ECB) mode (see figure 1-1). Here we simply take each individual block P_i, encrypt it with the block cipher B under the key k, and call the resulting ciphertext block C_i. The full ciphertext is then $C_1, \ldots, C_b$. To decrypt a message, one simply decrypts each block via the inverse of the block cipher.

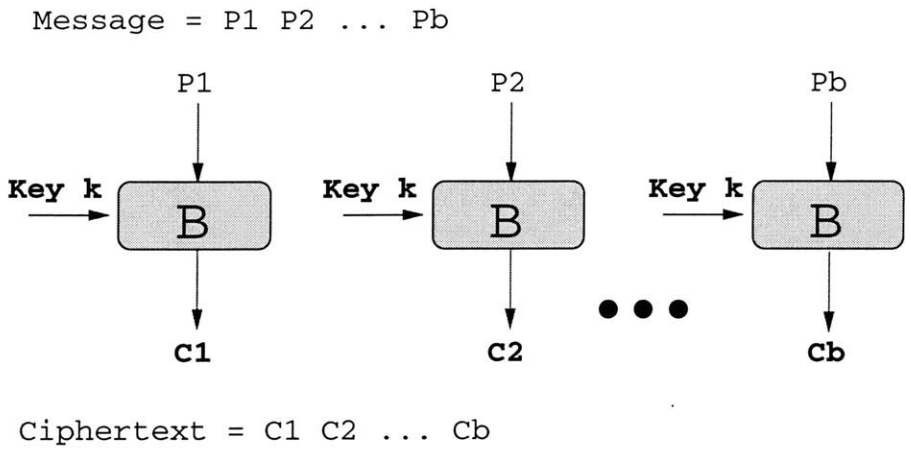

Figure 1-1: Encrypting Data in Electronic Codebook Mode

One drawback of the ECB mode of encryption is that if two distinct blocks P_i and P_j $(i \neq j)$ happen to contain the same text, then the corresponding ciphertext blocks C_i and C_j will look identical. This property is not terribly desirable. One alternate encryption mode, which overcomes this dilemma, is the cipher block chaining (CBC) mode (see figure 1-2). Here we start with a fixed and public initialization vector IV, which is a bit string whose length is the same as the block length. The ciphertext $C_1, \ldots, C_b$ is then computed as follows:

$$C_1 = B_k(IV \oplus P_1)$$

and for each $2 \leq i \leq b$,

$$C_i = B_k(C_{i-1} \oplus P_i).$$

Here the symbol $\oplus$ denotes the bit-wise exclusive-or operation (or bit-wise sum modulo 2). Thus, a given ciphertext block is a function of the bit-wise exclusive-or of the corresponding plaintext block with the previous ciphertext block. Decrypting a message simply involves decrypting each block in-turn, and applying the bit-wise exclusive-or.

Another well-known mode of operation is the output feedback mode (see figure 1-3). Here we start with a seed S, and compute

$$S_1 = B_k(S)$$

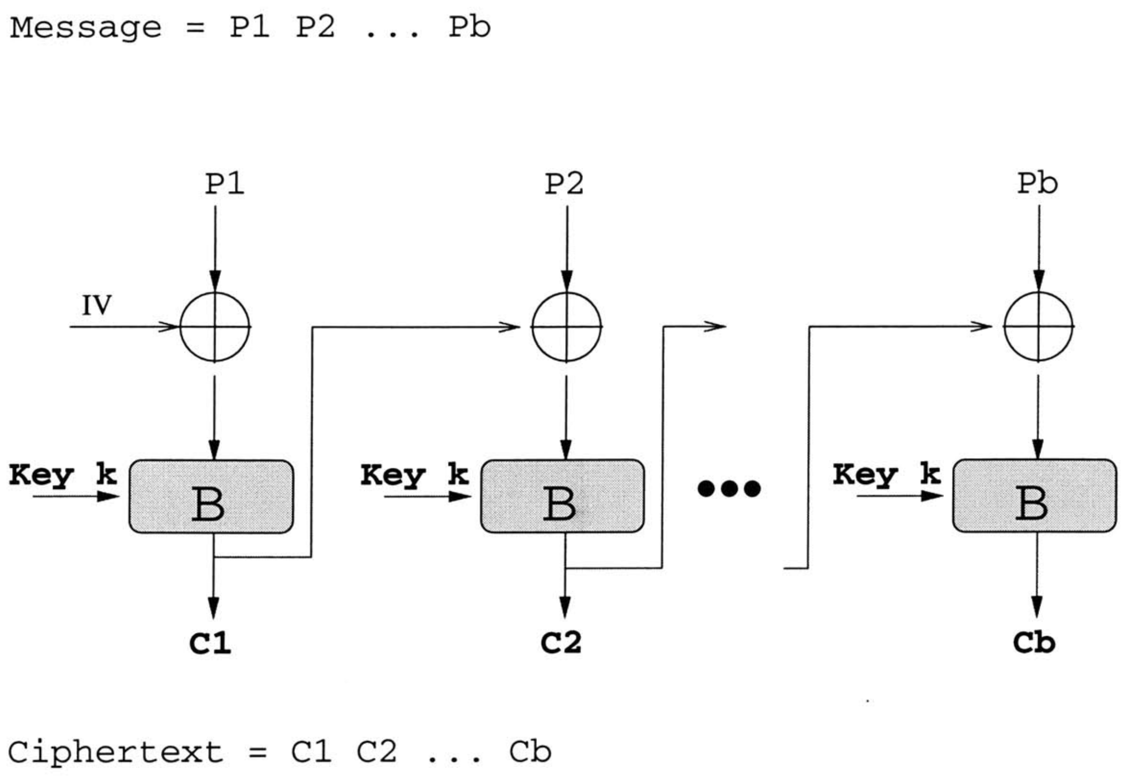

Figure 1-2: Encrypting Data in the Cipher Block Chaining Mode

and for each $2 \leq i \leq b$,

$$S_i = B_k(S_{i-1}).$$

The i^{th} ciphertext block is then defined by

$$C_i = P_i \oplus S_i.$$

In this mode of encryption, the block cipher is turned into what is called a *stream cipher.* We discuss stream ciphers in section 1.2.8.

The counter mode is another frequently used method for encryption. In counter mode, we start with a fixed counter value; for example 0. We encrypt the current counter value, and the ciphertext block is the bit-wise exclusive-or of the plaintext block and the encryption of the counter value. We increment the counter, and move on to the next block. The block cipher is again turned into a stream cipher in this mode of encryption. We discuss a minor variation on this mode when we discuss stream ciphers. For a more thorough discussion on

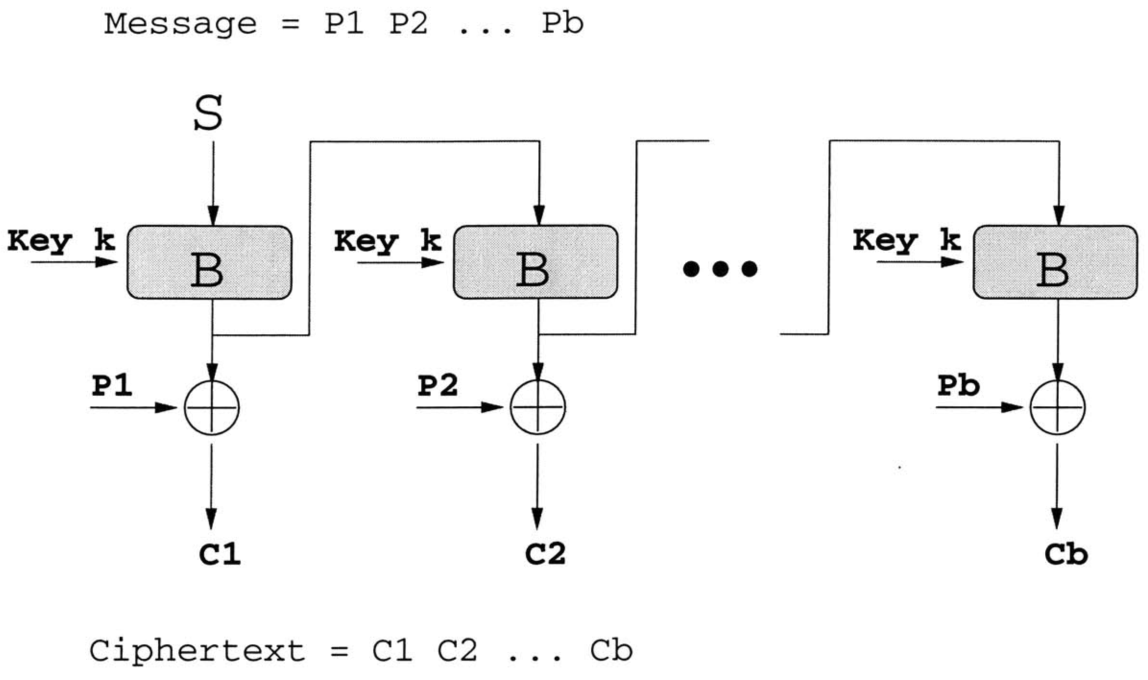

Figure 1-3: Encrypting Data in the Output Feedback Mode

various modes of encryption the reader may refer to the original specification for DES modes of operation [109], or to chapter 7 of the comprehensive text by Menezes, van Oorschot, and Vanstone [95].

1.2.3 Evaluating Block Ciphers

Now that we have explained, roughly, what a block cipher is, we explain how one might evaluate a block cipher. There are a variety of criteria that one may want to consider in judging a block cipher design proposal. We list a few below, though we stress that this list is by no means complete, and many of these criteria are related to one another.

1. Security: What degree of data confidentiality are we striving for? This is the most important property to consider. Since it is so important, we carefully and rigorously define what we mean by security later in this thesis. For now we give an intuitive definition. Informally, we say that a block cipher is secure if it appears to an adversary to behave like a truly random permutation even if the adversary has a certain kind of black-box access to the cipher. We explain this notion of a black box when we discuss the adversarial model in section 1.2.5. This gives us an "ultimate" form of security

since the adversary essentially gains no information about the plaintext (other than its approximate length) from the ciphertext. In certain situations, one may not always be willing to pay for such a strong level of security. In practice, one often evaluates security in terms of weaker notions such as how easy it is to either recover the secret key, or decrypt messages of one's choice, without a priori knowledge of the secret key. We prefer to consider the stronger notion of indistinguishability from a random permutation since other definitions may not be adequate for a variety of scenarios.

2. Encryption / Decryption Speed: How much time does it take to encrypt data? The answer may have a different interpretation depending on a variety of criteria. For example, it may depend on whether we are talking about encrypting a single block of data, or multiple blocks. We also need to consider the kinds of environments are we trying to optimize for, such as software, special purpose hardware, or particular microprocessors. We may attain radically different speed estimates depending on the exact nature of the environment; for example, DES works especially fast in hardware. On the other hand, it contains various transformations at the individual bit level, which are not handled efficiently by most modern microprocessors without careful and extensive optimization. Thus its software performance is relatively poor.

   Another consideration is that some modes of block cipher operation only require computing the forward direction of the permutation. Consider, for example, the output feedback mode or the counter mode of encryption. Even when one needs to decrypt the data, there is never a need to actually compute the inverse direction of the permutation. In such cases, the performance relies on the speed of computing the forward direction rather than the speed of computing the reverse direction. While these two speeds are nearly identical for many block ciphers, there are a number of examples for which the values are different. One particular example is the Rijndael cipher [41] which was chosen by the National Institute of Standards to be the next-generation United States encryption standard. In all current Rijndael implementations, the forward direction of the permutation takes less time to compute than the reverse direction.

   While achieving security is the most important design goal, there are a number of applications for which the costs associated with the cryptography are a very small portion of the overall cost of the system. Such applications include, but are not limited to, ATM and ISDN switches, Virtual Private Network (VPN) Routers, Database servers, and Firewalls. Thus, overall performance is an important issue to consider.

3. Memory Requirements: How much memory does it take to implement the cipher? If the cipher requires too much memory, it may not be possible to implement it effectively in certain environments (e.g. smartcards). Also, memory usage effects running time since the implementation may require extra references to memory in the form of LOAD / STORE instructions, rather than through much faster data registers. In some situations, one may be able to increase the speed via the use of extra memory. This phenomenon occurs with the Rijndael Cipher [41]. This feature may be useful for applications where speed is more important than memory; for example, if a large centralized server is handling a number of concurrent client requests.

   Of course, if memory is of no concern, one can always design a block cipher via a series of very large look-up tables, though this approach will most likely not be very practical.

4. Key Size: How long should the key be to provide adequate security for the application at hand? If the key is too short, then it may be possible for an adversary to do a brute-force search through the entire key space. Having a long key may hurt performance since it adds to memory requirements, which may in-turn, lead to a slower implementation. Also, key size may be related to communication complexity since symmetric keys may be agreed upon and exchanged over a network via a key exchange algorithm such as Diffie-Hellman [45] or RSA [125]. We stress, however, that having a long key does not automatically guarantee security since there may be other weaknesses in the block cipher. For a more thorough discussion on adequate key sizes for achieving commercial security in symmetric ciphers we refer the reader to the report by Blaze *et al.* [26].

5. Block Length: What is the ideal size of the message block on which the cipher will operate? If the block length is too short, then an adversary might be able to build a "code book" of plaintext / ciphertext pairs, and use it to determine information about a large number of message blocks. If the block length is too large, then this may hurt performance since a short message may have to be padded to the block length before it can be encrypted. This may result in extra storage and communication. It might

help to have some flexibility in the design so that there are versions of the cipher that can operate on various different block lengths. For example, the specification for the Rijndael [41] block cipher focuses on 128-bit, 192-bit, and 256-bit blocks, but the cipher also generalizes to larger block sizes.

There is a paper by Bellare and Rogaway [17] which explains how to take *any* block cipher on a given block length, and design a new cipher that has the same security as the original construction, but operates on all larger block lengths. Unfortunately, the throughput is reduced by a factor of about two.

6. Complexity of Design: How intricate is the design? It is not easy to quantify precisely what one means by "simplicity" though in general, one should aim for relatively simple block cipher designs for a variety of reasons. First of all, simpler designs tend to be easier to analyze from a security standpoint. As a result one may be able to obtain formal proofs that the cipher is secure. In practice, it is more likely that one can prove theorems which provide evidence that a particular construction is secure. Second, simpler designs are more likely to get implemented correctly. The incorrect implementation of a cipher may lead to a compromise in system security which is completely independent of the underlying cryptography. Finally, design complexity is often related to performance; for example, if one were doing a hardware implementation of a block cipher via a Field Programmable Gate Array (FPGA), a complex design may require many Configurable Logic Blocks (CLB), which would increase the circuit area, and hence the cost. There are a number of papers which discuss FGPA implementations of block ciphers which explain these kinds of issues in more detail [48, 55, 141]. These kinds of issues also apply to other hardware-based technologies.

7. Code Size: How many lines of code does it take to implement the cipher? In some sense this criterion is captured by some of the others, so it may not be worth worrying about. At the same time, there are a number of applications in which this criterion is especially important. For example, if the cipher is implemented in Javascript or in a Java Applet on a web page, then fewer lines of code results in faster download time. Also, one may want to include source code for cipher in a self-extracting archive – this idea appears in a number of commercial products such as DataSafe[2] and PC-

[2]`http://www.data-encryption.com`

Encrypt[3]. This last application may be viewed under the aegis of the more general concept of *self-describing cryptography*. This concept was discussed by Adida in his MIT Master's thesis [2]. In such cases, smaller code size will lead to an improvement in communication complexity, and bandwidth needs.

We stress that this list of criteria is by no means complete. Different applications call for different evaluation criteria. Also, as technology changes, new criteria may become more important; for example, many of the applications that are concerned with code size have arisen because of the Internet and cross platform languages such as Java. Similarly, the need for cryptography in embedded devices such as personal digital assistants, cellular phones, and smartcards has increased the need for constructions that are especially efficient.

1.2.4 Practical Approaches to Block Cipher Design

There are a number of principles and tools for block cipher design which are commonly seen in practice. We briefly survey some of these approaches. For a more involved treatment, we suggest the survey article by Robshaw [126], and the references contained therein.

We start with the principles of *confusion* and *diffusion*. These ideas are due to Shannon and appear in his seminal paper [133]. Roughly speaking, confusion refers to the idea of trying to obscure the relationship among the plaintext, ciphertext, and the key. In practice, confusion is almost always achieved via the use of substitution boxes (or S-Boxes), which are essentially carefully designed look-up tables. Diffusion refers to the idea of spreading the influence of individual plaintext bits and key bits over a sufficiently large portion of the ciphertext. Confusion, by itself, can be enough to build a secure block cipher; for example, one could essentially create a block cipher via an extremely large look-up table. This approach will probably not be terribly practical in most situations.

In practice, block ciphers repeatedly mix confusion and diffusion steps. Such ciphers are known as product ciphers. Often times the core of such a block cipher is a relatively simple function of the plaintext and key that incorporates a confusion step and a diffusion step. The cipher then consists of repeated applications of this so-called round function. There are number of well-known product ciphers. These include Lucifer [136], DES [108], LOKI [30], FEAL [134], PES [84, 85] (also known as IDEA), Khufu and Khafre [97].

[3]`http://www.pc-encrypt.com`

The *Feistel cipher* is a specific type of product cipher. Such ciphers are, arguably, the most commonly seen in practice; the DES block cipher is an example. In fact, the ciphers we study in this thesis fall into this category. At the core of a Feistel cipher is a so-called Feistel permutation. The eponym for this term is Horst Feistel, who was on the original IBM team that designed the block cipher DES. It is not clear, however, if Feistel was the sole inventor of the concept. In particular, Notz and Smith, who were also on the original DES design team, may have played a role in developing this important concept [39].

The Feistel permutation works as follows (see figure 1-4). We start with an input block m that is n bits long (where n is even), together with a keyed length-preserving round function f that operates on bit strings of length $n/2$. The Feistel transformation on m is then:

$$m^R \cdot (m^L \oplus f(m^R))$$

where m^L and m^R are the left and right halves of m respectively. Typically f is a function of the round key. We discuss this type of permutation in much greater detail throughout the thesis.

The one thing to note is that even if the function f is not a permutation, the overall Feistel construction using f still is. A block cipher can now be designed via repeated applications of a Feistel permutation where the round function depends on the key.

1.2.5 Security of Block Ciphers: The Adversarial Model

Having briefly discussed how block ciphers are designed, we now examine the kinds of resources an adversary will be able to access when trying to compromise the security of the cipher. The most basic assumption we make is that the adversary has access to the specifications for the underlying algorithm being used, though he does not have access to the key. For example, the adversary should know the key length, the block length, and how the cipher encrypts/decrypts data as a function of the key. This idea is known as *Kerckhoff's Principle* and was published in a two-part article in the January and February issues of the Journal des Sciences Militaires, 1883 [75].

While we assume that the adversary has no knowledge of the key, one may want to consider alternate models in which the adversary has partial knowledge of the key. Constructions which are secure in this setting have appeared in the literature [33, 47], and this

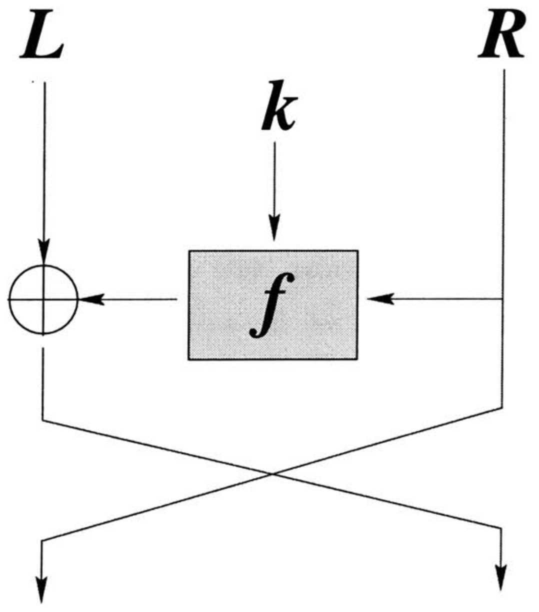

Figure 1-4: The basic Feistel permutation

branch of research is referred to as *exposure-resilient cryptography.*

We now look at the resources the adversary has with respect to plaintexts and ciphertexts produced by the cipher. There are two broad categories of attacks based on these types of resources: passive attacks and active attacks.

Passive Attacks

In passive attacks, we assume that the adversary can quietly eavesdrop. If the adversary has access to only various ciphertexts that are transmitted, then we classify any attack he makes as a *ciphertext only* attack. It is unlikely that any well-known cipher extensively used in practice can be compromised with this type of attack. A slightly more interesting case is the *known plaintext attack* in which the adversary gets to see various plaintext-ciphertext pairs. In its most generic form, no assumption should be made about the distribution of these known plaintext-ciphertext pairs, though we stress that many known plaintext attacks given in the literature assume that the plaintexts were generated at random. Indeed, many block cipher constructions can be broken if the adversary has access to enough randomly chosen plaintext-ciphertext pairs. See for example, Biham and Shamir's text on the differential cryptanalysis of DES [23].

Active Attacks

We now move on to the active attacks. In this scenario, we assume that the adversary has a certain kind of black-box access to the cipher. The adversary can query this cipher on inputs of his choice. For example, we can allow the adversary to see the encryption of *any message of his choice.* An attack based on such a resource is called a *chosen plaintext attack.* Similarly, if we allow the adversary to see the decryption of any ciphertext of his choice, we get a *chosen ciphertext attack.* Of course, it may even be possible that the adversary can mount both a chosen plaintext and ciphertext attack. One can further classify these types of attacks into static versus adaptive attacks. In a *static* attack, we assume that the adversary chooses all plaintexts (or ciphertexts) he is interested in encrypting (or decrypting) all at once. In contrast, an *adaptive* attack allows the adversary to choose future plaintexts (or ciphertexts) as functions of the answers it received to previous queries to its black box. In such cases he can interleave chosen plaintext queries with chosen ciphertext queries, which could be quite troublesome. We focus on this last scenario in this thesis.

At an initial glance, it may not be clear why active attacks have any practical importance. After all, if the adversary has access to a black box for decryption, he can decrypt any ciphertext he wishes. There are, however, practical scenarios in which these attacks are interesting. For example, the adversary may only have *temporary* access to a tamper-resistant device that encrypts and/or decrypts. Similarly, perhaps the adversary can obtain encryptions and/or decryptions for messages of a certain restricted form, but not necessarily for messages that might be of interest to him. There are a variety of other scenarios one can imagine. For this reason, it is important to consider the most general and powerful attacks the adversary can mount. The ciphers we propose in this thesis are *provably* secure against adaptive interleaved chosen plaintext and ciphertext attacks.

Other Attacks

The passive and active attacks are a very broad and general class which capture a number of important scenarios. There are several other attack models of interest which we briefly discuss here.

One recently proposed notion is that of *round security.* This concept was introduced in a paper by Ramzan and Reyzin [121], and is a novel contribution of this thesis. We discuss it in great detail in chapter 5. Briefly, the round security model allows an adversary black-box access to some of the internals of a particular cryptographic primitive in addition to allowing him black-box access to the primitive as a whole. The hope is to gain more insight into why particular constructions are secure or insecure.

Another method of interest is *power analysis* [78, 79]. The idea is that, in practice, any reasonably tamper-resistant device that performs cryptographic computation still leaks certain kinds of information; for example, one can measure power consumption, electromagnetic radiation, and the time taken by the device to perform the computation. Careful analysis of these quantities can sometimes lead to information about the underlying secret key involved in the algorithm. This type of attack is dangerous since it depends less on the underlying mathematics of a cryptographic scheme, and more on its concrete implementation.

1.2.6 Cryptanalysis of Block Ciphers

Having discussed the types of resources an adversary has when trying to compromise the security offered by a block cipher, we turn to various techniques in the cryptanalysis of block ciphers. The term cryptanalysis refers to the art of trying to "break" a cryptosystem. For the most part, the literature on the cryptanalysis of block ciphers has focused on the traditional passive and active attack models we just discussed. We briefly survey some of the more noteworthy results in the cryptanalysis of block ciphers.

The most basic mechanism for compromising the confidentiality of the data encrypted by a block cipher is via an exhaustive search over the entire key space. This method works if you have a few of the plaintext messages and their corresponding ciphertexts, so you can check if a given guess for the secret key is correct. Of course this method is grossly inefficient, but in some ways it serves as a benchmark. We can determine the quality of a cryptanalytic attack by comparing its efficiency to that of doing an exhaustive key search. For certain block ciphers, however, exhaustive key search is not impossible. For example, DES only has a 56-bit key, and many efforts have been made to defeat it by an exhaustive key search. One of the most noteworthy efforts is Deep Crack [54] which is a special purpose hardware device, that costs about $250,000, and can search through all 2^{56} possible DES keys in a matter of a few days. A number of techniques for building cryptanalytic hardware to do exhaustive key search are discussed in a paper by Goldberg and Wagner [59]. The paper appears in chapter 10 of the Electronic Frontier Foundation's book [54], and some of these techniques were taken from a paper by Weiner [143]. We now move on to more specific cryptanalytic techniques.

Differential Cryptanalysis

Perhaps one of the most prominent results in this area is *differential cryptanalysis* which is due to Biham and Shamir [23]. Differential cryptanalysis involves a chosen plaintext attack in which the adversary picks pairs of plaintexts that have a particular difference between them. He then studies the difference between the corresponding ciphertexts. By studying the probability distribution of ciphertext pair differences for given plaintext pair differences, he may be able to gain information about the secret key. The method often works well since many block ciphers combine simple linear transformations with non-linear transformations. A differential between plaintexts can be analyzed with respect to the linear components, and one can try to approximate its distribution through the non-linear components. Thus, this method is especially effective if this distribution is heavily skewed.

The technique, in its most basic form, appears in a paper by Murphy [98], which gives an attack on the four-round version of the FEAL block cipher [134] requiring only 20 chosen plaintexts. Since the four-round version was suggested for a variety of applications, and since the attack is especially efficient, it points to a serious weakness in FEAL. Biham and Shamir generalize the technique and apply it to many block ciphers [20, 21, 22, 23]. Most notable is their ability to mount an attack on DES [23]. The attack requires 2^{47} chosen plaintexts and about 2^{37} time steps,[4] which is still fairly high, even though it is less work than doing an exhaustive search.[5] In fact, many other DES-like cryptosystems are significantly more susceptible to differential cryptanalysis than DES itself [20]. In part, DES seems like a good candidate for differential cryptanalysis since all of its components, except for the S-boxes, are linear, and thus do not obfuscate differentials between pairs of inputs. Later, Coppersmith, who was part of the original DES design team at IBM admitted that their team knew about the technique, but could not tell anyone, and optimized the S-boxes in DES to resist the attack [38].

[4]The attack also works with 2^{55} *known* plaintexts, where the plaintexts are independently and identically distributed.

[5]The DES cipher has 2^{56} different keys, and so on average an exhaustive search will yield a key after roughly 2^{55} tries.

There are a number of variations on the basic theme of differential cryptanalysis. Among these are *higher-order* differential attacks, due to Lai [82], and *truncated* differential attacks, due to Knudsen [76]. There are also a number of theoretical approaches to designing block ciphers that are secure against differential attacks, which we will discuss in section 1.2.7.

Linear Cryptanalysis

Linear cryptanalysis, which is another major breakthrough in block cipher cryptanalysis, is due to Matsui [93]. The idea is to find relations involving the parity of individual plaintext, ciphertext, and key bits. The technique was first applied to DES, and can recover the key using 2^{45} *known* plaintexts. One can compute the necessary parity relations by examining the number of times the exclusive-or of certain input bits to the S-box coincides with the exclusive-or of certain output bits. By observing when these values differ from what one expects under the uniform distribution, one can derive a *linear* approximation for the S-boxes. Since the S-boxes are the only non-linear part of DES, a linear approximation for the S-boxes extends to a linear approximation for the cipher as a whole.

Other Cryptanalytic Techniques

While linear and differential cryptanalysis are perhaps the two most prominent cryptanalytic techniques, by no means do they exhaust the list of possible techniques one could use when trying to break a block cipher. One other technique is *differential-linear cryptanalysis* which is due to Hellman and Langford [86, 69]. By combining techniques from differential and linear cryptanalysis, they are able to break various reduced-round versions of DES slightly faster than what was previously known. Another technique, is the *interpolation attack* due to Jakobsen and Knudsen [72]. Here one attempts to approximate block ciphers by polynomials, and then performs Lagrange Interpolation using plaintext/ciphertext pairs as data points. For more information on techniques for block cipher cryptanalysis one can consult the article by Schneier [130] and the references mentioned therein.

1.2.7 Provable Security of Block Ciphers

Having discussed various approaches to compromising the security of block ciphers, we turn our attention to the problem of designing block ciphers which have some form of "provable security." The central focus of this dissertation involves gaining a deeper understanding of certain provably-secure constructions. We start with some of the existing work, point out some of the inherent gaps, and then briefly discuss how this dissertation attempts to fill these gaps.

The work on provable security of block ciphers can be broadly classified into two categories. First there is work which attempts to design ciphers that resist specific attacks such as linear and differential cryptanalysis. The other broad category involves designing ciphers which are, to a certain extent, resistant to *all* efficient attacks. Typically, the security proofs involve relating the security of the entire cipher to a (possibly conjectured) mathematical property of an underlying primitive used in the cipher's design.

Resistance to Specific Attacks

One approach to block cipher design involves designing ciphers so that they are immune to specific forms of cryptanalysis such as linear and differential. As is typical in most block cipher designs, one starts with a function f which takes as input a block of text and a key. The function is then iterated a number of times with different keys, and each iteration is termed a "round." The final outcome is the ciphertext. Now, one can attempt to understand the security of the entire cipher by determining various statistical properties of the function f.

This approach was utilized by Nyberg and Knudsen [106], who make minor modifications to simple algebraic functions. Specifically, they begin with the function $g(x) = x^3$ taken over $GF(2^{33})$. They discard one bit of the output to obtain a function h whose domain is $GF(2^{33})$ and whose range is $GF(2^{32})$. Next, they employ an affine expansion function $E : GF(2^{32}) \rightarrow GF(2^{33})$. Their round function $f : GF(2^{64}) \rightarrow GF(2^{64})$ is then

$$f(L \cdot R) = R \cdot (L + h(E(R) + k))$$

where L and R denote the leftmost and rightmost 32 bits of the 64-bit input, and k is the round key, and the symbol $\cdot$ denotes concatenation. By iterating the function six times with six different independently and identically distributed keys, one obtains a cipher that cannot be broken via differential cryptanalysis.

There are a number of other similar functions which one could use to obtain the same types of results, and more examples can be found in a paper by Nyberg [105]. A slightly more complicated round function that offers immunity to differential cryptanalysis when iterated sufficiently many times is the one used in PES [84]. The security analysis was done by Lai, Massey, and Murphy [83]; the first two authors were the co-designers of PES.

Another approach to the design of such an f function was developed by Vaudenay [140], and is called *decorrelation theory.* The idea is to use k-wise independent hash function families. Roughly speaking, a family of functions is k-wise independent if the outputs corresponding to k distinct inputs are uniformly distributed and independent of each other. In chapter 2 we formally define strongly universal hash functions which are the same as 2-wise independent hash functions. Vaudenay's main observation is that by utilizing k-wise independent function families in a Feistel ladder, one can design block ciphers which are "almost" k-wise independent permutation families. He then shows that these almost k-wise independent permutation families are secure against the traditional linear and differential attacks.

Since, in practice, one typically implements a simple function like f through S-boxes, another approach one can use is to design secure block ciphers by designing good S-boxes. To start with, one must first identify the criteria that make a particular S-box good. Dawson and Tavares have a paper on this topic [43]. Next, one must explain how to design S-boxes that meet some or all of these criteria. There have been a number of approaches along these lines. One approach, studied by Adams and Tavares [1] involves the use of *bent functions.* Another approach due to Nyberg [105] involves taking a simple algebraic function, tweaking it, and having the S-box compute this new tweaked function. One can also just generate random sequences of bits, form an S-box using them, and then check if the desired properties are met. This approach was utilized in the Serpent block cipher [7] which was a finalist for the Advanced Encryption Standard.

Immunity to Broad Classes of Attacks

One drawback of the previously mentioned approach is that it only thwarts certain kinds of attacks. Moreover, typically provable immunity is only achieved for the most plain flavor of these attacks. Thus there has been a great deal of work on designing ciphers that are essentially secure against all forms of attacks.

This approach was utilized by Anderson and Biham [6] in their Bear and Lion block ciphers. Both ciphers utilize *cryptographic hash functions* and *stream ciphers* (which we discuss in section 1.2.8) as underlying components. The security argument first involves making hallowed assumptions about these underlying components; namely that it is hard to find collisions or pre-images in the hash function, or that it is hard to find the seed of the stream cipher. Next, they show that if someone were to recover the key of the block cipher efficiently given one plaintext and its corresponding ciphertext, then they could violate one of the hallowed assumptions about the underlying component. While efficient key recovery is one threat, it is clearly not the only one to worry about. Thus, while both Bear and Lion are quite fast, the security guarantees one can make about them are of limited scope.

A more powerful result on designing secure block ciphers was obtained by Luby and Rackoff [88], and this dissertation studies their result in more detail. We now briefly survey some of the literature related to the Luby-Rackoff result; a more detailed treatment is presented throughout the dissertation. Luby and Rackoff provide a formal model for the security requirements of block ciphers in their seminal paper [88], and give a construction of a block cipher that achieved this level of security. Their work was motivated originally by the study of security of the block cipher DES [108]. They consider a block cipher to be secure, or super pseudorandom if, without knowing the key, a polynomial-time adversary with oracle access to both directions of the permutation is unable to distinguish it from a truly random permutation on the same message space. This notion captures the ultimate level of security one could desire under an adaptive interleaved chosen plaintext and chosen ciphertext attack. Luby and Rackoff also discuss the notion of a block cipher that is indistinguishable from random under just adaptive plaintext queries; they refer to this notion simply as a pseudorandom permutation. The Luby-Rackoff definition of a super pseudorandom permutation extends the the definition of a pseudorandom function, due to Goldreich, Goldwasser, and Micali [62]. In that scenario the adversary has oracle access only to the forward direction of the function.

The Luby-Rackoff construction of a super pseudorandom permutation was a theoretical breakthrough and stimulated a great deal of research. We use the term Luby-Rackoff cipher to describe constructions based on these principles. The original Luby-Rackoff construction is based on a pseudorandom function generator [62]. It consists of four rounds of *Feistel permutations* [51], each of which involves an application of a pseudorandom function and an exclusive-or operation. Each round's output is used for the next round's input, except for the last round, whose output is the output of the block cipher.

The cipher proposed by Luby and Rackoff operates on a $2n$-bit string $L \cdot R$, where L and R are n bits each, and can be described simply as follows:

$$\begin{aligned} S &= L \oplus f_1(R); \\ T &= R \oplus f_2(S); \\ V &= S \oplus f_3(T); \\ W &= T \oplus f_4(V). \end{aligned}$$

Here f_1, f_2, f_3, f_4 are independently-keyed pseudorandom functions whose domain and range are bit strings of length n, $\oplus$ represents the bit-wise exclusive-or operation, and the output is $V \cdot W$.

The original proofs of security due to Luby and Rackoff were subsequently simplified by Maurer [94]. Though his focus was on the non-adaptive case of the three-round Luby-Rackoff cipher. Following this work, Lucks [89] further generalized the proofs to include *unbalanced Feistel networks* and contributed the notion of a *difference concentrator*. This is a non-cryptographic primitive that replaces the pseudorandom function in the first round but still offers the same security. In parallel, much research has concentrated on obtaining variants of Luby-Rackoff constructions where the number of different pseudorandom functions used in the four rounds is minimized in order to save key material. For example, see the papers by Patarin and Pieprzyk [112, 118].

Following these works, Naor and Reingold [101], established a very efficient generalization where they formalize Lucks' treatment by using strongly universal hash functions and also provide a clean framework for proofs of security in the case of adaptive attacks. Naor and Reingold achieve an improvement in the time complexity by using only two pseudorandom function applications on n-bit strings to compute the value of a $2n$-bit pseudorandom permutation. The central idea is to sandwich a two rounds Feistel permutation involving pseudo-random functions between two pairwise independent $2n$-bit permutations. In other words, f is an n-bit to n-bit pseudorandom function and h_1, h_2 are two pairwise independent $2n$-bit permutations (for example $h_i(x) = a_i x + b_i$ over $GF(2^n)$, where a_i, b_i are uniformly distributed). Naor and Reingold prove that this construction is $2n$-bit super pseudorandom permutation. They generalize this construction by relaxing the pairwise independence condition on the exterior permutation rounds but changed the interior rounds to include two different pseudorandom functions. Their work fundamentally contributed to the understanding of Luby-Rackoff ciphers, and inspired much of the work in this dissertation. Subsequent work on optimizing the Naor-Reingold construction was done by Patel, Ramzan, and Sundaram [114], and this work will be described in great detail in chapter 3.

A number of variants to the original constructions were considered. For example, Patel, Ramzan, and Sundaram [115] examined the question of what happens when the exclusive-or operation, in the Feistel permutation, is replaced by other operations. We discuss these results in detail in chapter 4.

Ramzan and Reyzin [121] later proposed a more refined model of security for symmetric-key primitives and completely characterized the security of a number of Luby-Rackoff style ciphers in this model. This work will be presented in chapter 5.

1.2.8 Block Ciphers versus Stream Ciphers

Having discussed block ciphers in some detail, we take a step back and ask ourselves a more fundamental question: is a block cipher the best tool for encrypting data? The answer to this question is not clear. In this section we briefly discuss *stream ciphers* which are another symmetric-key primitive for encrypting data, and compare them to block ciphers. We stress, though, that block ciphers and stream ciphers are not completely disjoint notions; in fact, one can take a block cipher and build a stream cipher from it, and vice versa. Stream ciphers, in their most basic form, can typically be broken up into three constituent components:

1. Internal state: this component is often a sequence of bits, and initially the internal state is a function of the stream cipher's secret key. The key is often referred to as a *seed* in the context of stream ciphers.

2. Next state function: this component maps the existing state to some other state as a function of the key and possibly some portion of the message.

3. Final transformation: this component takes a piece of the plaintext and the current state as input, and computes the ciphertext.

The process defined by these three components is then repeated on subsequent plaintext blocks. Before proceeding further, we give an example of a stream cipher (see figure 1-5), which essentially involves using a block cipher in counter mode. Let us start with the DES

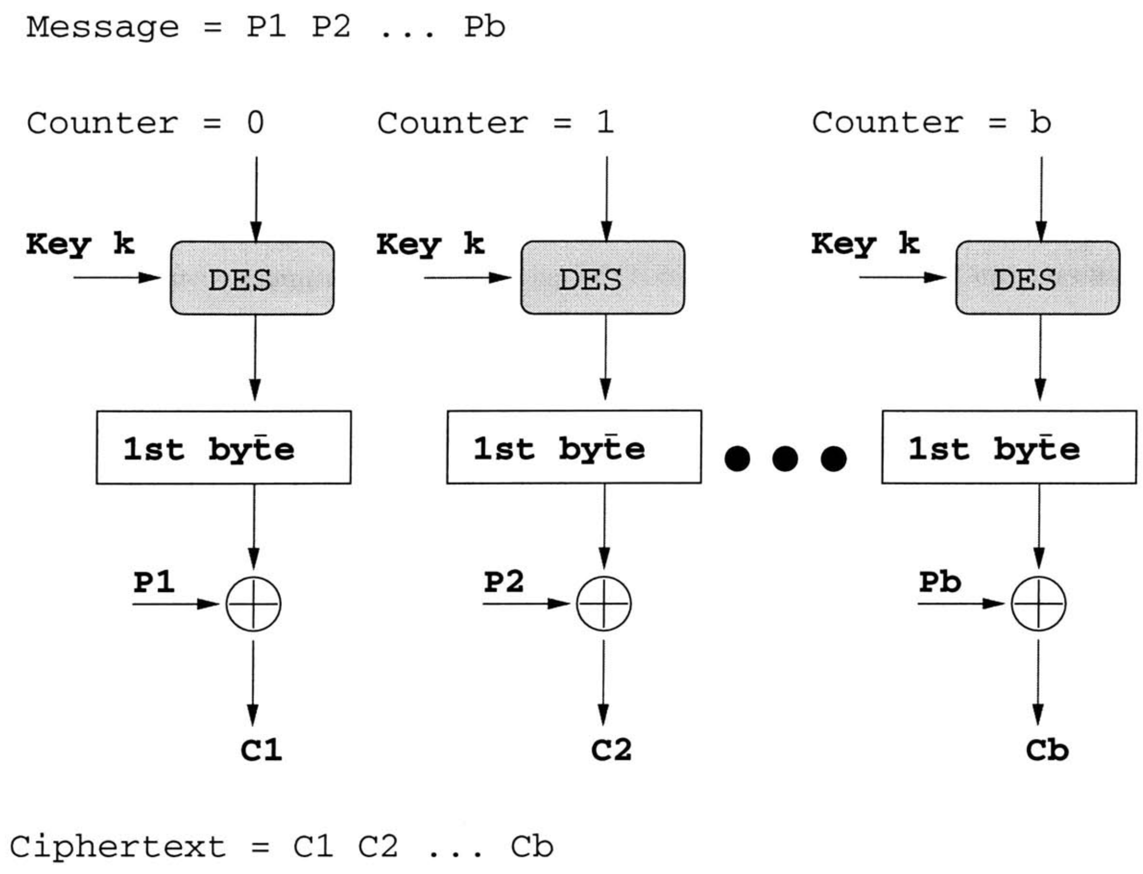

Figure 1-5: A stream cipher using DES

block cipher, and a counter which begins at 0. The seed to our stream cipher will be a 56-bit key k. We compute the ciphertext as follows. We encrypt the value of the counter, under DES with key k. We take the first byte of the output given by DES, and take the bit-wise exclusive-or with the first byte of the plaintext. The next state is then obtained by incrementing the counter and encrypting it under DES with the key k. The next block of ciphertext would thus be obtained by taking the first byte of the output given by DES and taking the bit-wise exclusive-or with the second byte of the plaintext. We repeat this process until we have encrypted all the data.

Now that we have described stream ciphers and given an example, we compare them to block ciphers. The first thing to notice is that while block ciphers and stream ciphers transform the plaintext one block at a time, block ciphers tend to encrypt larger input blocks than stream ciphers. For example, the DES block cipher encrypts messages in 64-bit blocks, whereas the stream cipher we described works on 8-bit blocks. Of course, in the above example, we could have designed the stream cipher to encrypt in 64-bit blocks as well, but we chose not to in order to illustrate the point that stream ciphers can work on smaller chunks of data. Since stream ciphers can encrypt smaller portions of data, they may be advantageous when encrypted messages must be transmitted over a network. In particular, for many common encryption modes, the length of the ciphertext transmitted by a block cipher will always be a multiple of the block length. On the other hand, if we use the stream cipher just mentioned, then the ciphertext has the same length as the plaintext.

The next observation to make is that stream ciphers may apply different transformations to different blocks of the plaintext. In particular, the output of the stream cipher is a function of its current state, and since the current state is sequentially updated, the corresponding transformations on two identical looking message blocks may be different. On the other hand, since a block cipher is a permutation, if we feed the same input to the permutation, then we get the same output. This feature may be problematic for certain encryption modes, such as the ECB mode. For this mode, if two different message blocks contain the same text, then the corresponding ciphertext blocks contain are identical. Of course, one can avoid this type of behavior with a block cipher by using it in a feedback or chaining mode.

A third observation is that, with a block cipher, both the plaintext and the key are extensively obfuscated in order to produce the ciphertext. Typically, in a stream cipher, a complex transformation is applied to the seed, though the final transformation is often simple. It is not clear that this observation directly leads to a weakness of one method versus another, but it primarily points out one fundamental difference between these two constructs.

One weakness of stream ciphers is that they are often *malleable*. That is, if the adversary intercepts a ciphertext, he can simply modify a few of the bits, and in many cases the plaintext corresponding to this corrupted ciphertext will look quite similar to the original pristine plaintext block, with those few bits changed. As a result, the decrypted message might appear to be valid, which results in miscommunication between the sender and the recipient, and could potentially be problematic. On the other hand, if a block cipher is used for encryption, then the decryption of the corrupted ciphertext block will look almost nothing like the original plaintext block, and will probably look like random junk. The recipient will probably be able to tell that the message block was corrupted en-route, and may ask for the message to be retransmitted. Of course we stress that one could avoid these problems through the use of *message authentication codes*, or MACs, which we discuss in section 1.3.

One issue that is related to the one just mentioned is that of error propagation. Suppose that encrypted messages are being transmitted over a noisy and unreliable environment, such as a wireless connection. As a result it may be possible that a single bit of the ciphertext gets flipped from a 0 to a 1, or vice versa. If the plaintext was encrypted with a stream cipher, the plaintext that results from decrypting this corrupted message with probably still look like the original plaintext, and so it may be easier to correct the transmission errors. On the other hand, if a block cipher had been used, the plaintext corresponding to the corrupted ciphertext would probably not look very much like the original plaintext. These types of problems could easily be overcome through the use of *error-correcting codes.* See the classic text by MacWilliams and Sloane [91] for a thorough exposition on error-correcting codes.

Given the points just mentioned, there is no overwhelming reason to prefer a block cipher over a stream cipher, or vice versa. It depends on the exact nature of the application, and in part on the personal preferences of the system designer. Surprisingly, however, block ciphers have received a significantly more extensive treatment in the literature. Much has been done in the way of developing general techniques for designing and cryptanalyzing block ciphers, whereas the body of academic literature on stream ciphers has been more sparse. For that reason, as well as the fact that block ciphers can easily be used in a stream cipher mode, we focus our own efforts on block ciphers. We refer the reader to the survey article by Robshaw [127] for more information on stream ciphers.

1.3 Message Authentication Codes

Message authentication is one of the most important tasks in cryptography. While it is not the main focus of this dissertation, the topic comes up in chapters 3, 5, and 6. For this reason, we give a high level description and brief history of the problem. In chapter 2 we give a more rigorous mathematical description.

Message authentication schemes involve communicating parties, and aim to fulfill two primary goals:

1. Authentication: messages are indeed sent by the person who is purported to have sent them.

2. Integrity: messages are not modified en-route.

The problem was first formally studied by Gilbert, MacWilliams, and Sloane [57]. The task of message authentication can be accomplished via the computation of a message authentication code, or MAC, on a message. The MAC should be designed so that knowledge of a secret key is required to both compute and verify the MAC.

In general, message authentication codes will be computed frequently and on inputs which are often thousands of bytes long. Moreover, computing and verifying tags is typically done in software, and may be done on relatively weak platforms. Additionally, the computations must often be done in real time. Therefore, developing techniques for optimizing MAC Algorithms while retaining the appropriate level of security is crucial.

There have been a number of approaches in designing MACs. One approach to message authentication involves using a secure block cipher, such as DES, in CBC mode. We simply take the last block produced by the CBC mode of DES, and encrypt it one more time under a different key, which is independent of the others, and uniformly chosen. Although this method was known for quite some time, very little was understood about its security. Eventually Bellare, Kilian, and Rogaway [14] showed that this method is secure under the assumption that the underlying block cipher is a pseudorandom permutation, and all messages are of the same length. Subsequently, Black and Rogaway [25], developed variants of the CBC MAC which are provably secure for variable-length messages.

Another approach to message authentication, often seen in practice, involves using cryptographic hash functions like MD5 [123]. For example, one approach is to envelope a message block with the secret key: $\mathsf{MD5}(x \cdot m \cdot x)$. Here x is the secret key for the MAC and m is a message block. One can iterate this type of construction to compute a MAC on the entire message. Unfortunately, several schemes of this type are vulnerable to a key-recovery attack due to Preneel and and van Oorschot [120]. Another construction that uses cryptographic hash functions in MACs is the HMAC, which is due to Bellare, Canetti, and Krawczyk [10]; their scheme is good because it utilizes fast and secure cryptographic building blocks. At an initial glance, it might seem that these techniques yield the best results. It turns out, however, that a very fundamental paradigm due to Wegman and Carter [142] allows us avoid using heavy duty cryptographic primitives on the entire input string via the use of *universal hash functions.* More remarkable is the fact that Wegman-Carter paradigm appeared in the literature over a decade before the formal security requirements for message authentication. In chapter 6, we describe the construction of the square hash, which is a fast universal hash function that can be used in the Carter-Wegman paradigm. The square hash can also be used with the block cipher constructions given in other parts of this thesis.

1.4 The Universal Hash Function Approach

In this approach, one starts with a family of ϵ-almost-Δ-universal hash functions H. We formally define this notion in chapter 2. In order to compute the authentication tag for a message m, the communicating parties secretly agree on a function $h \in H$ chosen at random, and on a sequence of random pads $p_1, p_2 \ldots$. To compute a MAC on the i^{th} message m_i, the sender computes $\mu_i = h(m_i) + p_i$. One remarkable aspect of this approach is that, even if a *computationally unbounded* adversary performs q black-box oracle queries to both algorithms used by the MAC, he has probability less than $q\epsilon$ to forge the MAC. In the Wegman-Carter construction one pre-processes a message quickly using universal hash functions, and then applies a cryptographic operation such as a one-time pad. In general, the one-time pad can be replaced by pseudorandom sequence [27, 87]. Then, the parties would have to pre-agree on the function h and on the seed s for the pseudorandom generator which produces this sequence. This approach to message authentication was suggested by Brassard [29]. If a pseudorandom sequence is used, then the resulting MAC is secure against a polynomial-time bounded adversary and typically the result only holds under the assumption that a given mathematical problem, such as computing a discrete logarithm, cannot be solved efficiently in the average case. Another technique employed in schemes such as UMAC [24], involves the use of pseudorandom functions [62]. Here one takes the output of the universal hash function, concatenates it with with a nonce and a counter, and feeds it into a pseudorandom function; the output produced by the pseudorandom function is the corresponding MAC. The nonce and counter are used to thwart statistical attacks based on the birthday paradox. The secret key for the MAC consists of the key for the hash function as well as the key for the pseudorandom function.

The original Wegman-Carter idea involves unconditionally secure message authentication. Prior to Wegman and Carter's original paper [142], unconditionally secure message authentication was studied by Gilbert, MacWilliams, and Sloane [57], which was the first paper to pose the problem of message authentication. The universal hash function approach for MACs was first studied by Wegman and Carter [142] and the topic has been heavily addressed in the literature [137, 80, 8, 68, 73, 135, 65]. In chapter 6, we develop a hash function based on the MMH scheme of Halevi and Krawczyk [65]. The MMH scheme achieves impressive software speeds and is substantially faster than many current software implementations of message authentication techniques and software implementations of universal hashing. Unfortunately, it is not always possible to do precise comparisons because the available data represents simulations done on various platforms, and often with code that was hand optimized. There are a number of papers in the literature that deal with experimental results of these types of constructions [135, 28, 104].

1.5 The Organization of this Dissertation

Having given some of the basic background and motivation, we outline the various chapters in this dissertation. We describe various preliminaries in chapter 2. We also discuss some of the notation, fundamental definitions, and mathematical tools necessary to understand all of the constructions and results we present throughout this thesis. In addition, we prove a number of general theorems which are used to analyze the constructions given in chapters 3, 4, and 5.

In chapter 3 we present a new construction of a more practical Luby-Rackoff cipher. The construction is efficient in terms of computation and key length. In addition, we consider an alternate security analysis of this construction, in which we make a weaker but more practical underlying assumption, and arrive at a weaker security claim. We also give an informal discussion on the optimality of Luby-Rackoff ciphers, and explain how the new construct we present fits into this framework.

Next, in chapter 4 we initiate a study of Luby-Rackoff ciphers in which we replace the exclusive-or operation in the Feistel ladder with addition in an arbitrary algebraic structure. This generalization opens up new lines of research, and we obtain a number of fascinating results. In particular, we construct a cipher which operates over various finite groups of characteristic greater than two, that is not only super pseudorandom, but also has better time/space complexity, and uses less key material than all previously considered Luby-Rackoff ciphers in the literature. Surprisingly, when we use the traditional bit-wise exclusive-or operation instead, the cipher can be distinguished from a random permutation with near certainty using only two queries. In addition to this result, we examine a number of other Luby-Rackoff ciphers which are known to be insecure when exclusive-or is used. In some cases, we can extend the attacks to our more general setting. The attacks are slightly more complicated. In other cases, it is unclear how to generalize the attacks, and determining the security of these new constructs is left as an open problem.

In chapter 5, we discuss a new model for analyzing the security of symmetric-key cryptographic primitives. The model makes use of the fact that many such primitives typically involve iterating simpler constructs for some number of rounds. The new model may be used to gain new insights into the security of these constructions. We completely characterize the security of a number of four-round Luby-Rackoff ciphers in this new model, and show that these ciphers remain secure even if the adversary is given black-box access to the middle two round functions. The powerful aspect of this result is that these extra resources offer the adversary *no advantage* in attacking the various ciphers we consider. We also describe situations in which Luby-Rackoff ciphers fail to remain secure in our model. These same techniques can also be applied to message authentication codes based on composing pseudorandom functions with universal hash functions, so we characterize their security in this new model as well.

Chapter 6 describes two novel ideas in the construction of fast universal hash functions. Such functions are used in numerous constructions throughout the thesis. In addition to their applicability in block cipher design, such functions can be used for message authentication. Universal hash functions possess various statistical properties which are useful in these cryptographic scenarios. Moreover, these properties are not conditioned on any kind of mathematical conjecture, so they always hold. In addition, since universal hash functions are not cryptographic, in and of themselves, they lend themselves to faster implementation. We introduce the square hash, which is a construction that performs especially well on modern microprocessor architectures. The first novel technique is to use squaring as opposed to multiplication, since the former can be implemented faster than the latter. The second novel technique involves ignoring various portions of the computation which, we can formally prove, does not significantly affect the required statistical properties for the applications at hand. One can think of this approach as "theoretical hacking." We discuss various implementation consideration and we substantiate our performance claims through hand optimized assembly language implementation results on an ARM processor.

We offer some concluding remarks in 7. Specifically, we summarize the results described throughout the thesis, and describe a number of possible avenues for future research. In particular, chapters 4 and 5 present entirely new paradigms in the study of provably-secure block ciphers, and thus open up new lines of work.

Finally, we have an appendix that describes the block cipher DES [108]. Luby and Rackoff were motivated by the design of DES, which led to their seminal paper [88], and eventually to the work in this dissertation. DES is a concrete example of a block cipher that uses the Feistel permutation, which we also use in the constructions we give.

Chapter 2

Preliminaries and Notation

2.1 Introduction

In this chapter we describe the notation to be used throughout the thesis, and we give various relevant definitions and prior constructions of Luby-Rackoff ciphers. Our presentation is in the "concrete" (or "exact") security model as opposed to the traditional complexity-theoretic model (though our results can be made to hold for either). Our treatment follows that of Bellare, Kilian, and Rogaway [14], and Bellare, Canetti, Krawczyk [11].

This chapter is organized as follows. We start by outlining some of the basic notation used throughout the thesis. In sections 2.3 and 2.4 we discuss basic tools such as finite function families, and various models of computation for our adversaries. These are necessary for understanding the cryptographic constructions given in this thesis. Then we discuss the concept of pseudorandomness, which we use to model the security of the block ciphers we consider. Next, we delve into the notion of a transcript, which helps us analyze security, and we state and prove various general theorems that relate transcripts to pseudorandomness. We go on to describe some of the specific constructions used in this dissertation such as universal hash function families and Feistel ladders. This paves the way for examining Luby-Rackoff ciphers, which are the central focus of this dissertation. Thereafter, we formally describe message authentication codes, which come up at various points in the thesis. In the next to last section we discuss the difference between the concrete security treatment and the traditional complexity-theoretic treatment, and then we end the chapter with some concluding remarks.

2.2 Notation

- We let I_n denote the set of bit strings of length n. In some cases, we use the notation $\{0,1\}^n$ to describe the same set.
- For a bit string x, we let $|x|$ denote its length.
- For bit strings x and y, we let $x \cdot y$ denote their concatenation. If x and y are the same length, then we sometimes write (x, y) to denote their concatenation.
- If x has even length, then x^L and x^R denote the left and right halves of the bits respectively; we sometimes write $x = (x^L, x^R)$ or $x^L \cdot x^R$.
- If x and y are two bit strings of the same length, $x \oplus y$ denotes their bit-wise exclusive-or.
- If n is a positive integer, we let $\mathrm{GF}(2^n)$ denote the Galois Field of order 2^n, and we let $\mathrm{GF}(2^n)^+$ denote the additive group attached to the field $\mathrm{GF}(2^n)$. Recall that elements of $\mathrm{GF}(2^n)$ can be represented as bit strings of length n, and that the addition operation on two elements merely amounts to taking their bit-wise exclusive-or.
- For functions f and g, where the range of f is contained in the domain of g, we let $g \circ f$ denote their functional composition; i.e. $x \mapsto g(f(x))$.
- If $\mathcal{D}$ and $\mathcal{R}$ are sets, then $\mathsf{Rand}^{\mathcal{D} \to \mathcal{R}}$ denotes the set of *all* functions with domain $\mathcal{D}$ and range $\mathcal{R}$.
- If k and l are positive integers, then $\mathsf{Rand}^{k \to l}$ denotes the set of all functions going from I_k to I_l.
- If $\mathcal{D}$ is a set, then $\mathsf{Perm}^{\mathcal{D}}$ denotes the set of *all* permutations on $\mathcal{D}$.
- If m is a positive integer, then Perm^m denotes the set of all permutations on the set I_m.
- If S is set whose elements can be sampled according to some pre-specified underlying probability distribution, then $x \overset{R}{\leftarrow} S$ denotes the process of picking an element x from S according to this distribution. Unless otherwise specified, the underlying distribution is assumed to be uniform.

- If S is a set contained in a universe U, then S^C denotes the set-theoretic complement of S – that is the set of elements of U that are not in S.
- If S is a set, and $\sigma = s_1, s_2, \ldots, s_n$ is a sequence consisting of elements from this set, then for $1 \leq i \leq n$ we let $\sigma^{(i)}$ denote the subsequence $s_1, \ldots, s_i$.

2.3 Finite Function Families

This thesis focuses on constructions of secure block ciphers. We model our ciphers as *pseudorandom permutation families.* Such permutation families are a type of finite function family. By a finite function (or permutation) family $\mathcal{F}$, we mean a set of functions with common domain and common range. In addition, the domain and range must be of finite size. Two particular examples are:

1. The set of all functions going from I_k to I_l, which we have denoted by $\mathsf{Rand}^{k \to l}$.
2. The set of all permutations on the set I_m, which we have denoted by Perm^m.

We can associate a key a with each function in a given finite function family. We permit the possibility that the same function is indexed by several different keys. We denote the function given by a key a as f_a. We let $\mathsf{Keys}(\mathcal{F})$ denote the set of keys for a keyed family $\mathcal{F}$. We assume that given the key a, it is possible to efficiently evaluate f_a at any point (as well as f_a^{-1} in the case of a keyed permutation family). For a given keyed function family, the *key length* is the maximum number of bits needed to denote any key for a function in the family. Often a key will be a string from I_s, in which case s is the key length. We also assume that the keys for our function families are picked according to some probability distribution. Typically, this distribution will be uniform.

2.4 Model of Computation

To discuss security, we need to explain our threat model. We do so by describing how an adversary will work. The adversary $\mathcal{A}$ is modeled as a program for a *Random Access Machine* (RAM) that has black-box access to some number k of oracles, each of which computes some specified function. The adversary $\mathcal{A}$ will have a one-bit output. If $(f_1, \ldots, f_k)$ is a k-tuple of functions, then $\mathcal{A}^{f_1, \ldots, f_k}$ denotes a k-oracle adversary who is given black-box oracle access to each of the functions $f_1, \ldots, f_k$. We define $\mathcal{A}$'s "running time" to be the number of time steps it takes plus the length of its description (to prevent one from embedding arbitrarily large lookup tables in $\mathcal{A}$'s description). This convention was used by Bellare, Kilian, and Rogaway [14].

Sometimes we abuse notation by listing an entire function family as the oracle, rather than just a single function. In this case, the oracle is considered to be a function (or some set of functions) chosen from the family, according to some induced probability distribution. That is, we can think of the oracle as a random variable, which denotes a function, and outputs the value of the function on any input queries it receives. For example, $\mathcal{A}^{\mathsf{Rand}^{n \to n}}$ would be used to denote an adversary whose oracle computes a function chosen at random from the set of all functions with domain and range I_n.

Similarly, it may be the case that an adversary has access to multiple oracles, all of which are drawn from the same family. For example, if we deal with oracles chosen from the family Perm^n, we could conceive of giving oracle access to the permutations $f, f^{-1} \in \mathsf{Perm}^n$. These kinds of scenarios apply when we talk about attacks on block ciphers, and we discuss them in much greater detail in the subsequent chapters of this thesis.

We also remark that oracles need not simply represent deterministic functions. Instead there could be some degree of randomness in their answers. In this case the oracle's output is determined by the input together with some internal coin tosses. Both deterministic and randomized oracles are used in this thesis.

We can also consider modeling our adversaries as traditional Turing machines (uniform model) or as circuits (non-uniform model). A Turing Machine would be equipped with a special oracle tape and instructions for making an oracle query, and a circuit would come with a special oracle gate. For our purposes, the choice is immaterial. The results we attain for one model can be translated for the other models. For a more in-depth discussion on these various models of computation, see the comprehensive text on complexity theory by Papadimitriou [111].

2.5 Pseudorandom Functions and Block Ciphers

The notion of pseudorandomness is extremely important since we use it to model the security of block ciphers. The pseudorandomness of a keyed function family $\mathcal{F}$ with domain $\mathcal{D}$ and range $\mathcal{R}$ captures its computational indistinguishability from $\mathsf{Rand}^{\mathcal{D}\to\mathcal{R}}$. That is, the "more" pseudorandom a function family $\mathcal{F}$, the "harder" it is for an adversary to distinguish between a function chosen randomly from $\mathcal{F}$ and a function chosen randomly from $\mathsf{Rand}^{\mathcal{D}\to\mathcal{R}}$.

Computational indistinguishability of two function families is captured by observing the average behavior of some adversary $\mathcal{A}$ when its oracle queries are answered by a function drawn from one family versus when its oracle queries are answered by a function drawn from the other family. In particular, we define the advantage the adversary has in distinguishing between two finite function families as follows:

Definition 1 *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two finite function families which both have domain $\mathcal{D}$ and range $\mathcal{R}$. Let $\mathcal{A}$ be a 1-oracle adversary. Then, the advantage of $\mathcal{A}$ in distinguishing between $\mathcal{F}_1$ and $\mathcal{F}_2$ is defined by*

$$\mathsf{Adv}_{\mathcal{A}}(\mathcal{F}_1, \mathcal{F}_2) = \Pr[a \overset{R}{\leftarrow} \mathsf{Keys}(\mathcal{F}_1) : \mathcal{A}^{f_a} = 1] - \Pr[a \overset{R}{\leftarrow} \mathsf{Keys}(\mathcal{F}_2) : \mathcal{A}^{f_a} = 1].$$

For any integers $q, t \geq 0$, we define an insecurity function:

$$\mathsf{Adv}_{\mathcal{F}_2}^{\mathcal{F}_1}(q, t) = \max_{\mathcal{A}}\{\mathsf{Adv}_{\mathcal{A}}(\mathcal{F}_1, \mathcal{F}_2)\},$$

where the above maximum is taken over choices of adversary $\mathcal{A}$ such that

- *$\mathcal{A}$ makes at most q queries to its oracle.*
- *The running time of $\mathcal{A}$, plus the time necessary to select the key a, and answer $\mathcal{A}$'s queries, is at most t.*

Definition 2 *We say that $\mathcal{F}_1$ is a (t, q, ϵ) indistinguishable from $\mathcal{F}_2$ if*

$$\mathsf{Adv}_{\mathcal{F}_2}^{\mathcal{F}_1}(q, t) \leq \epsilon.$$

The above definitions easily extend to the case of multiple-oracle adversaries. We explain the intuition behind these definitions after we discuss the concept of the pseudorandomness of a function family, which involves special cases of the above definitions.

We also remark that traditionally, in an asymptotic or complexity-theoretic analysis, the advantage is defined by taking the absolute value of the difference of the above probabilities (see, for example, Oded Goldreich's text [60]). This approach is intuitive since the advantage is then always a positive number. We, on the other hand, focus on the concrete security-theoretic treatment and do not use absolute values. The definition of advantage we use appears in a number of papers [14, 11]. For the concrete-security case we can ignore absolute value signs. In particular, for every adversary whose advantage is negative, we can construct an adversary whose advantage is positive; namely, consider the adversary whose responses are the exact opposite of another adversary. We can make this simplification since we are dealing with a fixed size function family, and hence a fixed value for the security parameter. Thus we only have to worry about a single adversary. In the asymptotic case, it is a little more difficult to remove the absolute value signs since primitives are defined as families of families; that is, one defines a family for each value of the security parameter. Thus one must define an adversary for each value of the security parameter. For example, in the uniform case one must deal with all values for the security parameters via a Turing machine that represents an infinite family of adversaries, so it is more difficult to get rid of the absolute value signs. In the non-uniform case, it is less of a concern since we essentially have one circuit (adversary) for every value of the security parameter.

It turns out that incorporating absolute value signs actually makes a small difference in the analysis of the block cipher constructions we propose in this dissertation; namely, the adversary's advantage increases by an additive factor of 2^{-2n}. This phenomenon occurs since certain terms do not cancel when absolute value signs are used. Since the difference in the actual analysis is minor, and given the reasoning mentioned above, we choose to ignore the absolute value signs.

We now move on to discussing the concept of pseudorandomness. The following definitions are based on ideas due to Goldreich, Goldwasser, and Micali [62]:

Definition 3 *Let $\mathcal{F}$ be a keyed function family with domain $\mathcal{D}$ and range $\mathcal{R}$. Let $\mathcal{A}$ be a 1-oracle adversary. Then we define $\mathcal{A}$'s advantage in distinguishing between $\mathcal{F}$ and* $\mathsf{Rand}^{\mathcal{D}\to\mathcal{R}}$ *as*

$$\mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(\mathcal{A}) = \Pr[a \overset{R}{\leftarrow} \mathsf{Keys}(\mathcal{F}) : \mathcal{A}^{f_a} = 1] - \Pr[f \overset{R}{\leftarrow} \mathsf{Rand}^{\mathcal{D}\to\mathcal{R}} : \mathcal{A}^{f} = 1].$$

For any integers $q, t \geq 0$, we define an insecurity function $\mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(q,t)$*:*

$$\mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(q,t) = \max_{\mathcal{A}}\{\mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(\mathcal{A})\},$$

where the above maximum is taken over choices of adversary $\mathcal{A}$ such that:

- *$\mathcal{A}$ makes at most q queries to its oracle.*
- *The running time of $\mathcal{A}$, plus the time necessary to select the key a, and answer $\mathcal{A}$'s queries, is at most t.*

Definition 4 *We say that $\mathcal{F}$ is a (t, q, ϵ)-secure pseudorandom function family if*

$$\mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(q,t) \leq \epsilon.$$

Let us examine the above definitions to gain more intuition. If $\mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(\mathcal{A}) = 0$, then $\mathcal{F}$ behaves exactly like $\mathsf{Rand}^{\mathcal{D}\rightarrow\mathcal{R}}$ from the adversary's point of view. Similarly, if $\mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(q,t) = 0$, then an adversary restricted to making q queries, and using at most t time steps cannot tell the difference between a function chosen from $\mathcal{F}$ and one chosen from $\mathsf{Rand}^{\mathcal{D}\rightarrow\mathcal{R}}$. If instead, $\mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(q,t) = \epsilon$, where ϵ is extremely small, then the adversary's ability to tell the difference between functions chosen from these two families is as small. We use the term pseudorandom function loosely to mean a function, drawn from a (t, q, ϵ)-secure finite family of functions, for "small" values of ϵ, and "large" values of t and q. Of course, in many ways, what we mean by "small" and "large" is relative, and depends on the particular application at hand. For the purposes of this dissertation, it is immaterial.

Goldreich, Goldwasser, and Micali [62] give a construction of a good pseudorandom function family based on the existence of any pseudorandom number generator. Subsequently, Hastad, Imagliazzo, Levin, and Luby show how to construct pseudorandom number generators from any one-way function [67], thus we can construct good pseudorandom function families under minimal cryptographic assumptions.

We are now ready to define a notion of security for block ciphers. This notion was first described by Luby and Rackoff [88]. The notion captures the pseudorandomness of a permutation family on $\mathcal{D}$ in terms of its indistinguishability from $\mathsf{Perm}^{\mathcal{D}}$, where the adversary is given oracle access to the forward direction of the permutation. If the adversary is given access to only the forward direction of the permutation, then he can receive the ciphertext corresponding to any plaintext of his choice. In this case, the adversary is given the ability to mount an adaptive chosen plaintext attack. A block cipher that cannot be distinguished from a random permutation against this type of attack is termed a *pseudorandom* *permutation.* More formally:

Definition 5 *Let $\mathcal{F}$ be a keyed permutation family with domain and range $\mathcal{D}$. Let $\mathcal{A}$ be a 1-oracle adversary. Then we say that $\mathcal{A}$ is an ϵ pseudorandom permutation distinguisher for $\mathcal{F}$ if*

$$\mathsf{Adv}^{\mathsf{prp}}_{\mathcal{F}}(\mathcal{A}) = \Pr[a \xleftarrow{R} \mathsf{Keys}(\mathcal{F}) : \mathcal{A}^{f_a} = 1] - \Pr[f \xleftarrow{R} \mathsf{Perm}^{\mathcal{D}} : \mathcal{A}^{f} = 1] \leq \epsilon.$$

For any integers $q, t \geq 0$, we define an insecurity function $\mathsf{Adv}^{\mathsf{prp}}_{\mathcal{F}}(q,t)$ similar to the one in definition 3:

$$\mathsf{Adv}^{\mathsf{prp}}_{\mathcal{F}}(q,t) = \max_{\mathcal{A}}\{\mathsf{Adv}^{\mathsf{prp}}_{\mathcal{F}}(\mathcal{A})\},$$

where the above maximum is taken over adversaries $\mathcal{A}$ who make at most q oracle queries, and whose running time, together with the time necessary to select the key a, and answer $\mathcal{A}$'s queries, is at most t.

Definition 6 *We say that $\mathcal{F}$ is a (t, q, ϵ)-secure pseudorandom permutation family if*

$$\mathsf{Adv}^{\mathsf{prp}}_{\mathcal{F}}(q,t) \leq \epsilon.$$

Now, since we are dealing with permutations, we can imagine that the adversary also has access to an oracle that computes the reverse or inverse direction of the permutation on any input. Having access to this direction of the permutation can be viewed as having an oracle which returns the plaintext corresponding to a particular ciphertext input. If the adversary has access to oracles which compute both directions of the permutation, then he is capable of mounting an interleaved adaptive chosen plaintext and ciphertext attack. A block cipher that is secure against this type of attack is termed a *super pseudorandom permutation.* We now give a more formal definition of the super pseudorandomness of a permutation family:

Definition 7 *Let $\mathcal{F}$ be a keyed permutation family with domain and range $\mathcal{D}$. Let $\mathcal{A}$ be a 2-oracle adversary. Then we say that $\mathcal{A}$ is an ϵ super pseudorandom permutation distinguisher for $\mathcal{F}$ if*

$$\mathsf{Adv}^{\mathsf{sprp}}_{\mathcal{F}}(\mathcal{A}) = \Pr[a \xleftarrow{R} \mathsf{Keys}(\mathcal{F}) : \mathcal{A}^{f_a, f_a^{-1}} = 1] - \Pr[f \xleftarrow{R} \mathsf{Perm}^{\mathcal{D}} : \mathcal{A}^{f, f^{-1}} = 1].$$

For any integers $q, t \geq 0$, we define an insecurity function $\mathsf{Adv}_F^{\mathsf{sprp}}(q,t)$ similar to the one in definition 3:

$$\mathsf{Adv}_{\mathcal{F}}^{\mathsf{sprp}}(q,t) = \max_{\mathcal{A}}\{\mathsf{Adv}_{\mathcal{F}}^{\mathsf{sprp}}(\mathcal{A})\},$$

where the above maximum is taken over adversaries $\mathcal{A}$ who make at most q oracle queries, and whose running time, together with the time necessary to select the key a, and answer $\mathcal{A}$'s queries, is at most t.

Definition 8 *We say that $\mathcal{F}$ is a (t, q, ϵ)-secure super pseudorandom permutation family if*

$$\mathsf{Adv}_{\mathcal{F}}^{\mathsf{sprp}}(q,t) \leq \epsilon.$$

In this thesis, we focus on adaptive interleaved chosen plaintext and ciphertext attacks. Thus we prove security of our block cipher constructions by showing that they are (t, q, ϵ)-secure super pseudorandom permutation families for the appropriate values of the parameters t, q, and ϵ. Finally, we remark that there are a number of examples of permutation families which are pseudorandom, but not super pseudorandom. One example, as we shall later see, is the original *three-round* Luby-Rackoff cipher.

2.6 Transcripts

The notion of a transcript is central to many of the mathematical arguments made in this thesis. The *transcript* records all of the communication between the adversary and its oracle. If a total of q queries are made to the oracle, then the transcript is of the form

$$\sigma = \langle Q_1, Q_2, \ldots, Q_q \rangle,$$

where each Q_i is a representation of the i^{th} query made to the oracle, together with the oracle's response. For example, Q_i might be of the form (x, y) where x is the input to the oracle and y is the oracle's answer. It is often convenient to talk about some prefix of the transcript sequence. Thus, we denote by $\sigma^{(i)}$ the sequence of the first i elements in the transcript σ: $Q_1, \ldots, Q_i$. In addition, we consider the notion of a function C which can determine the adversary's next query given the previous queries and answers.

Definition 9 *Let $C_{\mathcal{A}}[\sigma^{(i-1)}]$ denote the i^{th} query $\mathcal{A}$ makes to an oracle as a function of* *the first $i-1$ query-answer pairs in $\mathcal{A}$'s transcript. Here $1 \leq i \leq q$, and we assume that $\mathcal{A}$ makes exactly q queries. Let $C_{\mathcal{A}}[\sigma^{(q)}]$ denote the output that $\mathcal{A}$ would give as a function of the transcript. If the transcript is invalid for some reason (e.g. the inputs do not constitute proper queries or responses) then the output is undefined.*

If the underlying oracle is randomized, then the function C may also be randomized. We can modify the above definitions to handle this case. For the purposes of this thesis, however, we only need to consider deterministic C's. We also need to consider the notion of which transcripts can possibly be generated.

Definition 10 *Suppose $\sigma = \langle Q_1, \ldots, Q_q \rangle$ is a sequence. Then, σ is said to be a possible $\mathcal{A}$-transcript if it could be generated by an interaction of $\mathcal{A}$ with its oracle. In particular, for all $1 \leq i \leq q$, the i^{th} query made by $\mathcal{A}$ (as determined from σ) is*

$$C_{\mathcal{A}}[\langle Q_1, Q_2, \ldots, Q_{i-1} \rangle].$$

Also, if we have an adversary $\mathcal{A}$ with access to a set of oracles $\mathcal{O}$, then we often let the random variable $T_{\mathcal{O}}$ denote the transcript seen by $\mathcal{A}$ when its oracle queries are answered by $\mathcal{O}$. We also employ the shorthand $C_{\mathcal{A}}(\mathcal{O})$ to denote $C_{\mathcal{A}}(T_{\mathcal{O}})$.

2.7 Relating Transcripts and Computational Advantage

Since transcripts describe the adversary's interaction with its oracles, it should come as little surprise that we can bound the adversary's advantage in distinguishing between two function families in terms of probabilities related to the transcripts generated from these interactions. The following theorem is a slightly more generalized variant of one that has appeared in various forms in the literature [101, 114, 115, 121].

Theorem 1 *Suppose $\mathcal{A}$ is a deterministic oracle machine and let $\mathcal{O}_1$ and $\mathcal{O}_2$ be random variables denoting two possible oracles that $\mathcal{A}$ can use. Let $T_{\mathcal{O}_i}$ be a random variable denoting the transcript of $\mathcal{A}$'s interaction with oracle $\mathcal{O}_i$ (for $i = 1, 2$). Let P be a particular property (i.e. a boolean predicate that takes on the value 1 when the property is satisfied, and 0 otherwise) on the set of possible transcripts. Now, if all transcripts generated by $\mathcal{A}^{\mathcal{O}_1}$ have property P then:*

$$\mathsf{Adv}(\mathcal{O}_1,\mathcal{O}_2) \le \sum_{\sigma\in\Gamma} \Pr_{\mathcal{O}_1}[T_{\mathcal{O}_1}=\sigma] - \Pr_{\mathcal{O}_2}[T_{\mathcal{O}_2}=\sigma],$$

where Γ *is the set of all possible transcripts* σ *satisfying* P *and for which* $C_{\mathcal{A}}(\sigma)=1$

Proof: Let $\mathcal{T}$ denote all possible transcripts. Then:

$$\begin{aligned}
\mathsf{Adv}(\mathcal{O}_1,\mathcal{O}_2) &= \Pr_{\mathcal{O}_1}[\mathcal{A}^{\mathcal{O}_1}=1] - \Pr_{\mathcal{O}_2}[\mathcal{A}^{\mathcal{O}_2}=1]\\
&= \Pr_{\mathcal{O}_1}[C_{\mathcal{A}}(\mathcal{O}_1)=1] - \Pr_{\mathcal{O}_2}[C_{\mathcal{A}}(\mathcal{O}_2)=1]\\
&= \sum_{\sigma\in\mathcal{T}}\Big(\Pr_{\mathcal{O}_1}[C_{\mathcal{A}}(\mathcal{O}_1)=1 \mid T_{\mathcal{O}_1}=\sigma]\cdot\Pr_{\mathcal{O}_1}[T_{\mathcal{O}_1}=\sigma]\\
&- \Pr_{\mathcal{O}_2}[C_{\mathcal{A}}(\mathcal{O}_2)=1 \mid T_{\mathcal{O}_2}=\sigma]\cdot\Pr_{\mathcal{O}_2}[T_{\mathcal{O}_2}=\sigma]\Big)\\
&= \sum_{\sigma\in\mathcal{T}} \Pr_{\mathcal{O}_1}[C_{\mathcal{A}}(\sigma)=1]\cdot\Pr_{\mathcal{O}_1}[T_{\mathcal{O}_1}=\sigma] - \Pr_{\mathcal{O}_2}[C_{\mathcal{A}}(\sigma)=1]\cdot\Pr_{\mathcal{O}_2}[T_{\mathcal{O}_2}=\sigma]\\
&= \sum_{\sigma\in\Gamma} \Pr[C_{\mathcal{A}}(\sigma)=1]\cdot\Pr_{\mathcal{O}_1}[T_{\mathcal{O}_1}=\sigma] - \Pr[C_{\mathcal{A}}(\sigma)=1]\cdot\Pr_{\mathcal{O}_2}[T_{\mathcal{O}_2}=\sigma]\\
&+ \sum_{\sigma\in\Gamma^C} \Pr[C_{\mathcal{A}}(\sigma)=1]\cdot\Pr_{\mathcal{O}_1}[T_{\mathcal{O}_1}=\sigma] - \Pr[C_{\mathcal{A}}(\sigma)=1]\cdot\Pr_{\mathcal{O}_2}[T_{\mathcal{O}_2}=\sigma].
\end{aligned}$$

The derivation of the last equality from the previous requires a fairly subtle observation; namely, when we *fix* a particular transcript σ, then $C_{\mathcal{A}}(\sigma)$ is always either 0 or 1 since $\mathcal{A}$ is assumed to be deterministic in the theorem statement. As a result:

$$\Pr_{\mathcal{O}_1}[C_{\mathcal{A}}(\sigma)=1] = \Pr_{\mathcal{O}_2}[C_{\mathcal{A}}(\sigma)=1]$$

since neither probability actually *depends* on the choice of the oracle. We proceed:

$$\begin{aligned}
&\sum_{\sigma\in\Gamma} \Pr[C_{\mathcal{A}}(\sigma)=1]\cdot\Pr_{\mathcal{O}_1}[T_{\mathcal{O}_1}=\sigma] - \Pr[C_{\mathcal{A}}(\sigma)=1]\cdot\Pr_{\mathcal{O}_2}[T_{\mathcal{O}_2}=\sigma]\\
+&\sum_{\sigma\in\Gamma^C} \Pr[C_{\mathcal{A}}(\sigma)=1]\cdot\Pr_{\mathcal{O}_1}[T_{\mathcal{O}_1}=\sigma] - \Pr[C_{\mathcal{A}}(\sigma)=1]\cdot\Pr_{\mathcal{O}_2}[T_{\mathcal{O}_2}=\sigma]\\
=&\sum_{\sigma\in\Gamma}\Big(\Pr_{\mathcal{O}_1}[T_{\mathcal{O}_1}=\sigma] - \Pr_{\mathcal{O}_2}[T_{\mathcal{O}_2}=\sigma]\Big)\cdot\Pr[C_{\mathcal{A}}(\sigma)=1]
\end{aligned}$$

$$
\begin{aligned}
&+ \sum_{\sigma \in \Gamma^C} \left(\Pr_{\mathcal{O}_1}[T_{\mathcal{O}_1} = \sigma] - \Pr_{\mathcal{O}_2}[T_{\mathcal{O}_2} = \sigma] \right) \cdot \Pr[C_{\mathcal{A}}(\sigma) = 1] \\
&= \sum_{\sigma \in \Gamma} \left(\Pr_{\mathcal{O}_1}[T_{\mathcal{O}_1} = \sigma] - \Pr_{\mathcal{O}_2}[T_{\mathcal{O}_2} = \sigma] \right) && (2.1) \\
&+ \sum_{\sigma \in \Gamma^C} \left(\Pr_{\mathcal{O}_1}[T_{\mathcal{O}_1} = \sigma] - \Pr_{\mathcal{O}_2}[T_{\mathcal{O}_2} = \sigma] \right) \cdot \Pr[C_{\mathcal{A}}(\sigma) = 1]. && (2.2)
\end{aligned}
$$

The last equality follows from the previous, by definition, since $C_{\mathcal{A}}(\sigma) = 1$ whenever $\sigma \in \Gamma$. Now, if a given transcript σ is in the set-theoretic complement Γ^C, then exactly one of the following cases must occur:

1. $C_{\mathcal{A}}(\sigma) \neq 1$, or
2. $C_{\mathcal{A}}(\sigma) = 1$ and σ does not satisfy property P.

To complete the proof, it suffices to show that the value attained in equation 2.2 is never positive, in which case we are done since equation 2.1 is the upper bound we are trying to achieve. This fact follows from the following two observations:

1. If $C_{\mathcal{A}}(\sigma) \neq 1$ then $\Pr[C_{\mathcal{A}}(\sigma) = 1] = 0$.
2. If a transcript σ does not satisfy property P, then $\Pr[T_{\mathcal{O}_1} = \sigma] = 0$ since all possible transcripts generated by $\mathcal{A}^{\mathcal{O}_1}$ satisfy property P according to the theorem statement.

Thus, none of the terms in the summation are positive, which concludes the proof. ■

We now state another useful general theorem regarding the relationship between transcripts and indistinguishability. Like before, the theorem slightly generalizes and unifies theorems that have appeared in various forms in the literature [101, 114, 115, 121]. Roughly speaking, the theorem gives a bound on the distinguishing advantage with respect to probabilities associated with a transcript appearing in a given set. Here the set itself is defined by the particular choice of one of the oracles for the function families we wish to distinguish. By the term particular choice, we mean a specific function picked according to an underlying distribution; that is, if the oracle represents a function family, then we pick a single function from that family according to some underlying distribution. Here is the formal theorem statement:

Theorem 2 *Let $\mathcal{A}$, $\mathcal{O}_1$, $\mathcal{O}_2$, $T_{\mathcal{O}_1}$, $T_{\mathcal{O}_2}$, and Γ take on the same roles as in the previous theorem. Let $\mathsf{S}(\mathcal{O}_1)$ be a particular set of transcripts, where $\mathsf{S}(\mathcal{O}_1)$ depends on the choice of the specific function that $\mathcal{O}_1$ instantiates. Let σ^* denote the transcript which has the best chance of being in $\mathsf{S}(\mathcal{O}_1)$. That is,*

$$\sigma^* = argmax_\sigma \Pr_{\mathcal{O}_1}[\sigma \in \mathsf{S}(\mathcal{O}_1)].$$

Then

$$\begin{aligned}\mathsf{Adv}(\mathcal{O}_1, \mathcal{O}_2) &\leq \sum_{\sigma\in\Gamma} \left(\Pr_{\mathcal{O}_1}[T_{\mathcal{O}_1} = \sigma \mid \sigma \notin \mathsf{S}(\mathcal{O}_1)] - \Pr_{\mathcal{O}_2}[T_{\mathcal{O}_2} = \sigma] \right) \cdot \Pr_{\mathcal{O}_1}[\sigma \notin \mathsf{S}(\mathcal{O}_1)] \\ &+ \Pr_{\mathcal{O}_1}[\sigma^* \in \mathsf{S}(\mathcal{O}_1)].\end{aligned}$$

Proof: The proof follows by first applying theorem 1 and then performing some arithmetic manipulation. As before, let Γ be the set of all possible transcripts σ satisfying P and for which $C_{\mathcal{A}}(\sigma) = 1$.

$$\begin{aligned}\mathsf{Adv}(\mathcal{O}_1, \mathcal{O}_2) &\leq \sum_{\sigma\in\Gamma} \Pr_{\mathcal{O}_1}[T_{\mathcal{O}_1} = \sigma] - \Pr_{\mathcal{O}_2}[T_{\mathcal{O}_2} = \sigma] \\ &= \sum_{\sigma\in\Gamma} \left(\Pr_{\mathcal{O}_1}[T_{\mathcal{O}_1} = \sigma \mid \sigma \in \mathsf{S}(\mathcal{O}_1)] \cdot \Pr[\sigma \in \mathsf{S}(\mathcal{O}_1)] \right. \\ &+ \Pr_{\mathcal{O}_1}[T_{\mathcal{O}_1} = \sigma \mid \sigma \notin \mathsf{S}(\mathcal{O}_1)] \cdot \Pr[\sigma \notin \mathsf{S}(\mathcal{O}_1)] \\ &- \left. \Pr_{\mathcal{O}_2}[T_{\mathcal{O}_2} = \sigma] \cdot \left(\Pr_{\mathcal{O}_1}[\sigma \in \mathsf{S}(\mathcal{O}_1)] + \Pr_{\mathcal{O}_1}[\sigma \notin \mathsf{S}(\mathcal{O}_1)] \right) \right) \\ &= \sum_{\sigma\in\Gamma} \left(\Pr_{\mathcal{O}_1}[T_{\mathcal{O}_1} = \sigma \mid \sigma \notin \mathsf{S}(\mathcal{O}_1)] - \Pr_{\mathcal{O}_2}[T_{\mathcal{O}_2} = \sigma] \right) \cdot \Pr_{\mathcal{O}_1}[\sigma \notin \mathsf{S}(\mathcal{O}_1)] \\ &+ \sum_{\sigma\in\Gamma} \left(\Pr_{\mathcal{O}_1}[T_{\mathcal{O}_1} = \sigma \mid \sigma \in \mathsf{S}(\mathcal{O}_1)] - \Pr_{\mathcal{O}_2}[T_{\mathcal{O}_2} = \sigma] \right) \cdot \Pr_{\mathcal{O}_1}[\sigma \in \mathsf{S}(\mathcal{O}_1)].\end{aligned}$$

We complete the proof by obtaining the appropriate bound for the last expression:

$$\begin{aligned}&\sum_{\sigma\in\Gamma} \left(\Pr_{\mathcal{O}_1}[T_{\mathcal{O}_1} = \sigma \mid \sigma \in \mathsf{S}(\mathcal{O}_1)] - \Pr_{\mathcal{O}_2}[T_{\mathcal{O}_2} = \sigma] \right) \cdot \Pr_{\mathcal{O}_1}[\sigma \in \mathsf{S}(\mathcal{O}_1)] \\ &\qquad \leq \Pr_{\mathcal{O}_1}[\sigma^* \in \mathsf{S}(\mathcal{O}_1)] \cdot \sum_{\sigma\in\Gamma} \Pr_{\mathcal{O}_1}[T_{\mathcal{O}_1} = \sigma \mid \sigma \in \mathsf{S}(\mathcal{O}_1)] \\ &\qquad \leq \Pr_{\mathcal{O}_1}[\sigma^* \in \mathsf{S}(\mathcal{O}_1)],\end{aligned}$$

where the last equation follows from the previous since the sum of the probabilities of all the events in a particular sample space is at most 1. We thus get the desired bound. ■

2.8 Hash Functions

In many of the constructions given in this dissertation, we utilize particular finite function families with specific statistical properties of interest. The families we consider are called *universal hash function* families. We stress that such universal hash function families should not be confused with *cryptographic* hash functions such as MD5 [123] or the SHA family [102, 103]. The definitions of the universal hash function families we give follow those given in a number of works [36, 114, 142, 80, 128]. Stinson prepared a very nice note outlining the history of these definitions [138]. We review this history below.

In the following definitions, we assume H is a keyed function family with domain $\mathcal{D}$ and range $\mathcal{R}$. Moreover, we assume that $\mathcal{R}$ can be viewed as a group with additive notation ('+' and '-'). Finally, for the definitions below, we require that $\epsilon_1, \ldots, \epsilon_6$ are all greater than or equal to $1/|\mathcal{R}|$.

We start with the concept of universal hash function families, which are designed so that, on average, a function picked at random from the family will rarely map distinct elements in the domain to the same element in the range.

Definition 11 *We say that H is an ϵ_1-universal family of hash functions if for all $x \neq y \in \mathcal{D}$,*

$$\Pr[a \overset{R}{\leftarrow} \mathsf{Keys}(H) : h_a(x) = h_a(y)] \leq \epsilon_1.$$

The concept of a universal hash function is due to Carter and Wegman [36], and it was generalized to ϵ-universal by Stinson [137].

A *uniform* family of hash functions is a similar concept. Here, the function families should be designed so that no single fixed element gets mapped to another fixed element by a disproportionate number of functions chosen from the family.

Definition 12 *We say that H is an ϵ_2-uniform family of hash functions if for all $x \in \mathcal{D}, z \in \mathcal{R}$,*

$$\Pr[a \overset{R}{\leftarrow} \mathsf{Keys}(H) : h_a(x) = z] \leq \epsilon_2.$$

We note that in some of the past literature, hash functions are assumed to be uniform by default. We prefer to separate uniformity from the other properties we consider.

We now consider Δ-universal families, in which no value appears as the difference of the images of two distinct domain elements disproportionately often.

Definition 13 *We say that H is an ϵ_3-almost-Δ-universal if for all $x \neq y \in \mathcal{D}, z \in \mathcal{R}$,*

$$\Pr[a \overset{R}{\leftarrow} \mathsf{Keys}(H) : h_a(x) - h_a(y) = z] \leq \epsilon_3.$$

The concept of Δ-universal hash function families is due to Stinson [137], who generalizes the definition of Krawczyk [80] to arbitrary Abelian groups. We remark that Δ-universal families are often called XOR-universal when the range is I_l, for some positive integer l, and the bit-wise exclusive-or is used as the subtraction operation. This latter term is due to Rogaway [128]; the original concept is due to Krawczyk [80], who calls such function families ϵ-OTP-secure (OTP stands for one-time pad).

Next, we consider strongly universal hash function families. These families have the property that images of any pair of distinct inputs are uniformly distributed among the range elements. Thus, the function appears to be a truly random function whenever one only sees two input/output pairs. We remark that the synonymous terms *pairwise-independent* or *2-universal* hash function families are often seen in the literature.

Definition 14 *H is an ϵ_4-strongly universal family of hash functions if for all $x \neq y \in \mathcal{D}$, and $z_1, z_2 \in \mathcal{R}$,*

$$\Pr[a \overset{R}{\leftarrow} \mathsf{Keys}(H) : h_a(x) = z_1, h_a(y) = z_2] \leq \epsilon_4/|R|.$$

Carter and Wegman [36] were the first to define strongly universal hash functions. The notion was subsequently generalized to ϵ-strongly universal by Stinson [137]. The next two types of hash function families are novel contributions of this thesis. They respectively appear in two papers by Patel, Ramzan, and Sundaram [114], [115]. The first is the bisymmetric family which requires that the sum of the images of two (possibly identical) elements under two randomly chosen hash functions, never takes on a particular value disproportionately often.

Definition 15 *We say that H is an ϵ_5-bisymmetric family of hash functions if for all $x, y \in \mathcal{D}$ (here we allow $x = y$), $z \in \mathcal{R}$,*

$$\Pr[a_1 \xleftarrow{R} \mathsf{Keys}(H), a_2 \xleftarrow{R} \mathsf{Keys}(H) : h_{a_1}(x) + h_{a_2}(y) = z] \leq \epsilon_5.$$

The last family we consider is the monosymmetric family. It requires that the sum of the images of two (possibly identical) elements under the *same* randomly chosen hash function never takes on a particular value disproportionately often. It is similar to the previous definition, with the exception that the previous definition considers two (possibly distinct) hash functions, whereas the next one only involves one such function.

Definition 16 *We say that H is an ϵ_6-monosymmetric family of hash functions if for all $x, y \in \mathcal{D}$ (here we allow $x = y$) and $z \in \mathcal{R}$,*

$$\Pr[a \xleftarrow{R} \mathsf{Keys}(H) : h_a(x) + h_a(y) = z] \leq \epsilon_6.$$

An example of a family that has all four properties for $\epsilon_1 = \cdots = \epsilon_6 = 1/|\mathcal{R}|$ is a family keyed by a random pair $a = (a_1, a_2)$ with $a_1 \in \mathbb{Z}_p^*$, $a_2 \in \mathbb{Z}_p$, and $h_a(x) = a_1 x + a_2 \bmod p$ where p is a prime, $\mathbb{Z}_p^*$ is the multiplicative group modulo p, and $\mathbb{Z}_p$ is the additive group modulo p. Similarly, the affine transformation $ax + b$ where $a, b \in \mathrm{GF}(2^n)$, and $a \neq 0$ works as well. Another example that satisfies all of the above properties, except for strong universality, for fairly small values for ϵ, is the square hash family which we discuss in great detail in chapter 6.

We remark that we use the phrase "h is a universal (respectively uniform, Δ-universal, strongly universal, bisymmetric, monosymmetric) hash function" to mean "h is drawn from a universal (respectively uniform, Δ-universal, strongly universal, bisymmetric, monosymmetric) family of hash functions."

2.9 Constructions of Luby-Rackoff Ciphers

We now formally define Feistel ladders which are the main tool for constructing pseudo-random permutations on $2n$-bit strings from length-preserving functions on n-bits strings. Recall that we discuss this notion in chapter 1. Feistel ladders have been used in a large number of popular block cipher constructions such as DES. To start with, we define the basic Feistel permutation (see figure 1-4, in chapter 1).

Definition 17 (Basic Feistel Permutation) *Let f be a mapping from I_n to I_n. Let $x = (x^L, x^R)$ with $x^L, x^R \in I_n$. We denote by $\overline{f}$ the permutation on I_{2n} defined as $\overline{f}(x) = (x^R, x^L \oplus f(x^R))$. Note that it is a permutation because $\overline{f^{-1}}(y) = (y^R \oplus f(y^L), y^L)$, which can be computed if the function f is known.*

The Feistel ladder involves chaining several basic Feistel permutations together (see figure 2-1).

Definition 18 (Feistel Ladder) *If $f_1, \ldots, f_s$ are mappings with domain and range I_n, then we denote by $\Psi(f_1, \ldots, f_s)$ the permutation on I_{2n} defined as $\Psi(f_1, \ldots, f_s) = \overline{f_s} \circ \cdots \circ \overline{f_1}$*

Observe that a Feistel ladder is invertible since it is simply a composition of basic Feistel permutations. Sometimes we refer to Feistel ladders as *Feistel networks.* There is a natural "round" structure to these Feistel ladders. In particular, one can visualize that the plaintext input to the Feistel ladder is transformed via a series of individual rounds, where each round consists of the basic Feistel permutation $\overline{f_i}$. We discuss this round structure in more detail in chapter 5.

Luby and Rackoff [88] construct a pseudorandom permutation using three independently-keyed pseudorandom functions in a Feistel ladder. The main theorem in their paper is:

Theorem 3 (Luby-Rackoff PRP) *Let h_1, f_1, f_2 be independently-keyed functions from a keyed function family $\mathcal{F}$ with domain and range I_n and key space I_s. Let $\mathcal{B}$ be the family of permutations on I_{2n} with key space I_{3s} consisting of permutations of the form $P = \Psi(h_1, f_1, f_2)$ (the key for an element of $\mathcal{B}$ is simply the concatenation of keys for h_1, f_1, f_2). Then*

$$\mathsf{Adv}^{\mathsf{prp}}_{\mathcal{B}}(q, t) \le \mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(q, t + O(2s + 2q(t_f + n))) + \binom{q}{2}\left(2^{-n+1} + 2^{-2n}\right)$$

where t_f is the worst case time it takes to compute the value of a function from $\mathcal{F}$ on a given input.

We remark that in the original Luby-Rackoff paper, the main theorem statement is given in terms of complexity-theoretic security; we have recast the statement to the concrete

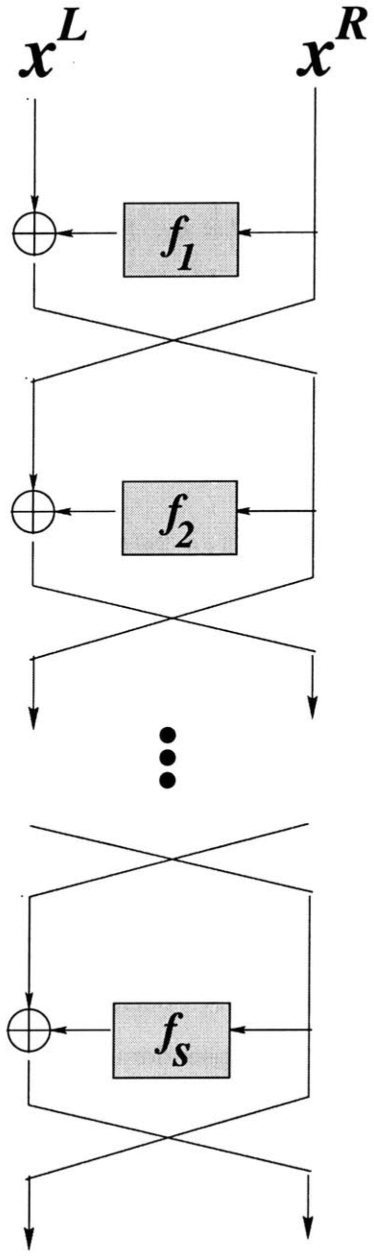

Figure 2-1: The s-round Feistel Ladder $\Psi(f_1, \ldots, f_s)$.

security setting. The same remark applies for the theorems due to Naor and Reingold which we describe shortly.

In their paper, Luby and Rackoff show that their three-round construction is not super pseudorandom by giving a very efficient distinguishing attack. We give variants on this attack, generalized to different settings, in chapters 4 and 5. Luby and Rackoff did observe, however, that by adding an extra round to their construction, they could achieve super pseudorandomness:

Theorem 4 (Luby-Rackoff SPRP) *Let h_1, f_1, f_2, h_2 be independently-keyed functions chosen from a keyed function family $\mathcal{F}$ with domain and range I_n and key space I_s. Let $\mathcal{B}$ be the family of permutations on I_{2n} with key space I_{4s} consisting of permutations of the form $P = \Psi(h_1, f_1, f_2, h_2)$ (the key for an element of $\mathcal{B}$ is simply the concatenation of keys for h_1, f_1, f_2, h_2). Then*

$$\mathsf{Adv}^{\mathsf{sprp}}_{\mathcal{B}}(q,t) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(q, t + O(3s + 3q(t_f + n))) + \binom{q}{2}\left(2^{-n+1} + 2^{-2n}\right)$$

where t_f is the worst case time it takes to compute the value of a function from $\mathcal{F}$ on a given input.

Naor and Reingold [101] optimize the above construction by removing the first and last rounds in the Feistel ladder, and replacing them with strongly universal hash functions (see figure 2-2). Of course, in this setting, the underlying hash function family must consist of permutations, in order for the cipher to be invertible. Their construction is more efficient since universal hash functions only involve specific statistical properties, so can typically be implemented much faster than good pseudorandom functions. Also, we can construct such universal hash functions without making any cryptographic assumption or conjecture. By reducing the number of pseudorandom function invocations from four to two, Naor and Reingold achieve a significant savings.

Theorem 5 (Naor-Reingold) *Let f_1 and f_2 be independently-keyed functions from a keyed function family $\mathcal{F}$ with domain and range I_n and key space I_{s_1}. Let h_1, h_2 be strongly-universal hash functions, keyed independently of each other and of f_1, f_2, from a keyed permutation family H with domain and range I_n and key space I_{s_2}. Let $\mathcal{B}$ be the family of permutations on I_{2n} with key space $I_{2s_1+2s_2}$ consisting of permutations of the form*

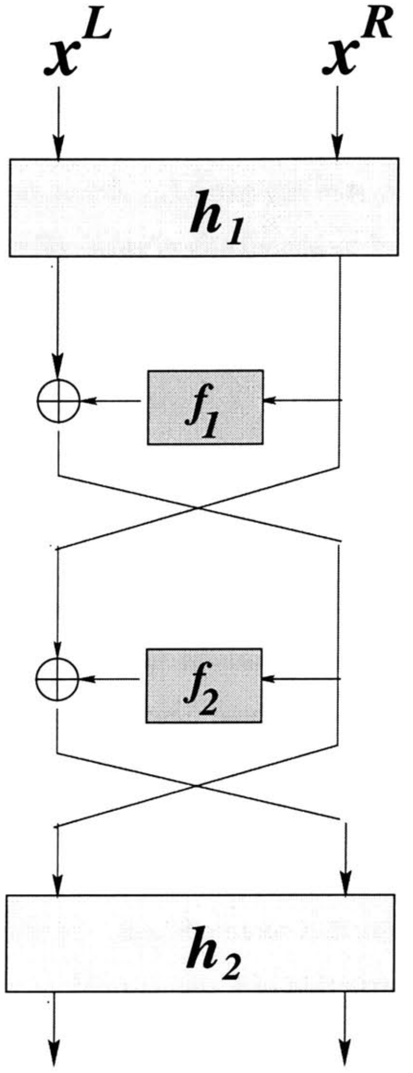

Figure 2-2: The Naor-Reingold construction $h_2^{-1} \circ \Psi(f_1, f_2) \circ h_1$.

$P = h_2^{-1} \circ \Psi(f_1, f_2) \circ h_1$. Then

$$\mathsf{Adv}^{\mathsf{sprp}}_{\mathcal{B}}(q,t) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(q, t + O(s_1 + 2s_2 + q(2t_h + t_f))) + \binom{q}{2}\left(2^{-n+1} + 2^{-2n}\right)$$

where t_f is the worst case time it takes to compute the value of a function from $\mathcal{F}$ on a given input, and t_h is the worst case time it takes to compute the value of a function from H on a given input.

Patel, Ramzan, and Sundaram [114], following a suggestion by Naor and Reingold [101], optimize the construction further to use the same pseudorandom function in each of the middle two rounds, thus reducing the key size. They require an additional condition on the hash function. We present the following theorem in chapter 3 of the thesis:

Theorem 6 (Patel-Ramzan-Sundaram) *Let f be a function from a keyed function family $\mathcal{F}$ with domain and range I_n and key space I_{s_1}. Let h_1, h_2 be ϵ_1-bisymmetric ϵ_2-almost Δ-universal hash functions, keyed independently of each other and of f, from a keyed function family H with domain and range I_n and key space I_{s_2}. Let $\mathcal{B}$ be the family of permutations on I_{2n} with key space $I_{s_1+2s_2}$ consisting of permutations of the form $P = \Psi(h_1, f, f, h_2)$. Then*

$$\mathsf{Adv}_{\mathcal{B}}^{\mathsf{sprp}}(q,t) \leq \mathsf{Adv}_{\mathcal{F}}^{\mathsf{prf}}(2q, t + O(2s_2 + 2q(t_h + n))) + q^2\epsilon_1 + \binom{q}{2}\left(2\epsilon_2 + 2^{-2n}\right).$$

We gain two advantages from using bisymmetric Δ-universal hash functions. The first is that we can construct such functions without using any type of underlying cryptographic assumption. The second is that since these functions only require specific statistical properties, they can be implemented much more efficiently than pseudorandom functions. For example, the square hash, which we present in chapter 6 only requires a single modular squaring. Also, the modulus, which is a prime, can be constructed in a specific manner to help make the modular reduction more efficient. On the other hand, the most efficient known number-theoretic construction of a pseudorandom function, due to Naor and Reingold [99], requires a few full-length modular exponentiations, each of which requires several hundred modular multiplications.

Another approach to designing pseudorandom functions is via the use of cryptographic hash functions such as SHA-1 [102]. The use of such functions as PRFs is more dubious since there is no simple precise cryptographic assumption one can make to justify their claimed pseudorandomness. At the same time, such cryptographic hash functions are much more efficient than number-theoretic constructions. Yet, they are still still slower than well-designed universal hash functions. For example, on a particular implementation on an ARM7 processor with hand-optimized assembly language code, a single call of the square hash function is about six times faster than a single call to the SHA-1 compression function. The square hash function in this test has 160-bit inputs and outputs, and the SHA-1 compression function has a 512-bit input and 160-bit output. We stress, however, that one would need to use at least one call to the SHA-1 compression function to achieve a PRF with a 160-bit output.

The above result of Patel, Ramzan, and Sundaram gives the most optimal known Luby-Rackoff cipher under the traditional assumptions. In this dissertation we examine some of these assumptions, and establish additional results related to such ciphers.

We note that we will often need to talk about the intermediate stages of the computation in a Feistel ladder as a plaintext is transformed to a ciphertext (and vice-versa). We denote the right halves of the values attained, as each of the successive basic Feistel permutations is applied, by the letters S, T, V, and W respectively. In addition, we refer to the left half and right halves of the plaintext input to the cipher as L and R respectively. Similarly, we refer to the left and right halves of the ciphertext output as V and W respectively. Consider, for example, the Patel-Ramzan-Sundaram construction $\Psi(h_1, f, f, h_2)$ [114]. We can now describe it by the following equations (see figure 2-3):

$$\begin{aligned}
S &= L \oplus h_1(R); \\
T &= R \oplus f(S); \\
V &= S \oplus f(T); \\
W &= T \oplus h_2(V).
\end{aligned}$$

For example, in the above case, the input to the second round is (R, S); the output after the second round is (S, T).

2.10 Message Authentication Codes

The notion of message authentication codes appears at various points in this thesis. Recall that message authentication codes are a secret-key construct which allow one party to efficiently transmit a message to another party so that the following two properties are satisfied:

1. Authentication: Messages are indeed sent by the person who is purported to have sent them.

2. Integrity: Messages are not modified en-route from the sender to the recipient.

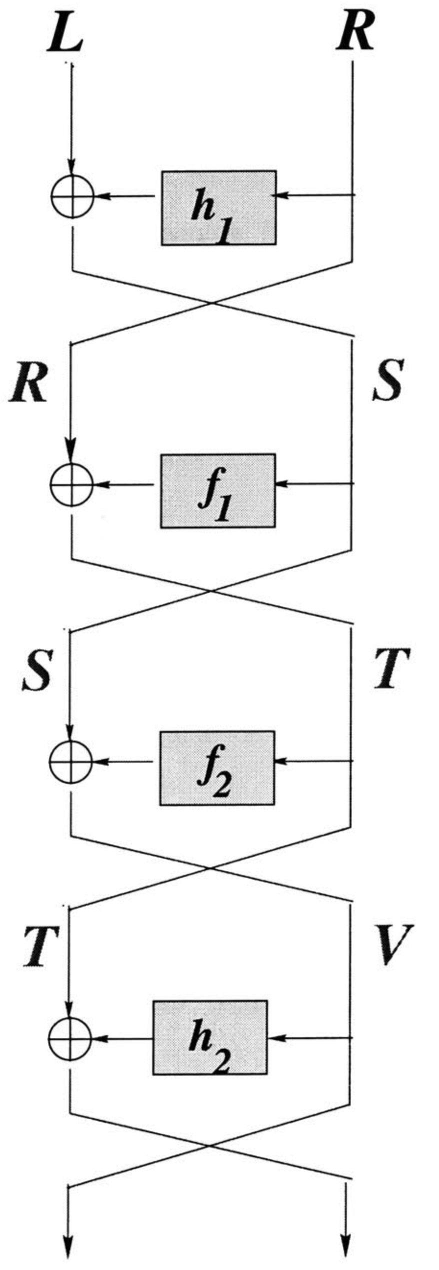

Figure 2-3: The Labeled Patel-Ramzan-Sundaram construction $\Psi(h_1, f_1, f_2, h_2)$.

We give a high-level description of the problem together with some of the relevant work in chapter 1. We now describe the setting in more rigorous detail. Two parties, Alice and Bob, are involved. We assume they have pre-agreed on a secret key a. In addition, there is a tagging algorithm S_a that applies a tag to a message, and a verification algorithm V_a that checks to see if the tag does indeed correspond to the message and the secret key a. In this thesis, we model both algorithms as keyed function families. Hence, once the key is chosen and fixed, the algorithms are deterministic. In addition, we also assume the algorithms are *stateless*. That is, they do not use counters or pass any other type of state information. We note that, in general, message authentication schemes may be probabilistic and may maintain some type of state information.

Let $\mathcal{M}$ denote a finite function family whose domain is the message space. We denote the key space by $\mathsf{Keys}(\mathcal{M})$. If Alice wants to send a message M to Bob, she first computes a *message authentication code*, or MAC, $\mu = \mathcal{M}_a(M)$. She sends (M, μ) to Bob, and upon receiving the pair, Bob checks to see if indeed $\mu = \mathcal{M}_a(M)$. The family $\mathcal{M}$ should be designed so that without knowledge of the secret key a, it should be infeasible for an adversary to construct a message and the corresponding tag. We stress that finite function families are not the only way to model message authentication codes, and there may be other ways which are more desirable.

Bellare, Kilian, and Rogaway [14] provided the first formal definition of security for a MAC. This definition is analogous to the formal security definition of a digital signature given by Goldwasser, Micali, and Rivest [63]. We adapt this definition for message authentication codes that are modeled as finite function families. In particular, we say that an adversary $\mathcal{A}$ breaks the MAC if, when given oracle access to $\mathcal{M}_a$, where a is kept secret, the adversary can come up with a pair (M^*, μ^*) such that $\mu^* = \mathcal{M}_a(M^*)$ but the message M^* was never given as an input to the oracle for $\mathcal{M}_a$. We now give a more formal treatment in terms of concrete security. We run the following experiment in which we use the adversary $\mathcal{A}$ to come up with a message and its corresponding message authentication code:

```
EXPERIMENT-FORGE($\mathcal{M}, \mathcal{A}$) :
    $a \overset{R}{\leftarrow} \mathsf{Keys}(\mathcal{M})$
    $(M^*, \mu^*) \leftarrow \mathcal{A}^{\mathcal{M}_a}$
    If $\mu^* = \mathcal{M}_a(M^*)$ and $\mathcal{A}$ never made $M^*$ a query to its oracle, then
        return 1
    else
        return 0.
```

Now we are ready to formally define security for a message authentication code.

Definition 19 *Let $\mathcal{M}$ be a keyed function family with domain I_k and range I_l. Let $\mathcal{A}$ be a 1-oracle adversary. Then,*

$$\mathsf{Adv}^{\mathsf{mac}}_{\mathcal{M}}(\mathcal{A}) = \Pr[\text{EXPERIMENT-FORGE}(\mathcal{M}, \mathcal{A}) \textit{ returns } 1].$$

For any integers $q, t \geq 0$, we define an insecurity function $\mathsf{Adv}^{\mathsf{mac}}_{\mathcal{M}}(q, t)$:

$$\mathsf{Adv}^{\mathsf{mac}}_{\mathcal{M}}(q, t) = \max_{\mathcal{A}}\{\mathsf{Adv}^{\mathsf{mac}}_{\mathcal{M}}(\mathcal{A})\}.$$

The above maximum is taken over choices of an adversary $\mathcal{A}$ restricted to running time at most t, and to at most q oracle queries.

Definition 20 *We say that $\mathcal{M}$ is a (t, q, ϵ)-secure family of message authentication codes if*

$$\mathsf{Adv}^{\mathsf{mac}}_{\mathcal{M}}(q, t) \leq \epsilon.$$

The intuition behind the above definitions is that the security of message authentication codes depends on the *unpredictability* of the keyed finite function family $\mathcal{M}$.

2.11 Construction of Universal Hash Function based Message Authentication Codes

As we mention in section 2.9, we can utilize universal hash functions in order to construct provably-secure block ciphers. Prior to their use in block ciphers, it was noticed that universal hash functions could be used for message authentication. As we discuss in chapter 1, the concept of a universal hash function based message authentication code was first seen in a paper by Wegman and Carter [142] – though in the unconditionally secure model. In the computational setting, one can construct these MACs by composing a universal hash function with a pseudorandom function. This construction yields a a pseudorandom function with a larger domain:

Theorem 7 *Let $\mathcal{F}$ be a keyed function family with domain I_n, range I_l and key space I_{s_1}. Let H be an ϵ-universal family of hash functions with domain I_k, range I_n, and key space I_{s_2}. Define the function family $\mathcal{M}$ with domain I_k, range I_l and key space $I_{s_1+s_2}$ via $\mathcal{M}_{k_1,k_2}(x) = f_{k_1}(h_{k_2}(x))$, where $f_{k_1} \in \mathcal{F}$ and $h_{k_2} \in H$. Then:*

$$\mathsf{Adv}^{\mathsf{prf}}_{\mathcal{M}}(q,t) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(q, t + O(s_2 + qt_h)) + \binom{q}{2}\epsilon,$$

where t_h is the worst case time it takes to compute the value of a function from H on a given input.

The above is a folklore result, and no published proof exists in the literature. One can easily prove the theorem using techniques from this thesis. Using a proposition due to Bellare, Kilian and Rogaway [14] (that analyzes the concrete security of using any pseudorandom function family $\mathcal{M}$ as a message authentication code) one can get a MAC construction. We can then prove the following theorem.

Theorem 8 *If we construct $\mathcal{M}$ as described above, then*

$$\mathsf{Adv}^{\mathsf{mac}}_{\mathcal{M}}(q,t) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathcal{M}}(q,t') + 1/2^l \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(q,t') + \binom{q}{2}\epsilon + 1/2^l,$$

where $t' = t + O(k + l + s_2 + qt_h)$.

We remark that the idea of using a pseudorandom function family for message authentication was seen much earlier [62, 61], and the above proposition primarily works out the concrete security of the construction.

2.12 Concrete Security versus Asymptotic Security

In this thesis we focus on proving our results in the concrete security model, rather than the complexity-theoretic security model. To justify our choice of the former over the latter, we discuss the relative merits and demerits of these models, though we stress that in many ways these two models are not terribly different.

Complexity-theoretic security analysis has been the focus of much research in theoretical cryptography. This treatment has its foundations in computational complexity theory. In this complexity-based cryptography, one typically talks about "polynomial-time" bounded adversaries, and "negligible" advantages. The approach was pioneered in the seminal paper of Goldwasser and Micali [64] on probabilistic encryption, and subsequently the complexity-theoretic approach was applied to pseudorandomness [27, 62, 144] and digital signatures [63]. This approach led to a number of important results, and helped lay the theoretical foundations for cryptography. For an excellent exposition on this aspect of cryptography, we refer the reader to Goldreich's text [60].

Often, in the complexity-theoretic model, one aims to arrive at a "proof of concept" that it is possible to construct a secure cryptographic system with certain properties. Thus, the various system parameters are often asymptotically approximated rather than precisely determined. The goal in concrete security is to determine the exact parameter values necessary in order to implement the system securely. It was developed in a series of papers by Bellare and Rogaway [15, 14, 13], and we refer the reader to Bellare's survey article [9] for a high-level exposition. In a concrete analysis, one tries to establish a tight approximation of the adversary's advantage as a function of the system's security parameters. In a complexity-theoretic analysis, however, it might be sufficient to say that a polynomial-time bounded adversary has negligible advantage; i.e. its advantage is bounded above by any inverse polynomial in the security parameters.

Now, if the security parameters are not set large enough, it might be possible for the adversary to break a real implementation of the system. For example, in our block cipher constructions, we build pseudorandom permutations out of pseudorandom functions, though we lose some security in the process. If the functions we start with are not "pseudorandom enough" then, because of the security loss, the permutations we build from them may not have adequate security for a particular application.

Thus the concrete security model requires us to be very precise in making our security claims. We stress that a complexity-theoretic analysis is still extremely useful since we can often determine the concrete parameters by working through it carefully. Similarly, given a security theorem and proof in the concrete setting, we can often come up with the analogous theorem and proof in the complexity-theoretic setting.

2.13 Conclusion

In this chapter we gave preliminary discussions on many of the notions relevant to this thesis. In addition to defining the notation to be used throughout the thesis, we gave some background information on Luby-Rackoff ciphers, and their constituent components like Feistel ladders and universal hash functions. We also explained the notion of a transcript since it is used in many of the proofs throughout the thesis. In addition, we defined message authentication codes (or MACs) which also appear throughout the thesis. Finally, we gave a brief discussion on concrete versus complexity-theoretic security analysis.

Chapter 3

Towards Optimal and Practical Luby-Rackoff Ciphers

3.1 Introduction

In this chapter we give new constructions of more practical Luby-Rackoff block ciphers which are efficient in terms of computation time and key length. In addition, we provide security guarantees for Luby-Rackoff ciphers under weaker and more practical assumptions about the underlying primitive. The results in this chapter appear in a paper by Patel, Ramzan, and Sundaram [114]. We start with the Naor-Reingold construction [101] and introduce new improvements in efficiency. Our constructions use the same pseudorandom function in rounds 2 and 3, and our universal hash functions in the 1st and 4th rounds operate on only half of the data as opposed to the entire data thereby improving on the Naor-Reingold construction.

We employ a novel construct called a *bisymmetric universal hash function*, that we define in chapter 2, which helps us attain more efficient constructions. In addition, we give an alternate security analysis which shows that even if the underlying round function is only a secure message authentication code (as opposed to the much stronger pseudorandom function) no adversary can easily invert Luby-Rackoff block ciphers.

This chapter is organized as follows. In section 3.2 we start by giving the Naor-Reingold construction and explain some of the difficulties associated with optimizing it. Also, we provide our construction, which overcomes these difficulties. In the next section, we analyze the super pseudorandomness of this construction. In section 3.4, we give an alternate security analysis under the assumption that the underlying round function is a secure message authentication code rather than a pseudorandom function, though our security claim is much weaker. In section 3.5 we initiate an informal discussion on the optimality of Luby-Rackoff ciphers, and explain where our cipher fits in. In the final section, we make some concluding remarks.

3.2 Improving Luby-Rackoff Ciphers

In this section we provide a construction and security proof of a more optimized Luby-Rackoff cipher. Our construction is more practical than the one given by Naor and Reingold [101] – which was the state of the art in Luby-Rackoff block ciphers. Recall the main theorem proven by Naor and Reingold:

Theorem 9 (Naor-Reingold [101]) *Let f_1 and f_2 be independently-keyed functions from a keyed function family $\mathcal{F}$ with domain and range I_n and key space I_{s_1}. Let h_1, h_2 be strongly-universal hash functions, keyed independently of each other and of f_1, f_2, from a keyed permutation family H, with domain and range I_n, and key space I_{s_2}. Let $\mathcal{P}$ be the family of permutations on I_{2n} with key space $I_{2s_1+2s_2}$ defined by permutations of the form $h_2^{-1} \circ \Psi(f_1, f_2) \circ h_1$. Then*

$$\mathsf{Adv}^{\mathsf{sprp}}_{\mathcal{P}}(q,t) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(q, t + O(s_1 + 2s_2 + q(2t_h + t_f))) + \binom{q}{2}\left(2^{-n+1} + 2^{-2n}\right)$$

where t_f is the worst case time it takes to compute the value of a function from $\mathcal{F}$ on a given input, and t_h is the worst case time it takes to compute the value of a function from H on a given input.

The Naor-Reingold construction significantly enhances the efficiency of the original Luby-Rackoff construction since it completely removes two calls of the expensive pseudorandom functions, and replaces them with much more efficient non-cryptographic strongly universal hash functions. In general, replacing pseudorandom functions with universal hash functions has a number of advantages, and we give a more thorough discussion in section 2.9 of chapter 2.

Naor and Reingold separately suggest two possible optimizations to their original construction and individually prove them to be secure block ciphers. The first is to use the same pseudorandom function in rounds two and three, thus saving key material: $h_2 \circ \Psi(f, f) \circ h_1$. The other possible optimization is to use the construction $\Psi(h_1, f_1, f_2, h_2)$ where the h_i are ϵ-almost Δ-universal hash functions which now operate on only *half* the data, as opposed to the entire $2n$-bit data. This construction saves running time and key material. Unfortunately, trying to realize both optimizations simultaneously ($\Psi(h_1, f, f, h_2)$) does not always lead to a secure cipher.

In particular, suppose that the ϵ-almost Δ-universal hash function family we use is the linear hash ($h_a(x) = ax$) where multiplication is performed over $GF(2^n)$. This family is known to be Δ-universal. Let us prove this.

Lemma 1 *The family H consisting of functions $h_a(x) = ax$ is a Δ-universal hash function.*

Proof: For all $x, y \in \mathrm{GF}(2^n)$, with $x \neq y$, and all $\delta \in \mathrm{GF}(2^n)$

$$\begin{aligned}
&\Pr_a[h_a(x) - h_a(y) = \delta] \\
&\qquad = \Pr_a[ax - ay = \delta] \\
&\qquad = \Pr_a[a = \delta \cdot (x-y)^{-1}] \\
&\qquad = 1/2^n.
\end{aligned}$$

The last equality follows from the previous since there is only a unique value of a satisfying the above equation, and there are 2^n possible choices. ■

We can distinguish the cipher $P = \Psi(h_1, f, f, h_2)$ from a random permutation by making one encryption query and one decryption query as follows. First we encrypt $L_1 \cdot R_1 = 0^{2n}$ (i.e. the $2n$-bit string of all 0's). If the underlying cipher is P, then following the equations for encryption, we get:

$$\begin{aligned}
S_1 &= L_1 \oplus h_1(R_1) \\
&= 0^n \oplus h_1(0^n) \\
&= h_1(0^n); \\
T_1 &= R_1 \oplus f(S_1)
\end{aligned}$$

$$
\begin{aligned}
&= 0^n \oplus f(h_1(0^n)) \\
&= f(h_1(0^n)); \\
V_1 &= S_1 \oplus f(T_1) \\
&= h_1(0^n) \oplus f^2(h_1(0^n)); \\
W_1 &= T_1 \oplus h_2(V_1) \\
&= f(h_1(0^n)) \oplus h_2(h_1(0^n) \oplus f^2(h_1(0^n))).
\end{aligned}
$$

Thus, the left half of the ciphertext output is:

$$V_1 = f^2(h_1(0^n)) \oplus h_1(0^n).$$

Next, we decrypt $V_2 \cdot W_2 = 0^{2n}$; i.e. the same string we encrypt in the first query. If the underlying cipher is P, then following the equations for decryption, we get:

$$
\begin{aligned}
T_2 &= W_2 \oplus h_2(V_2) \\
&= 0^n \oplus h_2(0^n) \\
&= h_2(0^n); \\
S_2 &= V_2 \oplus f(T_2) \\
&= 0^n \oplus f(h_2(0^n)) \\
&= f(h_2(0^n)); \\
R_2 &= T_2 \oplus f(S_2) \\
&= h_2(0^n) \oplus f^2(h_2(0^n)); \\
L_2 &= S_2 \oplus h_1(R_2) \\
&= f(h_2(0^n)) \oplus h_1(h_2(0^n) \oplus f^2(h_2(0^n))).
\end{aligned}
$$

Thus, the right half of the plaintext output is

$$R_2 = f^2(h_2(0^n)) \oplus h_2(0^n).$$

Now, if we use the linear hash $h_a(x) = ax$, we have that

$$h_1(0^n) = 0^n = h_2(0^n).$$

Thus:

$$\begin{aligned} V_1 &= h_1(0^n) \oplus f^2(h_1(0^n)) \\ &= 0^n \oplus f^2(0^n) \\ &= f^2(0^n). \end{aligned}$$

Similarly,

$$\begin{aligned} R_2 &= h_2(0^n) \oplus f^2(h_2(0^n)) \\ &= 0^n \oplus f^2(0^n) \\ &= f^2(0^n). \end{aligned}$$

Thus,

$$V_1 = f^2(0^n) = R_2$$

with certainty whenever P is the underlying cipher. This property would only hold with probability about $1/2^n$ if the cipher were a truly random permutation. Therefore, we have found an attack, which only requires two *non-adaptive* queries, and yields a distinguisher $\mathcal{A}$ with $\mathsf{Adv}^{\mathsf{prp}}_{\mathcal{F}}(\mathcal{A})$ exponentially close to 1. Observe that this attack imposes no restrictions on the underlying pseudorandom functions. In fact, the attack would still work if these functions were truly random rather than pseudorandom.

This raises the question of whether one can use the same pseudorandom function in rounds two and three and have an *efficient* non-cryptographic function operating on only half the bits in rounds 1 and 4. In this chapter, we give a construction which answers this question in the affirmative.

We employ ϵ_1-bisymmetric ϵ_2-almost Δ-universal hash functions. This novel construct will give us more efficient constructions than the strongly universal hash function based constructions of Naor and Reingold [101]. One universal hash function that satisfies these properties is the square hash, which can be implemented very efficiently on many platforms.

We discuss this function in great detail in chapter 6.

Using these ϵ_1-bisymmetric ϵ_2-almost Δ-universal hash functions in rounds one and four, and the same pseudorandom function in rounds 2 and 3, we can get a secure and efficient Luby-Rackoff cipher. Typically we want ϵ_1, ϵ_2 to be extremely small – around $1/2^n$ where the hash functions have domain and range I_n. In this chapter, we prove the following theorem:

Theorem 10 (Patel-Ramzan-Sundaram [114]) *Let f be a function from a keyed function family $\mathcal{F}$ with domain and range I_n and key space I_{s_1}. Let h_1, h_2 be ϵ_1-bisymmetric ϵ_2-almost Δ-universal hash functions, keyed independently of each other and of f, from a keyed function family H with domain and range I_n and key space I_{s_2}. Let $\mathcal{B}$ be the family of permutations on I_{2n} with key space $I_{s_1+2s_2}$ consisting of permutations of the form $B = \Psi(h_1, f, f, h_2)$. Then*

$$\mathsf{Adv}_{\mathcal{B}}^{\mathsf{sprp}}(q,t) \leq \mathsf{Adv}_{\mathcal{F}}^{\mathsf{prf}}(2q, t + O(2s_2 + 2q(t_h + n))) + q^2\epsilon_1 + \binom{q}{2}\left(2\epsilon_2 + 2^{-2n}\right),$$

where t_h is the time it takes (in the worst case) to compute the functions h_1 and h_2 on a given input.

We will focus our efforts on proving the following theorem, which utilizes truly random functions in the middle rounds rather than pseudorandom ones. Then, by applying standard techniques, the above theorem will follow as corollary.

Theorem 11 (Patel-Ramzan-Sundaram [114]) *Let f be chosen uniformly from the family $\mathsf{Rand}^{n\to n}$. Let h_1, h_2 be ϵ_1-bisymmetric ϵ_2-almost Δ-universal hash functions, keyed independently of each other and of f, from a keyed function family H with domain and range I_n. Let $\mathcal{P}$ be the family of permutations on I_{2n} consisting of permutations of the form $P = \Psi(h_1, f, f, h_2)$. Then*

$$\mathsf{Adv}_{\mathcal{P}}^{\mathsf{sprp}}(q,t) \leq q^2\epsilon_1 + \binom{q}{2}\left(2\epsilon_2 + 2^{-2n}\right).$$

Maurer [94] presents a very simple proof of security of the three-round Luby-Rackoff construction. His proof does not generalize to the four-round Luby-Rackoff construction, and does not deal well with adaptive adversaries. Naor and Reingold [101] give a more formal framework for proving adaptive security of Luby-Rackoff ciphers. Fortunately, the conditions that we need to satisfy for the security of our block cipher as in a Maurer-type treatment are the same as the conditions resulting from the more formal Naor-Reingold treatment. The proof that we sketch in the published version [114] uses a Maurer-style argument and does not include all the details. In addition it does not handle the adaptive case. In this chapter, with a view towards completeness and precision, we adopt the more general framework of Naor and Reingold, which allows us to present the proof of security in the setting of adaptive adversaries.

3.3 Proof of Security

We now analyze the security of our construction $P = \Psi(h_1, f, f, h_2)$ that we define in the above theorems. We proceed in the standard manner by showing that the our permutation $P = \Psi(h_1, f, f, h_2)$ is pseudorandom when f is truly random (instead of just pseudorandom). Our overall treatment, however, follows the nicely laid out framework of Naor and Reingold [101].

Recall that in our setting we model the adversary $\mathcal{A}$ as a program for a random access machine. The adversary gets a certain kind of black-box access to either a permutation sampled uniformly from the set of all possible permutations on $2n$ bits, or one sampled from the set of ciphers $P = \Psi(h_1, f, f, h_2)$. In the latter case, the sampling is done by randomly sampling functions h_1, h_2 from an ϵ_1-bisymmetric ϵ_2-almost Δ-universal family H according to its underlying distribution, and sampling f uniformly at random from $\mathsf{Rand}^{n \to n}$ (the family of all possible *functions* whose domain and range consists of n-bit strings).

The adversary must determine which of the two was actually sampled. Clearly, if the adversary cannot distinguish between the two then the cipher $P = \Psi(h_1, f, f, h_2)$ is, for all practical purposes, a truly random permutation, and no information about the plaintext can be derived from the ciphertext. This gives us the ultimate level of security.

The adversary $\mathcal{A}$ has access to two oracles: one oracle for each direction of the permutation. We can represent $\mathcal{A}$'s queries in two possible forms: $(+, x)$ which asks to obtain the value of $P(x)$, or $(-, y)$ which asks to obtain the value of $P^{-1}(y)$. For simplicity, we often write $L \cdot R$ to represent the left and right n bits of the plaintext x, and $V \cdot W$ to represent the left and right n bits of the ciphertext y. We define the query-answer pair for the i^{th} query as $\langle x_i, y_i \rangle \in \{0,1\}^{2n} \times \{0,1\}^{2n}$, where $\mathcal{A}$'s query is either $(+, x_i)$ and y_i is the answer it receives from P or its query is $(-, y_i)$ and x_i is the answer it receives. We assume that $\mathcal{A}$ makes exactly q queries, and we represent $\mathcal{A}$'s transcript by the sequence

$$\sigma = \langle (x_1, y_1), \ldots, (x_q, y_q) \rangle.$$

We restrict ourselves to the case that $\mathcal{A}$ is *deterministic*. This restriction does not really affect our results since we can fix the random tape that maximizes $\mathcal{A}$'s advantage, in addition to which the security bounds we attain are independent of the number of time steps needed by $\mathcal{A}$. Under this assumption, the i^{th} query made by $\mathcal{A}$ can be determined from the first $i-1$ query-answer pairs in $\mathcal{A}$'s transcript. Moreover, given the transcript and the description of the adversary, we can efficiently determine whether a given entry in the transcript represents a plaintext query or a ciphertext query, and we can determine the response (since $\mathcal{A}$ is deterministic, we can always simulate, and hence determine, its behavior).

From the way we have defined our transcripts, and using the notation discussed in section 2.6 of chapter 2, the only possible transcripts are ones for which:

$$C_{\mathcal{A}}[\sigma^{(i-1)}] \in \{(+, x_i), (-, y_i)\}, \tag{3.1}$$

where $\sigma^{(j)}$ represents the first j entries of σ, for $1 \leq j \leq q$. That is:

$$\sigma^{(j)} = \langle (x_1, y_1), \ldots, (x_j, y_j) \rangle.$$

We also assume that the adversary does not unnecessarily repeat queries.

Convention 1 *For any possible $\mathcal{A}$-transcript*

$$\sigma = \langle (x_1, y_1), \ldots, (x_q, y_q) \rangle$$

we assume from now on that if σ is consistent and if $i \neq j$ then both $x_i \neq x_j$ and $y_i \neq y_j$. This formalizes the concept that $\mathcal{A}$ never repeats a query if it can determine the answer from a previous query-answer pair.

We now consider another process for answering $\mathcal{A}$'s queries that will be useful to us.

Definition 21 *The random process $\tilde{R}$ answers the i^{th} query of $\mathcal{A}$ as follows:*

1. *If $\mathcal{A}$'s query is $(+, x_i)$ and for some $1 \leq j < i$ the j^{th} query-answer pair is (x_i, y_i), then $\tilde{R}$ answers with y_i. If more than one such query-answer pair exists, we pick the one with the smallest index.*

2. *If $\mathcal{A}$'s query is $(-, y_i)$ and for some $1 \leq j < i$ the j^{th} query-answer pair is (x_i, y_i), then $\tilde{R}$ answers with x_i. If more than one such query-answer pair exists, we pick the one with the smallest index.*

3. *If neither of the above happens, then $\tilde{R}$ answers with a uniformly chosen $2n$-bit string.*

Note that $\tilde{R}$'s answers may not be consistent with any function, let alone any permutation. We formalize this concept.

Definition 22 *Let*

$$\sigma = \langle (x_1, y_1), \ldots, (x_q, y_q) \rangle$$

be any possible $\mathcal{A}$-transcript. We say that σ is inconsistent if for some $1 \leq j < i \leq q$ the corresponding query-answer pairs satisfy:

$$\begin{aligned} x_i = x_j \quad &\textit{and} \quad y_i \neq y_j, \textit{ or} \\ x_i \neq x_j \quad &\textit{and} \quad y_i = y_j. \end{aligned}$$

Fortunately, we can show that the process $\tilde{R}$ often "behaves" exactly like a permutation. It turns out that if $\mathcal{A}$ is given oracle access to either $\tilde{R}$ or a function from Perm^{2n}, it will have a negligible advantage in distinguishing between the two. We prove this more formally in proposition 1 as was done by Naor and Reingold [101]. Before proceeding, recall that we can denote by the random variables $T_{\mathcal{P}}, T_{\mathsf{Perm}^{2n}}, T_{\tilde{R}}$ the transcript seen by $\mathcal{A}$ when its oracle queries are answered by $\mathcal{P}$, Perm^{2n}, $\tilde{R}$ respectively.

Proposition 1 *Let $\mathcal{A}$ be a 2-oracle adversary restricted to making a total of at most q queries to its oracles. Then*

$$\Pr_{\tilde{R}}[C_{\mathcal{A}}(T_{\tilde{R}}) = 1] - \Pr_{\mathsf{Perm}^{2n}}[C_{\mathcal{A}}(T_{\mathsf{Perm}^{2n}}) = 1] \leq \binom{q}{2} \cdot 2^{-2n}.$$

Proof: First, let Con denote the event that $T_{\tilde{R}}$ is consistent, and let $\neg\mathsf{Con}$ denote the complement of this event. Then, for any possible and consistent $\mathcal{A}$-transcript σ we have that:

$$\Pr_{\tilde{R}}[T_{\mathsf{Perm}^{2n}} = \sigma] = \frac{(2^{2n} - q)!}{2^{2n}!} = \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma \mid \mathsf{Con}].$$

Thus $T_{\mathsf{Perm}^{2n}}$ and $T_{\tilde{R}}$ have the same distribution conditioned on the event Con. We now bound the probability of $\neg\mathsf{Con}$. Recall that $T_{\tilde{R}}$ is inconsistent if there exists an i and j with $1 \leq j < i \leq q$ for which

$$\begin{array}{lll} x_i = x_j & \text{and} & y_i \neq y_j, \text{ or} \\ x_i \neq x_j & \text{and} & y_i = y_j. \end{array}$$

For a particular i and j this event happens with probability 2^{-2n}. So,

$$\Pr_{\tilde{R}}[\neg\mathsf{Con}] \leq \binom{q}{2} \cdot 2^{-2n}. \tag{3.2}$$

We complete the proof via a standard argument:

$$\begin{aligned}
&\Pr_{\tilde{R}}[C_{\mathcal{A}}(T_{\tilde{R}}) = 1] - \Pr_{\mathsf{Perm}^{2n}}[C_{\mathcal{A}}(T_{\mathsf{Perm}^{2n}}) = 1] \\
&\quad= \left(\Pr_{\tilde{R}}[C_{\mathcal{A}}(T_{\tilde{R}}) = 1 \mid \mathsf{Con}] - \Pr_{\mathsf{Perm}^{2n}}[C_{\mathcal{A}}(T_{\mathsf{Perm}^{2n}}) = 1]\right) \cdot \Pr_{\tilde{R}}[\mathsf{Con}] \\
&\quad+ \left(\Pr_{\tilde{R}}[C_{\mathcal{A}}(T_{\tilde{R}}) = 1 \mid \neg\mathsf{Con}] - \Pr_{\mathsf{Perm}^{2n}}[C_{\mathcal{A}}(T_{\mathsf{Perm}^{2n}}) = 1]\right) \cdot \Pr_{\tilde{R}}[\neg\mathsf{Con}] \\
&\quad= \left(\Pr_{\tilde{R}}[C_{\mathcal{A}}(T_{\tilde{R}}) = 1 \mid \neg\mathsf{Con}] - \Pr_{\mathsf{Perm}^{2n}}[C_{\mathcal{A}}(T_{\mathsf{Perm}^{2n}}) = 1]\right) \cdot \Pr_{\tilde{R}}[\neg\mathsf{Con}] \\
&\quad\leq \Pr_{\tilde{R}}[\neg\mathsf{Con}] \\
&\quad\leq \binom{q}{2} \cdot 2^{-2n},
\end{aligned}$$

which is the bound we desire. ■

We now proceed to obtain a bound on the advantage $\mathcal{A}$ will have in distinguishing between $T_{\mathcal{P}}$ and $T_{\tilde{R}}$. It turns out that $T_{\mathcal{P}}$ and $T_{\tilde{R}}$ are identically distributed unless some event depending on the choice of h_1, h_2 in $\mathcal{P}$ occurs. We call this event Bad and obtain a bound on the probability that it actually occurs. Intuitively Bad occurs whenever the internal function f in $\mathcal{P}$ would have been evaluated on the exact same point twice for

- two distinct chosen plaintext queries (forward direction of the permutation), or
- two distinct chosen ciphertext queries (reverse direction of the permutation), or
- one plaintext and one ciphertext query.

That is, Bad occurs whenever there is an *internal collision* in the function f during the computation of $\Psi(h_1, f, f, h_2)$. We formalize this as follows.

Definition 23 *Let $\sigma = \langle (x_1, y_1), \ldots, (x_q, y_q) \rangle$. Let $L_i \cdot R_i$ denote the leftmost n bits and rightmost n bits of x_i, and let $V_i \cdot W_i$ denote the leftmost n bits and rightmost n bits of y_i, for every $1 \le i \le q$. Then, for every specific pair of ϵ_1-bisymmetric ϵ_2-almost Δ-universal hash functions h_1, h_2 we define $\mathsf{Bad}(h_1, h_2)$ to be the set of all possible and consistent $\mathcal{A}$ − transcripts:*

$$\sigma = \langle (L_1 \cdot R_1, V_1 \cdot W_1), \ldots, (L_q \cdot R_q, V_q \cdot W_q) \rangle$$

satisfying:

- *event $\mathsf{B1}$: there exists $1 \le i < j \le q$ such that $h_1(R_i) \oplus L_i = h_1(R_j) \oplus L_j$, or*
- *event $\mathsf{B2}$: there exists $1 \le i < j \le q$ such that $W_i \oplus h_2(V_i) = W_j \oplus h_2(V_j)$, or*
- *event $\mathsf{B3}$: there exists $1 \le i, j \le q$ such that $h_1(R_i) \oplus L_i = W_j \oplus h_2(V_j)$.*

In the following proposition, we show that these bad events occur rarely.

Proposition 2 *Let $h_1, h_2 \in H$ be ϵ_1-bisymmetric ϵ_2-almost Δ-universal hash functions. Then, for any possible and consistent $\mathcal{A}$ − transcript*

$$\sigma = \langle (L_1 \cdot R_1, V_1 \cdot W_1), \ldots, (L_q \cdot R_q, V_q \cdot W_q) \rangle,$$

we have that

$$\Pr_{h_1, h_2}[\sigma \in \mathsf{Bad}(h_1, h_2)] \le q^2 \epsilon_1 + 2\binom{q}{2} \epsilon_2.$$

Proof: Recall that a transcript $\sigma \in \mathsf{Bad}(h_1, h_2)$ if event $\mathsf{B1}$, event $\mathsf{B2}$, or event $\mathsf{B3}$ occurs. We can determine the individual probabilities of each of these events separately, and obtain an upper bound on the desired probability by taking the sum. We start with bounding the probability of the first bad event:

$$\begin{aligned}
\Pr_{h_1}[\mathsf{B1}] &\leq \Pr_{h_1}[\exists 1 \leq i < j \leq q : h_1(R_i) \oplus L_i = h_1(R_j) \oplus L_j] \\
&\leq \sum_{1 \leq i < j \leq q} \Pr_{h_1}[h_1(R_i) \oplus L_i = h_1(R_j) \oplus L_j] \\
&\leq \sum_{1 \leq i < j \leq q} \Pr_{h_1}[h_1(R_i) \oplus h_1(R_j) = L_j \oplus L_i] \\
&\leq \binom{q}{2} \cdot \epsilon_2.
\end{aligned}$$

The last inequality follows from the previous since if $R_i \neq R_j$ then we can use the fact that h_1 is ϵ_2-almost Δ-universal, and if $R_i = R_j$ we know that $L_i \neq L_j$, since we assume the queries are distinct, in which case

$$\Pr_{h_1}[h_1(R_i) \oplus h_1(R_j) = L_j \oplus L_i] = 0.$$

Now, we bound the probability of the second event:

$$\begin{aligned}
\Pr_{h_2}[\mathsf{B2}] &\leq \Pr_{h_2}[\exists 1 \leq i < j \leq q : W_i \oplus h_2(V_i) = W_j \oplus h_2(V_j)] \\
&\leq \sum_{1 \leq i < j \leq q} \Pr[W_i \oplus h_2(V_i) = W_j \oplus h_2(V_j)] \\
&\leq \sum_{1 \leq i < j \leq q} \Pr[h_2(V_j) \oplus h_2(V_i) = W_j \oplus W_i] \\
&\leq \binom{q}{2} \cdot \epsilon_2.
\end{aligned}$$

The last inequality follows from the previous since if $V_i \neq V_j$ then we can use the fact that h_2 is ϵ_2-almost Δ-universal, and if $V_i = V_j$ we know that $W_i \neq W_j$, since we assume the queries are distinct, in which case

$$\Pr_{h}[h(V_i) \oplus h(V_j) = W_j \oplus W_i] = 0.$$

Finally, we must bound the probability of the third event:

$$\begin{aligned}\Pr_{h_1,h_2}[\mathsf{B3}] &\leq \Pr_h[\exists 1 \leq i,j \leq q : h_1(R_i) \oplus L_i = W_j \oplus h_2(V_j)] \\ &\leq \sum_{1\leq i,j\leq q} \Pr[h_1(R_i) \oplus L_i = W_j \oplus h_2(V_j)] \\ &\leq \sum_{1\leq i,j\leq q} \Pr[h_1(R_i) \oplus h_2(V_j) = W_j \oplus L_i] \\ &\leq q^2 \cdot \epsilon_1.\end{aligned}$$

The last inequality follows from the previous since h_1 and h_2 are ϵ_1-bisymmetric.

We thus get that:

$$\begin{aligned}\Pr_{h_1,h_2}[\sigma \in \mathsf{Bad}(h_1,h_2)] &\leq \Pr_{h_1,h_2}[\mathsf{B1}] + \Pr_{h_1,h_2}[\mathsf{B2}] + \Pr_{h_1,h_2}[\mathsf{B3}] \\ &\leq q^2\epsilon_1 + 2\binom{q}{2} \cdot \epsilon_2,\end{aligned}$$

which is the desired bound. ■

The following key lemma shows that the distribution of possible and consistent transcripts generated by $T_{\mathcal{P}}$ given that the bad conditions do not occur is identical to the distribution of possible and consistent transcripts generated by $T_{\tilde{R}}$. This lemma will later help us to determine a bound on the advantage our adversary $\mathcal{A}$ will have when trying to distinguish between these two cases in general.

Lemma 2 *Let σ be any possible and consistent $\mathcal{A}$-transcript, where*

$$\sigma = \langle (L_1 \cdot R_1, V_1 \cdot W_1), \ldots, (L_q \cdot R_q, V_q \cdot W_q) \rangle.$$

Then

$$\Pr_{\mathcal{P}}[T_{\mathcal{P}} = \sigma | \sigma \notin \mathsf{Bad}(h_1,h_2)] = \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma].$$

Proof: A fairly straightforward argument first provided by Naor and Reingold [101] gives us

$$\Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma] = 2^{-2nq}.$$

This equality follows from the fact that $\tilde{R}$ works by picking elements at random from I_{2n}. Thus, if we consider a fixed transcript entry, the probability that we can generate it is 2^{-2n}.

For q consistent transcript entries, $\tilde{R}$ can generate it by picking q elements independently from I_{2n}, which gives us the desired equality 2^{-2nq}. Let us consider the case in which σ was generated by $T_{\mathcal{P}}$; i.e. the oracle computes a function of the form $P = \Psi(h_1, f, f, h_2)$. Now, σ is a possible $\mathcal{A}$-transcript so $T_{\mathcal{P}} = \sigma$ if and only if $V_i \cdot W_i = P(L_i \cdot R_i)$ for all $1 \leq i \leq q$. We know that $L_i \cdot R_i$ and $V_i \cdot W_i$ must satisfy the following series of equations:

$$\begin{aligned} S_i &= L_i \oplus h_1(R_i); \\ T_i &= R_i \oplus f(S_i); \\ V_i &= S_i \oplus f(T_i); \\ W_i &= T_i \oplus h_2(V_i). \end{aligned}$$

So, in particular

$$V_i \cdot W_i = P(L_i \cdot R_i) \Leftrightarrow f(S_i) = T_i \oplus R_i \text{ and } f(T_i) = V_i \oplus S_i.$$

Next, suppose h_1, h_2 are ϵ_1-bisymmetric ϵ_2-almost Δ-universal hash functions for which $\sigma \notin \mathsf{Bad}(h_1, h_2)$. Then, for all $1 \leq i < j \leq q$, it follows that $S_i \neq S_j$ and $T_i \neq T_j$ (otherwise $\sigma \in \mathsf{Bad}(h_1, h_2)$). Similarly, for all $1 < i, j < q$, we have that $S_i \neq T_j$. So, if $\sigma \notin \mathsf{Bad}(h_1, h_2)$ all the inputs to f are distinct. Since f is chosen from $\mathsf{Rand}^{n \to n}$, for every fixed choice of h_1, h_2 such that $\sigma \notin \mathsf{Bad}(h_1, h_2)$ the probability (taken only over the choice of f) that $T_{\mathcal{P}} = \sigma$ is exactly 2^{-2nq}. Therefore:

$$\Pr_{\mathcal{P}}[T_{\mathcal{P}} = \sigma | \sigma \notin \mathsf{Bad}(h_1, h_2)] = 2^{-2nq},$$

which completes the proof of the lemma. ■

To complete the proof of the main theorem, we can follow the Naor-Reingold framework utilizing the above lemma where appropriate. Before going through the necessary steps, we state a preliminary proposition. In the following, we abuse notation by listing the adversary's oracles as $\langle \mathcal{P}, \mathcal{P}^{-1} \rangle$, and $\langle \mathsf{Perm}^{2n}, (\mathsf{Perm}^{2n})^{-1} \rangle$ to remind the reader that oracle access is given to both the forward and inverse directions of the permutation.

Proposition 3 *Let Γ be the set of all possible and consistent transcripts σ such that*

$C_{\mathcal{A}}(\sigma) = 1$. *Then:*

$$\mathsf{Adv}_{\mathcal{A}}(\langle \mathcal{P}, \mathcal{P}^{-1}\rangle, \langle \mathsf{Perm}^{2n}, (\mathsf{Perm}^{2n})^{-1}\rangle) \leq \sum_{\sigma\in\Gamma}(\Pr_{\mathcal{P}}[T_{\mathcal{P}} = \sigma] - \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma]) + \binom{q}{2}\cdot 2^{-2n},$$

where $\mathcal{A}$ is restricted to making a total of at most q queries to its oracles.

Proof: First, we break up the left hand side of the above inequality so that the distinguishability of both distributions are taken with respect to the process $\tilde{R}$:

$$\begin{aligned}&\mathsf{Adv}_{\mathcal{A}}(\langle \mathcal{P}, \mathcal{P}^{-1}\rangle, \langle \mathsf{Perm}^{2n}, (\mathsf{Perm}^{2n})^{-1}\rangle)\\ &\quad\leq \mathsf{Adv}_{\mathcal{A}}(\langle \mathcal{P}, \mathcal{P}^{-1}\rangle, \tilde{R}) + \mathsf{Adv}_{\mathcal{A}}(\tilde{R}, \langle \mathsf{Perm}^{2n}, (\mathsf{Perm}^{2n})^{-1}\rangle). \qquad (3.3)\end{aligned}$$

Now, since $\mathcal{A}$ with an oracle for $\mathcal{P}$ only generates possible and consistent transcripts, we can apply theorem 1 from chapter 2 to the first summand in equation 3.3. Next, we can apply proposition 1 from this chapter to the second summand. The proposition follows. ■

We now prove the main theorem:

Proof: (of Theorem 11) First we successively apply proposition 3 given above, and theorem 2 from chapter 2 with the set $\mathsf{Bad}(h_1, h_2)$:

$$\begin{aligned}&\mathsf{Adv}(\langle \mathcal{P}, \mathcal{P}^{-1}\rangle, \langle \mathsf{Perm}^{2n}, (\mathsf{Perm}^{2n})^{-1}\rangle)\\ &\quad\leq \sum_{\sigma\in\Gamma}(\Pr_{\mathcal{P}}[T_{\mathcal{P}} = \sigma] - \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma]) + \binom{q}{2}\cdot 2^{-2n}.\end{aligned}$$

Now, applying Theorem 2:

$$\begin{aligned}&\sum_{\sigma\in\Gamma}(\Pr_{\mathcal{P}}[T_{\mathcal{P}} = \sigma] - \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma])\\ \leq\ & \sum_{\sigma\in\Gamma}\left(\Pr_{\mathcal{P}}[T_{\mathcal{P}} = \sigma \mid \sigma \notin \mathsf{Bad}(h_1, h_2)] - \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma]\right)\cdot \Pr_{\mathcal{P}}[\sigma \notin \mathsf{Bad}(h_1, h_2)]\\ +\ & \Pr_{\mathcal{P}}[\sigma^* \in \mathsf{Bad}(h_1, h_2)].\end{aligned}$$

Observe that according to Lemma 2:

$$\left(\Pr_{\mathcal{P}}[T_\Psi = \sigma \mid \sigma \notin \mathsf{Bad}(h_1, h_2)] - \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma]\right) \cdot \Pr_{\mathcal{P}}[\sigma \notin \mathsf{Bad}(h_1, h_2)] = 0.$$

In addition, observe that proposition 2 tells us that

$$\Pr_{\mathcal{P}}[\sigma^* \in \mathsf{Bad}(h_1, h_2)] \leq q^2\epsilon_1 + 2\binom{q}{2} \cdot \epsilon_2.$$

Combining these observations, we get:

$$\begin{aligned}&\mathsf{Adv}(\langle \mathcal{P}, \mathcal{P}^{-1}\rangle, \langle \mathsf{Perm}^{2n}, (\mathsf{Perm}^{2n})^{-1}\rangle) \\ &\qquad\qquad \leq \; q^2\epsilon_1 + \binom{q}{2}\left(2\epsilon_2 + 2^{-2n}\right)\end{aligned}$$

which is the desired bound. ■

Having proven theorem 11, we now give the complete proof of theorem 10.

Proof: (of Theorem 10) The proof starts by analyzing the advantage an adversary has in distinguishing between the permutation families $\mathcal{B}$ and $\mathcal{P}$. Recall that $\mathcal{P}$ is the construction in which the functions in the middle two rounds are taken from $\mathsf{Rand}^{n\to n}$, and $\mathcal{B}$ is almost identical, with the exception that the middle two round functions are taken from $\mathcal{F}$ and may not be truly random. Combining this analysis with the result of theorem 11 gives us the desired result.

Consider an adversary $\mathcal{A}_1$ who tries to distinguish $\mathcal{B}$ and $\mathcal{P}$. We convert this adversary $\mathcal{A}_1$ into another adversary $\mathcal{A}_2$ that tries to distinguish between $\mathcal{F}$ and $\mathsf{Rand}^{n\to n}$ via the following reduction algorithm. We assume that $\mathcal{A}_1$ has access to an oracle $\mathcal{O}_1$ which either computes a permutation from $\mathcal{B}$ or $\mathcal{P}$ (or the inverse of such a permutation), and we assume that $\mathcal{A}_2$ has access to an oracle $\mathcal{O}_2$ which computes a function from $\mathcal{F}$ or $\mathsf{Rand}^{n\to n}$. We represent the permutation that the first oracle computes by p, and function that the second oracle computes by f.

```
Description of $\mathcal{A}_2^{\mathcal{O}_2}$ :
    $h_1, h_2 \leftarrow H$.
    **Repeat**
    Simulate $\mathcal{A}_1^{\mathcal{O}_1}$
        **If** $\mathcal{A}_1$ asks for $p(x)$ **then**
        compute $\Psi(h_1, f, f, h_2)$ using two queries to $\mathcal{O}_2$:
                $S = x^L \oplus h_1(x^R)$.
                $T = x^R \oplus f(S)$.
                $V = S \oplus f(T)$.
                $W = T \oplus h_2(V)$.
                Return $V \cdot W$ as the response to $\mathcal{A}_1$'s query for $p(x)$.
        **If** $\mathcal{A}_1$ asks for $p^{-1}(y)$ **then**
        compute $\Psi^{-1}(h_1, f, f, h_2)$ using two queries to $\mathcal{O}_2$:
                $T = y^R \oplus h_2(y^L)$.
                $S = y^L \oplus f(T)$.
                $R = T \oplus f(S)$.
                $L = S \oplus h_1(R)$.
                Return $L \cdot R$ as the response to $\mathcal{A}_1$'s query for $p^{-1}(y)$.
    **Until** $\mathcal{A}_1$ is done.
    Give the same output as $\mathcal{A}_1$.
```

We now explain what this algorithm does. It starts by picking two functions h_1, h_2 from the universal family H. This really amounts to picking two strings from I_{s_2}, and using these to key two functions. Next, the adversary $\mathcal{A}_1$ is simulated. As the simulation progresses, $\mathcal{A}_1$ can make two types of queries to its oracle: it can ask to have either the forward direction or inverse direction of the permutation on a given input. These would correspond to the encryption of a chosen plaintext, or the decryption of a chosen ciphertext. In either case, $\mathcal{A}_2$ can provide an answer to this query by actually performing the computation required in the Feistel ladder. To do so $\mathcal{A}_2$ must make two oracle calls to its oracle $\mathcal{O}_2$, which computes the function f. In addition, it must compute the value of h_1 on one input, and the value of h_2 on one input. Finally, when $\mathcal{A}_1$ is done, and outputs a value, $\mathcal{A}_2$ should give the same output.

Observe that for every query $\mathcal{A}_1$ makes in the simulation, $\mathcal{A}_2$ makes two queries to its oracle $\mathcal{O}_2$ and two universal hash function computations (since it must simulate a four-round Feistel network in which the outer two rounds each involve a hash function computation, and the middle two rounds each involve oracle calls). In addition, $\mathcal{A}_2$ must write various strings to its tape in order to make its oracle calls, and pick two random strings from I_{s_2} to choose functions from H. Thus, if the simulation of $\mathcal{A}_1$ takes t time steps, then the running time for $\mathcal{A}_2$ is $t + O(2s_2 + 2q(t_h + n))$. Now, suppose that

$$\mathsf{Adv}_{\mathcal{F}}^{\mathsf{prf}}(2q, t + O(2s_2 + 2q(t_h + n))) = \delta$$

for some value δ smaller than 1. Then, it follows that

$$\mathsf{Adv}_{\mathcal{P}}^{\mathcal{B}}(q, t) \leq \delta$$

since if the advantage were greater than δ, the algorithm we just described would be able to distinguish between $\mathcal{F}$ and $\mathsf{Rand}^{n \to n}$ with advantage better than δ. Consequently,

$$\mathsf{Adv}_{\mathcal{P}}^{\mathcal{B}}(q, t) \leq \mathsf{Adv}_{\mathcal{F}}^{\mathsf{prf}}(2q, t + O(2s_2 + 2q(t_h + n))).$$

Now, theorem 11 gives us that

$$\begin{aligned}\mathsf{Adv}(\langle \mathcal{P}, \mathcal{P}^{-1} \rangle, \langle \mathsf{Perm}^{2n}, (\mathsf{Perm}^{2n})^{-1} \rangle) \\ \leq \; q^2 \epsilon_1 + \binom{q}{2} \left(2\epsilon_2 + 2^{-2n}\right).\end{aligned}$$

Combining these two inequalities, we get that:

$$\mathsf{Adv}_{\mathcal{P}}^{\mathsf{sprp}}(q, t) \leq \mathsf{Adv}_{\mathcal{F}}^{\mathsf{prf}}(2q, t + O(2s_2 + 2q(t_h + n))) + q^2 \epsilon_1 + \binom{q}{2} \left(2\epsilon_2 + 2^{-2n}\right)$$

which is the desired bound. ■

3.4 Proving Security Under the MAC Assumption

We give an alternate security analysis of our construction. We utilize a weaker, but perhaps more practical, assumption, but make a weaker claim on the security of our construction. In particular, we show that if the underlying function f in our construction is a secure message authentication code (MAC), then it is infeasible for an adversary to come up with the plaintext corresponding to a randomly chosen ciphertext, or the ciphertext corresponding to a randomly chosen plaintext. Here the challenge plaintext (ciphertext) must be chosen at random from the set of all plaintexts (ciphertexts) that do not appear in any query made by the adversary. Some earlier work on the relationship between unpredictability (MACs) and indistinguishability was studied in a paper by Naor and Reingold [100].

While the security claim we make and prove is not strong from a purely theoretical point of view, it does have practical importance since unpredictability is a less stringent requirement than pseudorandomness on a function family. The weaker notions of security we discuss were formally defined in a paper by Even and Mansour [50]. The versions we present in the thesis are slightly adapted for our settings.

The first notion is security against the *cracking problem* or CP. In this scenario, the adversary has oracle access to a particular block cipher. He makes a series of (possibly adaptive) queries to these oracles. Finally, we he is given a challenge ciphertext that has never appeared either as a query, or as an answer to a query. We say that the adversary succeeds if it can successfully invert the ciphertext to produce the original plaintext (without making additional queries to the oracle).

The second notion is security against the *forgery problem* or FP. Here the adversary again makes a series of (possibly adaptive) queries to its oracles, and then is given a challenge *plaintext*, that has never appeared either as a query, or as an answer to a query. We say that the adversary succeeds if it can come up with the corresponding ciphertext.

There are several types of challenges that one can imagine. Our results involve the case of a *random challenge*. That is, the plaintext or ciphertext which is given as a challenge must be chosen at random. One can imagine other scenarios; for example, allowing the adversary to pick his or her own challenge. We make these definitions more formal.

Definition 24 *Let $\mathcal{P}$ be a permutation family over a set $\mathcal{D}$. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be adversaries,* *where $\mathcal{A}_1$ has access to two oracles. Then,*

$$\mathsf{Adv}^{\mathsf{CP}}_{\mathcal{P}}(\mathcal{A}_1,\mathcal{A}_2) = \Pr[a \leftarrow \mathsf{Keys}(\mathcal{P}); s \leftarrow \mathcal{A}_1^{p_a,p_a^{-1}}; c \overset{R}{\leftarrow} \mathcal{D}'; m \leftarrow \mathcal{A}_2(s,c) : p_k(m) = c].$$

Here the set $\mathcal{D}'$ represents all elements of $\mathcal{D}$ that do not appear as a component of a ciphertext oracle query to p_a^{-1} or as a ciphertext response of an oracle query to p_a. Thus the challenge ciphertext c is different from any ciphertext that has been given as a query or as a response to a query during the execution of $\mathcal{A}_1$. For any integers q,t, we define an insecurity function $\mathsf{Adv}^{\mathsf{CP}}_{\mathcal{P}}(q,t)$ as follows:

$$\mathsf{Adv}^{\mathsf{CP}}_{\mathcal{P}}(q,t) = \max_{\mathcal{A}_1,\mathcal{A}_2}\{\mathsf{Adv}^{\mathsf{CP}}_{\mathcal{P}}(\mathcal{A}_1,\mathcal{A}_2)\}.$$

The above maximum is taken over choices of adversary pairs $\mathcal{A}_1,\mathcal{A}_2$, that have combined running time at most t, and such that $\mathcal{A}_1$ makes at most q queries.

Definition 25 *We say that $\mathcal{P}$ is (t,q,ϵ)-secure against the cracking problem on random challenges if*

$$\mathsf{Adv}^{\mathsf{CP}}_{\mathcal{P}}(q,t) \leq \epsilon.$$

Similarly, we can establish a formal definition for the forgery problem:

Definition 26 *Let $\mathcal{P}$ be a permutation family over a set $\mathcal{D}$. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be adversaries, where $\mathcal{A}_1$ has access to two oracles. Then,*

$$\mathsf{Adv}^{\mathsf{FP}}_{\mathcal{P}}(\mathcal{A}_1,\mathcal{A}_2) = \Pr[a \leftarrow \mathsf{Keys}(\mathcal{P}); s \leftarrow \mathcal{A}_1^{p_a,p_a^{-1}}; m \overset{R}{\leftarrow} \mathcal{D}'; c \leftarrow \mathcal{A}_2(s,m) : p_k(m) = c].$$

Here the set $\mathcal{D}'$ represents all elements of $\mathcal{D}$ that do not appear as a component of a plaintext oracle query to p_a or as a plaintext response of an oracle query to p_a^{-1}. Thus the challenge plaintext m is different from any ciphertext that has been given as a query or as a response to a query during the execution of $\mathcal{A}_1$. For any integers q,t, we define an insecurity function $\mathsf{Adv}^{\mathsf{FP}}_{\mathcal{P}}(q,t)$ as follows:

$$\mathsf{Adv}^{\mathsf{FP}}_{\mathcal{P}}(q,t) = \max_{\mathcal{A}_1,\mathcal{A}_2}\{\mathsf{Adv}^{\mathsf{FP}}_{\mathcal{P}}(\mathcal{A}_1,\mathcal{A}_2)\}.$$

The above maximum is taken over choices of adversary pairs $\mathcal{A}_1,\mathcal{A}_2$, that have combined running time at most t, and such that $\mathcal{A}_1$ makes at most q queries.

Definition 27 *We say that $\mathcal{P}$ is (t,q,ϵ)-secure against the cracking problem on random* *challenges if*

$$\mathsf{Adv}_{\mathcal{P}}^{\mathsf{FP}}(q,t) \leq \epsilon.$$

These definitions are quite similar. In both cases we have two adversaries, one of which makes queries, and computes a "cookie" or some piece of state information s, and the other which takes this cookie s, together with a challenge ciphertext (plaintext), and tries to come up with the corresponding plaintext (ciphertext). While there are significantly stronger definitions of block cipher security (e.g. super pseudorandomness) the security measures we propose above are still useful; for example, they give guarantees against key-recovery attacks.

Theorem 12 *Let f be a function from a keyed function family $\mathcal{F}$, with domain and range I_n, and key space I_{s_1}. Let h_1, h_2 be two functions, keyed independently of each other and of f, chosen from a keyed function family H, which also has domain and range I_n, and key space I_{s_2}. Consider the permutation family $\mathcal{P}$, that consists of permutations of the form $\Psi(h_1, f, f, h_2)$, with domain and range I_{2n}, and key space $I_{s_1+2s_2}$. Then*

$$\mathsf{Adv}_{\mathcal{P}}^{\mathsf{CP}}(q,t) \leq \mathsf{Adv}_{\mathcal{F}}^{\mathsf{mac}}(2q, t + O(2s_2 + 2q(t_h + n))) + 2q/2^n,$$

where t_h is the worst case time to evaluate a function from H on a given input.

Proof: The overall idea is to show that given any adversary pair $\mathcal{A}_1, \mathcal{A}_2$ who can solve the cracking problem, we can construct an adversary $\mathcal{A}'$ who can break the underlying MAC f where $\mathcal{A}'$ has access to an oracle $\mathcal{O}_2$, which computes the MAC function f. We describe $\mathcal{A}'$ as follows.

```
Description of $\mathcal{A}'$ :
    $h_1, h_2 \leftarrow H$.
    **Repeat**
    Simulate $\mathcal{A}_1^{\mathcal{O}_1}$
        **If** $\mathcal{A}_1$ asks for $p(x)$ **then**
        compute $\Psi(h_1, f, f, h_2)$ using two queries to $\mathcal{O}_2$:
                $S = x^L \oplus h_1(x^R)$.
                $T = x^R \oplus f(S)$.
                $V = S \oplus f(T)$.
                $W = T \oplus h_2(V)$.
                Return $V \cdot W$ as the response to $\mathcal{A}_1$'s query for $p(x)$.
        **If** $\mathcal{A}_1$ asks for $p^{-1}(y)$ **then**
        compute $\Psi^{-1}(h_1, f, f, h_2)$ using two queries to $\mathcal{O}_2$:
                $T = y^R \oplus h_2(y^L)$.
                $S = y^L \oplus f(T)$.
                $R = T \oplus f(S)$.
                $L = S \oplus h_1(R)$.
                Return $L \cdot R$ as the response to $\mathcal{A}_1$'s query for $p^{-1}(y)$.
    **Until** $\mathcal{A}_1$ is done.
    Let $s$ denote the output of $\mathcal{A}_1$ ($s$ is a cookie).
    $V \cdot W \leftarrow I_{2n}$ (where $V, W \in I_n$).
    Simulate $\mathcal{A}_2(s, V \cdot W)$.
    Let $m = (L, R)$ denote the output of $\mathcal{A}_2$.
    Output: $(W \oplus h_2(V), V \oplus L \oplus h_1(R))$.
```

Here is what $\mathcal{A}'$ does. First it picks two functions h_1 and h_2 at random from some function family; for example, a family of bisymmetric Δ-universal hash functions. Then $\mathcal{A}'$ proceeds simulating $\mathcal{A}_1$. At some point $\mathcal{A}_1$ is going to make a query which could be in either of two forms: "Please give me the encryption of a message m" or "Please give me the decryption of a ciphertext c." In either case, $\mathcal{A}'$ must give a legitimate answer to the query that $\mathcal{A}_1$ makes. The adversary $\mathcal{A}'$ can do this easily by making two calls to the black box for f and simulating the encryption or decryption algorithms. For example, if the i^{th} query is an encryption query on a message $M_i = L_i \cdot R_i$, then $\mathcal{A}'$ computes values S_i, T_i, V_i, W_i according to equations:

$$\begin{aligned}
S_i &= L_i \oplus h_1(R_i); \\
T_i &= R_i \oplus f(S_i); \\
V_i &= S_i \oplus f(T_i); \\
W_i &= T_i \oplus h_2(V_i).
\end{aligned}$$

We see that $\mathcal{A}'$ calls the black box for f whenever it computes T_i and V_i. Decryption queries are handled in a similar fashion. Now, after $\mathcal{A}_1$ finishes making q queries, (which results in $\mathcal{A}'$ having made $2q$ queries), $\mathcal{A}'$ collects the state information s, that $\mathcal{A}_1$ computes. Next, $\mathcal{A}'$ picks a random ciphertext $c = V \cdot W$. It gives this ciphertext together with the state information s to $\mathcal{A}_2$. Call the response m, and divide it up into left and right halves: (L, R). Then, $\mathcal{A}'$ outputs:

$$(W \oplus h_2(V), V \oplus L \oplus h_1(R)).$$

Observe that if $\mathcal{A}_1$ makes q queries, then $\mathcal{A}'$ must make $2q$ queries to its oracle $\mathcal{O}_2$, and must compute the value of h_1 on q inputs, and the value of h_2 on q inputs. In addition, $\mathcal{A}'$ must record various values in order to make its oracle queries.

If the adversary pair $\mathcal{A}_1, \mathcal{A}_2$ successfully solves the cracking problem, then the above equation represents the pair $(T, f(T))$, which constitutes a candidate forgery on the MAC f. This event happens with some probability. This probability is almost identical to probability that EXPERIMENT-FORGE($\mathcal{A}'$) returns 1, where EXPERIMENT-FORGE is the experiment we defined when we discussed message authentication codes in chapter 2.

There is still one technicality remaining. Recall that in our definition of a secure message authentication code, we require that the adversary forge a message which is different from the messages that the adversary gives to the black box during the query phase. It turns out that, in the current scenario, this event happens with high probability. In particular, since $V \cdot W$ is chosen at random, and is independent of the other queries, the value $W \oplus h_2(V)$ is distributed uniformly, and is independent of any values that are given as input to f in the query phase. Since there were $2q$ input queries made to f, the probability that any of them equals $W \oplus h_2(V)$ is bounded above by $2q/2^n$.

Now, let us first compute the probability that $\mathcal{A}'$ fails to come up with a valid forgery. Observe that $\mathcal{A}'$ fails if at least one of the following two conditions occur:

1. The adversary pair $\mathcal{A}_1, \mathcal{A}_2$ does not correctly solve the cracking problem.
2. The value $W \oplus h_2(V)$ was already given as a query by $\mathcal{A}_1$ during the query phase of the simulation (i.e. $\mathcal{A}'$ did not forge a new message).

Suppose the first condition occurs with some probability $1 - \epsilon'$. We know that the second condition happens with probability $2q/2^n$. Thus, the probability that $\mathcal{A}'$ fails is bounded above by

$$1 - \epsilon' + 2q/2^n.$$

Consequently, the probability that $\mathcal{A}'$ succeeds is bounded from below by $\epsilon' - 2q/2^n$. Notice that by the statement of the theorem, the success probability of $\mathcal{A}'$ is bounded above by the probability that we can create a valid forgery; i.e. the probability that EXPERIMENT-FORGE($\mathcal{A}'$) returns 1. Also, the maximum value of ϵ' is $\mathsf{Adv}_{\mathcal{P}}^{\mathsf{CP}}(q,t)$. Thus:

$$\mathsf{Adv}_{\mathcal{P}}^{\mathsf{CP}}(q,t) - 2q/2^n \leq \mathsf{Adv}_{\mathcal{F}}^{\mathsf{mac}}(2q, t + O(2s_2 + 2q(t_h + n))).$$

The required bound follows. ■

We can state a similar theorem for security against the forgery problem under random challenges. We omit the full proof since it is similar to the previous proof. The only difference is that the adversary $\mathcal{A}'$ will construct a forgery of the form $(S, f(S))$ via the pair

$$(L \oplus h_1(R), W \oplus h_2(V) \oplus R).$$

Theorem 13 *Let f be a function from a keyed function family $\mathcal{F}$, with domain and range I_n, and key space I_{s_1}. Let h_1, h_2 be two functions, keyed independently of each other and of f, chosen from a keyed function family H, which also has domain and range I_n, and key* *space I_{s_2}. Consider the permutation family $\mathcal{P}$, that consists of permutations of the form $\Psi(h_1, f, f, h_2)$, with domain and range I_{2n}, and key space $I_{s_1+2s_2}$. Then*

$$\mathsf{Adv}_{\mathcal{P}}^{\mathsf{FP}}(q,t) \leq \mathsf{Adv}_{\mathcal{F}}^{\mathsf{mac}}(2q, t + O(2s_2 + 2q(t_h + n))) + 2q/2^n,$$

where t_h is the worst case time to evaluate a function from H on a given input.

3.5 A Discussion on Optimality

Having examined the class of ciphers $\Psi(h_1, f, f, h_2)$, under two distinct security analyses, we give an informal discussion on the optimality of a Luby-Rackoff cipher, and explain where our construction fits. Recall that since the path-breaking paper of Luby and Rackoff [88], considerable progress has been made with respect to making the construction more efficient. Specifically, as noted previously, most of the focus has been in "reducing" the number of invocations of a random function and the amount of key material used. Following the work of Lucks [89], Naor and Reingold [101] produce extremely efficient constructions with the help of universal hash functions and just two calls to a random function. In the present chapter, we describe a further generalization by using a different class of hash functions operating on only half the size of the input, in addition to a reduction in the key material. Is the end of progress in sight? We now discuss what it might mean for a Luby-Rackoff cipher to be optimal. We present various parameters of interest, and discuss how our proposal fits within this discussion.

1. Minimal number of rounds: In their original paper [88], Luby and Rackoff show that a cipher with only two Feistel rounds can easily be distinguished from random. In addition, the show that three rounds are necessary for pseudorandomness and four rounds are necessary for super pseudorandomness. Our construction also consists of four rounds.

2. Maximal security: Patarin [112] showed that the four-round Luby-Rackoff construction can be distinguished from a random permutation with probability $O(\frac{q^2}{2^n})$ with q queries. We meet this bound as stated in Theorem 1. We can reduce the distinguishing probability by increasing the number of rounds, but this would violate the previous criterion.

3. Minimal calls to cryptographic functions: Since the output of the block cipher is $2n$-bits long, it would seem that two calls to n-bit pseudorandom functions are necessary to ensure cryptographic security. We only make two pseudorandom function calls in rounds two and three respectively. For rounds one and four we use non-cryptographic ϵ_1-bisymmetric ϵ_2-almost Δ-universal hash functions, which results in a noticeable efficiency improvement.

4. Reusing the same pseudorandom function: It has been the goal of many papers to reduce the number of different pseudorandom functions that are used, ultimately hoping to use just one keyed pseudorandom function. Doing so saves key material, which may lead to an improvement in encryption / decryption time in practical implementations. We reuse the same key for the pseudorandom function in rounds two and three, so we achieve this goal as well.

5. Minimal data size operated on by non-cryptographic function: Our ϵ_1-bisymmetric ϵ_2-almost Δ-universal hash functions in rounds one and four operate on n-bits of the data. If we operated on any smaller part of the data then one could cause internal collisions with fewer queries via a birthday-type attack. The result would be an increase in the distinguishing probability and thus a decrease in security.

6. Reusing hash functions: It might be tempting to use the same universal hash function in rounds 1 and 4 (i.e. $\Psi(h, f, f, h)$) to save even more key material. However, using the same hash function h in both rounds, leads to an attack, which we now describe. First we encrypt a randomly chosen string $L_1 \cdot R_1$, where $L_1, R_1 \in I_n$. Following the equations for encryption:

$$\begin{aligned}
S_1 &= L_1 \oplus h(R_1); \\
T_1 &= R_1 \oplus f(S_1) \\
&= R_1 \oplus f(L_1 \oplus h(R_1)); \\
V_1 &= S_1 \oplus f(T_1) \\
&= L_1 \oplus h(R_1) \oplus f(R_1 \oplus f(L_1 \oplus h(R_1))); \\
W_1 &= T_1 \oplus h(V_1) \\
&= R_1 f(L_1 \oplus h(R_1)) \oplus h(L_1 \oplus h(R_1) \oplus f(R_1 \oplus f(L_1 \oplus h(R_1)))).
\end{aligned}$$

Now, we *decrypt* the same message, but with the left and right halves swapped. That is,

$$V_2 \cdot W_2 = R_1 \cdot L_1.$$

Following the equations for *decryption*, we get:

$$\begin{aligned}
T_2 &= W_2 \oplus h(V_2) \\
&= L_1 \oplus h(R_1); \\
S_2 &= V_2 \oplus f(T_2) \\
&= R_1 \oplus f(L_1 \oplus h(R_1)); \\
R_2 &= T_2 \oplus f(S_2) \\
&= L_1 \oplus h(R_1) \oplus f(R_1 \oplus f(L_1 \oplus h(R_1))); \\
L_2 &= S_2 \oplus h(R_2) \\
&= R_1 \oplus f(L_1 \oplus h(R_1)) \oplus h(L_1 \oplus h(R_1) \oplus f(R_1 \oplus f(L_1 \oplus h(R_1)))).
\end{aligned}$$

Now, observe that

$$L_2 \cdot R_2 = W_1 \cdot V_1$$

with certainty whenever the cipher is of the form $\Psi(h, f, f, h)$. If the cipher were truly random, this equation would only hold with negligible probability. Thus we have a distinguishing test. The attack only requires one chosen plaintext query and one chosen ciphertext query. Moreover, the attack is non-adaptive. The cipher is still insecure if we only allow plaintext queries (or only ciphertext queries), though requires the adversary to be adaptive. We illustrate another attack.

Suppose that we ask for the encryption of $L_1 \cdot R_1$, where this quantity is again chosen at random. Call the ciphertext output $V_1 \cdot W_1$. Now, we ask for the encryption of this quantity, but with the left and right halves swapped. That is,

$$L_2 \cdot R_2 = W_1 \cdot V_1.$$

Following the equations for encryption, we get:

$$\begin{aligned}
S_2 &= L_2 \oplus h(R_2) \\
&= W_1 \oplus h(V_1) \\
&= T_1; \\
T_2 &= R_2 \oplus f(S_2) \\
&= V_1 \oplus f(T_1) \\
&= S_1; \\
V_2 &= S_2 \oplus f(T_2) \\
&= T_1 \oplus f(S_1) \\
&= R_1; \\
W_2 &= T_2 \oplus h(V_2) \\
&= S_1 \oplus h(R_1) \\
&= L_1.
\end{aligned}$$

Consequently,

$$W_2 \cdot V_2 = L_1 \cdot R_1$$

which is our original plaintext. This equation holds with certainty whenever the underlying cipher is $\Psi(h, f, f, h)$, and only holds with negligible probability whenever the cipher is a truly random permutation. The attack only requires two plaintext queries, though is adaptive since the second query is a involves the answer of the first query.

These attacks are extremely effective since they require the adversary to have few computational resources. It thus seems that the cipher $\Psi(h, f, f, h)$ is not a terribly good choice. The attacks we give, however, rely on the *involutory* properties of the exclusive-or; that is, for any values a, b,

$$a \oplus b \oplus b = a$$

since $b \oplus b = 0$. In chapter 4 we show, surprisingly, that if we permit certain other operations besides the exclusive-or in the Feistel ladder, then we can, in fact, utilize the *same* hash functions in rounds 1 and 4, and create a *provably-secure* cipher of the form $\Psi(h, f, f, h)$.

3.6 Conclusion

In this chapter we have described some novel improvements to Luby-Rackoff ciphers. We introduced the concept of bisymmetric universal hash function families, which enable us to develop more efficient constructions. We also showed that under the weaker and more practical assumption that our round functions are unpredictable (rather than pseudorandom), the resulting ciphers are secure against certain inversion and forgery attacks. Finally, we gave an informal discussion on optimality criteria related to Luby-Rackoff ciphers and show that our cipher meets these criteria.

Chapter 4

Luby-Rackoff Ciphers Over Arbitrary Algebraic Structures

4.1 Introduction

In all of the previous Luby-Rackoff block cipher constructions given in the literature, the Feistel permutation takes as input the $2n$-bit block $L \cdot R$ and computes $R \cdot (L \oplus f(R))$ where f is the underlying round function. In this case, the Feistel permutation utilizes the exclusive-or operation. Stated abstractly: *all bit strings are treated as elements in the additive group of the Galois field of order* 2^n *and the Feistel permutation involves the addition operation in this group.* In this chapter we initiate the study of Luby-Rackoff ciphers over arbitrary algebraic structures. The hope is that utilizing different operations, with different algebraic properties, may yield new more powerful constructions.

Our efforts focus on the case of four-round Feistel networks that permute $2n$-bits of data as in the original Luby-Rackoff construction and some of its variants [88, 101, 114], such as the one we discuss in chapter 3. Our point of departure is that we treat the n-bit strings as elements in an arbitrary algebraic structure which is not necessarily $\mathrm{GF}(2^n)^+$.[1] This idea seems to open up new lines of research and we obtain results which are both surprising and useful. Specifically, we achieve the following:

- Our main result is the construction of a Luby-Rackoff style cipher, whose Feistel ladder

[1]Recall that $\mathrm{GF}(2^n)^+$ refers to the additive group attached to the field $\mathrm{GF}(2^n)$. The addition of two elements in this group amounts to computing their bit-wise exclusive-or.

operates over various *finite groups* of characteristic greater than 2, that is not only super-pseudorandom, but has better time/space complexity and uses fewer random bits than all previously considered Luby-Rackoff ciphers of equivalent security in the literature. Surprisingly, when we use the bit-wise exclusive-or operation instead, we can distinguish the cipher from random with near certainty using only two queries.

- We show that the requirements on our construction are precise when operations are performed in a *finite field.* In particular, eliminating one of the statistical requirements on the hash function used in the first and last round results in the cipher becoming distinguishable from random with near certainty using only a small constant number of queries.

- We examine various other Luby-Rackoff ciphers known to be insecure over $\mathrm{GF}(2^n)^+$. In some cases, we can break these ciphers over arbitrary *Abelian groups* – though we have to employ more complex attacks. In other cases, however, the security remains an open problem.

The main result is fairly surprising since then one can construct a Luby-Rackoff cipher that can be broken with two plaintext/ciphertext queries when the bit-wise exclusive-or is the group operation; yet, if one simply changes four of those bit-wise exclusive-or operations to, for example, additions mod 2^n, the cipher becomes completely secure against both adaptive chosen plaintext and ciphertext attacks. A more careful analysis shows that we only need to change the exclusive-or operation in the first and last round to addition mod 2^n to achieve security. Thus, in some sense, there are two very simple operations at the heart of the cipher's security.

Our main result utilizes the notion of a *monosymmetric* universal hash function. In particular the monosymmetric property does not really hold in groups of characteristic 2, which explains why our constructions fail to be secure in this case. Moreover our new constructions are practical; for example our constructions may work with addition modulo 2^n as the underlying operation, and we get security guarantees which are not otherwise attainable in the group $\mathrm{GF}(2^n)^+$ – even though both operations can be implemented very efficiently on most current microprocessors, and both are frequently used in popular block ciphers.

The impact of the underlying set and binary operation on bit strings with respect to a given problem can be seen in other settings, though we are not aware of any formal attempt to study it. Consider for example the *subset sum problem*:

Instance: A set S, of size n, consisting of elements from $\mathbb{Z}_{2^n}$, together with a value $B \in \mathbb{Z}_{2^n}$.

Problem: Find a subset $S' \subseteq S$ such that the sum of all the elements in S' (modulo 2^n) is exactly B:

$$\sum_{s \in S'} s \bmod 2^n = B$$

This problem is NP-hard when the inputs are treated as elements in the group of integers modulo 2^n [56, 74]. Yet, when we treat the inputs as elements of $\mathrm{GF}(2^n)^+$, the subset sum problem can be efficiently solved via a system of linear equations. In particular, suppose that the elements in our set S are $A_1, \ldots, A_n$. Each A_i can be represented as an n-bit quantity

$$A_i = A_i^1 A_i^2 \ldots A_i^n,$$

where each A_i^j is a single bit, for $1 \leq j \leq n$. Similarly, we can represent the bound B as:

$$B = B_1 \ldots B_n.$$

Now, consider the following linear system of equations over GF(2):

$$\begin{pmatrix} A_1^1 & A_2^1 & \ldots & A_n^1 \\ A_1^2 & A_2^2 & \ldots & A_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ A_1^n & A_2^n & \ldots & A_n^n \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{pmatrix}$$

By solving this system, we can construct the required subset S' as follows. For $1 \leq i \leq n$, if x_i is 1, then we include A_i in the subset S', and if x_i is 0, we do not include A_i in the subset S'. We thus have a procedure for finding the appropriate subset which requires solving a linear system of equations over a finite field. There are a number of efficient algorithms for accomplishing this task [40]. Thus the problem becomes much easier to solve over $\mathrm{GF}(2^n)^+$.[2]

The intuition behind this anomaly is that when we perform addition modulo 2^n on two bit strings, each bit of the output may be affected by several bits of each of the inputs. On the other hand, when we perform addition over $\mathrm{GF}(2^n)^+$, the i^{th} output bit is only a function of the i^{th} bits of each of the inputs. We see a similar phenomenon in our study of Luby-Rackoff ciphers over arbitrary algebraic structures and our results provide formal evidence for the intuition that groups of characteristic greater than 2 provide better mixing of data for cryptographic purposes.

The idea of modifying operations in existing block ciphers has been previously considered, though not studied in any formal context. For example, Biham and Shamir [23] show that replacing some of the exclusive-or operations in DES with additions modulo 2^n, makes their differential attack less powerful. Similarly, Carter, Dawson, and Nielsen [35] show a similar phenomenon when addition in DES is replaced by addition using a particular Latin Square. While these results show resistance to one particular type of attack, our results, in contrast, show provable security against *all attacks*. We thus pinpoint the precise reason that our cipher is secure.

The rest of this chapter is organized as follows. In section 4.2 we describe our main result in more detail. In the section thereafter, we give a detailed proof of security. In section 4.4 we give a brief discussion of monosymmetric hash functions, which are at the heart of our security analysis. In section 4.5 we show how to attack our cipher over arbitrary finite fields when the underlying hash functions do not satisfy the monosymmetric property. The next three sections discuss various Luby-Rackoff style ciphers known to be insecure when the underlying operation in the Feistel ladder is bit-wise exclusive-or, and show how to extend many of these results to the more general setting of arbitrary algebraic structures. In some cases, we cannot extend these results, so we leave these problems open. In the last section, we make some concluding remarks.

[2]We note that the above attack works over *any* finite field of *constant* characteristic greater than 2.

4.2 Main Result: A Minimal-Key Luby-Rackoff Cipher

The main result of this chapter is the construction of a Luby-Rackoff cipher which is secure when the underlying operation is addition in certain algebraic structures, but can be broken easily when addition is performed in $\mathrm{GF}(2^n)^+$.

We describe our construction. Suppose G is a group with additive notation. Recall that a group is a set of elements which is closed under a binary operation "+" and that satisfies the following properties:

1. All elements are associative under +:

$$\forall a, b, c \in G, (a + b) + c = a + (b + c).$$

2. There is an identity element 0 where:

$$\forall a \in G, a + 0 = 0 + a = a.$$

3. Every element $a \in G$ has a unique inverse element $-a$ where:

$$-a + a = a + -a = 0.$$

Furthermore if all elements commute under +:

$$\forall a, b \in G, a + b = b + a$$

then the group is said to be *Abelian.*

Let f be a function drawn from a family $\mathcal{F}$, with domain and range G. Let h be drawn from an ϵ_1-monosymmetric ϵ_2-almost Δ-universal family of hash functions. Remember that we define these families in chapter 2. Then, our construction is $\Psi(h, f, f, h)$ where addition in the underlying Feistel ladder is performed in the group G. Note that $\Psi(h, f, f, h)$ can be viewed as a permutation on $G \times G$. The security of this construction can be related in a precise manner to the parameters ϵ_1 and ϵ_2, and the pseudorandomness of the function family $\mathcal{F}$. It turns out that if this group G has characteristic 2, then $\epsilon_1 = 1$, and the cipher can easily be broken. On the other hand, for various other groups, the construction is provably secure. We now make some important observations about this construction:

1. The hash functions in the first and fourth rounds have the same randomly chosen key. Alternatively, we can replace h by f_2, where f_2 is a pseudorandom function keyed independently of f. Hence we also obtain the new result that $\Psi(f_2, f, f, f_2)$ is strongly pseudorandom as a corollary.

2. The pseudorandom functions in the second and third rounds have the *same* randomly chosen key (though this key should be chosen independently from the key for the hash functions).

3. If addition is performed over a group G of characteristic 2 (e.g. $G = \mathrm{GF}(2^n)^+$), then this cipher can easily be distinguished from random since it has involution-like properties; we discuss these properties in section 3.5 of chapter 3.

The cipher $\Psi(h, f, f, h)$ is a very efficient Luby-Rackoff style cipher: only two calls to the *same* pseudorandom function are made, the universal hash functions operate on half the input block, and the *same* universal hash function is used in the first and fourth rounds, which saves additional key material. We cite our main theorem:

Theorem 14 (Patel-Ramzan-Sundaram [115]) *Let G be a group, where each element of G can be represented by a bit string of length at most n. Let f be a function from a keyed function family $\mathcal{F}$ with domain and range G, and key space I_{s_1}. Let $h \in H$ be an ϵ_1-monosymmetric ϵ_2-almost Δ-universal hash function over the group G, with key space I_{s_2}. Let $\mathcal{B}$ be the family of permutations on $G \times G$, with key space $I_{s_1+s_2}$, consisting of permutations of the form $B = \Psi(h, f, f, h)$. Then:*

$$\mathsf{Adv}_{\mathcal{B}}^{\mathsf{sprp}}(q, t) \leq \mathsf{Adv}_{\mathcal{F}}^{\mathsf{prf}}(2q, t + O(s_2 + 2q(t_h + n))) + q^2\epsilon_1 + \binom{q}{2}\left(2\epsilon_2 + |G|^{-2}\right)$$

We remark that although Δ-universal hash functions are traditionally defined over Abelian groups, such a restriction is not necessary. One could easily extend the notion of Δ-universal hash functions over non-Abelian groups, and our above result would continue to hold for such groups.

We note that by modifying the appropriate definitions and using the same proof techniques, we can generalize these results to cases in which the each individual round involves a possibly different group operation. One particular result we can easily prove in this general model, using our techniques, is that $\Psi(h, f, f, h)$ is secure if the second and third rounds involve exclusive-or, but the first and fourth perform addition in certain other groups (for example the integers modulo 2^n). This is surprising since if we simply change two operations by making the first and fourth round use exclusive-or, then the cipher can be distinguished from random using only two queries. We also remark that our proof techniques do not utilize all of the group axioms. In fact, we only require an algebraic structure G in which:

- For all $a, b \in G$, there is one and only one value $x \in G$ such that $a + x = b$.
- For all $a, b \in G$, there is one and only one value $y \in G$ such that $y + a = b$.

Such an algebraic structure is called a *quasigroup*, and the addition table for a quasigroup can be represented by what is called a *latin square* [31]. However, since all of our concrete examples involve traditional groups, we prefer to cast our main result this way.

4.3 Proof of Security

We analyze the security of $\Psi(h, f, f, h)$ assuming addition is performed in an arbitrary group. We relate the security of the cipher to the value of ϵ attained by the monosymmetric ϵ-almost-Δ-universal hash function h. In the next section, we examine constructions of such functions over various algebraic structures.

We proceed in the standard manner by showing that our permutation $\Psi(h, f, f, h)$ is pseudorandom when f is truly random (instead of just pseudorandom). Our overall treatment, however, follows the nicely laid out framework of Naor and Reingold [101], but in the more general context of arbitrary finite groups. In particular we focus on proving the following theorem:

Theorem 15 (Patel-Ramzan-Sundaram [115]) *Let G be a group and let f be a function chosen uniformly from $\mathsf{Rand}^{G \to G}$. Let h be an ϵ_1-monosymmetric ϵ_2-almost-Δ-universal hash function with domain and range G, keyed independently of f, taken from a keyed function family H. Let $\mathcal{P}$ be the family of permutations on $G \times G$ consisting of permutations of the form $P = \Psi(h, f, f, h)$. Then:*

$$\mathsf{Adv}_{\mathcal{P}}^{\mathsf{sprp}}(q, t) \leq q^2 \epsilon_1 + \binom{q}{2} \left(2\epsilon_2 + |G|^{-2}\right).$$

The proof of theorem 15, like that of theorem 11 follows the framework of Naor and Reingold [101]. We start by recasting the original setting into the more general context of arbitrary finite groups. As usual, our adversary $\mathcal{A}$ is modeled as a program for a random access machine that gets black-box access to either a permutation uniformly sampled from $\mathsf{Perm}^{G\times G}$ or one sampled from the set of ciphers $P = \Psi(h, f, f, h)$. As was done previously, the adversary will have access to two oracles – one for computing each direction of the permutation. We denote a query for the forward direction of the permutation by $(+, x)$. Such a query asks to obtain the value $P(x)$. Similarly, we denote query in the reverse direction by $(-, y)$. Such a query asks to obtain the value for $P^{-1}(y)$. Like before, we write $L \cdot R$ to denote the left and right halves of the plaintext respectively, and we write $V \cdot W$ to denote the left and right halves of the ciphertext respectively. In this case, however, L and R are each elements of the group G, and we can think of $L \cdot R$ as an element of $G \times G$. The transcripts in our setting mimic those used in proof of theorem 11 of chapter 3. Also, in the following proof we make the standard assumption that the adversary $\mathcal{A}$ is deterministic.

We now consider a special random process for $\mathcal{A}$'s queries that will be useful to us. This random process is analogous to the one given in the proof of theorem 15 from chapter 3.

Definition 28 *The random process $\tilde{R}$ answers the i^{th} query of $\mathcal{A}$ as follows:*

1. *If $\mathcal{A}$'s query is $(+, x_i)$ and for some $1 \leq j < i$ the j^{th} query-answer pair is $\langle x_i, y_i \rangle$, then $\tilde{R}$ answers with y_i. If more than one such query-answer pair exists, we pick the one with the smallest index.*

2. *If $\mathcal{A}$'s query is $(-, y_i)$ and for some $1 \leq j < i$ the j^{th} query-answer pair is $\langle x_i, y_i \rangle$, then $\tilde{R}$ answers with x_i. If more than one such query-answer pair exists, we pick the one with the smallest index.*

3. *If neither of the above happens, then $\tilde{R}$ answers with a uniformly chosen pair $(g_1, g_2) \in G \times G$.*

As before, we note that $\tilde{R}$'s answers may not be consistent with any function, let alone any permutation. We formalize this concept.

Definition 29 *Let $\sigma = \langle (x_1, y_1), \ldots, (x_q, y_q) \rangle$ be any possible $\mathcal{A}$-transcript. We say that σ*

is inconsistent if for some $1 \leq j < i \leq q$ the corresponding query-answer pairs satisfy:

$$\begin{aligned} x_i = x_j \quad &\textit{and} \quad y_i \neq y_j, \textit{or} \\ x_i \neq x_j \quad &\textit{and} \quad y_i = y_j. \end{aligned}$$

Fortunately, we can show that the process $\tilde{R}$ often "behaves" exactly like a permutation over $G \times G$. It turns out that if $\mathcal{A}$ is given oracle access to either $\tilde{R}$ or $\mathsf{Perm}^{G\times G}$, it will have a negligible advantage in distinguishing between the two. We prove this more formally in proposition 4. Before proceeding, recall that we denote by the random variables $T_{\mathcal{P}}, T_{\mathsf{Perm}^{G\times G}}, T_{\tilde{R}}$ the transcript seen by $\mathcal{A}$ when its oracle queries are answered by $\mathcal{P}$, $\mathsf{Perm}^{G\times G}$, and $\tilde{R}$ respectively.

Proposition 4 *Let $\mathcal{A}$ be a 2-oracle adversary restricted to making a total of at most q queries to its oracles. Then:*

$$\Pr_{\tilde{R}}[C_{\mathcal{A}}(T_{\tilde{R}}) = 1] - \Pr_{\mathsf{Perm}^{G\times G}}[C_{\mathcal{A}}(T_{\mathsf{Perm}^{G\times G}}) = 1] \leq \binom{q}{2} \cdot |G|^{-2}.$$

Proof: The proof is analogous to the one given in proposition 1 from chapter 3, but encompasses arbitrary finite groups. We include the more general proof here for completeness.

First, let Con denote the event that $T_{\tilde{R}}$ is consistent, and let $\neg\mathsf{Con}$ denote the complement. Then, for any possible and consistent $\mathcal{A}$-transcript σ we have that:

$$\Pr_{\mathsf{Perm}^{G\times G}}[T_{\mathsf{Perm}^{G\times G}} = \sigma] = \frac{(|G| - q)!}{|G|!} = \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma \mid \mathsf{Con}].$$

Thus $T_{\mathsf{Perm}^{G\times G}}$ and $T_{\tilde{R}}$ have the same distribution conditioned on the event Con. We now bound the probability of $\neg\mathsf{Con}$. Recall that $T_{\tilde{R}}$ is inconsistent if there exists an i and j with $1 \leq j < i \leq q$ for which

$$\begin{aligned} x_i = x_j \quad &\text{and} \quad y_i \neq y_j, \text{or} \\ x_i \neq x_j \quad &\text{and} \quad y_i = y_j. \end{aligned}$$

For a particular i and j this event happens with probability $|G|^{-2}$. So,

$$\Pr_{\tilde{R}}[\neg\mathsf{Con}] \leq \binom{q}{2} \cdot |G|^{-2}.$$

We complete the proof via a standard argument:

$$
\begin{aligned}
\Pr_{\tilde{R}}[C_{\mathcal{A}}(T_{\tilde{R}}) = 1] - \Pr_{R}[C_{\mathcal{A}}(T_R) = 1] & \\
&= \left(\Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma \mid \mathsf{Con}] - \Pr_{R}[C_{\mathcal{A}}(T_R) = 1]\right) \cdot \Pr_{\tilde{R}}[\mathsf{Con}] \\
&+ \left(\Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma \mid \mathsf{Con}] - \Pr_{R}[C_{\mathcal{A}}(T_R) = 1]\right) \cdot \Pr_{\tilde{R}}[\neg\mathsf{Con}] \\
&= \left(\Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma \mid \mathsf{Con}] - \Pr_{R}[C_{\mathcal{A}}(T_R) = 1]\right) \cdot \Pr_{\tilde{R}}[\neg\mathsf{Con}] \\
&\leq \Pr_{\tilde{R}}[\neg\mathsf{Con}] \\
&\leq \binom{q}{2} \cdot |G|^{-2},
\end{aligned}
$$

which completes the proof. ■

We now proceed to obtain a bound on the advantage $\mathcal{A}$ will have in distinguishing between $T_{\mathcal{P}}$ and $T_{\tilde{R}}$. It turns out that $T_{\mathcal{P}}$ and $T_{\tilde{R}}$ are identically distributed unless some event depending on the choice of h in $\mathcal{P}$ occurs. We call this event Bad and obtain a bound on the probability that it actually occurs. Intuitively, Bad occurs whenever the internal function f in $\mathcal{P}$ would be evaluated on the exact same point twice for two distinct oracle queries – that is, whenever there is an internal collision. We formalize this as follows.

Definition 30 *For every specific monosymmetric ϵ-almost-Δ-universal hash function h, define* $\mathsf{Bad}(h)$ *to be the set of all possible and consistent $\mathcal{A}$-transcripts*

$$\sigma = \langle (L_1 \cdot R_1, V_1 \cdot W_1), \ldots, (L_q \cdot R_q, V_q \cdot W_q) \rangle$$

satisfying:

- *Event* B1*: there exists $1 \leq i < j \leq q$ such that $h(R_i) + L_i = h(R_j) + L_j$, or*
- *Event* B2*: there exists $1 \leq i < j \leq q$ such that $W_i - h(V_i) = W_j - h(V_j)$, or*
- *Event* B3*: there exists $1 \leq i, j \leq q$ such that $h(R_i) + L_i = W_j - h(V_j)$.*

Proposition 5 *Let $h \in H$ be an ϵ_1-monosymmetric ϵ_2-almost-Δ-universal hash function. Then, for any possible and consistent $\mathcal{A}$-transcript*

$$\sigma = \langle (L_1 \cdot R_1, V_1 \cdot W_1), \ldots, (L_q \cdot R_q, V_q \cdot W_q) \rangle,$$

we have that

$$\Pr_h[\sigma \in \mathsf{Bad}(h)] \leq 2\binom{q}{2} \cdot \epsilon_2 + q^2 \cdot \epsilon_1.$$

Proof: Recall that a transcript $\sigma \in \mathsf{Bad}(h)$ if event B1, event B2, or event B3 occurs. We can determine the individual probabilities of each of these events separately, and obtain an upper bound on the desired probability by taking the sum. We start with bounding the probability of the first event:

$$\begin{aligned}
\Pr_h[\mathsf{B1}] &\leq \Pr_h[\exists 1 \leq i < j \leq q : h(R_i) + L_i = h(R_j) + L_j] \\
&\leq \sum_{1 \leq i < j \leq q} \Pr_h[h(R_i) + L_i = h(R_j) + L_j] \\
&\leq \sum_{1 \leq i < j \leq q} \Pr_h[h(R_i) - h(R_j) = L_j - L_i] \\
&\leq \binom{q}{2} \cdot \epsilon_2.
\end{aligned}$$

The last inequality follows from the previous since if $R_i \neq R_j$ then we can use the fact that h is ϵ_2-almost Δ-universal, and if $R_i = R_j$ we know that $L_i \neq L_j$ since we assume the queries are distinct, in which case:

$$\Pr_h[h(R_i) - h(R_j) = L_j - L_i] = 0.$$

Now, we bound the probability of the second event:

$$\begin{aligned}
\Pr_h[\mathsf{B2}] &\leq \Pr_h[\exists 1 \leq i < j \leq q : W_i - h(V_i) = W_j - h(V_j)] \\
&\leq \sum_{1 \leq i < j \leq q} \Pr[W_i - h(V_i) = W_j - h(V_j)] \\
&\leq \sum_{1 \leq i < j \leq q} \Pr[h(V_j) - h(V_i) = W_j - W_i] \\
&\leq \binom{q}{2} \cdot \epsilon_2.
\end{aligned}$$

The last inequality follows from the previous since if $V_i \neq V_j$ then we can use the fact that h is ϵ_2-almost-Δ-universal, and if $V_i = V_j$ we know that $W_i \neq W_j$ since we assume the queries are distinct in which case $\Pr_h[h(V_i) - h(V_j) = W_j - W_i] = 0$. Finally, we must bound the probability of the third event:

$$\begin{aligned}
\Pr_h[\mathsf{B3}] &\leq \Pr_h[\exists 1 \leq i,j \leq q : h(R_i) + L_i = W_j - h(V_j)] \\
&\leq \sum_{1 \leq i,j \leq q} \Pr[h(R_i) + L_i = W_j - h(V_j)] \\
&\leq \sum_{1 \leq i,j \leq q} \Pr[h(R_i) + h(V_j) = W_j - L_i] \\
&\leq q^2 \cdot \epsilon_1.
\end{aligned}$$

Here the last inequality follows from the previous since h is ϵ_1-monosymmetric. We thus get that:

$$\begin{aligned}
\Pr_h[\sigma \in \mathsf{Bad}(h)] &\leq \Pr_h[\mathsf{B1}] + \Pr_h[\mathsf{B2}] + \Pr_h[\mathsf{B3}] \\
&\leq \binom{q}{2} \cdot \epsilon_2 + \binom{q}{2} \cdot \epsilon_2 + q^2 \cdot \epsilon_1 \\
&\leq 2\binom{q}{2} \cdot \epsilon_2 + q^2 \cdot \epsilon_1.
\end{aligned}$$

This concludes proof. ■

The following key lemma for proving theorem 15 shows that the distribution of possible and consistent transcripts generated by $T_{\mathcal{P}}$ given that the bad conditions do not occur is identical to the distribution of possible and consistent transcripts generated by $T_{\tilde{R}}$. This lemma will be useful when we try to determine a bound on the advantage our adversary $\mathcal{A}$ will have when trying to distinguish between these two cases in general; it is analogous to lemma 2, from chapter 3.

Lemma 3 *Let σ be any possible and consistent $\mathcal{A}$ − transcript, then*

$$\Pr_{\mathcal{P}}[T_{\mathcal{P}} = \sigma | \sigma \notin \mathsf{Bad}(h)] = \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma]. \tag{4.1}$$

Proof: First observe that $\Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma] = |G|^{-2q}$. This equality follows since $\tilde{R}$ picks elements from $G \times G$. Thus for a given fixed transcript entry, the probability that $\tilde{R}$ could generate it is $1/|G \times G|$ which equals $|G|^{-2}$. Now, for q consistent transcript entries, $\tilde{R}$ would generate them by picking q elements independently from $G \times G$, which gives us the desired probability of $|G|^{-2q}$.

Since σ is a possible $\mathcal{A}$-transcript, it follows that $T_{\mathcal{P}} = \sigma$ if and only if $V_i \cdot W_i = P(L_i \cdot R_i)$ for all $1 \leq i \leq q$. Next, suppose h is an ϵ_1-monosymmetric ϵ_2-almost-Δ-universal hash function for which $\sigma \notin \mathsf{Bad}(h)$. Now, we know that $L_i \cdot R_i$ and $V_i \cdot W_i$ must satisfy the following series of equations:

$$\begin{aligned} S_i &= L_i + h(R_i); \\ T_i &= R_i + f(S_i); \\ V_i &= S_i + f(T_i); \\ W_i &= T_i + h(V_i). \end{aligned}$$

So, in particular

$$(V_i, W_i) = P(L_i, R_i) \Leftrightarrow f(S_i) = T_i - R_i \text{ and } f(T_i) = V_i - S_i.$$

Now observe that for all $1 \leq i < j \leq q$, $S_i \neq S_j$ and $T_i \neq T_j$ (otherwise $\sigma \in \mathsf{Bad}(h)$). Similarly, for all $1 < i, j < q$, $S_i \neq T_j$. So, if $\sigma \notin \mathsf{Bad}(h)$ all the inputs to f are distinct. Since f is a random function, for every specific choice of h such that $\sigma \notin \mathsf{Bad}(h)$ the probability that $T_{\mathcal{P}} = \sigma$ is exactly $|G|^{-2q}$. Therefore:

$$\Pr_{\mathcal{P}}[T_{\mathcal{P}} = \sigma | \sigma \notin \mathsf{Bad}(h)] = |G|^{-2q},$$

which completes the proof of the lemma. ■

To complete the proof of the main theorem, we can follow the Naor-Reingold framework utilizing the above lemma where appropriate. Before doing so, we state and prove a preliminary proposition.

Proposition 6 *Let Γ be the set of all possible and consistent transcripts σ such that $C_{\mathcal{A}}(\sigma) = 1$. Then:*

$$\mathsf{Adv}(\langle \mathcal{P}, \mathcal{P}^{-1}\rangle, \langle \mathsf{Perm}^{G\times G}, (\mathsf{Perm}^{G\times G})^{-1}\rangle) \leq \sum_{\sigma\in\Gamma}(\Pr_{\mathcal{P}}[T_{\mathcal{P}} = \sigma] - \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma]) + \binom{q}{2} \cdot |G|^{-2},$$

where $\langle \mathcal{P}, \mathcal{P}^{-1}\rangle$ and $\langle \mathsf{Perm}^{G\times G}, (\mathsf{Perm}^{G\times G})^{-1}\rangle$ both represent a pair of oracles: one which computes the forward direction of the permutation, and the other which computes the inverse direction of the permutation.

Proof: First, we break up the left hand side of the above inequality so that the distinguishability of both distributions are taken with respect to the process $\tilde{R}$:

$$\begin{aligned}&\mathsf{Adv}(\langle \mathcal{P}, \mathcal{P}^{-1}\rangle, \langle \mathsf{Perm}^{G\times G}, (\mathsf{Perm}^{G\times G})^{-1}\rangle)\\ &\quad\leq \mathsf{Adv}(\langle \mathcal{P}, \mathcal{P}^{-1}\rangle, \tilde{R}) + \mathsf{Adv}(\tilde{R}, \langle \mathsf{Perm}^{G\times G}, (\mathsf{Perm}^{G\times G})^{-1}\rangle)\end{aligned} \tag{4.2}$$

Now, since $\mathcal{A}$ with an oracle for $\mathcal{P}$ only generates possible and consistent transcripts, we can apply theorem 1 from chapter 2 to the first summand in equation 4.2. Next, we can apply proposition 4 from this chapter to the second summand. The proposition follows. ■

We now prove the main theorem:

Proof: (of Theorem 15) First we successively apply proposition 6 given above, and theorem 2 from chapter 2 with the set $\mathsf{Bad}(h)$:

$$\begin{aligned}&\mathsf{Adv}(\langle \mathcal{P}, \mathcal{P}^{-1}\rangle, \langle \mathsf{Perm}^{G\times G}, (\mathsf{Perm}^{G\times G})^{-1}\rangle)\\ &\quad\leq \sum_{\sigma\in\Gamma}(\Pr_{\mathcal{P}}[T_{\mathcal{P}} = \sigma] - \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma]) + \binom{q}{2} \cdot |G|^{-2}.\end{aligned}$$

Now, applying Theorem 2:

$$\begin{aligned}&\sum_{\sigma\in\Gamma}(\Pr_{\mathcal{P}}[T_{\mathcal{P}} = \sigma] - \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma])\\ &\quad\leq \sum_{\sigma\in\Gamma}\left(\Pr_{\mathcal{P}}[T_{\mathcal{P}} = \sigma \mid \sigma \notin \mathsf{Bad}(h)] - \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma]\right) \cdot \Pr_{\mathcal{P}}[\sigma \notin \mathsf{Bad}(h)]\\ &\quad + \Pr_{\mathcal{P}}[\sigma^* \in \mathsf{Bad}(h)].\end{aligned}$$

Now, observe that according to lemma 3:

$$\left(\Pr_{\mathcal{P}}[T_{\mathcal{P}} = \sigma \mid \sigma \notin \mathsf{Bad}(h)] - \Pr_{\mathsf{Perm}^{G\times G}}[T_{\mathsf{Perm}^{G\times G}} = \sigma]\right) \cdot \Pr_{\mathcal{P}}[\sigma \notin \mathsf{Bad}(h)] = 0.$$

In addition, observe that proposition 5 tells us that

$$\Pr_{\mathcal{P}}[\sigma^* \in \mathsf{Bad}(h)] \leq q^2\epsilon_1 + 2\binom{q}{2} \cdot \epsilon_2.$$

Combining these observations, we get:

$$\begin{aligned}\mathsf{Adv}(\langle \mathcal{P}, \mathcal{P}^{-1}\rangle, \langle \mathsf{Perm}^{G\times G}, (\mathsf{Perm}^{G\times G})^{-1}\rangle)& \\ \leq\ & q^2\epsilon_1 + \binom{q}{2}\left(2\epsilon_2 + |G|^{-2}\right),\end{aligned}$$

which completes the proof. ■

Thus we have proven theorem 15. Theorem 14 follows as a corollary using fairly standard techniques. The proof looks nearly identical to the proof of theorem 10 given in chapter 3.

4.4 Monosymmetric Δ-Universal Hash Functions over various Groups

In the previous section we construct a Luby-Rackoff style cipher that uses the same hash function h in rounds one and four, and the same round function f rounds two and three. The security of this construction rests upon the pseudorandomness of the round function f, and the parameters ϵ_1, ϵ_2 associated with the hash function h.

In this section we initially demonstrate that over certain algebraic structures, small values of ϵ_1, ϵ_2 are easy to attain. Next, we show that in other groups, the value of ϵ_1 will always be quite large. Our first example concerns the family $\mathsf{Rand}^{G\to G}$, for a group G.

Lemma 4 *Let G be any group in which no element has order 2. Then, the set of all possible functions $\mathsf{Rand}^{G\to G}$ is $1/|G|$-monosymmetric $1/|G|$-almost-Δ-universal.*

Proof: First we show that $\mathsf{Rand}^{G\to G}$ is $1/|G|$-almost-Δ-universal. Consider three values $x, y, \delta \in G$, with $x \neq y$. The values $f(x)$ and $f(y)$ are uniformly distributed when f is chosen at random from $\mathsf{Rand}^{G \to G}$. Thus, their difference is uniformly distributed, and takes on any value in the range G with equal probability. Consequently,

$$\forall x \neq y \in G, \forall \delta \in G : \Pr[f \xleftarrow{R} \mathsf{Rand}^{G \to G} : f(x) - f(y) = \delta] = 1/|G|.$$

Now we examine the monosymmetric property. Again, we pick values $x, y, \delta \in G$. There are two cases to consider. If $x \neq y$, then using an argument very similar to the one above, we get:

$$\forall x \neq y \in G, \forall \delta \in G : \Pr[f \xleftarrow{R} \mathsf{Rand}^{G \to G} : f(x) + f(y) = \delta] = 1/|G|.$$

The only remaining case is when $x = y$. In this case, $f(x) + f(y) = 2f(x)$. Thus we are then left with an equation of the form $2f(x) = \delta$. We claim that there is at most one value of $f(x)$ that satisfies this equation. If this claim were true, then we are done since $f(x)$ is uniformly distributed since f is chosen at random from $\mathsf{Rand}^{G \to G}$. We now prove the claim by contradiction. Suppose that there are two values x_1, x_2 such that $2f(x_1) = 2f(x_2) = \delta$, but $f(x_1) \neq f(x_2)$. Then, it follows that $2(f(x_1) - f(x_2)) = 0$. However, $f(x_1) - f(x_2) \in G$, and $f(x_1) - f(x_2) \neq 0$. This contradicts the assumption that G does not contain any element of order 2. ■

Unfortunately, it is difficult to efficiently sample from $\mathsf{Rand}^{G \to G}$ because it contains $|G|^{|G|}$ elements, which is quite large for our purposes. Though, if one looks at the above proof, it is not hard to see that we did not need truly random functions. Instead, strongly universal families of functions suffice for the proof since they appear random whenever one considers only two input / output pairs. We thus have the following corollary.

Corollary 1 *Let G be any group in which no element has order 2. Then any strongly universal family of functions is $1/|G|$-monosymmetric $1/|G|$-almost-Δ-universal.*

If G is the additive group attached to a finite field F, then one such family of strongly universal hash functions is the linear congruential hash family: $h_{a,b}(x) = ax + b$ where arithmetic is performed in F. Since there are finite fields, of characteristic greater than 2, of size p^k for any odd prime p, and any natural number k, we can construct good monosymmetric Δ-universal hash function families for sets of these size. We now show that we can construct reasonably good families of such hash functions for sets of *any* size.

Lemma 5 *Let m be a natural number, and let G be the cyclic group $\mathbb{Z}_m$. Let p be the smallest prime such that $m \leq p$. Let H be any ϵ_1-monosymmetric ϵ_2-almost-Δ-universal hash function family over the additive group attached to the finite field F_p. Consider the family of functions H', which is defined as follows:*

$$H' = \{h'_a : \mathbb{Z}_m \to \mathbb{Z}_m \mid a \in \mathsf{Keys}(H)\},$$

where the functions h'_a are defined as:

$$h'_a(x) = h_a(x) \bmod m.$$

Here h_a is chosen from H according to key a. We are also using the natural representation of elements in $\mathbb{Z}_m$ and $\mathbb{Z}_p$ whereby we utilize the smallest non-negative integer. Then H' is $4\epsilon_1$-monosymmetric $4\epsilon_2$-almost-Δ-universal over the group $\mathbb{Z}_m$.

Proof: We first start with Bertrand's postulate,[3] which states that for each integer $m \geq 2$, there is a prime number p with $m \leq p < 2m$. First, we examine the Δ-universal property. Let $x, y, \delta \in \mathbb{Z}_m$ be chosen arbitrarily, with $x \neq y$. Let a be a key such that $h'_a(x) - h'_a(y) = \delta$. Equivalently:

$$(h_a(x) - h_a(y)) \bmod m = \delta. \tag{4.3}$$

Now, since H operates over the additive group attached to the finite field F_p, it follows that $h_a(x), h_a(y) \in \{0, \dots, p-1\}$. Therefore,

$$h_a(x) - h_a(y) \in \{-p+1, \dots, p-1\} \tag{4.4}$$

Moreover, by equation (4.3) given above, it follows that:

$$h_a(x) - h_a(y) \in \{\delta, \delta + m, \delta - m, \delta + 2m, \delta - 2m, \dots\}.$$

Now, the set of possible differences is finite, so there must be some least possible element. Denote this element by l_0. It must be of the form $l_0 = \delta + im$, for some integer i. Consider values of the form $l_r = \delta + (i+r)m$, where $r \geq 4$. We claim such values can never occur as a

[3]A proof of Bertrand's postulate can be found in the classic number theory text of Hardy and Wright [66]. The proof given there is due to Erdös, and is much simpler than the original proof given by Chebyshev.

possible difference. This follows from the fact that $l_r - l_0 \geq rm \geq 4m > 2p$. But, according to equation (4.4), there are at most $2p$ values in the range. Thus, l_r cannot appear as a difference for $r \geq 4$. This observation implies that there at most four candidate values for the difference. Thus,

$$\begin{aligned}
&\Pr[a \xleftarrow{R} \mathsf{Keys}(H) : h'_a(x) - h'_a(y) = \delta] \\
&\quad \leq \Pr[a \xleftarrow{R} \mathsf{Keys}(H') : h_a(x) - h_a(y) \in \{l_0, l_1, l_2, l_3\}] \\
&\quad \leq \Pr[a \xleftarrow{R} \mathsf{Keys}(H) : h_a(x) - h_a(y) = l_0] \\
&\quad + \Pr[a \xleftarrow{R} \mathsf{Keys}(H) : h_a(x) - h_a(y) = l_1] \\
&\quad + \Pr[a \xleftarrow{R} \mathsf{Keys}(H) : h_a(x) - h_a(y) = l_2] \\
&\quad + \Pr[a \xleftarrow{R} \mathsf{Keys}(H) : h_a(x) - h_a(y) = l_3] \\
&\quad \leq 4\epsilon_2
\end{aligned}$$

The last equation follows from the previous since H is ϵ_2-almost-Δ-universal. Thus, we have shown that H' is $4\epsilon_2$-almost-Δ-universal. We can use the same technique to show that H' is $4\epsilon_1$-monosymmetric. ■

In specific cases we may be able to get a tighter bound by exploiting either the algebraic structure of the hash function itself or the relationship between p and m (for example, if $m < p \leq 3m/2$, then we can achieve values of $3\epsilon_1$ and $3\epsilon_2$). For the case $m = 2^n$, these functions are especially important since addition modulo 2^n is easily implemented on most processors. Therefore, our constructions have very important practical implications. Another very efficient family of monosymmetric ϵ-almost-Δ-universal hash functions for which ϵ is small ($2/2^n$) is the square hash family [49], which we discuss in great detail in chapter 6.

We now consider groups for which there are no good families of monosymmetric-Δ universal hash functions. The most striking example occurs in the additive group of the Galois Field of order 2^n, $\mathrm{GF}(2^n)^+$. In particular, if H is *any* family of functions whose range is $\mathrm{GF}(2^n)^+$, then $\Pr_{h \in H}[h(x) + h(y) = \delta] = 1$ whenever $x = y$ and $\delta = 0$. Thus, it is not possible to get a value of ϵ_1 smaller than 1 for the monosymmetry property. Another example of a group which may not possess good monosymmetry properties is the *multiplicative* group modulo m, $\mathbb{Z}_m^*$. In this case, when we set $x = y$, then the expression

$\Pr_{h \in H}[h(x) \cdot h(y) = \delta]$ may be very high if δ has many square roots. In particular, if the prime factorization of m consists of k distinct odd primes $p_1, \ldots, p_k$ then it follows from the Chinese Remainder Theorem that certain elements of $\mathbb{Z}_m^*$ may have up to 2^k square roots. Thus it might be difficult to construct good examples of monosymmetric ϵ-almost-Δ-universal hash functions over the multiplicative group modulo m if there are many primes in m's factorization. There are many other groups one may want to examine – an extensive study of this topic is beyond the scope of this thesis.

4.5 Necessity of the Monosymmetric Property

In this section we provide evidence that our minimal-key construction is fairly optimal, and that the monosymmetry property is needed. In particular, we show that $\Psi(h_1, f, f, h_2)$ is not necessarily secure if h_1 and h_2 are independently keyed Δ-universal hash functions that do not satisfy the additional monosymmetry property. Note that in this case, we consider a cipher for which h_1 and h_2 may be different hash functions, so our attack is more general. The attack also works if they are the same hash function. In chapter 3, we show that this particular cipher could be broken when operations were performed in $\mathrm{GF}(2^n)^+$; in this chapter we extend the result to the case when operations are performed over arbitrary *finite fields*, though we need to resort to different techniques to do so. We note that this result is shown not for arbitrary finite groups, but for arbitrary finite fields. Extending it to hold for arbitrary groups is left as an open problem.

We describe the attack. Suppose that h_1 and h_2 are taken from the linear hash family. That is $h_1(x) = a_1 \cdot x$ and $h_2(x) = a_2 \cdot x$, where multiplication is performed with respect to the underlying finite field. This family is known to be Δ-universal, and we show this in section 3.2 of chapter 3. Pick a value α at random, and obtain both the encryption of $x = \alpha \cdot 0$ and the decryption of $0 \cdot \alpha$. Working through the equations for the encryption of $x = \alpha \cdot 0$:

$$\begin{aligned} S_1 &= h_1(R_1) + L_1 \\ &= h_1(0) + \alpha \\ &= 0 + \alpha \end{aligned}$$

$$\begin{aligned}
&= \alpha; \\
T_1 &= f(S_1) + R_1 \\
&= f(\alpha); \\
V_1 &= f(T_1) + S_1 \\
&= f^2(\alpha) + \alpha; \\
W_1 &= h_2(V_1) + T_1 \\
&= a_2 \cdot V_1 + f(\alpha).
\end{aligned}$$

Now we work through the equations for *decrypting* $V_2 \cdot W_2 = 0 \cdot \alpha$:

$$\begin{aligned}
T_2 &= W_2 - h_2(V_2) \\
&= \alpha - 0 \\
&= \alpha; \\
S_2 &= V_2 - f(T_2) \\
&= 0 - f(\alpha) \\
&= -f(\alpha); \\
R_2 &= T_2 - f(S_2) \\
&= \alpha - f(-f(\alpha)); \\
L_2 &= S_2 - h_1(R_2) \\
&= -f(\alpha) - a_1 \cdot R_2.
\end{aligned}$$

Next, let $A_1 = L_2 + W_1$. Observe that:

$$\begin{aligned}
A_1 &= L_2 + W_1 \\
&= a_2 \cdot V_1 - a_1 \cdot R_2.
\end{aligned}$$

Now, we repeat the same process as above. In particular, we ask for the encryption of $L_3 \cdot R_3 = \alpha' \cdot 0$ and call the result $V_3 \cdot W_3$. We also ask for the decryption of $V_4 \cdot W_4 = 0 \cdot \alpha'$ and call the result $L_4 \cdot R_4$. Let A_2 denote $L_4 + W_3$. By an argument similar to the one given above, $A_2 = a_2 \cdot V_3 - a_1 \cdot R_4$. We now have a system of equations:

$$\begin{aligned} -a_1 \cdot R_2 + a_2 \cdot V_1 &= A_1; \\ -a_1 \cdot R_4 + a_2 \cdot V_3 &= A_2. \end{aligned}$$

Since we know $R_2, R_4, V_1, V_3, A_1, A_2$ the only unknowns are a_1, a_2. With high probability, this system of equations has full rank, and we can solve for a_1 and a_2. This procedure allows us to compute the keys to the hash functions in the first and fourth rounds. Knowing these keys reduces the problem to distinguishing the two round Luby-Rackoff cipher $\Psi(f, f)$ from random, which can easily be done in two queries [88].

4.6 Attacking the Three-Round Luby Rackoff Cipher over Arbitrary Abelian Groups

Luby and Rackoff show that the three-round variant of their cipher $\Psi(f_1, f_2, f_3)$ is pseudorandom, but not super pseudorandom [88]. In particular, one can distinguish the cipher from random with high probability by making two plaintext queries, and one ciphertext query. We generalize their attack to work when the operation in the Feistel ladder is addition over an arbitrary Abelian group G. We stress that the cipher is still pseudorandom over these other algebraic structures – it is just not super pseudorandom. We describe the attack.

```
Description of attack against $\Psi(f_1, f_2, f_3)$.

1. Choose a random plaintext: $(L_1, R_1) \overset{R}{\leftarrow} G \times G$.
2. Query for the encryption of $L_1 \cdot R_1$ and call the result $T_1 \cdot V_1$.
3. Choose a value $L_2$ at random: $L_2 \overset{R}{\leftarrow} G$.
4. If $L_2 = L_1$ repeat the above step.
5. Query for the encryption of $L_2 \cdot R_1$ and call the result $T_2 \cdot V_2$.
6. Query for the decryption of $T_2 \cdot (V_2 + L_1 - L_2)$, and call the result $L_3 \cdot R_3$.

If $R_3 = T_2 + R_1 - T_1$ then
    return that the cipher is not random
else
    return that the cipher is random.
```

We claim that this attack works with overwhelming probability. The analysis now follows. Assume the underlying cipher is the three-round Luby-Rackoff cipher $\Psi(f_1, f_2, f_3)$. We denote the three round Luby-Rackoff cipher in the usual manner. Suppose that we want to encrypt $L_i \cdot R_i$. The process can be described by the following equations:

$$\begin{aligned} S_i &= L_i + f_1(R_i); \\ T_i &= R_i + f_2(S_i); \\ V_i &= S_i + f_3(T_i). \end{aligned}$$

The ciphertext output is $T \cdot V$. Now, the process of decrypting the pair $T \cdot V$ can be described by the following equations:

$$\begin{aligned} S_i &= V_i - f_3(T_i); \\ R_i &= T_i - f_2(S_i); \\ L_i &= S_i - f_1(R_i). \end{aligned}$$

The plaintext output is $L \cdot R$. Now, when we carry out the attack previously mentioned, we get the following crucial observation:

$$\begin{aligned} S_1 - S_2 &= (L_1 + f(R_1)) - (L_2 + f(R_1)) \\ &= L_1 - L_2. \end{aligned} \tag{4.5}$$

which follows since $S_1 = L_1 + f_1(R_1)$, and $S_2 = L_2 + f_2(R_1)$. Let us observe what happens when we make the ciphertext query (query number 3). Decrypting $T_2 \cdot (V_2 + L_1 - L_2)$ we get:

$$\begin{aligned} S_3 &= V_3 - f_3(T_3) \\ &= V_2 + L_1 - L_2 - f_3(T_2) \\ &= S_2 + L_1 - L_2 \\ &= S_1. \end{aligned}$$

The third equation follows from the second since $S_2 = V_2 - f_3(T_2)$ by definition. Also, the last equation follows from the previous by our crucial observation (equation 4.5). Continuing on with the decrypting process:

$$\begin{aligned} R_3 &= T_3 - f_2(S_3) \\ &= T_2 - f_2(S_1). \end{aligned}$$

The last equation follows from the previous since $S_3 = S_1$. Now, observe that:

$$\begin{aligned} R_3 + T_1 &= (T_2 - f_2(S_1)) + (R_1 + f_2(S_1)) \\ &= T_2 + R_1. \end{aligned}$$

Therefore,

$$R_3 = T_2 + R_1 - T_1.$$

So, if the cipher is the three-round Luby-Rackoff cipher $\Psi(f_1, f_2, f_3)$, the above property holds with certainty (and we can check it since R_1, R_3, T_1, T_2 are all known). If the cipher were a random permutation, this property would hold with negligible probability.

From the above equations, one can see that the attack requires the group to be commutative. We are unable to develop an attack that works for non-Abelian groups, in general. At the same time, we note that in certain cases, there are non-Abelian groups which are still very "commutative." Consider, for example, the dihedral group D_{2n}, which represents the group of symmetries (rotations and flips) of a regular n-gon [70]. In this case, any two rotations with no flips commute with each other. Thus, if two elements are picked uniformly at random from D_{2n}, they commute with probability at least $1/4$. In fact, the probability is higher since other randomly chosen pairs of elements commute; for example, every element commutes with itself. We can thus extend our above attack to work for Dihedral groups, though the success probability will diminish by a constant factor.

4.7 Attacking $\Psi(f, f, f)$

Rueppel [129] shows that $\Psi(f, f, f)$, the three-round Feistel cipher in which all the round functions are identical, is not even pseudorandom when the operation in the Feistel ladder is the bit-wise exclusive-or. The idea behind his attack is that when we use the same function f in all three rounds and addition is performed in a group of characteristic 2, $\Psi(f, f, f)$ has certain involution-like properties. Rueppel left open the problem of generalizing his attack to Feistel ladders that operate over other algebraic structures.

When operations are performed over an arbitrary algebraic structure, the involution-like properties that Rueppel exploits in his original attack no longer seem to hold. It turns out that this cipher is still insecure, but requires a different attack. We emphasize that the attack requires the underlying group to be Abelian – we leave as open the problem of determining the security when non-Abelian groups are considered. We present this attack here.

```
Description of attack against $\Psi(f,f,f)$.

1. Query for the encryption of $0 \cdot 0$ and call the result $T_1 \cdot V_1$.
2. Query for the encryption of $0 \cdot T_1$ and call the result $T_2 \cdot V_2$.
3. Query for the encryption of $(V_1 - V_2) \cdot T_2$ and call the result $T_3 \cdot V_3$.

If $T_3 = T_1 + T_2$ then
        return that the cipher is not random
else
        return that the cipher is random.
```

Here is the analysis. We show that if the cipher is really $\Psi(f,f,f)$ then this condition holds with certainty; if the cipher is a truly random permutation, then the condition $T_3 = T_1 + T_2$ holds with negligible probability. This gives us a distinguisher. Now we show why $T_3 = T_1 + T_2$ with certainty when the cipher is $\Psi(f,f,f)$. First let $L_1 \cdot R_1 = 0 \cdot 0$. Then, following the equations for encryption of a three-round Luby-Rackoff cipher:

$$\begin{aligned} S_1 &= f(0); \\ T_1 &= f^2(0); \\ V_1 &= f^3(0) + f(0). \end{aligned}$$

Now, we let $L_2 \cdot R_2 = 0 \cdot T_1$. When we encrypt:

$$\begin{aligned} S_2 &= f(R_2) + L_2 \\ &= f(T_1) \\ &= f^3(0); \\ T_2 &= f(S_2) + R_2 \\ &= f^4(0) + f^2(0); \\ V_2 &= f(T_2) + S_2 \end{aligned}$$

$$= f(T_2) + f^3(0).$$

Finally, when we encrypt $L_3 \cdot R_3 = (V_1 - V_2) \cdot T_2$ we get:

$$\begin{aligned}
S_3 &= f(R_3) + L_3 \\
&= f(T_2) + (V_1 - V_2) \\
&= f(T_2) + (f^3(0) + f(0)) - (f(T_2) + f^3(0)) \\
&= f(0) \\
&= S_1; \\
T_3 &= f(S_3) + R_3 \\
&= f(S_1) + R_3 \\
&= f^2(0) + T_2 \\
&= T_1 + V_2.
\end{aligned}$$

Thus $V_3 = T_1 + V_2$. This event happens with probability 1 when the underlying cipher is $\Psi(f, f, f)$ but happens with probability that is exponentially small in the underlying group size when the underlying cipher is truly random. The exact same attack can be used to show that $\Psi(f_1, f_2, f_1)$ is insecure, where f_1 and f_2 are independently-keyed random functions.

4.8 Attacking $\Psi(f^i, f^j, f^k)$ When k is Multiple of $i + j$

Another interesting class of ciphers is $\Psi(f^i, f^j, f^k)$ where f is a pseudorandom function, and f^i represents the i-fold composition of f with itself. Zheng, Matsumoto, and Imai [145] show how to break this class of ciphers for arbitrary i, j, k, when working over $\mathrm{GF}(2^n)^+$. The attack more heavily depends on the involutory properties of $\mathrm{GF}(2^n)^+$. When considering arbitrary finite groups, we know how to break the cipher only when i, j, and k satisfy certain relations; in particular, when k is a multiple of $i + j$. We are unable to produce an attack that works in general, nor are we able to prove security in any of the other cases. We now describe the attack.

```
Description of attack against $\Psi(f^i, f^j, f^k)$ where $k$ is a multiple of $i+j$.

1. Let $\alpha = k/(i+j)$.
2. Query for the encryption of $(0,0)$ and call the result $(T_1, V_1)$.
3. Query for the encryption of $(0, T_1)$ and call the result $(T_2, V_2)$.
4. For $l = 3$ to $\alpha + 1$ do
        4a. Query for the encryption of $(0, T_{l-1} - T_{l-2})$ and call the result $(T_l, V_l)$.
5. Query for the encryption of $(T_{\alpha+1} - T_\alpha - V_1, 0)$, and call the result $(T_{\alpha_2}, V_{\alpha_2})$.
6. Query for the encryption of $(-T_1, T_{\alpha+2})$, and call the result $(T_{\alpha_3}, V_{\alpha_3})$.
If $T_{\alpha+3} = T_1 + T_1$ then
        return that the cipher is not random
else
        return that the cipher is random.
```

We claim that this attack works with high probability. Here is the analysis. It is not hard to see that if the cipher were truly random, then the above attack would return the correct answer with probability extremely close to 1. We now claim that if the cipher is of the form $\Psi(f^i, f^j, f^k)$ with $k = \alpha(i+j)$ then the above attack will always output that the cipher is not random. First, consider what happens when we encrypt $0 \cdot 0$:

$$\begin{aligned} S_1 &= f^i(0); \\ T_1 &= f^{i+j}(0); \\ V_1 &= f^i(0) + f^{i+j+k}(0). \end{aligned}$$

Next, consider what happens when we encrypt $0 \cdot T_1$:

$$\begin{aligned} S_2 &= f^i(T_1); \\ T_2 &= f^{i+j}(T_1) + T_1; \\ V_2 &= f^i(T_1) + f^k(f^{i+j}(T_1) + T_1). \end{aligned}$$

Thus, $T_2 - T_1 = f^{i+j}(T_1) = f^{2(i+j)}(0)$. In general, when we encrypt $0 \cdot x$:

$$\begin{aligned} S &= f^i(x); \\ T &= f^{i+j}(x) + x; \\ V &= S + f^k(f^{i+j}(x) + x). \end{aligned}$$

Thus,

$$T - x = f^{i+j}(x).$$

Using this observation, we see that:

$$\begin{aligned} T_{\alpha+1} - T_\alpha &= f^{(i+j)(\alpha+1)}(0) \\ &= f^{(i+j)\alpha+(i+j)}(0) \\ &= f^{i+j+k}(0). \end{aligned}$$

Thus $T_{\alpha+1} - T_\alpha - V_1 = -f^i(0)$. During query $\alpha + 2$ this value is on the left hand side of the encryption, and 0 is on the right hand side. The result of this encryption is:

$$\begin{aligned} S_{\alpha+2} &= 0; \\ T_{\alpha+2} &= f^j(0); \\ V_{\alpha+2} &= f^{k+j}(0). \end{aligned}$$

Finally, when we encrypt $(-T_1, T_{\alpha+2})$, we get:

$$\begin{aligned} S_{\alpha+3} &= 0; \\ T_{\alpha+3} &= f^j(0) + f^j(S_{\alpha_3}); \\ V_{\alpha+3} &= f^k(f^j(0) + T_{\alpha+2}). \end{aligned}$$

Which implies that

$$T_{\alpha+3} = f^j(0) + f^j(0)$$

$$= T_1 + T_1.$$

The claim now follows.

4.9 Conclusion

This chapter initiated a study of Luby-Rackoff ciphers over arbitrary finite algebraic structures. In particular, we surprisingly discovered that there are certain Luby-Rackoff cipher constructions which are secure when the Feistel operation is taken with respect to certain groups, though become insecure when operations are taken with respect to other groups. For example, when we replace bit-wise exclusive-or by addition modulo 2^n we turn an insecure cipher into a secure one. Our results spawn new areas for research, and motivate a need to re-examine the literature on Luby-Rackoff ciphers and determine the extent to which the old results hold when we look at arbitrary finite algebraic structures.

Chapter 5

On the Round Security of Symmetric-Key Cryptographic Primitives

5.1 Introduction

In this chapter we put forward a new model for analyzing the security of symmetric-key cryptographic primitives, such as block ciphers. The model involves giving the adversary black-box access to some of the internal keyed components of the primitive; it captures the fact that many such primitives often consist of iterating simpler constructs for a number of rounds, and may provide insight into the security of such designs. The model can be though of as a refinement to the traditional model of giving the adversary black-box access to the entire primitive.

We completely characterize the security of four-round Luby-Rackoff style ciphers in our model, and show that the ciphers remain secure even if the adversary is given black-box access to the middle two round functions. Our results not only apply to the four-round Feistel construction that appeared in the original paper of Luby and Rackoff [88], but also to some of the variants that we discussed in chapters 3 and 4. We can also apply these techniques to study message authentication codes based on universal hash functions. The results in this chapter appear in a paper by Ramzan and Reyzin [121].

We obtain our results by utilizing a transcript argument, similar to the one used in chapters 3 and 4. In the present case, however, the argument is more complex because the adversary has more power since he is given black-box access to internal components of the cipher.

We organize the remainder of this chapter as follows. In section 5.2 we give a motivating discussion on the natural round structure of symmetric-key cryptographic primitives. In the section that follows, we give a high-level overview of our contributions. In section 5.4 we discuss our new model of looking at the "round security" and in the subsequent section we discuss how this new model applies to four-round Luby-Rackoff ciphers. Sections 5.6 and 5.7 respectively explain our negative and positive results. The negative results describe the situations for which the Luby-Rackoff style ciphers fail to remain secure in our model, and the positive results describe the situations in which security is upheld. Remarkably, we cover the universe of possible scenarios in these sections, so our characterization of Luby-Rackoff ciphers in this new model is complete. In section 5.8 we discuss the round security of universal hash function based message authentication codes. Finally, in section 5.9 we make some concluding remarks.

5.2 The Natural Round Structure of Symmetric-Key Primitives

Recall that in their path-breaking paper, Luby and Rackoff [88] give a formal definition of a secure block cipher, and then show how to construct such a cipher using pseudorandom functions. Their block cipher consists of four rounds of Feistel permutations, each of which consists of an application of a pseudorandom function and an exclusive-or operation. Each round's output is used for the next round's input, except for the last round, whose output is the output of the block cipher (see figure 5-1).

Much of the theoretical research that followed the work of Luby and Rackoff [88] focuses on efficiency improvements to this construction – thus far this thesis has discussed some of this work. All of these variations also possess the same natural round structure as the original Luby-Rackoff construction.

This theme of an inherent round structure in block ciphers is also seen extensively in practice. For example, a number of ciphers, including DES [108] (which motivated Luby and Rackoff's work) and many of the AES submissions [110] have an inherent round structure (though not necessarily involving Feistel permutations), where the output of one round is used as input to the next.

In addition to block ciphers, constructions of other cryptographic primitives often also proceed in rounds. For example, universal-hash-function-based message authentication codes (Wegman-Carter MACs [142]), which we discuss in section 2.11 of chapter 2, can be viewed as consisting of two rounds. Moreover, cryptographic hash functions (e.g., MD5 [123]), and the various message authentication schemes that are built on top of them (e.g., HMAC [10]), have an induced round structure as well.

Consequently, it should come as little surprise that cryptanalysts have often considered looking at individual rounds in order to better understand the security properties of a given design; for example, a large number of papers have been written analyzing reduced-round variants of block ciphers and hash functions. See, for example, Biham and Shamir's text on the cryptanalysis of DES [23], or Schneier's self-study course on block cipher cryptanalysis [130], and the references therein.

It thus seems that a theoretical framework incorporating the notion of rounds would be desirable. This chapter proposes such a framework. Although our model is a simple extension of the classical models of security for symmetric primitives ([88], [62], [11]), it allows one to obtain a number of interesting results not captured by the traditional models. In particular, we analyze the security of the original Luby-Rackoff construction, some of its variants, and Wegman-Carter message authentication codes within our framework.

5.3 Our Contributions

5.3.1 A New Model

The definition of a secure block cipher from Luby and Rackoff [88], or of a secure message authentication code from Bellare, Kilian, and Rogaway [14], allows the adversary only black-box access to the full primitive. We discuss this model in detail in section 1.2.5 of chapter 1. It is, perhaps, the most widely used model in the research literature. We develop the notion of *round security*, which considers what happens when the adversary has additional black-box access to some of the internal rounds of the computation of the primitive. Although we focus on block ciphers, our techniques can be extended to other primitives such as message authentication codes.

For example, in the case of block ciphers, we study what happens when the adversary is allowed, in addition to its chosen-plaintext and chosen-ciphertext queries, to input a value directly to some round i of the block cipher and view the output after some round j, with restrictions on i and j. The adversary's job is still the same: to determine whether the chosen-ciphertext and chosen-plaintext queries are being answered by the block cipher or by a random permutation. The queries to internal rounds are always answered by the block cipher.

As discussed below, this model allows us gain a better understanding of what makes symmetric constructions secure, and enables us to make statements about security that are not captured by the traditional model.

5.3.2 Round Security of Luby-Rackoff Ciphers

We completely characterize the round security of the original Luby-Rackoff construction and its more efficient variants, such as the ones presented in chapters 3 and 4. That is, we precisely specify the sets of rounds that the adversary can access for the cipher to remain secure, and show that access to other sets of rounds will make the cipher insecure.

Recall that the cipher proposed by Luby and Rackoff operates on a $2n$-bit string $L \cdot R$ and can be described simply as follows:

$$\begin{aligned} S &= L \oplus h_1(R); \\ T &= R \oplus f_1(S); \\ V &= S \oplus f_2(T); \\ W &= T \oplus h_2(V), \end{aligned}$$

where h_1, h_2, f_1, f_2 are pseudorandom functions, $\oplus$ represents the exclusive-or, and the output is $V \cdot W$.

Subsequent work by Naor and Reingold [101], and Patel, Ramzan, and Sundaram [114] (which we describe in chapter 3), demonstrates that the pseudorandom functions h_1 and h_2 can be replaced by bisymmetric Δ-universal hash functions, and that f_1 could equal f_2. This work suggests that strong randomness is important only in the middle two rounds. We extend this observation by showing that, in fact, secrecy is important in the first and last rounds, while randomness (but no secrecy) is needed in the middle two rounds. Specifically, we show that:

- The cipher *remains secure* even if the adversary has oracle access to both f_1 and f_2.
- The cipher becomes *insecure* if the adversary is allowed access to any other round oracles.

Moreover, we demonstrate that instantiating h_1 and h_2 as hash functions instead of as pseudorandom functions does not significantly lower the round security of the block cipher, thus supporting the observation that strong randomness is not needed in the first and last rounds of the Luby-Rackoff construction.

5.3.3 Round Security of Universal Hash Function MACs

Using techniques in this thesis, one can also characterize the round security of a class of universal hash function based message authentication codes (Wegman-Carter MACs). Recall that we can construct such MACs with the following two-round process. In the first round, we apply a universal hash function h to a relatively large message, to get a shorter intermediary string. Then, in the second round, we use a pseudorandom function f on the shorter string to get a final tag. It turns out that:

- The MAC remains *secure* if the adversary has oracle access to f.
- The MAC is, in general, *insecure* if the adversary has oracle access to h.

There are other types of Wegman-Carter MACs in which one uses other cryptographic primitives instead of pseudorandom functions. We discuss some of these alternatives in section 1.4 of chapter 1. We do not study these other types of MACs in this chapter since we are unable to effectively characterize them in our model.

5.3.4 Implications for the Random Oracle Model

Our work has interesting implications for Luby-Rackoff ciphers and Wegman-Carter MACs in the random oracle model. In this model, one assumes the existence of a publicly computable function that behaves randomly. One can easily define security of block ciphers and message authentication codes in this model given the work of Bellare and Rogaway [16]: one simply allows all parties (including the adversary) access to the same public random oracle, and the adversary has to succeed for a random choice of the oracle.

Our results imply that the Luby-Rackoff cipher remains secure in the random oracle model if one replaces the functions f_1 and f_2 with random oracles. That is, in the random oracle model, keying material will only be necessary for h_1 and h_2, which, as we show in chapter 3, can be just universal hash functions.

Similarly, the Wegman-Carter MAC remains secure if the pseudorandom function, used in the second round, is replaced with a random oracle. Thus, again, in the random oracle model, keying material is needed only for the hash function.

Block ciphers have been analyzed in the random-oracle model before. For example, Even and Mansour [50] construct a cipher using a public random *permutation* oracle P (essentially, the construction is $y = P(k_1 \oplus x) \oplus k_2$, where k_1 and k_2 constitute the key, x is the plaintext, and y is the resulting ciphertext). They show their construction is hard to invert and to existentially forge. We can recast their construction in our model, as a three-round cipher, where the adversary has access to the second round. Using the techniques in this chapter we can, in fact, obtain a stronger result; namely, that their cipher is super pseudorandom. We stress, however, that their construction involves access to a random *permutation* oracle, which is a more stringent requirement.

Of course, whether a scheme in the random oracle model can be instantiated securely in the real world (that is, with polynomial-time computable functions in place of random oracles) is uncertain, particularly in light of the results of Canetti, Goldreich and Halevi [34], who show that there are schemes which are secure in the random oracle model, but become insecure when the random oracle is replaced by any polynomial-time publicly computable function. However, our results open up an interesting direction: is it possible to replace pseudorandom functions with unkeyed functions in any of the constructions we discuss?

5.3.5 Implications for Tamper Resistance

Our model also has some implications for designing tamper-resistant devices. For example, suppose that one wants to implement a block cipher (or some other primitive) on a smart card and one wants to determine whether or not the entire smart card should be resistant to tampering. This consideration has some importance since it may cost more to make the entire card tamper resistant. If certain parts of the card are not tamper resistant, then it may be possible for an adversary to probe various parts of the card and see the data as it is being transformed from the plaintext to the ciphertext. For example, the adversary may see the data between rounds i and $i+1$. In our round security model, this probing capability is achieved by giving the adversary access to an oracle which allows it to put input into round one and see the results after round i (and an oracle that allows the adversary to place an input after the last round, and see the corresponding output before round $i+1$). For four-round Luby-Rackoff ciphers, we show that we cannot allow the adversary to probe the data being transformed between any two rounds. Similarly, for Wegman-Carter MACs, we cannot allow the adversary to probe the data being transformed between the first and second round.

5.4 New Model: Round Security

We now discuss our new model of round security. The definitions can be easily extended to other symmetric primitives, such as MACs. We start by defining the notion of a round decomposition.

Definition 31 *Let $\mathcal{P}, \mathcal{F}^1, \mathcal{F}^2, \ldots, \mathcal{F}^r$ be keyed permutation families, each with domain and range I_l and key length s. Suppose that for any key $a \in I_s$, it turns out that*

$$p_a = f_a^r \circ \ldots \circ f_a^1,$$

where p_a denotes picking the function in $\mathcal{P}$ whose key corresponds to a. Then $\mathcal{F}^1, \ldots, \mathcal{F}^r$ is called an r-round decomposition for $\mathcal{P}$. For $i \leq j$, we denote by $(i \rightarrow j)_a$ the permutation

$$f_a^j \circ \ldots \circ f_a^i,$$

and we denote by $(i \leftarrow j)_a$ the permutation

$$\left(f_a^j \circ \ldots \circ f_a^i\right)^{-1}.$$

We denote by $i \rightarrow j$ and $i \leftarrow j$ the corresponding keyed function families.

Note that having oracle access to a member of $i \rightarrow j$ means being able to give inputs to round i of the forward direction of a block cipher and view outputs after round j. Likewise, having oracle access to $i \leftarrow j$ corresponds to being able to give "inputs" to round j of the *reverse* direction of the block cipher and view outputs "before" round i. Note that it makes sense to talk about having oracle access to $i \rightarrow i$ since it allows one to give input before round i, and and see the output after round i has been processed. We also observe that if we implement the cipher on a device which allows the adversary to probe the data as it is being transformed between rounds i and $i+1$, then this can be modeled by giving oracle access to $1 \rightarrow i$ and $i+1 \leftarrow r$. Also observe that the oracle for $1 \rightarrow r$ corresponds to an oracle for a chosen plaintext attack $\mathcal{P}$, and the oracle for $1 \leftarrow r$ corresponds to an oracle for chosen ciphertext attack $\mathcal{P}^{-1}$. Thus, one can think of our model as a refinement of the traditional models which only give oracle access to $\mathcal{P}$ or $\mathcal{P}^{-1}$.

We are now ready to define security in this round-based model. This definition closely mimics definition 5 from chapter 2. The difference is that the adversary is allowed oracle access to some subset K of the set

$$\{i \rightarrow j, i \leftarrow j : 1 \leq i \leq j \leq r\},$$

and the insecurity function additionally depends on K.

Definition 32 *Let $\mathcal{P}$ be a block cipher with domain and range I_l, key length s and some r-round decomposition $\mathcal{F}^1, \ldots, \mathcal{F}^r$. Fix some subset $K = \{\pi^1, \ldots, \pi^k\}$ of the set $\{i \rightarrow j, i \leftarrow j : 1 \leq i \leq j \leq r\}$, and let $\mathcal{A}$ be a $k+2$-oracle adversary. Then we define $\mathcal{A}$'s advantage as*

$$\begin{aligned}&\mathsf{Adv}^{\mathsf{sprp}}_{\mathcal{P},\mathcal{F}^1,\ldots,\mathcal{F}^r,K}(\mathcal{A}) = \\ &\qquad \Pr[a \overset{R}{\leftarrow} I_s : \mathcal{A}^{p_a, p_a^{-1}, \pi_a^1, \ldots, \pi_a^k} = 1] - \Pr[p \overset{R}{\leftarrow} \mathsf{Perm}^l, a \overset{R}{\leftarrow} I_s : \mathcal{A}^{p, p^{-1}, \pi_a^1, \ldots, \pi_a^k} = 1]\end{aligned}$$

For any integers $q, t \geq 0$ and set K, we define an insecurity function

$$\mathsf{Adv}^{\mathsf{sprp}}_{\mathcal{P},\mathcal{F}^1,\ldots,\mathcal{F}^r}(q, t, K) = \max_{\mathcal{A}}\{\mathsf{Adv}^{\mathsf{sprp}}_{\mathcal{P},\mathcal{F}^1,\ldots,\mathcal{F}^r}(\mathcal{A})\}$$

similarly to the one in definition 5. Again, the above maximum is taken over adversaries $\mathcal{A}$ who make at most q total queries (to all of their oracles), and whose running time, together with the time to select the appropriate keys and answer oracle queries, is at most t.

We remark that the notion of round security is taken with respect to a particular round decomposition of the underlying primitive. In some sense, if the round decomposition is not "interesting" then the results obtained may not be "interesting" either. For example, if $\mathcal{P}$ is the cipher, then $\mathcal{P}, \mathsf{id}, \mathsf{id}, \ldots, \mathsf{id}$, where id is the identity function, is a legitimate round decomposition of the cipher. Yet, at the same time, this decomposition will not yield additional insights beyond what we know from analyzing security with respect to the traditional models.

We also note that our definition of security involves comparing a block cipher, with additional round oracles, to a truly random permutation. Another plausible definition for round security might be to compare a block cipher with itself, where in the first instance we give round oracles corresponding to the cipher, and in the other instance we give round oracles which are independent of the cipher. This alternate approach turns out not to be terribly different from the originally proposed approach. In particular, if the cipher already is a super pseudorandom permutation, then one can show that the two approaches are equivalent. On the other hand, if the cipher is not a super pseudorandom permutation, then there are already security concerns, so it is not clear if one should expend much effort in analyzing this scenario.

5.5 Round Security of Luby-Rackoff Ciphers

Having developed a round security framework for block ciphers, we examine the specific case of the original four-round Luby-Rackoff cipher, which we described in chapter 2. Our goal is to characterize the insecurity function defined above depending on the set K of oracles. We are able to do so completely, in the following sense. We place *every* set K in one of two categories: either the insecurity function is unacceptably high, or it is almost as low as in the standard model. That is, we completely characterize the acceptable sets of oracles for the construction to remain secure in our model.

Moreover, we do so for all three ciphers presented in section 2.9 although we need to add an ϵ-uniformity condition on the hash functions in the second and third constructions in order for them to remain secure; this is a mild condition, often already achieved by a hash function family. Recall that we formally define the ϵ-uniformity condition in chapter 2.

As it turns out, the round security of the three constructions is the same. Specifically, all three ciphers remain secure if the adversary is given access to the second and third rounds. These results suggest, in some sense, that the so-called "whitening" steps, performed in the first and last rounds, require secrecy but only weak randomness, whereas the middle rounds require strong randomness but no secrecy.

We present our results in two parts. First, in Section 5.6, we examine what combinations of oracles make the cipher insecure. Then, in Section 5.7, we show that any other combination leaves it secure.

5.6 Negative Results

In this section we demonstrate which oracles make the cipher insecure. Our negative results are strong, in the sense that they hold regardless of what internal functions h_1, h_2, f_1, f_2 are used. That is, the cipher can be distinguished from a random permutation even if each of these functions is chosen truly at random. Thus, our results hold for all the four-round ciphers presented in this dissertation. For now, we concentrate on the ciphers whose underlying Feistel ladder operates over $\mathrm{GF}(2^n)^+$ (bit-wise exclusive-or), though our techniques can apply to Luby-Rackoff style ciphers over other algebraic structures, such as the ones we consider in chapter 4.

Theorem 16 *Regardless of how the functions h_1, f_1, f_2, h_2 are chosen from the set of all possible functions with domain and range I_n, let $P = \Psi(h_1, f_1, f_2, h_2)$. Let t be the time required to compute 17 n-bit exclusive-or operations, a comparison of two n-bit strings, and 9 oracle queries.*[1] *Then*

$$\mathsf{Adv}^{\mathsf{sprp}}_{\mathcal{P},\overline{h_1},\overline{f_1},\overline{f_2},\overline{h_2}}(9, t, K) \geq 1 - 2^{-n},$$

as long as K is not a subset of $\{2 \rightarrow 2, 2 \leftarrow 2, 3 \rightarrow 3, 3 \leftarrow 3, 2 \rightarrow 3, 2 \leftarrow 3\}$. That is, P is insecure as long as the adversary has access to an oracle that includes the first or fourth rounds.

In the above theorem recall that $\overline{h_1}$ (respectively $\overline{f_1}$, $\overline{f_2}$, $\overline{h_2}$) denotes the basic Feistel permutation mapping (L, R) to $(R, L \oplus h_1(R))$. This notation appears in definition 17 in chapter 2.

We will prove the theorem by eliminating oracles that allow the adversary to distinguish the cipher from a random permutation. This involves using the attack against a three-round cipher from the original Luby-Rackoff paper [88]. If we consider ciphers whose underlying

[1]The values 17 and 9 can be reduced by more careful counting; it is unclear, however, if there is any reason to expend effort finding the minimal numbers that work.

Feistel ladder does not use the bit-wise exclusive-or, then we can use the attack given in section 4.6. We now give the complete proof. First, we note the following fact.

Lemma 6 *If we give a way to compute the values of h_1 on arbitrary inputs, then there exists an adversary $\mathcal{A}$ that asks three queries to h_1, two queries to the chosen-plaintext oracle p, and one query to the chosen-ciphertext oracle p^{-1}, performs 8 exclusive-or operations, and has an advantage of $1 - 2^{-n}$.*

Proof: This is so because access to h_1 allows the adversary to "peel off" the first round of the cipher, and then use the original Luby-Rackoff attack [88] against a three-round cipher. Consider the following attack, which assumes the existence of an oracle to compute the first round function h_1 (i.e. $1 \rightarrow 1$).

```
Description of adversary who has oracle access to $(1 \rightarrow 1)$.

1. Pick three arbitrary $n$-bit strings: $L_1 \overset{R}{\leftarrow} I_n$, $R_1 \overset{R}{\leftarrow} I_n$, $R_2 \overset{R}{\leftarrow} I_n$.
2. Query for the encryption of $L_1 \cdot R_1$ and call the result $V_1 \cdot W_1$.
3. Query the $(1 \rightarrow 1)$ oracle on $R_1$ and $R_2$. Call the results $h_1(R_1)$ and $h_1(R_2)$.
4. Let $L_2 = L_1 \oplus h_1(R_1) \oplus h_1(R_2)$.
5. Query for the encryption of $L_2 \cdot R_2$ and call the result $V_2 \cdot W_2$.
6. Let $V_3 = V_2$.
7. Let $W_3 = W_2 \oplus R_1 \oplus R_2$.
8. Query for the decryption of $V_3 \cdot W_3$ and call the result $L_3 \cdot R_3$.
9. Query the $(1 \rightarrow 1)$ oracle on $R_3$. The result is $h_1(R_3)$.

If $h_1(R_3) \oplus L_3 = V_1 \oplus V_2 \oplus L_1 \oplus h_1(R_1)$ then
    return that the cipher is not random.
else
    return that the cipher is random.
```

If the plaintext and ciphertext oracles are truly random, then the above attack will output the incorrect answer with probability 2^{-n}, because V_1 and L_3 are then random and independent of the rest of the terms. However, if the plaintext and ciphertext oracles are for the block cipher, then the adversary will output the correct answer with certainty. Here is why.

Let S_i, T_i $(1 \leq i \leq 3)$ be the intermediate values computed in rounds 1 and 2 of the block cipher for the three queries. Now, observe that:

$$\begin{aligned} L_2 &= L_1 \oplus h_1(R_1) \oplus h_1(R_2); \\ V_3 &= V_2; \\ W_3 &= W_2 \oplus R_1 \oplus R_2. \end{aligned}$$

Note that:

$$\begin{aligned} S_1 &= L_1 \oplus h_1(R_1) \\ &= L_2 \oplus h_1(R_2) \\ &= S_2. \end{aligned}$$

Then:

$$\begin{aligned} T_3 &= W_3 \oplus h_2(V_3) \\ &= W_2 \oplus R_1 \oplus R_2 \oplus h_2(V_2) \\ &= T_2 \oplus R_1 \oplus R_2 \\ &= f_1(S_2) \oplus R_2 \oplus R_1 \oplus R_2 \\ &= f_1(S_1) \oplus R_1 \\ &= T_1. \end{aligned}$$

Finally,

$$\begin{aligned} h_1(R_3) \oplus L_3 &= S_3 \\ &= V_3 \oplus f_2(T_3) \\ &= V_2 \oplus f_2(T_1) \\ &= V_2 \oplus V_1 \oplus S_1 \\ &= V_2 \oplus V_1 \oplus L_1 \oplus h_1(R_1), \end{aligned}$$

which completes the proof. ■

Note that this fact can be similarly shown for h_2. The lemma above allows us to prove the following result.

Lemma 7 *If K contains at least one of the following oracles: $1 \rightarrow 4$, $1 \leftarrow 4$, $2 \rightarrow 4$, $2 \leftarrow 4$, $1 \rightarrow 3$, $1 \leftarrow 3$, $1 \rightarrow 1$, $1 \rightarrow 2$, $1 \leftarrow 1$, $1 \leftarrow 2$, $4 \leftarrow 4$, $3 \leftarrow 4$, $4 \rightarrow 4$ or $3 \leftarrow 4$, then there exists $\mathcal{A}$ making no more than 9 queries to the oracles and performing no more than 17 exclusive operations whose advantage is $1 - 2^{-n}$.*

Proof: If K contains $1 \rightarrow 4$ or $1 \rightarrow 3$, then $\mathcal{A}$ can input an arbitrary pair $L \cdot R$ to either of these and receive $V \cdot W$ or $T \cdot V$. $\mathcal{A}$ then inputs $L \cdot R$ to the chosen plaintext oracle p to receive $V' \cdot W'$, and checks if $V = V'$. A similar attack holds if K contains $1 \leftarrow 4$ or $2 \leftarrow 4$.

If K contains $2 \rightarrow 4$, then $\mathcal{A}$ can input an arbitrary pair $R \cdot S$ to it to receive $V \cdot W$. In this case, $\mathcal{A}$ inputs $V \cdot W$ to the chosen ciphertext oracle p^{-1} to receive $L \cdot R'$ and checks if $R = R'$. A similar attack holds for $1 \leftarrow 3$.

If K contains $1 \rightarrow 1$ or $1 \rightarrow 2$, then $\mathcal{A}$ can input $L \cdot R$ and receive, in particular, $S = L \oplus h_1(R)$. $\mathcal{A}$ can then compute $h_1(R) = S \oplus L$, and use the procedure of Lemma 6. Access to $1 \leftarrow 1$ allows $\mathcal{A}$ to input $R \cdot S$ and receive $(L = S \oplus h_1(R), R)$. $\mathcal{A}$ can then compute $h_1(R) = L \oplus S$. Access to $1 \leftarrow 2$ allows $\mathcal{A}$ to compute $h_1(R)$ as follows:

1. Query the $1 \leftarrow 2$ oracle on an arbitrary pair $S_1 \cdot T_1$ to get $L_1 \cdot R_1$.

2. Let $T_2 = T_1 \oplus R_1 \oplus R$ and $S_2 = S_1$.

3. Query the $1 \leftarrow 2$ oracle on $S_2 \cdot T_2$ to get $L_2 \cdot R_2$. Then

$$\begin{aligned} R_2 &= T_2 \oplus f_1(S_2) \\ &= (T_1 \oplus R_1 \oplus R) \oplus (R_1 \oplus T_1) \\ &= R; \end{aligned}$$

4. Compute $h_1(R) = L_2 \oplus S_2$.

Thus, any of the oracles $1 \rightarrow 1, 1 \rightarrow 2, 1 \leftarrow 1, 1 \leftarrow 2$ gives $\mathcal{A}$ access to h_1 and thus makes the cipher insecure. A similar argument holds for access to any of the oracles $4 \leftarrow 4, 3 \leftarrow 4$, $4 \rightarrow 4$ and $3 \rightarrow 4$. ■

Finally, to prove Theorem 16, note that there are 20 possible oracles. Of those, 14 are ruled out by the above lemma, leaving only 6 possible oracles to choose from.

5.7 Positive Results

In this section, we prove what is essentially the converse of the results of the previous section. Namely, we show that if K is the set given in Theorem 16, then the cipher is secure. Of course, if K is a subset of it, then the cipher is also secure. In particular, we prove the following theorem.

Theorem 17 (Ramzan-Reyzin [121]) *Suppose* $K \subseteq \{2 \to 2, 2 \leftarrow 2, 3 \to 3, 3 \leftarrow 3, 2 \to 3, 2 \leftarrow 3\}$.
Let h_1, f_1, f_2, h_2, t_f *and* $\mathcal{B}$ *be as in the original Luby-Rackoff construction (Theorem 4). Then*

$$\mathsf{Adv}^{\mathsf{sprp}}_{\mathcal{B},\overline{h_1},\overline{f_1},\overline{f_2},\overline{h_2}}(q,t,K) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(q,t') + \binom{q}{2}\left(2^{-n+1} + 2^{-2n}\right) + q^2\left(2^{-n-1}\right),$$

where

$$t' = t + O(3s + 3q_c(t_f + n) + q_o n).$$

If $h_1, f_1, f_2, h_2, t_h, t_f$ *and* $\mathcal{B}$ *are as in the Naor-Reingold construction (Theorem 5), with the additional condition that* h_1 *and* h_2 *be* ϵ_3*-uniform, then*

$$\mathsf{Adv}^{\mathsf{sprp}}_{\mathcal{B},\overline{h_1},\overline{f_1},\overline{f_2},\overline{h_2}}(q,t,K) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(q,t') + \binom{q}{2}\left(2\epsilon + 2^{-2n}\right) + q^2\epsilon_3/2,$$

where

$$t' = t + O(s_1 + 2s_2 + q_c(2t_h + t_f) + q_o n).$$

Finally, if h_1, f, h_2, t_h, t_f *and* $\mathcal{B}$ *are as in Patel-Ramzan-Sundaram construction of chapter 3 (Theorem 6), with the additional condition that* h_1 *and* h_2 *be* ϵ_3*-uniform, then*

$$\mathsf{Adv}^{\mathsf{sprp}}_{\mathcal{B},\overline{h_1},\overline{f},\overline{f},\overline{h_2}}(q,t) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(2q,t') + q^2(\epsilon_1 + 2\epsilon_3) + \binom{q}{2}\left(2\epsilon_2 + 2^{-2n}\right),$$

where

$$t' = t + O(2s_2 + 2q_c(t_h + n) + q_o n).$$

We focus our proof on the last part of the theorem. The proofs of the other cases are very similar. Our proof technique is a generalization of the techniques seen in chapters 3 and 4, but is designed to deal with the extra queries. We continue to analyze security in the concrete or exact model, rather than the asymptotic model.

We proceed in the standard manner by showing that the permutation $\Psi(h_1, f, f, h_2)$ is pseudorandom when f is truly random (rather than pseudorandom). We also note that access to the oracles of K is equivalent to access to the oracle for f (although one query to $2 \to 3$ or $3 \to 2$ can be simulated by two queries to f). Thus, it suffices to prove the following theorem.

Theorem 18 *Let f be chosen uniformly from $\mathsf{Rand}^{n \to n}$. Let h_1, h_2 be ϵ_1-bisymmetric ϵ_2-almost-Δ-universal ϵ_3-uniform hash functions with domain and range I_n. Let $\mathcal{P}$ be the family of permutations on I_{2n} consisting of permutations of the form $P = \Psi(h_1, f, f, h_2)$. Then, for any 3-oracle adversary $\mathcal{A}$ that makes at most q_c queries to its first two oracles and at most q_o queries to its third oracle ($q = q_o + q_t$),*

$$\mathsf{Adv}^{\mathsf{sprp}}_{\mathcal{P},\overline{h_1},\overline{f},\overline{f},\overline{h_2}}(q, t, f) \leq q_c^2 \epsilon_1 + 2q_o q_c \epsilon_3 + \binom{q_c}{2}\left(2\epsilon_2 + 2^{-2n}\right).$$

The remainder of this section gives the proof of this theorem. To summarize, the first part of the proof focuses on the transcript (a.k.a. the "view") of the adversary, and shows that each possible transcript is about as likely to occur when $\mathcal{A}$ is given $\mathcal{P}$ as when $\mathcal{A}$ is given Perm^{2n}. The second part uses a probability argument to show that this implies that $\mathcal{A}$ will have a small advantage in distinguishing $\mathcal{P}$ from Perm^{2n}.

Proof of Theorem 18

We now analyze the security of our construction $P = \Psi(h_1, f, f, h_2)$ under the round security model. The technique used involves a more complex transcript argument, similar in spirit to the arguments used in chapters 3 and 4.

Recall that in our setting we model the adversary $\mathcal{A}$ as a program for Random Access Machines. The adversary gets a certain kind of black-box access to either a permutation uniformly sampled from the set of all possible permutations on $2n$ bits, or one sampled from some the set of ciphers $\Psi(h_1, f, f, h_2)$. Let P denote this first oracle. In addition, the adversary gets access to an oracle that computes the function f – we denote this oracle by $\mathcal{O}^f$. In the case that $P = \Psi(h_1, f, f, h_2)$, $\mathcal{O}^f$ computes the same function as the one used in the second and third rounds for P. In the case that P is a truly random permutation, then $\mathcal{O}^f$ still computes the kind of function used in the second and third rounds of a cipher of the form $\Psi(h_1, f, f, h_2)$. In this case, however, the function f is completely independent of $P \overset{R}{\leftarrow} \mathsf{Perm}^{2n}$. Now, $\mathcal{A}$ has two possibilities for queries to the oracle P: $(+, x)$ which asks to obtain the value of $P(x)$, or $(-, y)$ which asks to obtain the value of $P^{-1}(y)$ – where both x and y are in I_{2n}. We call these cipher queries. As per our usual convention, we often write $L \cdot R$ to represent the left and right n bits of the plaintext x, and $V \cdot W$ to represent the left and right n bits of the ciphertext y. We define the query-answer pair for the i^{th} cipher query as $\langle x_i, y_i \rangle \in I_{2n} \times I_{2n}$ if $\mathcal{A}$'s query was $(+, x_i)$ and y_i is the answer it received from P or its query was $(-, y_i)$ and x_i is the answer it received. We assume that $\mathcal{A}$ makes exactly q_c queries and we call the sequence

$$\langle (x_1, y_1), \ldots, (x_{q_c}, y_{q_c}) \rangle_P$$

the cipher-transcript of $\mathcal{A}$.

In addition, $\mathcal{A}$ can make queries to $\mathcal{O}^f$. We call these internal queries. We denote these queries as: $(\mathcal{O}^f, x')$ which asks to obtain $f(x')$. We define the query-answer pair for the i^{th} internal query as $\langle x'_i, y'_i \rangle \in I_n \times I_n$ if $\mathcal{A}$'s query was $(\mathcal{O}^f, x'_i)$ and the answer it received was y'_i. We assume that $\mathcal{A}$ makes q_o queries to this oracle. We call the sequence

$$\langle (x'_1, y'_1), \ldots, (x'_{q_o}, y'_{q_o}) \rangle_{\mathcal{O}^f}$$

the internal-transcript of $\mathcal{A}$.

Now, we can make the standard assumption that $\mathcal{A}$ is deterministic (since we can always fix the optimal random choices). Under this assumption, the exact next query made by $\mathcal{A}$ can be determined by the previous queries and the answers received. We now formalize this notion by generalizing the function C that we defined in our section on transcripts (section 2.6 of chapter 2).

Definition 33 *Let*

$$C_{\mathcal{A}}[\langle(x_1, y_1), \ldots, (x_i, y_i)\rangle_P, \langle(x'_1, y'_1)\rangle, \ldots, (x'_j, y'_j)\rangle_{\mathcal{O}^f}],$$

where $i < q_c$ or $j < q_o$, denote the $i + j + 1^{st}$ query $\mathcal{A}$ makes as a function of the first $i + j$ query-answer pairs in $\mathcal{A}$'s cipher and oracle transcripts. Let

$$C_{\mathcal{A}}[\langle(x_1, y_1), \ldots, (x_{q_c}, y_{q_c})\rangle_P, \langle(x'_1, y'_1), \ldots, (x'_{q_o}, y'_{q_o})\rangle_{\mathcal{O}^f}]$$

denote the output $\mathcal{A}$ gives as a function of its cipher and oracle transcripts.

Definition 34 *Let σ be the pair of sequences*

$$\left(\langle(x_1, y_1), \ldots, (x_{q_c}, y_{q_c})\rangle_P, \langle(x'_1, y'_1), \ldots, (x'_{q_o}, y'_{q_o})\rangle_{\mathcal{O}^f}\right),$$

where for $1 \le i \le q_c$ we have that $\langle x_1, y_1\rangle \in I_{2n} \times I_{2n}$, and for $1 \le j \le q_o$, we have that $\langle x', y'\rangle \in I_n$. Then, σ is a consistent $\mathcal{A}$-transcript if for every $1 \le i \le q_c$:

$$C_{\mathcal{A}}[\langle(x_1, y_1), \ldots, (x_i, y_i)\rangle_P, \langle(x'_1, y'_1), \ldots, (x'_j, y'_j)\rangle_{\mathcal{O}^f}] \in \{(+, x_{i+1}), (-, y_{i+1}), (\mathcal{O}^f, x'_{j+1})\}.$$

We will make use of the random process $\tilde{R}$ (see definition 21 in chapter 3). The main difference is that in this scenario, the random process $\tilde{R}$ will *only* answer the cipher queries of $\mathcal{A}$. The internal queries will always be answered by round functions (or the composition of round functions) from the block cipher. As before, $\tilde{R}$'s answers may not be consistent with any function, let alone any permutation. In this case, we say that the underlying transcript is inconsistent. We also assume, as we did before, that $\mathcal{A}$ never asks a query if it can determine the answer directly from a previous query-answer pair.

Recall that the process $\tilde{R}$ often "behaves" exactly like a permutation, and if $\mathcal{A}$ is given oracle access to either $\tilde{R}$ or Perm^{2n} to answer its cipher queries, it will have a negligible advantage in distinguishing between the two. We prove this formally in proposition 1 in chapter 3. In fact, if $\mathcal{A}$ only makes q_c cipher queries, then his advantage in distinguishing between $\tilde{R}$ and Perm^{2n} is $\binom{q_c}{2} \cdot 2^{-2n}$. Note that internal queries are immaterial for obtaining this bound since not only are internal query oracles independent of the cipher query oracles, but they are identically distributed for both $\tilde{R}$ and Perm^{2n} since they are always drawn directly from the construction of the cipher.

Before proceeding, recall that we can denote by the random variables $T_{\mathcal{P}}, T_{\mathsf{Perm}^{2n}}, T_{\tilde{R}}$ the cipher-transcript / internal-transcript pair seen by $\mathcal{A}$ when its cipher queries are answered by $\mathcal{P}$, Perm^{2n}, $\tilde{R}$ respectively, and its internal queries are all answered by $\mathcal{O}^f$.

We now proceed to obtain a bound on the advantage that $\mathcal{A}$ will have in distinguishing between $T_{\mathcal{P}}$ and $T_{\tilde{R}}$. It turns out that $T_{\mathcal{P}}$ and $T_{\tilde{R}}$ are identically distributed unless some bad event depending on the choice of h_1, h_2 in $\mathcal{P}$ occurs. We call this event Bad and obtain a bound on the probability that it occurs. Similar to the previous chapters, Bad occurs whenever the function f in $\mathcal{P}$ would have been evaluated on the same input twice during the process of answering queries. For example, this event would occur if the input to the internal query oracle were identical to the value on which f is evaluated during a normal cipher query (either in the plaintext or ciphertext direction). We formally define Bad in the next definition, and in proposition 7 we obtain a bound on the probability that it actually occurs. Then, in lemma 8 we show that $T_{\mathcal{P}}$ and $T_{\tilde{R}}$ are identically distributed conditioned on the event Bad occurring.

Definition 35 *For every specific pair of functions h_1, h_2 define $\mathsf{Bad}(h_1, h_2)$ to be the set of all possible and consistent transcripts*

$$\sigma = \langle (x_1, y_1), \ldots, (x_{q_c}, y_{q_c}) \rangle_P, \langle (x'_1, y'_1), \ldots, (x'_{q_o}, y'_{q_o}) \rangle_{\mathcal{O}^f}$$

satisfying at least one of the following events:

- *Event* $\mathsf{B1}$*: there exists $1 \leq i < j \leq q_c$ such that $h_1(x_i^R) \oplus x_i^L = h_1(x_j^R) \oplus x_j^L$, or*
- *Event* $\mathsf{B2}$*: there exists $1 \leq i < j \leq q_c$ such that $y_i^R \oplus h_2(y_i^L) = y_j^R \oplus h_2(y_j^L)$, or*
- *Event* $\mathsf{B3}$*: there exists $1 \leq i, j \leq q_c$ such that $h_1(x_i^R) \oplus x_i^L = y_j^R \oplus h_2(y_j^L)$, or*
- *Event* $\mathsf{B4}$*: there exists $1 \leq i \leq q_c$, $1 \leq j \leq q_o$ such that $h_1(x_i^R) \oplus x_i^L = x'_j$, or*
- *Event* $\mathsf{B5}$ *: there exists $1 \leq i \leq q_c$, $1 \leq j \leq q_o$ such that $y_i^R \oplus h_2(y_i^L) = x'_j$.*

Proposition 7 *Let h_1, h_2 be ϵ_1-bisymmetric ϵ_2-almost-Δ-universal ϵ_3-uniform hash functions. Then, for any possible and consistent $\mathcal{A}$ − transcript σ, we have that*

$$\Pr_{h_1,h_2}[\sigma \in \mathsf{Bad}(h_1,h_2)] \le q_c^2\epsilon_1 + 2q_oq_c\epsilon_3 + \binom{q_c}{2} \cdot 2\epsilon_2.$$

Proof: Recall that a transcript $\sigma \in \mathsf{Bad}(h_1,h_2)$ if at least one of the events B1 through B5 occur. We can determine the individual probabilities of each of these events separately, and obtain an upper bound on the desired probability by taking the sum. We start with bounding the probability of the first event:

$$\begin{aligned}
\Pr_{h_1}[\mathsf{B1}] &\le \Pr_{h_1}[\exists 1 \le i < j \le q_c : h_1(R_i) \oplus L_i = h_1(R_j) \oplus L_j] \\
&\le \sum_{1\le i<j\le q_c} \Pr_{h_1}[h_1(R_i) \oplus L_i = h_1(R_j) \oplus L_j] \\
&\le \sum_{1\le i<j\le q_c} \Pr_{h_1}[h_1(R_i) \oplus h_1(R_j) = L_j \oplus L_i] \\
&\le \binom{q_c}{2} \cdot \epsilon_2.
\end{aligned}$$

The last inequality follows from the previous since if $R_i \ne R_j$ then we can use the fact that h_1, h_2 are ϵ_2-almost-Δ-universal; if $R_i = R_j$, we know that $x_i^L \ne x_j^L$ (since we assume the cipher queries are distinct), in which case

$$\Pr_h[h(R_i) \oplus h(R_j) = L_j \oplus L_i] = 0.$$

Now, we bound the probability of the second event:

$$\begin{aligned}
\Pr_{h_2}[\mathsf{B2}] &\le \Pr_{h_2}[\exists 1 \le i < j \le q_c : W_i \oplus h_2(V_i) = W_j \oplus h_2(V_j)] \\
&\le \sum_{1\le i<j\le q_c} \Pr[W_i \oplus h_2(V_i) = W_j \oplus h_2(V_j)] \\
&\le \sum_{1\le i<j\le q_c} \Pr[h_2(V_j) \oplus h_2(V_i) = W_j \oplus W_i] \\
&\le \binom{q_c}{2} \cdot \epsilon_2.
\end{aligned}$$

The last inequality follows from the previous since if $V_i \ne V_j$ then we can use the fact that h is ϵ_2-almost-Δ-universal; if $V_i = V_j$, we know that $W_i \ne W_j$ (since we assume the queries are distinct), in which case

$$\Pr_h[h(V_i) \oplus h(V_j) = W_j \oplus W_i] = 0.$$

A bound for the probability of the third event similarly follows, from ϵ_1-bisymmetry of h_1, h_2:

$$\begin{aligned}
\Pr_{h_1,h_2}[\mathsf{B3}] &\leq \Pr_h[\exists 1 \leq i,j \leq q_c : h_1(R_i) \oplus L_i = W_j \oplus h_2(V_j)] \\
&\leq \sum_{1\leq i,j\leq q_c} \Pr[h_1(R_i) \oplus L_i = W_j \oplus h_2(V_j)] \\
&\leq \sum_{1\leq i,j\leq q_c} \Pr[h_1(R_i) \oplus h_2(V_j) = W_j \oplus L_i] \\
&\leq {q_c}^2 \cdot \epsilon_1.
\end{aligned}$$

We bound the remaining two events using ϵ_3-uniformity.

$$\begin{aligned}
\Pr_{h_1}[\mathsf{B4}] &\leq \Pr_{h_1}[\exists 1 \leq i \leq q_c, 1 \leq j \leq q_o : h_1(R_i) \oplus L_i = x'_j] \\
&\leq \sum_{1\leq i\leq q_c, 1\leq j\leq q_o} \Pr_{h_1}[h_1(R_i) = x'_j \oplus L_i] \\
&\leq (q_c \cdot q_o) \cdot \epsilon_3.
\end{aligned}$$

And finally,

$$\begin{aligned}
\Pr_{h_2}[\mathsf{B5}] &\leq \Pr_{h_1}[\exists 1 \leq i \leq q_c, 1 \leq j \leq q_o : W_i \oplus h_2(V_i) = x'_j] \\
&\leq \sum_{1\leq i\leq q_c, 1\leq j\leq q_o} \Pr_{h_2}[h_2(V_i) = W_i \oplus x'_j] \\
&\leq (q_c \cdot q_o) \cdot \epsilon_3.
\end{aligned}$$

We thus get that:

$$\Pr_{h_1,h_2}[\sigma \in \mathsf{Bad}(h_1,h_2)] \;\leq\; \Pr_{h_1,h_2}[\mathsf{B1}] + \ldots + \Pr_{h_1,h_2}[\mathsf{B5}]$$

$$\leq \; q_c^2\epsilon_1 + 2q_oq_c\epsilon_3 + \binom{q_c}{2} \cdot 2\epsilon_2,$$

which concludes the proof. ■

The following key lemma shows that the distribution of possible and consistent transcripts generated by $T_{\mathcal{P}}$ given that the bad conditions do not occur is identical to the distribution of possible and consistent transcripts generated by $T_{\tilde{R}}$. This lemma will later help us to determine a bound on the advantage our adversary $\mathcal{A}$ will have when trying to distinguish between these two cases in general. The lemma is the analogue of lemma 2 from chapter 3, but takes the internal query transcript into account as well.

Lemma 8 *Let*

$$\sigma = \langle (x_1, y_1), \ldots, (x_{q_c}, y_{q_c}) \rangle_P, \langle (x'_1, y'_1), \ldots, (x'_{q_o}, y'_{q_o}) \rangle_{O^f}$$

be any possible and consistent $\mathcal{A}$-transcript, then

$$\Pr_{\mathcal{P}}[T_{\mathcal{P}} = \sigma | \sigma \notin \mathsf{Bad}(h_1, h_2)] = \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma].$$

Proof: By the same reasoning in the proof of lemma 2 from chapter 3, one can show that

$$\Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma] = 2^{-(2q_c+q_o)n}.$$

The only difference in the argument is the extra multiplicative factor of 2^{-q_on}. This factor arises since we make q_o internal queries, resulting in $q_o \cdot n$ bits of total output; moreover, these bits are uniformly distributed and independent of each other since the queries are answered by a random function.

Now, we fix h_1, h_2 such that $\sigma \notin \mathsf{Bad}(h_1, h_2)$. We will compute $\Pr_f[T_{\mathcal{P}} = \sigma]$ (note that the probability is now only over the choice of f). Since σ is a possible $\mathcal{A}$-transcript, it follows that $T_{\mathcal{P}} = \sigma$ if and only if $y_i = \Psi(h_1, f, f, h_2)(x_i)$ for all $1 \leq i \leq q_c$ and $y'_j = f(x'_j)$ for all $1 \leq j \leq q_o$. If we define

$$\begin{aligned} S_i &= L_i \oplus h_1(R_i); \\ T_i &= W_i \oplus h_2(V_i), \end{aligned}$$

then

$$V_i \cdot W_i = P(L_i \cdot R_i) \Leftrightarrow f(S_i) = T_i \oplus R_i \text{ and } f(T_i) = V_i \oplus S_i.$$

Now observe that for all $1 \leq i < j \leq q_c$, $S_i \neq S_j$ and $T_i \neq T_j$ (otherwise $\sigma \in \mathsf{Bad}(h_1, h_2)$). Similarly, for all $1 < i, j < q_c$, $S_i \neq T_j$. In addition, it follows again from the fact that $\sigma \notin \mathsf{Bad}(h_1, h_2)$ that for all $1 \leq i \leq q_c$ and $1 \leq j \leq q_o$, $x_i' \neq S_j$ and $x_i' \neq T_j$. So, if $\sigma \notin \mathsf{Bad}(h_1, h_2)$ all the inputs to f are distinct. Since f is a random function, $\Pr_f[T_{\mathcal{P}} = \sigma] = 2^{-(2q_c+q_o)n}$ (the cipher transcript contributes 2^{-2nq_c} and the oracle transcript contributes $2^{-q_o n}$ to the probability).

Thus, for every choice of h_1, h_2 such that $\sigma \notin \mathsf{Bad}(h_1, h_2)$, the probability that $T_{\mathcal{P}} = \sigma$ is exactly the same: $2^{-(2q_c+q_o)n}$. Therefore:

$$\Pr_{\mathcal{P}}[T_{\mathcal{P}} = \sigma | \sigma \notin \mathsf{Bad}(h_1, h_2)] = 2^{-(2q_c+q_o)n}.$$

which completes the proof of the lemma. ■

The rest of the proof consists of using the above lemma and propositions 1 and 7 in a probability argument. In the following, we abuse notation by listing the adversary's oracles as $\langle \mathcal{P}, \mathcal{P}^{-1}, f \rangle$, and $\langle \mathsf{Perm}^{2n}, (\mathsf{Perm}^{2n})^{-1}, f \rangle$ to remind the reader that the adversary is given oracle access to both the forward and inverse directions of the permutation, as well as to an internal round function f. We start with the following preliminary proposition.

Proposition 8 *Let Γ be the set of all possible and consistent transcripts σ such that $C_{\mathcal{A}}(\sigma) = 1$. Then:*

$$\mathsf{Adv}(\langle \mathcal{P}, \mathcal{P}^{-1}, f \rangle, \langle \mathsf{Perm}^{2n}, (\mathsf{Perm}^{2n})^{-1}, f \rangle) \leq \sum_{\sigma \in \Gamma} (\Pr_{\mathcal{P}}[T_{\mathcal{P}} = \sigma] - \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma]) + \binom{q_c}{2} \cdot 2^{-2n},$$

where $\mathcal{A}$ is restricted to making at most q_c cipher queries to its oracles.

Proof: First, we break up the left hand side of the above inequality so that the distinguishability of both distributions are taken with respect to the process $\tilde{R}$:

$$\begin{aligned} &\mathsf{Adv}(\langle \mathcal{P}, \mathcal{P}^{-1}, f \rangle, \langle \mathsf{Perm}^{2n}, (\mathsf{Perm}^{2n})^{-1}, f \rangle) \\ &\quad \leq \mathsf{Adv}(\langle \mathcal{P}, \mathcal{P}^{-1}, f \rangle, \langle \tilde{R}, f \rangle) + \mathsf{Adv}(\langle \tilde{R}, f \rangle, \langle \mathsf{Perm}^{2n}, (\mathsf{Perm}^{2n})^{-1}, f \rangle). \end{aligned} \tag{5.1}$$

Now, since $\mathcal{A}$ with an oracle for $\mathcal{P}$ only generates possible and consistent transcripts, we can apply theorem 1 from chapter 2 to the first summand in equation 5.1. Next, we can apply proposition 1 from chapter 3 to the second summand. The proposition follows. ■

We now prove the main theorem:

Proof: (of Theorem 18) First we successively apply proposition 8 given above, and theorem 2 from chapter 2 with the set $\mathsf{Bad}(h_1, h_2)$:

$$\begin{aligned}&\mathsf{Adv}(\langle \mathcal{P}, \mathcal{P}^{-1}, f\rangle, \langle \mathsf{Perm}^{2n}, (\mathsf{Perm}^{2n})^{-1}, f\rangle)\\ &\quad \le \sum_{\sigma\in\Gamma}(\Pr_{\mathcal{P}}[T_{\mathcal{P}} = \sigma] - \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma]) + \binom{q_c}{2}\cdot 2^{-2n}\end{aligned}$$

Now, applying theorem 2:

$$\begin{aligned}&\sum_{\sigma\in\Gamma}(\Pr_{\mathcal{P}}[T_{\mathcal{P}} = \sigma] - \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma])\\ &\le \sum_{\sigma\in\Gamma}\left(\Pr_{\mathcal{P}}[T_{\mathcal{P}} = \sigma \mid \sigma \notin \mathsf{Bad}(h_1, h_2)] - \Pr_{\tilde{R}}[T_{\tilde{R}} = \sigma]\right)\cdot \Pr_{\mathcal{P}}[\sigma \notin \mathsf{Bad}(h_1, h_2)]\\ &+ \Pr_{\mathcal{P}}[\sigma^* \in \mathsf{Bad}(h_1, h_2)].\end{aligned}$$

Observe that according to lemma 8:

$$\left(\Pr_{\mathcal{P}}[T_{\mathcal{P}} = \sigma \mid \sigma \notin \mathsf{Bad}(h_1, h_2)] - \Pr_{\mathsf{Perm}^{2n}}[T_{\mathsf{Perm}^{2n}} = \sigma]\right)\cdot \Pr_{\mathcal{P}}[\sigma \notin \mathsf{Bad}(h_1, h_2)] = 0.$$

In addition, observe that proposition 7 tells us that

$$\Pr_{\mathcal{P}}[\sigma^* \in \mathsf{Bad}(h_1, h_2)] \le q_c^2\epsilon_1 + 2q_oq_c\epsilon_3 + \binom{q_c}{2}\cdot 2\epsilon_2.$$

Combining these observations, we get:

$$\begin{aligned}&\mathsf{Adv}(\langle \mathcal{P}, \mathcal{P}^{-1}\rangle, \langle \mathsf{Perm}^{2n}, (\mathsf{Perm}^{2n})^{-1}\rangle)\\ &\qquad\qquad \le q_c^2\epsilon_1 + 2q_oq_c\epsilon_3 + \binom{q_c}{2}\cdot(2\epsilon_2 + 2^{-2n}),\end{aligned}$$

which completes the proof. ■

Theorem 17 follows as a corollary using standard techniques. The proof looks nearly identical to the proof of theorem 10 given in chapter 3.

5.8 Round Security of Universal Hash Function Based Message Authentication Codes

We can analyze the round security of the Wegman-Carter MAC similarly to the round security of the Luby-Rackoff construction. Recall that the first round of a Wegman-Carter MAC is a hash function h, and the second round is a pseudorandom function f. Thus the number of possibilities for the set K of oracles for internal rounds is smaller: the adversary already has access to $1 \to 2$; so the only oracles of interest are $K = 1 \to 1$ (an oracle for h) and $K = 2 \to 2$ (an oracle for f).

Negative Results

If we give the adversary oracle access to h, then the scheme is not, in general, secure. This is so because, for many hash functions (in particular, for the one presented in the example in Section 2), given sufficiently many input-output pairs one can construct a new pair of values x, y such that $h(x) = h(y)$, but $x \neq y$. In such a case, $S_a(x) = S_a(y)$. So, once the adversary queries S_a on x, it will be able to output a valid MAC for y.

Positive Results

Surprisingly, it turns out that if we give the adversary oracle access to f, the Wegman-Carter MAC continues to be secure, provided h is uniform, in addition to being universal. This is not a serious restriction on h, because most natural examples of universal hash functions exhibit good uniformity properties.

Theorem 19 *Let $\mathcal{F}$, H, $\mathcal{U}$, $\mathcal{M}$ be as in Theorems 7 and 8, with the additional property that H is ϵ_1-uniform. Let $K = \{2 \to 2\}$, let q_o denote the number of queries made to the oracle K, and let q_m denote the number of queries made to the function family $\mathcal{M}$ ($q = q_m + q_o$). Then*

$$\mathsf{Adv}^{\mathsf{mac}}_{\mathcal{M},H,\mathcal{F}}(q,t,K) \leq \mathsf{Adv}^{\mathsf{prf}}_{\mathcal{F}}(q,t') + \binom{q}{2}\epsilon + q^2\epsilon_1/4 + 1/2^l,$$

where $t' = t + O(k + l + s_2 + q_m t_h + q_o n)$.

The theorem can be proven via the techniques used for the block cipher case – in fact, the argument is simpler because there are fewer bad conditions to analyze. We omit the proof for this reason. We point out that our results imply the somewhat paradoxical fact that S remains pseudorandom even if the adversary has oracle access to its only pseudorandom component f.

5.9 Conclusion

In this chapter we discussed a new model for analyzing the security of symmetric-key cryptographic primitives which accounts for the fact that many such primitives have an induced round structure. The model allows the adversary black-box access to some of the internal keyed functions in the primitive. We analyzed the security of various four-round Luby-Rackoff ciphers in this new model, and showed that the cipher remains secure even if we give the adversary black-box access to the round function f. We explained how our results also apply to universal hash function based message authentication codes, and to the Even-Mansour [50] block cipher.

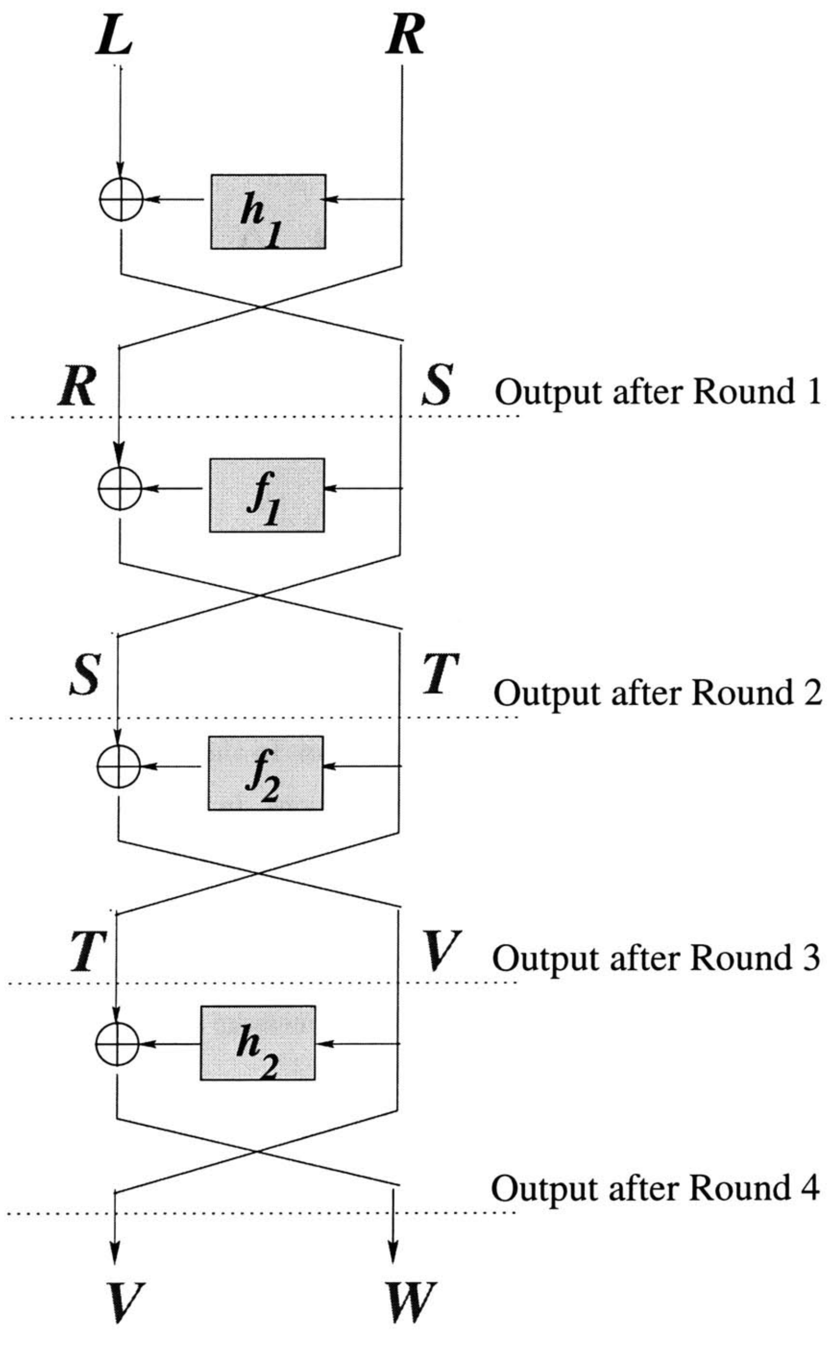

Figure 5-1: The natural round structure of Luby-Rackoff style ciphers

Chapter 6

Square Hash: A Fast Δ-universal Hash Function

6.1 Introduction

Many of the previously described results of the thesis utilize universal families of hash functions. Such hash functions provably possess particular statistical properties and can be implemented without any cryptographic assumptions. In this chapter we discuss two new ideas in the construction of fast universal hash functions. In addition to their applicability in block cipher design as we describe in chapters 3, 4, 5, these hash functions can actually be used for the purposes of message authentication, which is where we focus our attention in this chapter. A less detailed description of these results appeared in a paper by Etzel, Patel, and Ramzan [49]. We describe the problem of message authentication, together with various approaches to solving it, in chapters 1 and 2.

The starting point of our construction is a simple but novel family of universal hash functions that is more efficient than many standard constructions. We compare our hash functions to the MMH family studied by Halevi and Krawczyk [65]. We then introduce additional techniques for speeding up our constructions; these techniques apply to MMH and may apply to other universal hash functions. The techniques involve ignoring certain parts of the computation, while still retaining the necessary statistical properties for secure message authentication. We substantiate our claims through implementation results on an ARM (Advanced RISC Machine) processor. Our constructions are general and can be used in any setting where universal hash functions are needed. Consequently, they may be of independent interest.

The remainder of this chapter is organized as follows. In section 6.2 we outline the basic square hash. In section 6.3 we compare the basic square hash to MMH and describe the optimizations to our basic construction. In the two sections thereafter, we discuss various implementation considerations, and give concrete implementation results. Finally, in section 6.6 we make some concluding remarks.

6.2 The Square Hash

The square hash introduces two new ideas in the construction of fast universal hash functions. We start with a simple, but novel family of universal hash functions which is more efficient than certain well-known hash function families. The efficiency lies in the fact that whereas other common constructions involve integer multiplications, our construction involves integer squaring. It turns out that squaring a large integer requires fewer basic word multiplications than multiplying two large integers [95], so we get a speed-up. In certain architectures, multiplication takes significantly more time than other basic arithmetic operations, so we can get good savings with this approach. Our second idea is to optimize the implementation of this hash function by ignoring certain parts of the computation; moreover, we formally prove that, despite ignoring these parts of the computation, the bound ϵ on the resulting optimized hash function is still low enough to provide for a very secure MAC. One can think of this approach as "theoretical hacking." Specifically, the second new idea discussed in this chapter is to completely ignore some of the *carry bits* when performing the computation of the hash function in our basic construction. Since carry bits can be cumbersome to deal with, we can save computational effort in this manner. We stress that this savings will primarily occur when our tag size is several words long since some microprocessors, such as the Intel Pentium, allow you to multiply two 32-bit words, and get a 64-bit result with all the carries "for free."

At first it seems counterintuitive that we can simply ignore what appears to be a crucial part of the computation. However, we are able to show that our MAC algorithms are still secure for natural choices of the parameters. Square hash builds on some of the ideas in the MMH construction of Halevi and Krawczyk [65]; Knudsen independently proposed a construction similar to square hash for use in block cipher design [77]. We start with an underlying hash function which is similar to the one used in MMH; however, our new hash function performs fewer multiplications. In MMH, the final carry bit of the output is ignored – in square hash we extend this idea by showing that we can ignore almost all of the carry bits and still get quite a reasonable trade-off in security. Hence, in theory, square hash should be a strong alternative to MMH. We have implementation results on an ARM processor to substantiate this claim. Moreover, since word blocks in the input can be worked on independently, our constructions are parallelizable.

We now describe some basic constructions of universal hash function families based on squaring. We also examine modifications that enable faster implementations at negligible costs in collision probability. In the definitions and theorems that follow, we work over the integers modulo p where p is a prime.

6.2.1 The Basic Construction

Definition 36 *Define the SQH family of functions from $\mathbb{Z}_p$ to $\mathbb{Z}_p$ as: $SQH \equiv \{h_x : \mathbb{Z}_p \longrightarrow \mathbb{Z}_p | x \in \mathbb{Z}_p\}$ where the functions h_x are defined as:*

$$h_x(m) \equiv (m + x)^2 \bmod p.$$

Theorem 20 *The family SQH is Δ-universal.*

Proof: For all $m \neq n \in \mathbb{Z}_p$, and $\delta \in \mathbb{Z}_p$:

$$\begin{aligned}\Pr_x[h_x(m) - h_x(n) = \delta] &= \Pr_x[(m + x)^2 - (n + x)^2 = \delta] \\ &= \Pr_x[m^2 - n^2 + 2(m - n)x = \delta] \\ &= 1/p.\end{aligned}$$

The last inequality follows since for all $m \neq n \in \mathbb{Z}_p$ and $\delta \in \mathbb{Z}_p$ there is a unique x which satisfies the equation $m^2 - n^2 + 2(m - n)x = \delta$. ■

6.2.2 Comparison with Linear Congruential Hash

We compare the square hash to the linear congruential hash, which is one of the most heavily referenced universal hash functions in the literature. We define the linear congruential hash (LCH) family of functions to be: $LCH \equiv \{h_{x,b} : \mathbb{Z}_p \longrightarrow \mathbb{Z}_p | x, b \in \mathbb{Z}_p\}$ where each of the functions $h_{x,b}$ are defined as:

$$h_{x,b}(m) \equiv mx + b \bmod p.$$

In most cases, the SQH family requires less computation time than LCH. The speed-up occurs because squaring an n-bit number requires roughly half the number of basic word multiplications than multiplying two n-bit numbers [95]; thus we can save when dealing with quantities that are several words long. One major caveat is that SQH is not a permutation. Thus one should not use it an application that requires a strongly-universal permutation. We now compare square hash with the MMH construction [65].

6.3 Comparison with MMH

Halevi and Krawczyk [65] studied a family of Δ-universal hash functions entitled MMH. MMH was originally defined by Carter and Wegman [36]. Halevi and Krawczyk discovered techniques to speed up the software implementation at negligible costs in the collision probabilities. These hash functions are suitable for very fast software implementation. They apply to hashing variable sized data and to fast cryptographic message authentication. In this section we compare our SQH family to MMH. We show that in theory SQH is more efficient with respect to both computation time and key sizes than MMH. We also show that all of the clever software optimizations discovered by Halevi and Krawczyk for MMH can be applied to SQH as well. Finally, we further optimize square hash by disregarding many of the carry bits in the computation. We now describe MMH* which is the basic non-optimized version of MMH.

6.3.1 Description of MMH*

Definition 37 *[65] Let $k > 0$ be an integer. Let $x = \langle x_1, \ldots, x_k \rangle$, and $m = \langle m_1, \ldots, m_k \rangle$, $x_i, m_i \in \mathbb{Z}_p$, The* MMH* *family of functions from $\mathbb{Z}_p^k$ to $\mathbb{Z}_p$ is defined as follows:* MMH* $\equiv$

$\{g_x : \mathbb{Z}_p^k \longrightarrow \mathbb{Z}_p \mid x \in \mathbb{Z}_p^k\}$ *where the functions* g_x *are defined as*

$$g_x(m) = m \cdot x = \sum_{i=1}^{k} m_i x_i \bmod p.$$

Theorem 21 [Halevi and Krawczyk]: MMH* *is a* Δ*-universal family of hash functions.*

Halevi and Krawczyk [65] also discussed a way to generalize these functions so that their range is $\mathbb{Z}_p^l$ rather than just $\mathbb{Z}_p$. This can be done via a Cartesian product type idea due to Stinson [137]. Specifically, we hash the message l times using l independently chosen keys and we concatenate the hashes. This yields a collision probability of $1/p^l$. At first this requires a much larger key size, but that can be reduced by applying a Toeplitz matrix type idea due to Krawczyk [80]; namely (for the case $l = 2$), choose $k+1$ scalars $x_1, \ldots, x_{k+1}$ and set the first key to be $\langle x_1, \ldots, x_k \rangle$ and the second key to be $\langle x_2, \ldots, x_{k+1} \rangle$. The collision probability reduces to roughly $1/p^2$.

6.3.2 A Variant of Square Hash Similar to MMH*

Definition 38 *Let* $k > 0$ *be an integer. Let* $x = \langle x_1, \ldots, x_k \rangle$, *and* $m = \langle m_1, \ldots, m_k \rangle$, $x_i, m_i \in \mathbb{Z}_p$. *The SQH* family of functions from* $\mathbb{Z}_p^k$ *to* $\mathbb{Z}_p$ *is defined as follows: SQH** $\equiv$ $\{g_x : \mathbb{Z}_p^k \longrightarrow \mathbb{Z}_p \mid x \in \mathbb{Z}_p^k\}$ *where the functions* g_x *are defined as*

$$g_x(m) = \sum_{i=1}^{k} (m_i + x_i)^2 \bmod p \qquad (6.1)$$

Theorem 22 *SQH* is a* Δ*-universal family of hash functions.*

Proof: Let $m \neq n \in \mathbb{Z}_p^k$ with $m = \langle m_1, \ldots, m_k \rangle$, $n = \langle n_1, \ldots, n_k \rangle$ and $m_i, n_i \in \mathbb{Z}_p$. Let $\delta \in \mathbb{Z}_p$. Since $m \neq n$ there is some i for which $m_i \neq n_i$. Without loss of generality, suppose $m_1 \neq n_1$. Now, we show: for all $x_2, \ldots, x_k \Pr_{x_1}[g_x(m) - g_x(n) = \delta] = 1/p$ (where $x = \langle x_1, \ldots, x_k \rangle$) which implies the lemma. So,

$$\begin{aligned} &\Pr_{x_1}[g_x(m) - g_x(n) = \delta] \\ &\quad = \Pr[\sum_{i=1}^{k} (x_i + m_i)^2 - \sum_{i=1}^{k} (x_i + n_i)^2 = \delta] \end{aligned}$$

$$
\begin{aligned}
&= \Pr[2(m_1 - n_1)x_1 = \delta - m_1^2 + n_1^2 - \sum_{i=2}^{k}(x_i + m_i)^2 + \sum_{i=2}^{k}(x_i + n_i)^2] \\
&= 1/p.
\end{aligned}
$$

The last equation follows since $(m_1 - n_1) \neq 0$ implies that there is a unique $x_1 \in \mathbb{Z}_p$ satisfying the equation inside the probability. ■

6.3.3 Comparing SQH^* to MMH*

SQH^* should be faster than MMH* because squaring can be implemented so it requires roughly half the number of basic word multiplications as multiplying two numbers. In particular, here is a fast squaring algorithm which is adapted from the one given by Menezes, Vanstone, and van Oorschot [95].

```
Description of fast squaring algorithm.
Input: $x = x_{w-1} \cdots x_0$ where $x_i \in \{0,1\}^l$, $0 \le i \le w-1$.
Output: $x^2$ as a concatenation of $l$-bit words.

1. **For** $i = 1$ to $2w - 1$ **do**:
        1a. $t_i \leftarrow 0$.
2. **For** $i = 0$ to $w - 1$ **do**:
        2a. $(u, v) \leftarrow t_{2i} + x_i \cdot x_i$, $t_{2i} \leftarrow v$, $c \leftarrow u$.
        2b. **For** $j = (i+1)$ to $(w-1)$ **do**:
                $(u, v) \leftarrow t_{i+j} + 2x_j x_i + c, t_{i+j} \leftarrow v, c \leftarrow u$.
        2c. $t_{i+t} \leftarrow u$.
3. $(u, v) \leftarrow t_{2w-2} + x_{w-1} \cdot x_{w-1}, t_{2w-2} \leftarrow v, t_{2w-1} \leftarrow u$.
4. Return $t_{2w-1} t_{2w-2} \cdots t_0$.
```

The above algorithm essentially makes use of the following observation. If you have positive integer $x = x_{w-1} \cdots x_0$ where each x_i $(0 \le i \le w-1)$ is an l-bit quantity, then you have to compute $x_i \cdot x_j$ for all $0 \le i < j \le w - 1$. This computation requires $w(w+1)/2$ basic l-bit word multiplications. On the other hand, if you want to multiply x with another quantity $y = y_{w-1} \cdots y_0$, then you would have to compute $x_i \cdot y_j$ for all $0 \le i, j \le w -$

1. This computation requires w^2 basic l-bit word multiplications. Thus squaring gives a speed-up factor of $(w(w+1)/2)/w^2 = (w+1)/2w$ in terms of basic word multiplications. Since multiplication is relatively expensive on many architectures, we may save considerably through the use of efficient squaring. There are a number of other techniques one may use to enhance the efficiency of these types of universal hash functions. In particular, Halevi and Krawczyk made several clever software optimizations on MMH; the same optimizations apply to SQH as well.

6.3.4 Speeding up MMH*

Halevi and Krawzyck [65] define MMH_{32}, which is an optimized version of MMH*:

Definition 39 *Set $p = 2^{32} + 15$ and let k be a positive integer. Let $x = \langle x_1, \ldots, x_k \rangle$, and $m = \langle m_1, \ldots, m_k \rangle$, $x_i, m_i \in \mathbb{Z}_p$. Define the MMH_{32} family of functions from $(\{0,1\}^{32})^k$ to $\{0,1\}^{32}$ as: $\mathsf{MMH}_{32} \equiv \{h_x : (\{0,1\}^{32})^k \longrightarrow \{0,1\}^{32} \mid x \in (\{0,1\}^{32})^k\}$ where the functions h_x are defined as*

$$h_x(m) = (((\sum_{i=1}^{k} m_i x_i) \bmod 2^{64}) \bmod p) \bmod 2^{32}.$$

Theorem 23 **[Halevi and Krawczyk]:** *MMH_{32} is an ϵ-almost-Δ-universal family of hash functions with $\epsilon \leq 6 \cdot 2^{-32}$.*

The same optimization applies to SQH^*, and we discuss this now.

6.3.5 Speeding up SQH^*

Here is a variant of square hash, called SQH_{asm}, which is suited for assembly language implementation since both its domain and range consist of bit strings of length l. Ideally, l should be chosen as a multiple of the word size on the processor on which the algorithm will be implemented in order to enhance efficiency.

Definition 40 *Let l and k be positive integers, and let $2^l < p < 2^l + 2^{l-1}$. Let $x = \langle x_1, \ldots, x_k \rangle$, and $m = \langle m_1, \ldots, m_k \rangle$, $x_i, m_i \in \mathbb{Z}_p$. The SQH_{asm} family of functions from $\mathbb{Z}_p^k$ to $\{0,1\}^l$ is defined as follows: $SQH_{asm} \equiv \{g_x : \mathbb{Z}_p^k \longrightarrow \{0,1\}^l \mid x \in \mathbb{Z}_p^k\}$, where the* *functions g_x are defined as*

$$g_x(m) = ((\sum_{i=1}^{k}(m_i + x_i)^2) \bmod p) \bmod 2^l.$$

Theorem 24 *SQH_{asm} is an ϵ-almost-Δ-universal family of hash functions with $\epsilon \leq 3 \cdot 2^{-l}$.*

Proof: Let $\delta \in \{0,1\}^l$ be chosen arbitrarily. Let $m \neq n$ be arbitrary message vectors. Let x be a key such that $h_x(m) - h_x(n) \equiv \delta \pmod{2^l}$, where $h \in SQH^*$. Equivalently,

$$h'_x(m) - h'_x(n) \equiv \delta \pmod{2^l}$$

where $h' \in SQH_{asm}$. Now, both $h_x(m)$ and $h_x(n)$ are in the range $0, \ldots, p-1$. Therefore, their difference taken over the integers lies in the range $-p+1, \ldots, p-1$. If we denote $p = 2^l + t$ where $0 < t < 2^{l-1}$ then:

$$h'_x(m) - h'_x(n) \in \begin{cases} \{\delta, \delta - 2^l\} & t \leq \delta \leq 2^l - t \\ \{\delta - 2^l, \delta, \delta + 2^l\} & 0 \leq \delta \leq t - 1 \\ \{\delta, \delta - 2^l, \delta - 2^{l+1}\} & 2^l - t < \delta \leq 2^l - 1 \end{cases}$$

Therefore, there are at most three values for the quantity $h_x(m) - h_x(n)$ which cause $h_x(m) - h_x(n) \equiv \delta \pmod{2^l}$. Since SQH^* is a Δ-universal hash function, it follows that for any $\delta' \in \{0,1\}^l$ there is at most one choice of the key x for which

$$h_x(m) - h_x(n) \equiv \delta' \bmod p.$$

So, at most 3 keys satisfy the equation

$$h'_x(m) - h'_x(n) \equiv \delta' \pmod{2^l}.$$

Therefore: $\Pr_x[h_x(m) - h_x(n) \equiv \delta \pmod{2^l}] \leq 3 \cdot 2^{-l}$. ■

6.3.6 A Further Speed-Up

There is a minor weakness in SQH_{asm}. Since arithmetic is done modulo p, with $2^l < p < 2^l + 2^{l-1}$, elements in $\mathbb{Z}_p$ are $l+1$ bits long. We would, however, like to make l the word size of the machine on which we are implementing our code in order to speed up computations. Typically $l = 32$ (e.g. Pentium processors), or may be has high as 64 (Intel Merced). Having to deal with $l+1$ bit quantities means that we have to store and square *several* extra words. A first solution is to restrict both m_i and x_i to be at most l bits since we have flexibility over the choice of domain. Unfortunately, $m_i + x_i$ may be an $l+1$ bit quantity which means we still need to store and square extra words. It turns out that we simply can ignore the most significant bit of $m_i + x_i$ at just a minor cost in the important statistical properties of the new hash function. We give another square hash variant and prove that it performs well.

Definition 41 *Let l and k be positive integers with $2^l < p < 2^l + 2^{l-1}$. Let $x = \langle x_1, \ldots, x_k \rangle$, and let $m = \langle m_1, \ldots, m_k \rangle$, $x_i, m_i \in \{0,1\}^l$. Define the SQH_{asm2} family of functions from $(\{0,1\}^l)^k$ to $\{0,1\}^l$ as: $SQH_{asm2} \equiv \{g_x : (\{0,1\}^l)^k \longrightarrow \{0,1\}^l \mid x \in \{0,1\}^l\}$ where the functions g_x are defined as*

$$g_x(m) = ((\sum_{i=1}^{k}((m_i + x_i) \bmod 2^l)^2) \bmod p).$$

So, all we are doing is ignoring the most significant bit of $x_i + m_i$. This means that the sum will fit into l bits, which means that we do not have to use an extra word to both store and square.

Theorem 25 *SQH_{asm2} is an ϵ-almost-Δ-universal family of hash functions with $\epsilon \leq 2 \cdot 2^{-l}$.*

Proof: Let $m \neq n \in \{0,1\}^l$ with $m = \langle m_1, \ldots, m_k \rangle$, $n = \langle n_1, \ldots, n_k \rangle$ and $m_i, n_i \in \{0,1\}^l$. Let $\delta \in \mathbb{Z}_p$. Since $m \neq n$ there is some i for which $m_i \neq n_i$. Without loss of generality, suppose $m_1 \neq n_1$. Now, we show that for all $x_2, \ldots, x_k$:

$$\Pr_{x_1}[g_x(m) - g_x(n) = \delta] \leq 2/2^l$$

(where $x = \langle x_1, \ldots, x_k \rangle$), which implies the lemma. First, let

$$A \quad = \quad \sum_{i=2}^{k}((x_i + m_i) \bmod 2^l)^2 \bmod p,$$

and let

$$B = \sum_{i=2}^{k} ((x_i + n_i) \bmod 2^l)^2 \bmod p.$$

Then:

$$\begin{aligned}
&\Pr_{x_1}[g_x(m) - g_x(n) = \delta] \\
&= \Pr_{x_1}[(((x_1 + m_1) \bmod 2^l)^2 + A) - (((x_1 + n_1) \bmod 2^l)^2 + B) \equiv \delta \pmod{p}] \\
&= \Pr_{x_1}[((x_1 + m_1) \bmod 2^l)^2 - ((x_1 + n_1) \bmod 2^l)^2 \equiv B - A + \delta \pmod{p}].
\end{aligned}$$

Since x_1 and m_1 are both l-bit quantities, their sum will be at most $2^{l+1} - 2$, which means that to reduce this quantity mod 2^l we have to subtract off at most 2^l. Therefore,

$$((x_1 + m_1) \bmod 2^l) = x_1 + m_1 - 2^l c(x_1, m_1),$$

where $c(x_1, m_1)$ is some value in $\{0, 1\}$. In this case, c is the *carry* or *overflow* bit associated with adding x_1 and m_1. Similarly, we can write

$$((x_1 + n_1) \bmod 2^l) = x_1 + n_1 - 2^l c(x_1, n_1).$$

Replacing these equations into the above and performing some arithmetic manipulation, we get:

$$\begin{aligned}
&\Pr_{x_1}[((x_1 + m_1) \bmod 2^l)^2 - ((x_1 + n_1) \bmod 2^l)^2 \equiv B - A + \delta \pmod{p}] \\
&\quad = \Pr_{x_1}[2x_1((m_1 - n_1) + (c(x_1, n_1) - c(x_1, m_1)) \cdot 2^l) \equiv \delta' \pmod{p}].
\end{aligned}$$

Where

$$\delta' = B - A + \delta + (n_1 - c(x_1, n_1)2^l)^2 - (m_1 - c(x_1, m_1)2^l)^2.$$

Now,

$$(c(x_1, n_1) - c(x_1, m_1)) \cdot 2^l \in \{-2^l, 0, 2^l\}.$$

However, since $m_1, n_1 \in \{0,1\}^l$ and since $m_1 \neq n_1$, it follows that

$$(m_1 - n_1) \in \{(1-2^l), \ldots, -1, 1, \ldots, (2^l - 1)\},$$

which implies that

$$((m_1 - n_1) + (c(x_1, n_1) - c(x_1, m_1)) \cdot 2^l) \neq 0,$$

and for a given $c(x_1, m_1)$ and $c(x_1, n_1)$ there is at most one value of x_1 satisfying the above equations. Finally, we have

$$\begin{aligned}
&\Pr_{x_1}[2x_1((m_1 - n_1) + (c(x_1, n_1) - c(x_1, m_1)) \cdot 2^l) \equiv \delta'] \\
&\qquad \leq \Pr_{x_1}[2x_1((m_1 - n_1) - 2^l) \equiv \delta' \pmod p \mid c(x_1, n_1) - c(x_1, m_1) = -1] \\
&\qquad + \Pr_{x_1}[2x_1(m_1 - n_1) \equiv \delta' \pmod p \mid c(x_1, n_1) - c(x_1, m_1) = 0] \\
&\qquad + \Pr_{x_1}[2x_1((m_1 - n_1) + 2^l) \equiv \delta' \pmod p \mid c(x_1, n_1) - c(x_1, m_1) = 1] \\
&\qquad \leq 3/2^l.
\end{aligned}$$

This gives us a bound of $3/2^l$. We can improve this to $2/2^l$ by observing that for fixed values of m and n, $c(x_1, n_1) - c(x_1, m_1)$ cannot simultaneously take on the values $+1$ and -1 for varying choices of x_1. In particular, if $n_1 > m_1$ then we claim that $c(x_1, n_1) - c(x_1, m_1) \geq 0$. This follows because $c(x_1, n_1) - c(x_1, m_1) = -1$ implies $x_1 + n_1 < 2^l$ and $x_1 + m_1 \geq 2^l$ which implies that $m_1 > n_1$. Similarly, it can be shown that $n_1 < m_1$ implies $c(x_1, n_1) - c(x_1, m_1) \leq 0$. Thus $c(x_1, n_1) - c(x_1, m_1)$ takes on at most two possible values and ϵ is bounded by $2/2^l$.
■

6.3.7 Ignoring Carry Bits in the Computation

We now describe a way to further speed up square hash at a small tradeoff in the collision probability. The idea is novel, and applies not only to MMH but perhaps to other constructions of universal hash functions. Basically, we show that we can ignore many of the carry bits in the computation of square hash and still get very strong performance for cryptographic applications. In some sense this extends the ideas of Halevi and Krawczyk who speed up MMH by ignoring only the most significant carry bit. We start by describing the notion of carry bits and explain why computation can speed up if you ignore them. We then define variants of square hash in which you can ignore some of the carry bits, and show that the resulting security guarantee is still excellent for cryptographic applications. Finally, we define yet another variant of square hash in which you can ignore even more carry bits and show that the security guarantee is still strong for cryptographic applications under suitable settings for the parameters.

Carry Bits

When two words are added, there is usually an overflow or carry that takes place in the computation. For example, if the word size is 8, and you compute 11001001 + 10010001 you get 101011010. Since the word size is 8, the most significant bit 1 is called the carry or overflow bit because it overflowed from the usual 8 bit word size. Now, when arithmetic operations are performed on very long integers, as is usually the case for cryptographic applications, the carry bits between individual word operations are used for the next operation. So, if the word size is 8, and you are trying to compute 1011010100110101 + 1010101111100101 then the computation is broken up into parts. First, each bit string is broken up to take word size into account. The first string is broken up into two parts which we label A and B respectively: $A = 10110101$ and $B = 00110101$. The second string would be broken up into two parts: $C = 10101011$ and $D = 11100101$. Now, the computation is carried out as follows: first we compute $B + D$ store the answer in a word, and store the carry c_0 separately. Denote by E the word in which we store $B + D$. Then $E = 00011010$ and the carry bit $c_0 = 1$. Now, we compute $F = A + C + c_0$ and store the carry bit in c_1. Then $F = 01100001$ and the carry bit c_1 is 1. The total answer is the concatenation c_1FE: 10110000100011010. So, it is necessary to keep track of a carry bit as you do the computations on integers that require more than one word to represent. Unfortunately, certain instructions on processors do not deal with carry bits effectively (for example the Multiply with Accumulate instruction on the ARM). Also, even if an instruction saves the carry bit (for example, in the condition code register), this information may get destroyed when other instructions are executed. In addition, most high level programming languages do not deal with carry bits effectively; this increases the computation time of arithmetic instructions over integers that are several words long because it becomes necessary to explicitly keep track of the carry bit. High level programming languages are often preferable because they are portable and they facilitate software development. Also, various cross-platform languages, such as Java and Javascript, tend not to handle carry bits effectively. We show that we can overcome these dilemmas by ignoring the carry bits altogether. We call these variants of square hash SQH_c and SQH_{c2} since they can be effectively implemented with high level programming languages such as C. We also prove that we get strong performance despite ignoring what seems to be a crucial part of the computation.

Ignoring Carry Bits in the Outer Summation

We describe a preliminary speedup in which we ignore the carry bits in the outer summation, and show that we still get a good approximation to a Δ-universal hash function. First, we start with some notation.

Definition 42 *Let w, l be a positive integers, and let let $a_1, \ldots, a_k$ denote positive integers, each of which can be represented by wl-bits. We denote by:*

$$\sum_{1 \le i \le k}^{\langle l-\oplus \rangle} a_i$$

the value you get if you compute the sum $\sum_{i=1}^{n} a_i$ but ignore *the carry bits between the words, where l is the word length.*

For example, if you let the word size be 8 and compute $a_1 = 1011010100110101$ and $a_2 = 1010101111100101$ as in the above example, then

$$\sum_{1 \le i \le 2}^{\langle l-\oplus \rangle} a_i = 0110000000011010.$$

The normal sum $a_1 + a_2$ is 10110000100011010. But recall that the 9th least significant bit is the result of the carry from the summation of first pair of words, and the most significant bit is the result of the carry from the second pair of words. Since you are ignoring the carry bits, the function $\sum_{1 \le i \le k}^{\langle l-\oplus \rangle} a_i$ can be implemented much more efficiently than just the normal sum $\sum_{i=1}^{n} a_i$. This is especially true if the a_i are large integers (several words long). We now formally define a new variant of square hash and show that it still gives us strong performance.

Definition 43 *Let l and k be positive integers with $2^l < p < 2^l + 2^{l-1}$. Let $x = \langle x_1, \ldots, x_k \rangle$,* *and $m = \langle m_1, \ldots, m_k \rangle$, $x_i, m_i \in \mathbb{Z}_p$. The SQH_c family of functions from $\mathbb{Z}_p^k$ to $\mathbb{Z}_p$ is defined as follows: $SQH_c \equiv \{g_x : \mathbb{Z}_p^k \longrightarrow \mathbb{Z}_p \mid x \in \mathbb{Z}_p^k\}$ where the functions g_x are defined as*

$$g_x(m) = \sum_{1 \le i \le k}^{\langle l-\oplus \rangle} (m_i + x_i)^2 \bmod p.$$

Theorem 26 *Let l be the word size of the processor on which you are computing and let w be the number of words it takes to store x_i. Then SQH_c is an ϵ-almost-Δ-universal family of hash functions with $\epsilon \le 3^{2w}/2^{lw}$.*

Proof: Fix a value $a \in \mathbb{Z}_p$ and let $m = \langle m_1, \ldots, m_k \rangle \ne m' = \langle m'_1, \ldots, m'_k \rangle$ be your two messages. Assume without loss of generality that $m_1 \ne m'_1$. We prove that for all $x_2, \ldots, x_k \in \{0,1\}^{lw}$

$$\Pr_{x_1}[g_x(m) - g_x(m') = a \bmod p] \le 3^{2w}/2^{lw},$$

where $x = \langle x_1, \ldots, x_k \rangle$, which implies the theorem. Now, let us fix some choice of $x_2, \ldots, x_k$ and let

$$s = \sum_{2 \le i \le k}^{\langle l-\oplus \rangle} (m_i + x_i)^2.$$

That is, s denotes the sum of all but the first term where the carry bits are ignored between the words. Then,

$$\sum_{1 \le i \le k}^{\langle l-\oplus \rangle} (m_i + x_i)^2 = (x_1 + m_1)^2 + s - c,$$

where $c \in \{0,1\}^{2l+1}$ is a "correction vector" in which the i^{th} bit of c (counting from the left) contains a 1 if there was an overflow of 1 at that position (and contains a 0 otherwise). Observe that the correction taking place here is actually between the quantity $(x_1 + m_1)^2$ and the quantity s, which represents the remaining terms *after the carry bits have already been ignored.* In the example above with a_1 and a_2 the correction vector c is: 1000000100000000. Similarly, if we let

$$s' = \sum_{2 \le i \le k}^{\langle l-\oplus \rangle} (m'_i + x_i)^2,$$

then

$$\sum_{1\le i\le k}^{\langle l-\oplus\rangle} (m_i' + x_i)^2 = (x_1 + m_1')^2 + s' - c',$$

where c' is the associated correction vector. Therefore,

$$\begin{aligned}
&\Pr_{x_1}[g_x(m) - g_x(m') \equiv a \pmod p] \\
&\quad= \Pr_{x_1}[(x_1 + m_1)^2 + s - c - (x_1 + m_1')^2 - s' + c' \equiv a \pmod p] \\
&\quad= \Pr_{x_1}[x_1 \equiv (a + (c - c') + s' - s + m_1'^2 - m_1^2)/2(m_1 - m_1') \pmod p] \\
&\quad\le (\text{The number of distinct values } c - c' \text{ can take}) \cdot 2^{-lw}.
\end{aligned}$$

So we must derive a bound for the number of distinct values $c - c'$ can take. Now, c and c' consist mostly of 0's. In fact, the only positions of c in which there could be a 1 are the ones where there could be a carry. The first thing to observe is that s (and s') represent the sum of the remaining terms after the carry bits have been ignored, so the correction vectors c and c' only need to correct the effects of adding two quantities. The second thing to observe is that carries only occur at the boundaries between words. The first observation tells us that for a given pair of words, the correction vector only needs to correct for at most a single bit of error, and the second observation tells us where these correction bits must be. Thus only positions

$$l + 1, 2l + 1, 3l + 1, \ldots, 2wl + 1$$

of the correction vectors c and c' can possibly contain a 1. Now, for $c_{il+1} - c'_{il+1} \in \{-1, 0, 1\}$ for $1 \le i \le 2w$. Since there are only $2w$ bits that can get affected and 3 different values for their difference, the total number of different vectors $c - c'$ is bounded by 3^{2w}. So, we have that

$$\Pr_{x_1}[g_x(m) - g_x(m') \equiv a \pmod p] \le 3^{2w}/2^{lw},$$

which proves the theorem. ■

Now, observe that the quantity $3^{2w}/2^{lw}$ is actually rather small. We see this if we substitute suitable values for the parameters. If the word size l is 32 bits, then the ensuing value ϵ for output tags of size 2,3,4, or 5 words is at most 2^{-57}, 2^{-86}, 2^{-115}, and 2^{-144}

respectively. These are negligible and are smaller that what one may need for a reasonably secure MAC implementation. Also, keep in mind that these bounds are derived for the case that all carry bits in between word additions are ignored. In practice, one may only want to ignore some subset of these carry bits.

We now consider the question of whether we can optimize further at a slightly greater cost in security. The next section works towards this aim by showing that we can ignore even more carry bits at an increased cost in collision probability.

Ignoring Carry Bits When Squaring

Since the process of squaring can be expressed entirely in terms of doing basic word multiplications, shifts, and addition operations, we can consider the idea of ignoring the carry bits when performing additions during a squaring operation to further speed up our hash functions. We show that if we also ignore the carry bits that occur when the quantity $(x_i + m_i)$ is squared, then the resulting value of ϵ still provides adequate security for suitable values for the parameters. We start with some notation.

Definition 44 *Let w, l be a positive integers, and let let $a_1, \ldots, a_k$ denote positive integers, each of which can be represented by some multiple-word quantity. We denote by:*

$$a^{2\langle l-\oplus\rangle}$$

the value you get if you compute a^2, but ignore *the carry bits between the word when you perform the necessary additions. Here l is the word length.*

Now, we define a new square hash variant that ignores the carry bits when squaring.

Definition 45 *Let l and k be positive integers, with $2^l < p < 2^l + 2^{l-1}$. Let $x = \langle x_1, \ldots, x_k \rangle$, and $m = \langle m_1, \ldots, m_k \rangle$, $x_i, m_i \in \{0,1\}^l$. The SQH_{c2} family of functions from $(\{0,1\}^l)^k$ to $\mathbb{Z}_p$ is defined as follows: $SQH_{c2} \equiv \{g_x : (\{0,1\}^l)^k \longrightarrow \mathbb{Z}_p \mid x \in (\{0,1\}^l)^k\}$ where the functions g_x are defined as*

$$g_x(m) = \left(\sum_{1 \le i \le k}^{\langle l-\oplus\rangle} (m_i + x_i)^{2\langle l-\oplus\rangle} \bmod p \right). \qquad (6.2)$$

Note that we ignore carry bits when we square and when we take the sum over the $(x_i+m_i)^2$. However, we *do not* ignore the carry bits when we actually compute $(x_i + m_i)$. We can get away with ignoring these carry bits as well, but for simplicity of analysis we do not ignore them. We now state our main theorem about how well this new family of hash functions performs:

Theorem 27 *Let l be the word size of the architecture on which you are computing and let w be the number of words it takes to store x_i. Then SQH_{c2} is an ϵ-almost-Δ-universal family of hash functions with $\epsilon \leq (\prod_{i=1}^{w}(4i+1)^2)/2^{lw}$.*

Proof: The proof uses similar ideas to the proof of the previous theorem about ignoring carry bits in the outer summation. In particular, we define correction vectors c and c' in the same manner as before, and bound the number of distinct possibilities for their difference $c-c'$. We first observe that for the i^{th} word of c (counting from the right for $1 \leq i \leq w$) there are 1's in at most the $\log(2i+1)$ least significant bit positions of that word – this observation follows from the fact that the value at that word is obtained by adding $2i+1$ words (when squaring). The remaining bits must be 0. So, only the least significant $\log(2i+1)$ bits in word i are undetermined. Similarly, for word j with $w+1 \leq j \leq 2w$, the least significant $\log(4w+3-2j)$ are undetermined. Now, if the b least significant bits of each of two different words are undetermined, then the value attained in each case lies in the range $\{0,\ldots,2^b-1\}$. Thus, the difference of those two words lies in the range $\{-2^b+1,\ldots,2^b-1\}$, hence the difference can take on at most

$$(2^b-1)-(-2^b+1)+1 = 2^{b+1}-1$$

different values. The number of distinct possible values for $c-c'$ is the product of the number of different possible values each of the individual words can take. This quantity equals:

$$\begin{aligned}
(\prod_{i=1}^{w}(2^{\log(2i+1)+1}-1) \cdot \prod_{j=m+1}^{2w}(2^{\log(4w+3-2j)+1}-1)) \\
&= (\prod_{i=1}^{w} 2^{\log(2i+1)+1}-1)^2 \\
&= (\prod_{i=1}^{w} 4i+1)^2,
\end{aligned}$$

which yields the desired bound. ■

Although this expression looks large, for suitable values of the parameters it still gives good security. Keep in mind that typically $1 \leq w \leq 5$. In particular, if the word size l is 32 bits, and we hash down to 2,3,4, or 5 words, then *computationally unbounded* adversaries will fail to forge the MAC with probability better than 2^{-53}, 2^{-77}, 2^{-101}, or 2^{-124} respectively. Again, these bounds occur for the case that all carry bits are ignored – in practice, one may want to ignore some subset of the bits and obtain better security guarantees.

6.3.8 Fully Optimized Square Hash

We present the fully optimized version of square hash:

Definition 46 *Let l and k be positive integers with $2^l < p < 2^l + 2^{l-1}$. Let $x = \langle x_1, \dots, x_k \rangle$, and $m = \langle m_1, \dots, m_k \rangle$, $x_i, m_i \in \{0,1\}^l$. The SQH_E family of functions from $(\{0,1\}^l)^k$ to $\{0,1\}^l$ is defined as follows: $SQH_E \equiv \{g_x : (\{0,1\}^l)^k \longrightarrow \{0,1\}^l \mid x \in \{0,1\}^l\}$, where the functions g_x are defined as*

$$g_x(m) = \left(\left(\sum_{1 \leq i \leq k}^{\langle l - \oplus \rangle} ((m_i + x_i) \bmod 2^l)^{2 \langle l - \oplus \rangle} \right) \bmod p \right) \bmod 2^l. \tag{6.3}$$

Theorem 28 *Let l be the word size on which you are computing and w is the total number of words needed to store x_i. Then SQH_E is an ϵ-almost-Δ-universal family of hash functions with $\epsilon \leq (6 \cdot \prod_{i=1}^{w} (4i+1)^2)/2^{lw}$*

Proof: The theorem follows by combining the statements and proofs of the previous theorems. ■

6.3.9 Comparison to NMH

At the end of their paper, Halevi and Krawczyk [65] briefly discussed another family of Δ-universal hash functions called NMH. It would be interesting to do a detailed comparison between NMH and SQH that studies speed and required key sizes. Another interesting area for future research would be to apply some of our techniques of ignoring carry bits to MMH and NMH.

6.4 Considerations on Implementing Square Hash

In this section, we discuss various important implementation considerations, and in the next we give actual implementation results. To start with, square hash should be faster since we use squaring instead of multiplication. The speed-up factor for squaring an w-word integer versus multiplying two w word integers (in terms of basic word multiplications) is $(w-1)/2w$. Typically, MACs have tag sizes between 32 and 160 bits, depending on the level of security needed. Therefore, on 32-bit architectures, $1 \leq w \leq 5$ and we get speed up factors of %0, %25, %33, %38, and %40 for the different values of w. Now, on most slower architectures, multiplications require many more clock cycles than other simple arithmetic operations such as addition. For example, on the original Pentium processor, the ratio between number of clock cycles for unsigned multiplication versus addition is about 5:1. This ratio probably gets much larger on weaker processors such as those on cellular phones, embedded devices, smart-cards, etc,. Moreover, for these types of smaller processors, word sizes may be smaller, hence the number of words we multiply increases, and the savings we achieve by using squaring rather than multiplication greatly increases. Thus, we recommend using square hash on such architectures. On some of the more modern processors such as the Pentium Pro and Pentium II, multiplications do not take much more time than additions (closer to 2:1, [42]), so square hash is not advantageous is such cases.

Another important implementation consideration is the memory architecture of the processor on which you are implementing. In our case, we need extra data registers to quickly implement squaring. On Pentium architectures there are only 4 32-bit data registers [42]. Hence, we may need to make additional memory references which could slow things down. On the other hand, the PowerPC has 32 32-bit general purpose registers [122], which allows us to get fast squaring.

6.5 Implementation Results

We used the ARM (i.e. ARM7) processor to create hand-optimized assembly code to compare speeds of various algorithms. The ARM7 is a popular RISC processor and is used inside cellular phones, PDAs, smartcards, etc. It is a 32-bit processor with 16 general purpose registers. Basic operations like addition require 1 cycle whereas multiplication usually requires 6 cycles.

	MMH	*SQH1*	*SQH2 (some carries dropped)*	HMAC-SHA-1
Cycles	2061	1659	1575	4000+
Speedup over MMH	1x	1.24x	1.31x	.52x
Speedup over SHA-1	1.94x	2.41x	2.54x	1x
Security - ϵ	$6.25 \cdot 2^{-90}$	$6 \cdot 2^{-96}$	$2.53 \cdot 2^{-90}$	
Code Size (bytes)	408	3040	2704	4000+
Hash key Size (random bits)	2208 (2112+96)	2112	2112	

Table 6.1: Assembly implementations on ARM7 with 96 bit output and input block size of 2112 bits.

Our results show a significant speedup for square hash over MMH, and thus validate our theoretical results. For long messages and same or better security than MMH, square hash is 1.31 times faster than MMH (Table 1).

Message authentication using universal hash functions is typically performed by breaking up a long message (e.g 1 megabyte) in to smaller blocks (e.g. 2112 bits) and reducing each block, using an equivalent size hash key, down to the size of the MAC output or tag size (e.g. 96 bits). These tags are then concatenated, and the procedure is repeated until we are left with a single 96-bit tag. We can describe this process by an upside-down tree. The leaf nodes correspond to the blocks of the original message, and the data at each level represents one application of the above procedure. The root node is the final tag or output for the MAC. This tree hashing procedure adds about 10% overhead to both square hash and MMH and we omit it in the calculations presented in the tables for purposes of simplifying comparison. The security parameter ϵ as reported in the tables would have to be multiplied by the height of the tree [65].

We report results for a tag size of 96 bits since we believe it is a popular choice for message authentication in Internet standards (e.g. HMAC). Larger output sizes of 128 and 160 bits could further improve speedup factors due to greater savings on multiplications. We also report cycle counts for SHA-1 on an ARM7 to verify that we are faster than traditional non-universal hash based MACs (e.g. HMAC). To create the MAC, in actual use, MMH and square hash would have to encrypt the 96 bit output and HMAC-SHA-1 would need to perform a further SHA-1. We exclude this in the cycle counts in the tables to simplify

	MMH	*SQH1*	*SQH2 (some carries dropped)*	HMAC-SHA-1
Cycles	1086	856	816	2000
Speedup over MMH	1x	1.27x	1.33x	.54x
Speedup over SHA-1	1.84x	2.34x	2.45x	1x
Code Size	220	1544	1384	4000+
Hash key Size (random bits)	1152 (1056+96)	1056	1056	

Table 6.2: Assembly implementations on ARM7 with 96 bit output and input block size of 1056 bits.

comparison.

First for 2112-bit blocks (a multiple of 96) we compare MMH, $SQH1$, $SQH2$, and HMAC-SHA-1. $SQH1$ is the basic square hash function SQH_{asm} with the minor optimization of SQH_{asm2} giving an overall security of $6 \cdot 2^{-96}$ compared to the security of $6.25 \cdot 2^{-90}$ for 96 bit MMH. $SQH2$ is the same as $SQH1$, except that some carry bits are dropped in the squaring until the security is similar or better than that of MMH. As a result of dropping some carries, computation time decreases.

SHA-1 requires more than 1000 operations on 512-bit input and thus requires more than 4000 operations on 2112 bit input. All 3 universal hashes are significantly faster than the SHA-1 based MAC. $SQH1$ is 1.24 times as fast as MMH and $SQH2$ is 1.31 times as fast as MMH. Code sizes are somewhat large because of loop unrolling. Without unrolling additional computational time will be added to all three universal hashes to handle looping. The hash key (random bits) for MMH is 96 bits larger than square hash if the Toeplitz construction is used [65].

In Table 2 we also report cycle counts for 1056-bit blocks. Since 1024 bit blocks, as used by Halevi and Krawczyk [65], are not a multiple of 96, we used 1056 (a multiple of 32 and 96) as the input length. We ran experiments with messages that had the same size as the tag, and we noticed similar speedups. We also tested C versions of MMH and square hash and we saw similar speedups. Table 6.3 gives a break down of the instruction and cycle counts for both MMH and $SQH2$.

	Instructions	*Cycles*
MMH 66 (2112/32) words	105	687
key + message loading	28	188
Multiply + Accumulate	66	461
mod p reduction	5	9
function call	6	29
Total (3 times MMH 66 words)	105	2061
SQH2 3 words	29	69
key + message loading	2	10
Multiply	5	30
Multiply + Accumulate	1	7
Adds	21	21
22 (2112/96) times SQH2 (3 words)	638	1518
mod p reduction	16	25
function overhead	10	32
Total	664	1575

Table 6.3: MMH and SQH2 cycle count break downs: 96 bit output and block size of 2112 bits.

6.6 Conclusion

We described a new family of universal hash functions geared towards high speed message authentication. On some platforms, our hash functions appear to be faster than the MMH family, which itself is considered to be one of the fastest universal hash function implementations. We also introduced additional techniques for speeding up our constructions. These constructions and techniques are general and may be of independent interest.

Chapter 7

Conclusions and Extensions

This thesis has focused on both the highly theoretical and highly practical aspects of secure block cipher design. In this chapter, we briefly summarize these results, and point to some potential areas for future research.

7.1 Summary

In this dissertation we engaged in a detailed study of the Luby-Rackoff block cipher construction [88]. Our aim was to better understand the security of these ciphers, and attempt to make them more practical. We began by providing new constructions of Luby-Rackoff ciphers which were simultaneously faster and utilized less key material than previous constructions in the literature. In addition, we examined Luby-Rackoff ciphers when the underlying round function was merely unpredictable, rather than pseudorandom. In this case, we showed that the resulting block cipher had some unpredictability properties as well.

Next, we initiated a study of Luby-Rackoff ciphers over arbitrary finite algebraic structures. Our main result was the existence of a Luby-Rackoff cipher that is secure in one algebraic structure, yet completely insecure in another. The surprising aspect of this result is that cipher is broken when the operation is a bit-wise exclusive-or, but is secure when the operation is addition modulo 2^n, even though both operations are used extensively in practical block cipher design.

This cipher is the most optimal in terms of time complexity and key material, than any other Luby-Rackoff style cipher in the literature. Besides constructing a new cipher, we extended particular attacks for Luby-Rackoff ciphers that worked in one algebraic structure to attacks that worked in more arbitrary structures. These attacks required new ideas.

We also proposed a new notion called *round security* for studying symmetric-key primitives. This new way of thinking was inspired by the fact that many symmetric-key primitives, such as block ciphers, have an induced round structure. We completely characterized the original Luby-Rackoff construction [88], and some of its variants [101, 114], in this new model. This analysis led to a number of insights into the security of these schemes.

Finally, we proposed the construction of a new hash function called the square hash which can be used in many of the above constructions. Square hash was designed to be fast on modern microprocessors, and in some sense involved "theoretical hacking" in order to develop constructions which provably had various statistical properties, and which were fast. We implemented square hash on an ARM processor using hand-optimized assembly language, and validated our theoretical results.

7.2 Extensions and Future Work

Beyond providing faster constructions and novel notions of security, this dissertation opens up a number of interesting areas for future research, which we now outline.

7.2.1 Weaker Round Functions

In chapter 3, we considered the notion of what happens when the round function in a Feistel ladder is unpredictable instead of pseudorandom. In some sense this question is part of a broader area of study: what happens when you replace the pseudorandom function by a non-pseudorandom function in a Feistel ladder? In many ways gaining insight into a question like this would bridge the gap between theory and practice since most practical block cipher constructions involve taking a *simple* round function and iterating it many times. A related question was addressed by Maurer [94], Naor and Reingold [101], and Vaudenay [140]. In these papers, the authors replace the pseudorandom functions with k-wise independent functions, and examine the extent to which the whole cipher is a k-wise independent permutation. It would be interesting to build a general theory around Luby-Rackoff ciphers with weak round functions.

7.2.2 Arbitrary Algebraic Structures

In chapter 4, we initiated a study of Luby-Rackoff ciphers over arbitrary algebraic structures. We observed that there are situations in which a cipher is insecure when the traditional operation is the bitwise exclusive-or, but secure otherwise. In some sense many of the attacks that work in one setting do not translate so easily to the other. Thus, a general area for research would be to examine the various known results for Luby-Rackoff ciphers in the traditional setting, and examine the extent to which they hold in this more generalized setting.

We did this for a number of results pertaining to the cipher $\Psi(f^i, f^j, f^k)$ where f is a pseudorandom function. This cipher was broken for all i, j, and k by Zheng, Matsumoto, and Imai [145]. The attack involves the involutory properties of the exclusive-or operator. While we could extend the attack to generic algebraic structures for specific values of i, j, and k, we were unable to solve the general case. We leave this as an open problem.

Similarly, many of our attacks on the above ciphers required specific axioms, such as associativity or commutivaty of the underlying algebraic structure. It would be interesting to come up with attacks that work over non-associative or non-commutative algebraic structures. Or, more interestingly, prove that the schemes are secure in this setting.

7.2.3 Round Security of Other Constructions

In chapter 5, we introduced a new notion of security for symmetric-key primitives. The hope was to gain better insight into what makes these constructions secure. We successfully analyzed the Luby-Rackoff construction under this new model, and gained a number of interesting insights. We also successfully analyzed universal hash function based message authentication codes (Wegman-Carter MACs) in this model. One area for future research would be to apply this model to other constructions. For example, trying to cryptanalyze some of the AES finalist block ciphers (or idealized versions of them where some of the round functions are replaced with pseudorandom functions). There is much work to be done in this area.

7.2.4 Theory of Substitution-Linear Transformation Block Ciphers

This dissertation focused on ciphers that involve the Feistel ladder – indeed many ciphers seen in practical use fall into this category. Another design, known as the substitution-linear transformation network (SLN), has become fashionable recently. In fact Rijndael [41], which was named by the National Institute of Standards to be the advanced encryption standard (AES), is one such example. Also, Serpent [7] which, together with Rijndael, was one of the five finalists for the AES, is also a substitution-linear transformation network.

In a substitution-linear transformation network, essentially every operation is a permutation. Contrast this with a Feistel cipher in which the round function is usually not a permutation, though the Feistel ladder builds a permutation around it. In an SLN, the substitution portion usually involves an S-box that is invertible. The linear transformation portion typically involves one or more affine functions which are applied in a given round.

It would interesting to come up with a theory of secure SLNs. To do so might require a new approach. In particular, with Feistel ciphers, we made the assumption that the round function is pseudorandom, and proved that the entire cipher is a pseudorandom permutation. For an SLN, assuming that an entire component is pseudorandom would be too strong, since every component is already a permutation, which would leave nothing more to prove. To make this approach work, one might have to start with an assumption that is not too strong. With the advent of SLNs, and the dearth of good theoretical results about them, there are a number of viable research possibilities.

7.2.5 Design of Fast Universal Hash Functions

In chapter 6 we introduced the square hash which involves two novel ideas in the construction of fast universal hash functions. The natural question to ask is whether we can develop something faster. One effort along these lines is the UMAC construction [24]. It was developed after we developed square hash, and it is geared primarily towards the task of message authentication. In particular, the UMAC is not length preserving, and its statistical properties are not as good as those of square hash. Consequently, it cannot be immediately used for block cipher design. The UMAC construction, is however, extremely fast, so it may be possible to develop a new function, which involves several calls to UMAC, but is able to achieve the properties we need.

Appendix A

Description of the Data Encryption Standard

We now describe the Data Encryption Standard (or DES) [108]. It is perhaps the most widely used symmetric-key encryption algorithm. DES has been heavily studied, and even though it was first submitted for public comment in 1975, the only practical attack on DES is via a brute-force search over the entire key space. In this chapter we give a brief history of the development of DES, as well as a technical description.

A.1 History

In the early 1970s IBM was asked by one of its banking customers to design an algorithm for encrypting data sent between automatic teller machines (ATM) and a central server. IBM formed a team which consisted of people at its Kingston and Yorktown heights sites. The following people were on the team, among other consultants: Roy Adler, Don Coppersmith, Horst Feistel, Edna Grossman, Alan Konheim, Carl Meyer, Bill Notz, Lynn Smith, Walt Tuchman, and Bryant Tuckerman. Around this same time, the National Bureau of Standards (NBS), now known as the National Institute of Standards (NIST), issued an initial call for submissions for an encryption algorithm that would protect unclassified data.

Unfortunately, this initial call for submissions was not terribly succesful. Thus in August 1994, the NBS reissued its call for submissions, and the IBM team proposed an algorithm called Lucifer [136], that it had designed. The National Security Agency (NSA) became involved with this process to ensure that the chosen algorithm would be secure. Subsequently, IBM, in consultation with the NSA designed DES, which was based around Lucifer. DES was made available for public comment in 1975, and it was published in the Federal Register [108] as an official standard in 1977. Although the technical specification of the algorithm was made public, the original design criteria was not. As a result, people suspected that the NSA had embedded some type of hidden "trap door" so they could easily invert the algorithm. This claim was never validated. In 1994, Coppersmith, a member of the original DES design team, published an article that dispelled this notion [38]. His article described the various design considerations surrounding DES, and how it was, in fact, strengthened to prevent an attack technique known as differential cryptanalysis, which was known to the NSA at that time, but was discovered by the academic community much later [23].

A.2 Technical Description of DES

Having given some of the history of DES, we move on to a more technical description of the algorithm. DES is a block cipher that operates on 64-bit blocks, and has a 56-bit secret key. We first give a high level overview of DES, and then provide more details. DES encryption involves the following steps:

1. The 56-bit secret key k is expanded into 16 *round keys* $k_1, \ldots, k_{16}$. Each round key is 48-bits long, and is computed by permuting a particular subset of the bits in k.

2. The 64-bit plaintext message block P is subject to an initial permutation IP which simply changes the order of the individual bits. The initial permutation is a fixed function. We write $P_0 = \mathsf{IP}(P)$ to denote the value of P after the initial permutation is applied.

3. The value P_0 is subject to 16-rounds of a particular transformation, which works as follows. The input is divided into two halves $P_0 = L_0 \cdot R_0$ where L_0 and R_0 are both 32-bits long. Now, for $1 \leq i \leq 16$, the values L_i and R_i are computed as follows:

$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \oplus f(R_{i-1}, k_i). \end{aligned}$$

   Here f is a *round function* which we will specify later. This computation is known as a *Feistel permutation*, and is named after Horst Feistel, who was on the original DES design team.

4. Finally we apply the inverse of the initial permutation to the string $R_{16} \cdot L_{16}$. The ciphertext C is then defined by $C = \mathsf{IP}^{-1}(R_{16} \cdot L_{16})$.

We now describe how the round function f works. It takes two arguments: a 32-bit message block R_{i-1}, and a 48-bit key k_i. First, it applies an *expansion function* E to R_{i-1} to obtain a 48-bit quantity. Next, it takes the bit-wise exlusive-or of this quantity with the round key k_i. Let T denote the resulting value. That is

$$T = E(R_{i-1}) \oplus k_i.$$

Now, T_i, which is a 48-bit value, is broken up piecewise into eight 6-bit values $T_1, \ldots, T_8$. Next, a substitution box (S-box) is applied to each piece. In DES, there are eight substitution boxes, $S_1, \ldots, S_8$, which map 6-bit inputs to 4-bit outputs. For $1 \le j \le 8$, we let V_j denote $S_j(T_j)$. Thus $V = V_1 \ldots V_8$ is a 32-bit quantity. Now, the bits in this 32-bit quantity are permuted according to a fixed permutation P.

We now describe the details of the individual functions used in DES, and then go on to discuss the key scheduler. First, we describe the initial permutation IP. It can be described according to the following table:

Initial Permutation IP							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

The table can be interpreted as follows. The 58th bit of the plaintext P becomes the first bit of $IP(P)$. The 50th bit if P becomes the second bit of $IP(P)$, and so on. The inverse of IP is the described by the following table:

Initial Permutation Inverse IP^{-1}							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

We now describe the details of the round function f. To begin with we must specify the expansion function E. It can be described by the following table:

Expansion Function E					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

The S-boxes $S_1, \ldots, S_8$ will described according to the following 4 x 16 tables, which can be interpreted as follows. Suppose the input to the S-box is $b = b_1 \ldots b_6$, where for each $1 \leq j \leq 6$, b_j is a single bit. We use the pair of bits b_1, b_6 to determine a row of the table via the binary representation. That is, 00 would index the top-most row, $(0, 1)$ would index the second row from the top, 10 would index the third row from the top, and 11 would represent the bottom row. Now, we use the middle bits $b_2 b_3 b_4 b_5$ to represent a column via the binary representation. Thus, 0000 would index the leftmost column, and 1111 would index the rightmost column. Now to compute the value specified according to the S-box, we merely break the string up, and locate the value in the appropriate row and column. The output of the S-box is the binary representation of this value. As an example, the string 011000 would get mapped to top-most row, and the 13^{th} column from the left (note that

13th column is specified by the binary representation of 12 since the leftmost column starts at 0). Thus, the value attained if 011000 were given to the first S-box as input would be 0101, which is the binary representation of 5.

S-box S_1															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S-box S_2															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S-box S_3															
10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S-box S_4															
7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S-box S_5															
2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S-box S_6															
12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S-box S_7															
4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S-box S_8															
13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

In order to finish our specification of the f function, we must describe the permutation P which is applied to the output of the S-box. The following table describes P:

P Permutation			
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

The only remaining component left to discuss is the DES key scheduler. This algorithm takes as input a 56-bit key k, and produces 16 round keys $k_1, \ldots, k_{16}$, each of which is 48-bits. The first step is expanding the 56-bit key into a 64-bit key by inserting 8 parity check bits. The bits are inserted into positions 8, 16, 24, ..., 64. They are defined so that after each extra bit is added, the corresponding 8-bit block will have an odd number of ones. For example, if the initial 56-bit string is

0100100

0111001

0100100

0101010

1111001

0001110

1111110

1110011

then, after the parity check bits are inserted, the new string will be

01001001

01110011

01001001

01010100

11110010

00011100

11111101

11100110.

These extra bits are primarily used for error detection, and are ignored by the key scheduler. We have included them here since they were a part of the original DES specification [108]. Now, given the 64-bit key k, the first step is to ignore the parity bits, and compute the following fixed permutation PC1 on the remaining bits.

Permutation PC1						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

We write $C_0 \cdot D_0 = \mathsf{PC1}(k)$ where C_0 denotes the leftmost 28 bits pf $\mathsf{PC1}(k)$, and D_0 denotes the rightmost 28 bits. Now, for $1 \leq i \leq 16$, we compute values $C_i = \mathsf{LROT}_i(C_{i-1})$ and $D_i = \mathsf{LROT}_i(D_{i-1})$. The function LROT denotes a left rotation (cyclic shift) of the its input. For i = 1,2,9, and 16, LROT_i rotates its argument to the left by one bit. For all other values of i, LROT_i rotates its argument to the left by two bits.

Finally, the round key k_i is computed as $k_i = \mathsf{PC2}(C_i \cdot D_i)$, where PC2 is a fixed permutation defined according to the following table:

Permutation PC2					
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

We have described all the necessary components for encrypting a message with DES. To decrypt a message, all can utilize the same algorithm, but with the round keys in reverse order: $k_{16}, k_{15}, \ldots, k_1$. This approach works since the initial permutation and final permutation are inverses of each other, and computation of the inverse in a single round of DES, which is a Feistel permutation, looks identical to the computation associated with the forward direction of the permutation.

Bibliography

[1] C. Adams and S. Tavares. The use of bent sequences to achieve higher-order strict avalanche criterion. Technical Report 90–013, Queen's University, 1990.

[2] B. Adida. Self-describing cryptography. Master's thesis, Massachusetts Institute of Technology, June 1999. Available from: `http://ben.adida.net/`.

[3] V. Afanassiev, C. Gehrmann, and B. Smeets. Fast message authentication using efficient polynomial evaluation. In *Proc. Fast Software Encryption 97*, volume 1267 of *Springer-Verlag*, pages 190–204, 1997.

[4] W. Aiello and R. Venkatesan. Foiling birthday attacks in length-doubling transformations. In *Proc. EUROCRYPT 96*, Lecture Notes in Computer Science, 1996.

[5] J. An and M. Bellare. Constructing vil-macs from fil-macs: Message authentication under weakened assumptions. In M. Wiener, editor, *Proc. CRYPTO 99*, volume 1666 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[6] R. Anderson and E. Biham. Bear and Lion: Two practical and provably secure block ciphers. In *Proc. Fast Software Encryption 96*, pages 113–120, 1996.

[7] R. Anderson, E. Biham, and L. Knudsen. Serpent: A proposal for the Advanced Encryption Standard. NIST AES Proposal, June 1998.

[8] M. Atici and D. Stinson. Universal hashing and multiple authentication. In *Proc. CRYPTO 96*, Lecture Notes in Computer Science. Springer-Verlag, 1996.

[9] M. Bellare. Practice-oriented provable security. In *Proceedings of First International Workshop on Information Security (ISW 97)*, volume 1396 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[10] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *Advances in Cryptology—CRYPTO '96*. Springer-Verlag, 1996.

[11] M. Bellare, R. Canetti, and H. Krawczyk. Pseudorandom functions revisited: The cascade construction and its concrete security. In *37th Annual Symposium on Foundations of Computer Science* [71], pages 514–523.

[12] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatement of symmetric encryption: Analysis of the DES modes of operation. In *Proc. 38th IEEE Symp. on Foundations of Comp. Science*, 1997.

[13] M. Bellare, R. Guérin, and P. Rogaway. XOR MACs: New methods for message authentication using finite pseudorandom functions. In D. Coppersmith, editor, *Proc. CRYPTO 95*, volume 963 of *LNCS*, pages 15–28. Springer-Verlag, 1995.

[14] M. Bellare, J. Kilian, and P. Rogaway. The security of cipher block chaining. In Desmedt [44], pages 341–358.

[15] M. Bellare and P. Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Proc. CRYPTO 93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer-Verlag, 1993.

[16] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *First ACM Conference on Computer and Communications Security*, pages 62–73, Fairfax, 1993.

[17] M. Bellare and P. Rogaway. On the construction of Variable-Input-Length ciphers. In L. Knudsen, editor, *Proc. Fast Software Encryption 99*, volume 1636 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[18] D.J. Bernstein. How to stretch random functions: The security of protected counter sums. *Journal of Cryptology*, 12(3):185–192, Summer 1999.

[19] E. Biham and N. Keller. Cryptanalysis of reduced variants of Rijndael. Available from: `http://aes.nist.gov/`, 2000.

[20] E. Biham and A. Shamir. Cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991.

[21] E. Biham and A. Shamir. Differential cryptanalysis of FEAL and N-hash. In *Proc. EUROCRYPT 91*, Lecture Notes in Computer Science, pages 1–16. Springer-Verlag, 1991.

[22] E. Biham and A. Shamir. Differential cryptanalysis of Snefru, Khafre, REDOC-II, LOKI, and Lucifer. In *Proc. CRYPTO 91*, Lecture Notes in Computer Science, pages 156–171. Springer-Verlag, 1992.

[23] E. Biham and A. Shamir. *Differential Cryptanalysis of the Data Encryption Standard.* Springer Verlag, 1993. ISBN: 0-387-97930-1, 3-540-97930.

[24] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway. UMAC: fast and secure message authentication. In M. Wiener, editor, *Proc. CRYPTO 99*, volume 1666 of *Springer-Verlag*, pages 216–233, August 1999. Full version can be found at: `http://www.cs.ucdavis.edu/~rogaway/umac`.

[25] J. Black and P. Rogaway. CBC MAC's for arbitrary length messages: The three-key constructions. In M. Bellare, editor, *Proc. CRYPTO 2000*, volume 1880 of *Springer-Verlag*, pages 197–215, August 2000.

[26] M. Blaze, W. Diffie, R. L. Rivest, B. Schneier, T. Shimomura, E. Thompson, and M. Wiener. Minimal key lengths for symmetric ciphers to provide adequate commercial security, January 1996.

[27] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *Siam Journal on Computing*, 13(4):850–864, 1984.

[28] A. Bosselaers, R. Govaerts, and J. Vandewalle. Fast hashing on the Pentium. In *Proc. CRYPTO 96*, Lecture Notes in Computer Science. Springer-Verlag, 1996.

[29] G. Brassard. On computationally secure authentication tags requiring short secret shared keys. In *Proc. CRYPTO 82*, Lecture Notes in Computer Science, pages 79–86, 1982.

[30] L. Brown, J. Pieprzyk, and J. Seberry. LOKI—a cryptographic primitive for authentication and secrecy applications. In *Proc. AUSCRYPT 90*, Lecture Notes in Computer Science, pages 229–236. Springer-Verlag, 1990.

[31] R.H. Bruck. What is a loop? In A. A. Albert, editor, *Studies in Modern Algebra*, volume 2 of *Studies in Mathematics*, pages 59–99. Prentice-Hall, 1963.

[32] C. Burwick, D. Coppersmith, E. D'Avignon, R. Gennaroand S. Halevi, C. Jutla, S. M. Matyas Jr., L. O'Conner, M. Peyravian, D. Safford, and N. Zunic. Mars – a candidate cipher for the AES. NIST AES Proposal, June 1998.

[33] R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz, and A. Sahai. Exposure-resilient functions and all-or-nothing transforms. In Bart Preneel, editor, *Proc. EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 453–469. Springer-Verlag, May 2000.

[34] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. In *Proc. 30th ACM Symp. on Theory of Computing*, 1998.

[35] G. Carter, E. Dawson, and L. Nielsen. DESV: A Latin Square variation of DES. In *Proceeding of Workshop on Selected Areas of Cryptography*, 1995.

[36] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, April 1979.

[37] S. A. Cook. The complexity of theorem proving procedures. In *Proc. 3rd ACM Symp. on Theory of Computing*, pages 151–158, 1971.

[38] D. Coppersmith. The Data Encryption Standard (DES) and its strength against attacks. *IBM Journal of Research and Development*, 38(4):243–250, May 1994.

[39] D. Coppersmith. The development of DES. CRYPTO, August 2000. Invited Talk.

[40] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[41] J. Daemen and V. Rijmen. AES proposal Rijndael. NIST AES Proposal, June 1998.

[42] S. P. Dandamudi. *Introduction to Assembly Language Programming From 8086 to Pentium Proecessors.* Springer-Verlag New York, 1998.

[43] M. Dawson and S. Tavares. An expanded set of s-box design criteria based on information theory and its relation to differential-like attacks. In *Proc. EUROCRYPT 91*, Lecture Notes in Computer Science, pages 353–367. Springer-Verlag, 1991.

[44] Yvo G. Desmedt, editor. *Advances in Cryptology—CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*. Springer-Verlag, 21–25 August 1994.

[45] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22:644–654, 1976.

[46] H. Dobbertin, A. Bosselaers, and B. Preneel. RIPEMD-160, a strengthened version of RIPEMD. In D. Gollman, editor, *Proc. Fast Software Encryption 96*, volume 1039 of *Lecture Notes in Computer Science*, pages 71–82. Springer-Verlag, 1996.

[47] Y. Dodis. *Exposure-Resilient Cryptography.* PhD thesis, Massachusetts Institute of Technology, August 2000.

[48] A.J. Elbirt, W. Yip, B. Chetwynd, and C. Paar. An FPGA implentation and performance evaluation of the AES block cipher candidate algorithm finalists. In *Proceedings of the Third Advanced Encryption Standard Candidate Conference*, pages 13–27, 2000.

[49] M. Etzel, S. Patel, and Z. Ramzan. Square hash: Fast message authentication via optimized universal hash functions. In *Proc. CRYPTO 99*, Lecture Notes in Computer Science. Springer-Verlag, 1999.

[50] S. Even and Y. Mansour. A construction of a cipher from a single pseudorandom permutation. *Journal of Cryptology*, 10(3):151–162, Summer 1997.

[51] H. Feistel. Cryptography and computer privacy. *Scientific American*, 228(5):15–23, May 1973.

[52] H. Feistel. Block cipher cryptographic system. U.S. Patent # 3,798,359, March 1974.

[53] N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner, and D. Whiting. Improved cryptanalysis of Rijndael. In *Proc. Fast Software Encryption 2000*, Lecture Notes in Computer Science. Springer-Verlag, 2000.

[54] Electronic Frontier Foundation. *Cracking DES: Secrets of Encryption Research, Wiretap Politics and Chip Design.* O'reilly, 1998.

[55] K. Gaj and P. Chodoweic. Comparison of the hardware performance of the AES candidates using reconfigurable hardware. In *Proceedings of the Third Advanced Encryption Standard Candidate Conference*, pages 40–54, 2000.

[56] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.

[57] E. Gilbert, F.M. Williams, and N. Sloane. Codes which detect deception. *Bell System Technical Journal*, 53(3):405–424, 1974.

[58] H. Gilbert, M. Girault, P. Hoogvorst, F. Noilhan, T. Pornin, G. Poupard, J. Stern, and S. Vaudenay. Decorrelated Fast Cipher:- an AES candidate. NIST AES Proposal, June 1998.

[59] I. Goldberg and D. Wagner. Architectural considerations for cryptanalystic hardware. Appears as chapter 10 of [54], 1996.

[60] O. Goldreich. *The Foundations of Cryptography – Volume 1.* Cambridge University Press, 2000.

[61] O. Goldreich, S. Goldwasser, and S. Micali. On the cryptographic applications of random functions (extended abstract). In *Proc. CRYPTO 84*, pages 276–288. Springer-Verlag, 1984.

[62] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[63] O. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *Siam Journal of Computing*, 17(2):281–308, 1988.

[64] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.

[65] S. Halevi and H. Krawczyk. MMH: Message authentication in software in the gbit/second rates. In *Proceedings of the 4th Workshop on Fast Software Encryption*, Lecture Notes in Computer Science. Springer-Verlag, 1997.

[66] G. H. Hardy and E. M. Wright. *An introduction to the theory of numbers.* Oxford University Press, 5 edition, 1980.

[67] J. Hastad, R. Impagliazzo, L. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, August 1999.

[68] T. Helleseth and T. Johansson. Universal hash functions from exponential sums over finite fields. In *Proc. CRYPTO 96*, Lecture Notes in Computer Science. Springer-Verlag, 1996.

[69] M. Hellman and S. Langford. Differential-linear cryptanalysis. In *Proc. CRYPTO 94*, Lecture Notes in Computer Science, pages 17–25. Springer-Verlag, 1996.

[70] I. N. Herstein. *Topics in Algebra.* Blaisdell Publishing Company, 1964.

[71] IEEE. *37th Annual Symposium on Foundations of Computer Science*, 1996.

[72] T. Jakobsen and L. Knudsen. The interpolation attack on block ciphers. In *Proc. Fast Software Encryption 97*, volume 1267 of *Lecture Notes in Computer Science*, pages 28–40. Springer-Verlag, 1997.

[73] T. Johansson. Bucket hashing with small key size. In *Proc. EUROCRYPT 97*, Lecture Notes in Computer Science. Springer-Verlag, 1997.

[74] R. Karp. Reducibility among combinatorial problems. in Complexity of Computer Computations, 1972.

[75] A. Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX, 1883. pp. 5–38, Jan. 1883, pp. 161–181, Feb. 1883.

[76] L. Knudsen. *Block Ciphers - Analysis, Design, and Applications.* PhD thesis, Aarhus University, 1994.

[77] L. Knudsen. Truncated and higher order differentials. In *Proceedings of the 2nd Workshop on Fast Software Encryption*, Lecture Notes in Computer Science. Springer-Verlag, 1995.

[78] P. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Proc. CRYPTO 96*, Lecture Notes in Computer Science, pages 104–113. Springer-Verlag, 1996.

[79] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Proc. CRYPTO 99*, Lecture Notes in Computer Science, pages 388–397. Springer-Verlag, 1999.

[80] H. Krawczyk. LFSR-based hashing and authentication. In *Proc. CRYPTO 94*, Lecture Notes in Computer Science. Springer-Verlag, 1994.

[81] H. Krawczyk. New hash functions for message authentication. In *Proc. EUROCRYPT 95*, Lecture Notes in Computer Science. Springer-Verlag, 1995.

[82] X. Lai. *On the Design and Security of Block Ciphers.* PhD thesis, ETH Zurich, 1992.

[83] X. Lai, J. Massey, and S. Murphy. Markov ciphers and differential cryptanalysis, 1991.

[84] X. Lai and J.L. Massey. A proposal for a new block encryption standard. In I.B. Damgård, editor, *Proc. EUROCRYPT 90*, volume 473 of *Lecture Notes in Computer Science*, pages 389–404. Springer-Verlag, 1990.

[85] X. Lai and J.L. Massey. Device for conversion of a digital block and use of the same. U.S. Patent # 5,214,703, 1993.

[86] S. Langford. *Differential-Linear cryptanalysis and threshold signatures.* PhD thesis, Stanford University, June 1995.

[87] M. Luby. *Pseudorandomness and Cryptographic Applications.* Princeton University Press, 1996.

[88] M. Luby and C. Rackoff. How to construct pseudorandom permutations and pseudorandom functions. *SIAM J. Computing*, 17(2):373–386, April 1988.

[89] S. Lucks. Faster Luby-Rackoff ciphers. In *Proc. Fast Software Encryption 96*, Lecture Notes in Computer Science. Springer-Verlag, 1996.

[90] S. Lucks. Attacking seven rounds of Rijndael under 192-bit and 256-bit keys. In *Proc. Third Advanced Encryption Standard Conference, 2000*, 2000.

[91] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes.* North Holland Publishing Company, New York, 1977.

[92] M. Matsui. Linear cryptanalysis method for DES cipher. In *Proc. EUROCRYPT 93*, Lecture Notes in Computer Science, pages 386–397. Springer-Verlag, 1993.

[93] M. Matsui. New block encryption algorithm MISTY. In E. Biham, editor, *Proc. Fast Software Encryption 97*, volume 1267 of *Lecture Notes in Computer Science*, pages 54–68. Springer-Verlag, 1997.

[94] U. Maurer. A simplified and generalized treatment of Luby-Rackoff pseudorandom permutation generators. In *Proc. EUROCRYPT 92*, Lecture Notes in Computer Science. Springer-Verlag, 1992.

[95] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[96] R. Merkle and M. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*, IT-24:525–530, 1978.

[97] R. C. Merkle. Fast software encryption functions. In A.J. Menezes, editor, *Proc. CRYPTO 90*, volume 537 of *Lecture Notes in Computer Science*, pages 476–501. Springer-Verlag, 1990.

[98] S. Murphy. Cryptanalysis of FEAL using 20 chosen plaintexts. *Journal of Cryptology*, 2(3):145–154, 1990.

[99] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *Proc. 38th IEEE Symp. on Foundations of Comp. Science*, 1997.

[100] M. Naor and O. Reingold. From unpredictability to indistinguishability: A simple construction of pseudo-random functions from MACs. In *Proc. CRYPTO 98*, Lecture Notes in Computer Science. Springer-Verlag, 1998.

[101] M. Naor and O. Reingold. On the construction of pseudo-random permutations: Luby-Rackoff revisited. *J. of Cryptology*, 12:29–66, 1999. Preliminary version in: *Proc. STOC 97*.

[102] National Institute of Standards and Technology (NIST). FIPS publication 180-1: Secure hash standard (SHS), April 1995.

[103] National Institute of Standards and Technology (NIST). Descriptions of SHA-256 SHA-384 and SHA-512, 2001. To appear.

[104] W. Nevelsteen and B. Preneel. Software performance of universal hash functions. In *Proc. EUROCRYPT 99*, Lecture Notes in Computer Science. Springer-Verlag, 1999.

[105] K. Nyberg. Differentially uniform mappings for cryptography. In *Proc. EUROCRYPT 93*, volume 765 of *Lecture Notes in Computer Science*, pages 55–65. Springer-Verlag, 1994.

[106] K. Nyberg and L. Knudsen. Provable security against a differential attack. *Journal of Cryptology*, 8(1):27–37, 1995.

[107] U.S. Department of Commerce / National Institute of Standards. FIPS publication 180: Secure Hash Algorithm, 1995. Federal Information Processing Standards Publication 180.

[108] National Bureau of Standards. FIPS publication 46: Data encryption standard, 1977. Federal Information Processing Standards Publication 46.

[109] National Bureau of Standards. FIPS publication 81: DES modes of operation, 1980. U.S. Department of Commerce.

[110] National Institute of Standards and Technology. Advanced encryption standard home page. `http://www.nist.gov/aes`.

[111] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley Publishing Company, 1994.

[112] J. Patarin. New results on pseudorandom permutation generators based on the DES scheme. In *Proc. CRYPTO 91*, Lecture Notes in Computer Science. Springer-Verlag, 1991.

[113] J. Patarin. Improved security bounds for pseudorandom permutations. In *4th ACM Conference on Computer and Communications Security*, pages 140–150, 1997.

[114] S. Patel, Z. Ramzan, and G. Sundaram. Towards making Luby-Rackoff ciphers optimal and practical. In *Proc. Fast Software Encryption 99*, Lecture Notes in Computer Science. Springer-Verlag, 1999.

[115] S. Patel, Z. Ramzan, and G. Sundaram. Luby-rackoff ciphers over arbitrary algebraic structures or why xor is not so exclusive. In submission, 2000.

[116] S. Patel, Z. Ramzan, and G. S. Sundaram. Sha-Zam: a block cipher fast as DES, secure as SHA. Appears as a contribution to the Ad-Hoc Authentication Group (AHAG), Mobile Telecommunications industry, 1998. Modified version appears as a contribution to the Third Generation Partnership Project (3GPP), Mobile Telecommunications industry, 1999.

[117] S. Patel, Z. Ramzan, and G. S. Sundaram. Sha-Zam: short ciruiting cryptanalysis. Appears as a contribution to the Ad-Hoc Authentication Group (AHAG), Mobile Telecommunications industry, 1999.

[118] J. Pieprzyk. How to construct pseudorandom permutations from single pseudorandom functions. In *Proc. EUROCRYPT 90*, Lecture Notes in Computer Science. Springer-Verlag, 1990.

[119] B. Preneel and P. van Oorschot. MDx-MAC and building fast MACs from hash functions. In *Proc. CRYPTO 95*, volume 963 of *Lecture Notes in Computer Science*, pages 1–14. Springer-Verlag, 1995.

[120] B. Preneel and P.C. van Oorschot. On the security of two MAC algorithms. In *Proc. EUROCRYPT 96*, Lecture Notes in Computer Science. Springer-Verlag, 1996.

[121] Z. Ramzan and L. Reyzin. On the round security of symmetric-key cryptographic primitives. In Mihir Bellare, editor, *Proc. CRYPTO 2000*, number 1880 in Lecture Notes in Computer Science, pages 376–393. International Association for Cryptologic Research, Springer-Verlag, August 2000.

[122] *Rhapsody Assembler Reference.* Available via `http://developer.apple.com/`.

[123] R. Rivest. The MD5 message digest algorithm. IETF RFC-1321, 1992.

[124] R. Rivest, M. Robshaw, R. Sidney, and Y.L. Yin. The RC6 block cipher. NIST AES Proposal, June 1998.

[125] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[126] M.J.B. Robshaw. Block ciphers. Technical Report TR-601, RSA Laboratories, August 1995.

[127] M.J.B. Robshaw. Stream ciphers. Technical Report TR-701, RSA Laboratories, July 1995.

[128] P. Rogaway. Bucket hashing and its application to fast message authentication. In *Proc. CRYPTO 95*, Lecture Notes in Computer Science. Springer-Verlag, 1995.

[129] R. A. Rueppel. On the security of Schnorr's pseudo random generator. In *Proc. EUROCRYPT 89*, Lecture Notes in Computer Science. Springer-Verlag, 1989.

[130] B. Schneier. A self-study course in block cipher cryptanalysis. Available from: `http://www.counterpane.com/self-study.html`, 1998.

[131] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. Twofish: a 128-bit block cipher. NIST AES Proposal, June 1998.

[132] C. P. Schnorr. On the construction of random number generators and random function generators. In *Proc. EUROCRYPT 88*, Lecture Notes in Computer Science. Springer-Verlag, 1988.

[133] C.E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28:657–715, 1949.

[134] A. Shimzu and S. Miyaguchi. Fast data encipherment algorithm FEAL. In D. Chaum and W.L. Price, editors, *Proc. EUROCRYPT 87*, volume 304 of *Lecture Notes in Computer Science*, pages 267–280. Springer-Verlag, 1987.

[135] V. Shoup. On fast and provably secure message authentication based on un iversal hashing. In *Proc. CRYPTO 96*, Lecture Notes in Computer Science. Springer-Verlag, 1996.

[136] A. Sorkin. Lucifer, a cryptographic algorithm. *Cryptologia*, 8(1):22–41, 1984.

[137] D. R. Stinson. Universal hashing and authentication codes. *Design, Codes and Cryptography*, 4:369–380, 1994. Preliminary version appeared at CRYPTO91.

[138] D. R. Stinson. Comments on definitions of universal hash families, August 2000. Available from: `http://cacr.math.uwaterloo.ca/~dstinson/`.

[139] G. Tsudik. Message authentication with one-way hash functions. In *Proceedings of Infocom '92*. IEEE Press, 1992.

[140] S. Vaudenay. Provable security for block ciphers by decorrelation. In *Proceedings of STACS '98*, number 1373 in Lecture Notes in Computer Science, pages 249–275. Springer-Verlag, 1998.

[141] N. Weaver and J. Wawrzynek. A comparison of the AES candidates amenability to FPGA implementation. In *Proceedings of the Third Advanced Encryption Standard Candidate Conference*, pages 28–39, 2000.

[142] M. N. Wegman and J. L. Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265–279, June 1981.

[143] M. Weiner. Efficient DES key search. In *Proc. CRYPTO 93*, Lecture Notes in Computer Science. Springer-Verlag, 1993.

[144] A. C. Yao. Theory and applications of trapdoor functions (extended abstract). In *Proc. 23rd IEEE Symp. on Foundations of Comp. Science*, pages 80–91, 1982.

[145] Y. Zheng, T. Matsumoto, and H. Imai. Impossibility and optimality results on constructing pseudorandom permutations. In *Proc. EUROCRYPT 89*, Lecture Notes in Computer Science. Springer-Verlag, 1989.