Elastic Block Ciphers

Debra Lee Cook

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2006

ABSTRACT

Elastic Block Ciphers

Debra Lee Cook

Standard block ciphers are designed around one or a small number of block sizes. From both a practical and a theoretical perspective, the question of how to efficiently support a range of block sizes is of interest. In applications, the length of the data to be encrypted is often not a multiple of the supported block size. This results in the use of plaintext-padding schemes that impose computational and space overheads. Furthermore, a variable-length block cipher ideally provides a variable-length pseudorandom permutation and strong pseudorandom permutation, which are theoretical counterparts of practical block ciphers and correspond to ideal properties for a block cipher.

The focus of my research is the design and analysis of a method for creating variable-length block ciphers from existing fixed-length block ciphers. As the heart of the method, I introduce the concept of an *elastic block cipher,* which refers to stretching the supported block size of a block cipher to any length up to twice the original block size while incurring a computational workload that is proportional to the block size. I create a structure, referred to as the *elastic network*, that uses the round function from any existing block cipher in a manner that allows the properties of the round function to be maintained and results in the security of the elastic version of a block cipher being directly related to that of the original version. By forming a reduction between the elastic and original versions, I prove that the elastic version of a cipher is secure against round-key recovery attacks if the original cipher is secure against such attacks. I illustrate the method by creating elastic versions of four existing block ciphers. In addition, the elastic network provides a new primitive structure for use in symmetric-key cipher design. It allows for the creation of variable-length pseudorandom permutations and strong pseudorandom permutations in the range of b to $2b$ bits from round functions that are independently chosen pseudorandom permutations on b bits.

Contents

List of Figures

List of Tables

I am grateful for the guidance my advisors, Moti Yung and Angelos Keromytis, provided over the past four years. I want to thank Steve Bellovin for his comments on my research. In addition to Moti, Angelos and Steve, I want to thank the remainder of my thesis committee, Charanjit Jutla and Sal Stolfo, for their time.

I wish to acknowledge my former AT&T co-workers for their support and friendship over the past several years, especially Paul Zahray for supporting my changing schedule at AT&T and for his advice, Linda Kohm and Paul for providing chocolate, and Bob Marchiano for helping to convince me that I should either pursue a Ph.D. in computer science or become a veterinarian. I chose the first option. Due to this potentially being my only chance to mention their names in anything I write, I wish to thank my cats, Kitty and Pi, for their devoted attention and for frequently interrupting me as I was typing this thesis.

Chapter 1

Introduction

1.1 Overview

The focus of my research is the design and analysis of a method for creating variable-length block ciphers, which I refer to as *elastic block ciphers*. Standard block ciphers are designed around one or a small number of block sizes, with most supporting 128-bit blocks. From both a practical and a theoretical perspective, the question of how to efficiently support a range of block sizes is of interest. In applications, the length of the data to be encrypted is often not a multiple of the supported block size. This results in the use of plaintext-padding schemes [Lab93] that impose computational and space overheads by appending bits to the data until it is an integral number of blocks. When the data being encrypted is relatively small, the padding becomes a significant portion of the encrypted data. For example, encrypting a database at the field or row level to allow for efficient querying can easily result in a substantial amount of padding. When encrypting network traffic, padding can easily cause the packet to exceed the maximum transmission unit, resulting in packet fragmentation. When the plaintext is a length between one and two blocks, an elastic block cipher allows all of the bits to be encrypted as a single block, avoiding the need to use a mode of encryption and creating a stronger binding across the ciphertext bits compared to the ciphertext produced by a mode of encryption, such as CBC. In addition, a variable-

length block cipher ideally provides a variable-length pseudorandom permutation (PRP) and strong PRP (SPRP), which are theoretical counterparts to practical block ciphers.

Previous proposals for converting existing block ciphers into variable-length ones focused on treating a block cipher as a black box and combining it with other operations [BR99, PRS04]. While such an approach allows for a formal proof of security of the variable-length block cipher under certain assumptions about the original block cipher, the resulting constructions require multiple applications of the original block cipher, making them computationally inefficient compared to padding. These methods may be valuable in providing modes of encryption that preserve the length of the data but they do not address how to design block ciphers to support variable-length blocks. There have also been ad-hoc attempts to design a variable-length block cipher from scratch [Ree92, Sch98].

My work provides a solution to the problems of how to create variable-length block ciphers for practical use and how to create variable-length PRPs and SPRPs in theory. I accomplish this by taking a new approach to designing variable-length block ciphers that avoids the shortcomings of designing a cipher from scratch while offering an improvement over the proposals that treat a cipher as a black box. The approach creates a variable-length block cipher from any existing block cipher in a manner that allows the computational workload for encrypting data to be proportional to the block size. Furthermore, the security of the variable-length version against key recovery attacks is directly related to the security of original cipher against such attacks. My research is also of value in that it produced a cryptographic primitive, specifically a structure that I call the *elastic network*. The creation of new primitives is valuable as existing algorithms may succumb to attacks either as result of new analysis or increased computing resources, and in offering mechanisms that allow for improved utilization of resources and that provide capabilities for new applications. The elastic network may be of future use in the design of hash functions and modes of encryption, in addition to its use in block ciphers. In relation to cryptographic building blocks, I discuss the concept of a generic key schedule for block ciphers, which can provide implementation benefits by eliminating the need to support a key schedule for each block cipher.

1.2 Contributions

My main contributions are:

- Introducing the concept of an *elastic block cipher,* which refers to stretching the supported block size of a block cipher to any length up to twice the original block size while incurring a computational workload that is proportional to the block size. I call the resulting variable-length cipher the elastic version of the original cipher. My method for constructing elastic block ciphers is novel in how it builds upon existing block ciphers. I created a structure, the *elastic network*, that uses the round function from any existing block cipher and allows bits beyond the supported block size to be combined with bits in the supported block size that form the input to the round function. This allows the properties of the round function to be maintained and results in the security of the elastic version of a block cipher being directly related to that of the original version. The approach used to create elastic block ciphers falls between that of a black box approach and an ad-hoc design approach. The round function is treated as a black box with operations added between rounds.

- Proving that variable-length PRPs and SPRPs can be created from fixed-length PRPs by using the elastic network. Specifically, I prove that three-round elastic network is a variable-length PRP, the inverse of a four-round elastic network is a variable-length PRP and a five-round elastic network is a variable-length SPRP when the round functions are independently chosen PRPs. I also show that these are the minimum number of rounds required. This allows for the creation of variable-length (in the range of b to $2b$ bits) PRPs and SPRPs from b-bit PRPs. By combining the elastic network with an existing construction for $2b, 3b, ...mb$-bit SPRPs, where m is an integer, I can create $2b$ to $2mb$-bit PRPs and SPRPs from b-bit PRPs. By combining the elastic network with a Feistel network, b to $2b$-bit PRPs and SPRPs can be created from $\frac{b}{2}$-bit pseudorandom funtions (PRF).

- Defining the security of the elastic version of a block cipher in terms of the original block cipher. By forming a reduction between the elastic and original versions of a cipher, any attack on the elastic version that recovers round-key bits can be converted

into a polynomial time and memory related attack that recovers the round-key bits for the original cipher. All practical attacks attempt to recover round-key bits (*e.g.,* linear [Mat93], differential [BS93], higher order differential [Knu95] and related key [Bih93] attacks). The significance of this result is that it proves the elastic version of a cipher is secure against known practical attacks if the original cipher has been proven secure against such attacks. This eliminates the need to analyze the elastic version against every attack, instead allowing the results of such analysis on the original cipher to be reused. My approach is the first such reduction for a variable-length block cipher that shows how to directly use a practical attack on the variable-length version of a cipher to attack the original cipher.

- Showing how to convert a linear attack on the elastic version of a block cipher to a linear attack on the original fixed-length version of the block cipher in polynomial time and memory. I also extend the result to show that any attack consisting of algebraic equations (as opposed to only linear equations) on an elastic version of a block cipher can be converted in polynomial time and memory to a set of equations with which to attack the original version of the block cipher.

- Illustrating the method for constructing elastic block ciphers by applying it to four existing block ciphers that were finalists in standards competitions: AES [NIS01b], Camellia [AIK+00], MISTY1 [Mat00b] and RC6 [RRSY98]. I implemented elastic versions of these ciphers in software. In order to quantify the potential space savings achieved with an elastic block cipher, I calculated the amount of padding eliminated when encrypting a database corresponding to customer information for an online bookstore using an elastic block cipher.

- Showing how the probability of a differential attack on an elastic version of a block cipher can be bounded using the probabilities for differential characteristics from the original cipher's round function. I apply the method to elastic versions of AES and MISTY1 to bound the probability a differential characteristic occurs in the elastic versions.

- Proposing a generic key schedule for block ciphers. I discuss the benefits of having

a key schedule that is independent of any block cipher and that outputs pseudorandom bits. I demonstrate that this is feasible using an existing algorithm as the key schedule, thus showing that the concept of a generic key schedule is practical and possible without devising a new algorithm. Having a single key schedule allows for a common implementation to be used with multiple block ciphers. The generation of pseudorandom expanded-key bits eliminates certain types of attacks on ciphers and assists in preventing attacks that attempt to recover round-key bits. This is due to the elimination of well-defined relationships (lack of randomness) between expanded-key bits found in most existing block ciphers' key schedules.

- Illustrating how the elastic network can be used to create new modes of encryption. I propose two new modes as examples and provide a preliminary analysis of their benefits and weaknesses.

1.3 Organization

The remainder of my dissertation is organized as follows: In Chapter 2, I provide an overview of block cipher design criteria and describe previous approaches to designing variable-length block ciphers. The first part of my research covers the design and general analysis of the elastic network and the elastic block cipher algorithm, and theoretical aspects of my work. These results are presented in Chapters 3 to 5. The second part of my research consists of instantiations and applications of elastic block ciphers and the elastic network. These components of my work are described in Chapters 6 to 10.

In Chapter 3, I define the method for constructing elastic block ciphers. I first describe the elastic network and explain why I could not use an existing structure, specifically an unbalanced Feistel network. Then I present the algorithm for converting fixed-length block ciphers to elastic block ciphers. In Chapter 4, I show that the elastic network can be used to create variable-length PRPs and SPRPs under a minimum number of rounds when the round functions are independently chosen PRPs. I describe how the elastic network can be combined with an existing construction for $2b, 3b, ...mb$-bit SPRPs to create variable-length PRPs and SPRPs on $2b$ to $2mb$ bits from b-bit PRPs, and how the elastic network can be

combined with a Feistel network to create variable-length PRPs and SPRPs on b to $2b$ bits from $\frac{b}{2}$-bit PRFs. In Chapter 5, I analyze the security of elastic block ciphers in general. I show that any attack that recovers the round-key bits of the elastic version of a cipher implies such an attack exists on the original, fixed-length version of the cipher by creating a reduction from the elastic version to the original version. Therefore, if the original version is immune to such attacks, the elastic version is also immune to such attacks. In order to provide a specific example of cryptanalysis of elastic block ciphers that is independent of the reduction method, I consider linear cryptanalysis. I show how any set of linear equations for an elastic block cipher can be converted in polynomial and memory time to a set of equations for the original cipher. I extend the result to any attack involving algebraic equations.

In Chapter 6, I describe elastic versions of AES, Camellia, MISTY1 and RC6 that I created to demonstrate the feasibility of elastic block ciphers. For each cipher, I compare the performance of the elastic version to the original version with padding and I apply statistical tests to measure the randomness of the output. In Chapter 7, I discuss differential cryptanalysis of elastic block ciphers. I define states, in terms of the differential input to a round, that an elastic block cipher can achieve. By using the differential bounds for the round function from the original version of a block cipher and the possible sequence of states, I bound the probability that a differential characteristic can occur for the elastic block cipher. This method is illustrated with the elastic versions of AES and MISTY1. In Chapter 8, I discuss the options for key schedules of elastic block ciphers and introduce the concept of a generic key schedule. I define requirements for a generic key schedule in order to provide an improvement over existing key schedules. I show that an existing algorithm, the RC4 stream cipher [Riv96], satisfies these requirements.

In the next two chapters I discuss two uses of elastic block ciphers. First, in Chapter 9, I describe how the use of an elastic block cipher can be beneficial to database encryption by reducing the size of the ciphertext. I use a database representing customer records of an online bookstore as an example. Second, in Chapter 10, I illustrate how support for variable block sizes allows for new modes of encryption by proposing two new modes of encryption.

In Chapter 11, I provide a summary of my results and describe some open problems

related to elastic block ciphers and the elastic network. The appendices contain descriptions of the block ciphers AES, Camellia, MISTY1 and RC6, the results for the performance and statistical analysis of their elastic versions, and a description of RC4.

Chapter 2

Background

2.1 Overview

This chapter provides background information on the general design criteria for block ciphers and on previous proposals for variable-length block ciphers. I describe the typical structures used in block ciphers and I review the criteria used for selecting standard block ciphers in the AES and NESSIE competitions. I describe a previous proposal by Bellare and Rogaway for creating variable-length block ciphers that uses any existing block cipher as a black box, and a modification by Patel, *et al.*, to their proposal. I also briefly discuss modes of encryption, which are covered in more detail in Chapter 10, and review ciphertext stealing, which is a modification to modes of encryption that eliminates padding.

2.2 Block Cipher Definition and Structures

A block cipher is a type of symmetric-key cipher. A block cipher operating on b-bit inputs is a family of permutations on b bits with the key given to the block cipher used to select the permutation. Using the following notation:

- Let k be a q-bit key.
- Let P be a b-bit string denoting a plaintext.
- Let C be a b-bit string denoting a ciphertext.

Definition 1. *A block cipher, G, is defined as:*

- *An encryption function:* $E = \{E_k\}$ *is a family of* 2^q *permutations on b bits indexed by k.*
- *A decryption function:* $D = \{D_k\}$ *is a family of* 2^q *permutations on b bits indexed by k such that* D_k *is the inverse of* E_k*.*

Given a plaintext, P, and key, k, if $C = E_k(P)$ *then* $P = D_k(C)$*.*

In practice, a block cipher will take as input a secret key, k, and apply a function, F, called a key schedule, to k that expands k into an expanded key, ek, *i.e.*, $ek = F(k)$. In practice, k is usually 128, 192 or 256 bits and ek is often more than 100 bytes. As discussed in Chapter 8, the key schedules of existing block ciphers used in practice are designed to be computationally efficient at the cost of a lack of randomness in the expanded-key bits.

Practical block ciphers can be categorized as substitution - permutation networks (SPN). A SPN is a general term encompassing any algorithm constructed entirely from a series of substitutions and permutations performed on the data. The terms diffusion and confusion are commonly used to indicate the basic properties required of a block cipher. Diffusion refers to bits influencing each other. Ideal diffusion is achieved when every bit impacts every other bit in a manner such that changing one bit of plaintext will change each bit of ciphertext with 50% probability. Confusion refers to the property that it should not be possible to determine anything about the key (and thus the plaintext) when given the ciphertext; in other words, any relationship between the key and ciphertext should be indiscernable. The concepts of SPNs, diffusion and confusion in block ciphers were initiated by Shannon in 1949 [Sha49]. While there is no standard formula for creating block ciphers, the practice has been to use a series of rounds, with a round consisting of one or more substitutions and/or permutations applied to the data. Typically, a subset of the expanded-key bits are used in each round. Using rounds eliminates the need to devise one function that provides sufficient diffusion and confusion on a single application, instead allowing less complex and less computationally intense functions to be applied multiple times. This reduces both the difficulty of analyzing the cipher and implementing the cipher. In the competitions mentioned in Section 2.3, most finalists have taken one of two forms: either a

series of rounds that process all bits per round, as done in AES and Serpent [ABK98], or a Feistel network (shown in Figure 2.1) that consists of a series of rounds, but with only some of the bits, typically half, processed in each round. One reason a Feistel network is useful is because its inverse is the network run in reverse, thus the round function is the same for both encryption and decryption and does not need to be invertible. Block ciphers using Feistel networks include DES [NIS99a], MISTY1, and Camellia. Other examples include MARS [Cop99], that uses a three-level Feistel network, and Twofish [SKW+98], that is not a true Feistel network because it applies a one-bit rotation to part of the data between rounds.

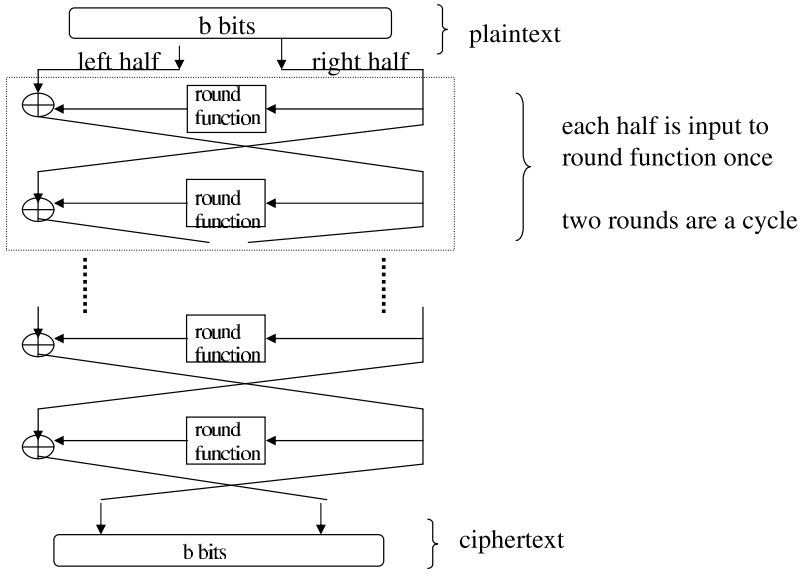

Figure 2.1: Balanced Feistel Network

Block ciphers are designed to minimize the impact (in terms of resources) of encryption when processing data. Thus, simple operations such as logical operations, shifts, rotations and table lookups are common; whereas, expensive arithmetic operations are avoided unless they can be implemented as a table lookup. Substitution boxes (S-boxes), where specific bit sequences are replaced via a table lookup, are designed to minimize differential and linear

relationships. Another common step is whitening, the process of XORing the entire data block with expanded-key bits. Whitening is typically performed at the beginning or end of each round. It contributes to confusion by limiting an attacker's ability to know the exact bit values entering or exiting a round. Whitening is not beneficial in preventing differential cryptanalysis because it will cancel out when computing the XOR of two data blocks.

2.3 Design and Selection of Standard Block Ciphers

The design criteria of block ciphers typically take into consideration factors related to permitting widespread implementation in addition to security; however, the basic question of what block size applications require is never mentioned. Instead, block sizes are dictated to the applications. While a minimum block size, currently 128 bits, is needed for security to prevent exhaustive searches over all plaintexts when given ciphertexts, the lack of support for variable-length blocks above this minimum is more likely due to a general lack of proposals than security and performance issues. It should also be noted that as the block size increases, the probability of a collision (identical ciphertext blocks) decreases when encrypting streaming data with a mode of encryption, such as CBC mode. The two most well-known recent competitions for block ciphers with the goal of producing standards were the US National Institute of Standards and Technology (NIST) Advanced Encryption Standard (AES) and the European Union's New European Schemes for Signatures, Integrity and Encryption (NESSIE) competitions. NIST selected Rijndael in 2001 [NIS01b], henceforth referred to as AES. NESSIE recommended multiple ciphers in February, 2003. For block ciphers, MISTY1 (64-bit blocks), Camellia (128-bit blocks), AES (128-bit blocks) and SHACAL-2 (256-bit blocks) were selected. A description of each is available in NESSIE's report [NES03]. The Information-Technology Security Center (ISEC) in Japan held Cryptec from 2000 to 2003 [Inf03]. Cryptec was a public call for various cryptographic functions, including block and stream ciphers, with the intent of recommending algorithms for use by Japan's government. The 128-bit block ciphers deemed "practically" secure from Cryptec's 2001 call for proposals and presented in Cryptec's 2003 results are AES, Camellia, CIPHERUNICORN-A [TKMN00], Hierocrypt-3 [OSK+01] and SC2000 [SYY+01]. Compared to AES and NESSIE,

Cryptec's call for proposals did not state formal goals and evaluation criteria for the submissions (only saying the ciphers would be evaluated against a list of known attacks), and posted only high-level results. Therefore, I focus on NIST's and NESSIE's criteria in my discussion on traditional block cipher design. Besides security, the AES and NESSIE selection criteria for block ciphers focused on performance and applicability for general use. Neither required variable block sizes. NESSIE's call for submission placed simplicity above flexibility in key and block size, stating "Simplicity and clarity of design are important considerations. Variable parameter sizes are less important." [NES03]

The AES selection criteria consisted of three general areas [NIS99b]. Most important was security based on resistance to cryptanalysis, the soundness of the mathematical basis and the randomness of the ciphertext. Second, the amount of system resources required and monetary costs were considered. The system resources required for both software and hardware implementations were taken into account. The third area referred to as algorithm and implementation characteristics covered the ability to utilize the block cipher for other cryptographic purposes including using it as a hash function, a random bit generator and a stream cipher (*i.e.*, such as via CTR mode [NIS01c]). General algorithm characteristics included in this area were encryption and decryption using the same algorithm, ability to implement the algorithm in both software and hardware, and simplicity of implementations. Simplicity is important in reducing implementation errors and impacts costs, such as power consumption, number of hardware gates and execution time. Some submissions included the ability to handle keys and block sizes other than the 128-bit requirement. However, with one exception, the variations from 128-bit keys and blocks only manifested themselves in submissions allowing keys greater than 128 bits and supporting 256-bit blocks.

The NESSIE selection criteria were divided into four areas: security, market requirements, performance and flexibility [NES00]. Security was loosely defined as resistance to cryptanalysis. Market requirements covered the feasibility of implementing the algorithm both from a technical perspective (cost-efficient implementations) and business perspective (free of licensing restrictions). Performance and flexibility covered the range of environments in which the algorithm could efficiently be implemented. Software considerations included 8-bit processors (as found in inexpensive smart cards), 32-bit and 64-bit proces-

sors. For hardware, both field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs) were considered. Finally, the flexibility of the algorithm for use in multiple applications and to satisfy multiple purposes was considered. NESSIE had three categories of block ciphers. "High security" required keys of at least 256 bits and a block length of 128 bits. "Normal security" required keys of at least 128 bits and a block length of 128 bits. "Normal legacy" required keys of at least 128 bits and a block length of 64 bits. In all categories there was a requirement that the minimal attack workload was at least on the order of 2^{80} triple DES encryptions.

2.4 Variable-Length Block Ciphers

Block ciphers are, ideally, pseudorandom permutations (PRPs), which are a subset of pseudorandom functions (PRFs). There has been little previous work on variable-length PRFs. The focus has been on variable-length inputs with fixed-length outputs as applicable to MACs and hash functions [AB99, BCK96, Ber99, BR00] and on multiples of the original block length [HR03a, HR03b, LR88, NR99] (although the same goal is accomplished by modes of encryption, for which there are numerous examples used in practice, *e.g.,* CBC, OFB, CFB, CTR ...). There has also been work on using PRPs to create PRFs [HWKS98]. While there have been proposals for variable-length block ciphers in the past, such as the Hasty Pudding Cipher (HPC) [Sch98] and CMEA [WSK97], my intent is not to design an *ad-hoc* new cipher, but to systematically build upon existing block ciphers. While designing a cipher from scratch allows the designer to incorporate new features, such as support for a range of block sizes, it also requires analyzing the cipher against all known attacks. HPC was deemed to be insecure in the first round of the AES competition [NIS00]. CMEA was used for encryption of cellular signaling messages. Aside from flaws in CMEA's design, its application to short blocks of data and implementations with no chaining mode contributed to the insecurity of products using it [WSK97]. Ciphertext stealing, described in Section 2.5, is another way to avoid padding, but it does not provide computational savings.

The first proposal (and one of only two proposals using existing block ciphers without modification) for a variable-length block cipher that converts any block cipher into one that

accepts variable block lengths is by Bellare and Rogaway [BR99]. Their method is shown in

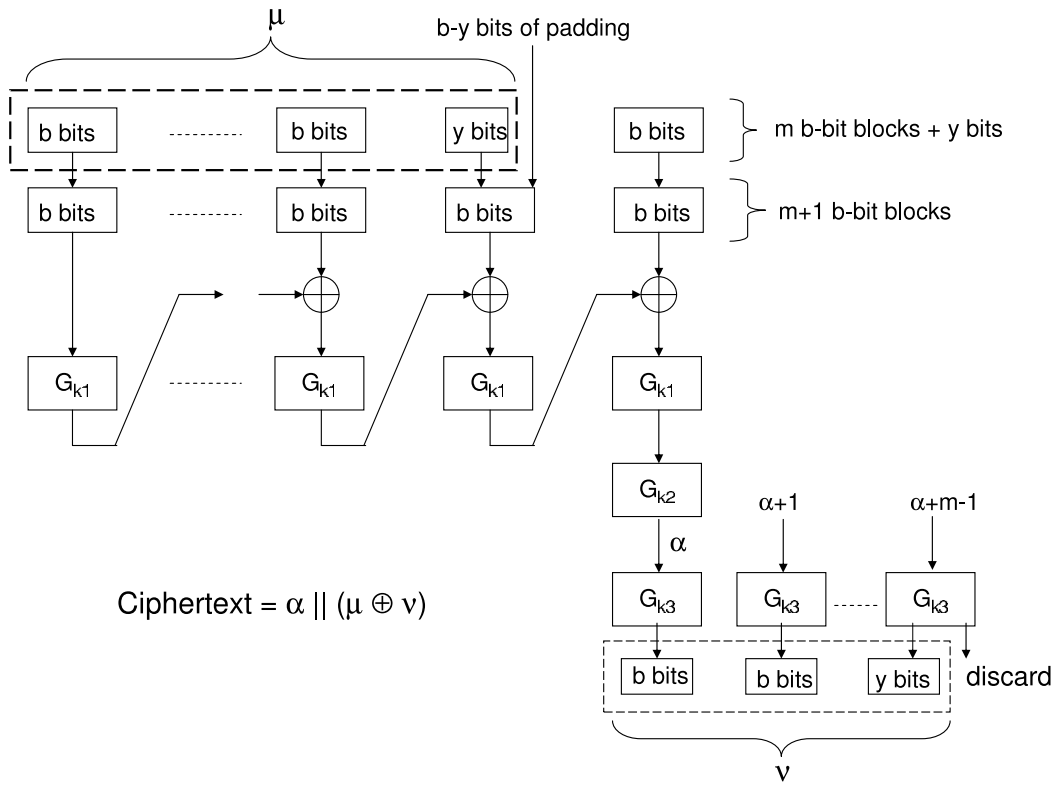

Figure 2.2: Bellare and Rogaway's Variable-Length Block Cipher

Figure 2.2. Unlike my construction, Bellare and Rogaway do not modify the original block cipher, but instead add operations around it. They treat the original cipher as a black box and analyze the construction independently of the specific block cipher. The security of their variable-length block cipher is defined in terms of the original cipher. Given a $(mb + y)$-bit segment of plaintext and a b-bit block cipher, for $0 < y < b$ and $m > 1$, $b - y$ bits of padding are added before the last (rightmost) b-bit block to produce $m + 1$ b-bit blocks. The data is then encrypted in CBC mode. The last block output from the CBC mode is encrypted again and the resulting b-bit output, α, is used as input to the block cipher in CTR mode. CTR mode is run until $(m - 1)b + y$ bits are output. These bits are XORed with the leftmost $(m - 1)b + y$ bits of input, to produce $u \oplus v$. The ciphertext is the concatentation of α and and $u \oplus v$. Encrypting one plus a fractional block, $b + y$ bits, involves four applications of the block cipher (two applications in CBC mode, one more

application to obtain α and one application in CTR mode) plus additional operations, thus requiring more than four times the work of the original block cipher to encrypt one plus a fractional block regardless of the number of extra bits actually encrypted (*e.g.,* even if the data is one bit longer than the original block length). Therefore, while their approach preserves the length of the data, it requires at least twice the work of padding to encrypt one plus a fractional block. The other proposal by Patel, *et al.*, shown in Figure 2.3 is a

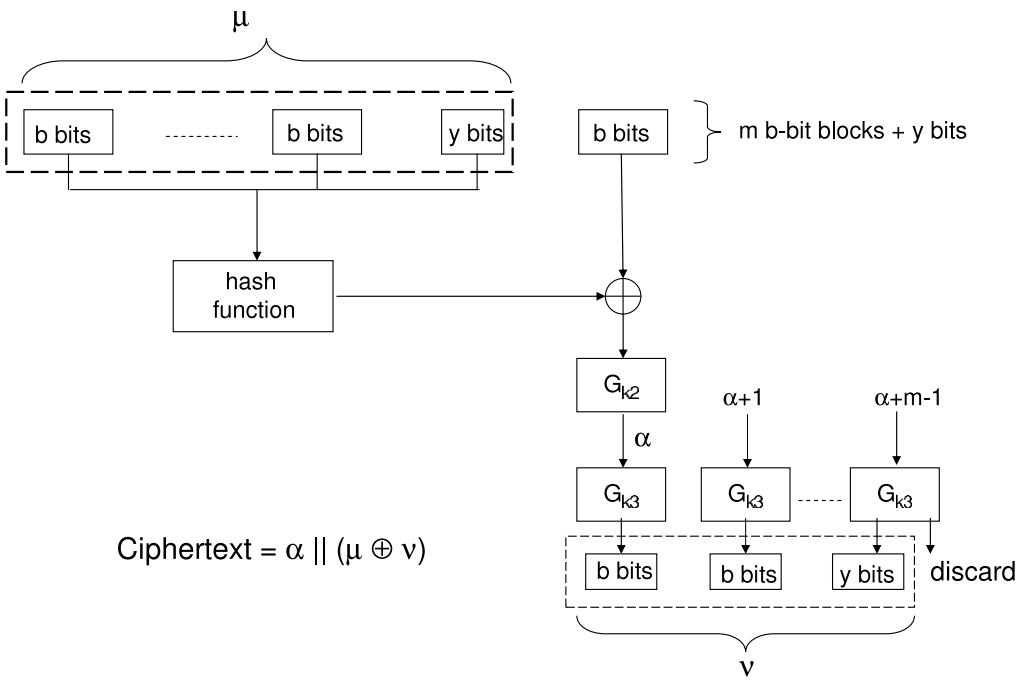

Figure 2.3: Variable-Length Block Cipher from Patel, *et.al.*

modification of Bellare and Rogaway's method [PRS04]. The CBC portion is replaced by a hash function, potentially reducing the amount of computation in this component of the algorithm. Now block sizes between one and two times the original block length require one application of a hash function and two applications of the cipher instead of four applications of the cipher. This modified version also treats the original block cipher as a black box with operations added around it and is computationally less efficient than padding. As I describe in Chapter 3, elastic block ciphers are created by adding steps between rounds to

allow the cipher to adapt to the block size. The number of rounds varies with the block size such that the workload gradually expands to twice that of the original block cipher as the block length expands. Only the round function of the original block cipher is treated as a black box when creating its elastic version instead of the entire cipher being used as a black box. This is the major methodological difference between my work and the proposals by Bellare and Rogaway, and Patel, *et al.*

A related construction, and one that I will later use in combination with the elastic network, is the CBC-Mask-CBC (CMC) mode of encryption by Halevi and Rogaway [HR03b]. The CMC mode allows for the creation of $2b, 3b, ...mb$-bit SPRPs from b-bit PRPs, for some integer $m > 2$. The CMC mode is computationally inefficient, but allows for the creation of $2b$ to $2mb$-bit PRPs and SPRPs when used as the round function in the elastic network. Refer to Chapter 4.8 for a description of CMC mode.

2.5 Modes of Encryption

In addition to the security of the block cipher, the actual manner in which it is applied (the mode of encryption) to data must be considered. Encrypting data one block at a time (ECB mode) permits several attacks. Plaintext patterns can easily be recognized since identical plaintext blocks will produce identical ciphertext blocks. Ciphertext blocks can be inserted, removed or replaced if no form of authentication is performed. Therefore, modes such as CBC have been created to prevent such attacks. The recommendations by NIST have remained CBC, OFB, CFB and CTR for the past three years [NIS01c]. Chapter 10 contains descriptions of these modes. More recently CCM, which uses CBC to create a message authentication code (MAC) followed by CTR mode to encrypt the MAC and plaintext, has also been recommended by NIST [NIS04]. However, weaknesses still exist. For example, CBC, which is the only mode recommended that does not involve using the block cipher as a stream cipher, is subject to block-wise adaptive attacks [JMV02] and splicing attacks. These attacks are described in Chapter 10. While splicing of ciphertext blocks encrypted in CBC mode will garble one plaintext block with probability close to 1, this can go unnoticed, especially when the data altered is part of an image as opposed to

text. A MAC combined with CBC mode will indicate the data has been altered, but not what portion has been altered. CTR, OFB and CFB are modes that utilize a block cipher to output a key stream. Using a key stream is not suitable for all applications due to the inability to reuse a key (a desirable feature when encrypting multiple files at separate times) and, in scenarios where data is transmitted between two entities, when synchronization of the key stream between the sending and receiving entities may be an issue.

I note that existing modes of encryption can be used to encrypt data without expanding the length of the plaintext by using what is called ciphertext stealing. This method does not provide any computational savings over padding; instead, it adds minor computational overhead. Ciphertext stealing pads the last plaintext block using some of the ciphertext from the previous block, and does not output the ciphertext bits used for the padding in order to maintain the length of the plaintext. Ciphertext stealing generally works as follows: For a block size of b bits, when encrypting $nb+y$ bits, for an integer $n \geq 1$ and integer y where $0 < y < b$, the mode of encryption proceeds as normal through the last full block. $b-y$ bits of the last full ciphertext block are prepended to the remaining y bits of plaintext to form the $(n+1)^{st}$ block. The ciphertext consists of the output from the mode of encryption on the first $n-1$ blocks, the y bits from the n^{th} block of output that were not prepended to the remaining y bits of plaintext and the entire b bits of output from the $(n+1)^{st}$ block. When decrypting, the last block is decrypted before the next to last block, and bits from the last plaintext block appended to the next to last (the partial) ciphertext block. This requires switching the order of the last two blocks when decrypting and computing the length of the last block to determine how many bits from the plaintext must be appended to the next to last ciphertext block. Minor tweaks to these steps are needed depending on the exact encryption mode used. For example, in CBC mode, the point at which the $b-y$ bits from the n^{th} ciphertext block are prepended to the remaining y bits to form the input to the last application of the block cipher is after the XOR of plaintext and ciphertext bits. The need to switch the order of the last two blocks when decrypting is a disadvantage of ciphertext stealing because changing the block order impacts the performance of high-speed hardware encryptors. Ciphertext stealing works with any block cipher without requiring modification to the block cipher, the only impact is to how a mode of encryption is applied to the last

one plus fractional block of data.

2.6 Summary

A tradition of using fixed-length block ciphers and the fact that fixed-length block ciphers are easier to design and implement than variable-length ones has contributed to lack of proposals for variable-length block ciphers. Furthermore, although the standards competitions for block ciphers have not explicitly requested submissions only support specific block sizes, they have not required that submissions support variable-length blocks. The few previous proposals for variable-length block ciphers fall into two categories, designing a cipher from scratch or treating an existing cipher as a black box and using it as a component around which to construct a variable-length block cipher. In the first category, there are currently no variable-length block ciphers that are proven to be secure in practice. In the second category, Bellare and Rogaway succeeded in creating a variable-length block cipher whose security is defined in terms of the original cipher. However, both their creation and a modified version of it are computationally inefficient compared to padding. Ciphertext stealing is another option that produces ciphertext that is the same length as the original, unpadded, plaintext. While it works with any block cipher and does not require any alteration of the block cipher itself, it does require minor alterations to modes of encryption and offers no computational savings compared to padding.

Chapter 3

Elastic Block Cipher Construction

3.1 Overview

In this chapter, I describe my algorithm for creating elastic block ciphers and the underlying structure, the elastic network, that serves as the basis for the algorithm. The algorithm converts the encryption and decryption functions of existing block ciphers to accept blocks of size b to $2b$ bits, where b is the block size of the original block cipher. My method uses a new network structure, the elastic network, into which the round function of the original block cipher is inserted. This allows the properties of the original block cipher's round function to be reused. The elastic network creates a permutation on $b+y$ bits from a round function that processes b bits, where $0 \leq y \leq b$. I neither modify the round function of the block cipher nor decrease the number of rounds applied to each bit; instead, the method allows bits beyond the supported block size to be combined with bits in the supported block size.

First, I describe the elastic network and explain why I could not use an existing structure, specifically, an unbalanced Feistel network [SK96]. Second, I describe the steps for converting any fixed-length block cipher to a variable-length block cipher. Four instantiations of elastic block ciphers are described in Chapter 6.

3.2 Elastic Network

Before introducing the elastic network, I define the following terms concerning diffusion that are used in the descriptions of the elastic network and the elastic block cipher construction.

Definition 2. *Bit Influence: Let $x1$ be the i^{th} bit of input to a b-bit block cipher. Let $x2$ be the j^{th} bit of output from round q of the block cipher. $x1$ influences $x2$ if changing $x1$ while holding all other $b-1$ input bits to the block cipher constant causes $x2$ to change with probability > 0.*

Definition 3. *Rate of Diffusion: Let x be one bit of input to a b-bit block cipher. The rate at which x influences all bit positions is measured in terms of the number of rounds and number of bit positions impacted.*

Definition 4. *Complete Diffusion: If every input bit to a b-bit block cipher influences the value in all b bits after q rounds, then the block cipher is said to have complete diffusion in q rounds.*

Complete diffusion does not imply security and is not the same as diffusion in an ideal block cipher where changing a single bit of input will cause each individual bit of output to change with 50% probability. In complete diffusion, the probability each individual bit of output changes must only be $> 0\%$ and may be 100%.

Definition 5. *Active Bit: A bit (position) input to a block cipher is called active in round j if the bit is input to the round function in round j.*

Definition 6. *Cycle: A cycle in a fixed-length, b-bit block cipher is the sequence of steps in which all b bits have been processed by the round function.*

For example, in AES, the round function is a cycle. In a balanced Feistel network, a sequence of two applications of the round function, which processes $\frac{b}{2}$ bits in each application, is a cycle. In RC6, the sequence of four applications of the round function is a cycle.

Definition 7. *Round Function for an Elastic Block Cipher: A round function in the elastic version of a fixed-length, b-bit block cipher is a cycle of the b-bit block cipher.*

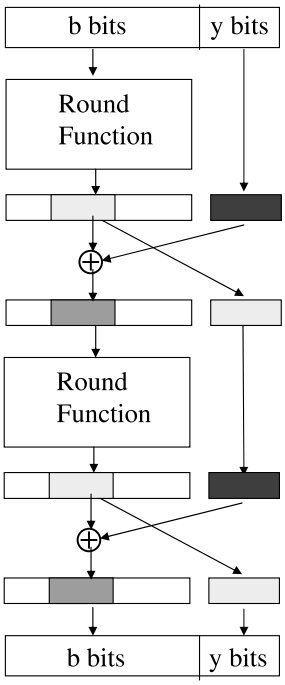

Figure 3.1: Two-Round Elastic Network

The purpose for creating the elastic network is to have a structure that enables existing fixed-length block ciphers to be converted to variable-length block ciphers by adding steps between rounds of the block cipher. While I did not want to use an existing block cipher as a black box in order to gain computational efficiency compared to padding data to an integral number of blocks, I did want to use the round function of the block cipher in order to leverage its properties. Therefore, one of my goals was to create a structure in which operations can be inserted between rounds of a block cipher that are independent of the round function of the block cipher. The properties I require of the structure are:

- It provides a permutation on $b + y$ bits for any $0 \leq y \leq b$ where b is the block size of the fixed-length block cipher.
- It is a single, generic, construction that can be used with any block cipher.
- The round function of any existing b-bit block cipher becomes a component of the

structure without any modifcation required to the round function.

- The number of rounds is not set by the structure, but rather the round function can be applied as many times as needed by a specific cipher.
- The rate of diffusion for $b+y$ bits is defined in terms of the rate of diffusion for b bits in the fixed-length block cipher.
- The operations involved in the structure allow for efficient implementations in terms of time and memory requirements.

The elastic network satisfies these properties. A two-round version of the network is shown in Figure 3.1. It works by inserting the unmodified round function (cycle) of the original, b-bit block cipher into the network. To create a permutation on $b+y$ bits, b bits are input to the round function, as would normally occur in the original block cipher, and y bits (let Y denote these bits) are omitted from the round function. After the round function is applied, but before its output is given as input to the next application of the round function, y of the b bits output from the round function (let X denote these y bits) are XORed with Y, allowing Y to become part of the b bits input to the next application of the round function. X becomes the y bits omitted from the next application of the round function. If the original cipher is a Feistel network, a cycle of the Feistel network is used as the round function. This allows all b bits to be processed by the round function of the original cipher before the swap of X and Y occurs. Similarly, if the original cipher is designed such that only a portion of the b bits is processed by the round function in each round, the round function is defined to be the cycle in which all portions of the b bits have been processed. RC6 is an example of a block cipher that is not a Feistel network and whose round function impacts only a subset of its input. Any number of rounds of the elastic network can be applied. The operations added around the round function are simple, involving only the XOR of bits and swapping of bit segments. Finally, the rate of diffusion is defined in terms of the rate of diffusion of the original cipher. Complete diffusion refers to the point at which every bit of the input to the block cipher has influenced every other bit. The elastic network requires at most one more round than the original block cipher to obtain complete diffusion.

Claim 3.1. *If complete diffusion occurs after q rounds (cycles) in the original, fixed-length version of a block cipher, it occurs after at most $q + 1$ rounds in the elastic version of the block cipher.*

Proof. A round in the elastic version of the block cipher uses a cycle from the original version of the cipher followed by the swapping of bits. By the end of the first round, the y bits left out of the round function have not impacted any other bits. The rate of diffusion for the b bits input to the first round function is the same as in the original cipher. The inputs to the second through the last application of the round function are influenced by all $b + y$ bits because of the XOR in the swap step after each round. Thus beginning at the second round, all $b + y$ bits influence the input to the round function and complete diffusion will occur within q rounds as in the original version. The b bits output from the $(q + 1)^{st}$ round function have been influenced by all $b + y$ bits; therefore, after the swap of bits that follows the $(q + 1)^{st}$ round, all $b + y$ bits have influenced the leftmost b bits and the rightmost y bits resulting from the swap step. Therefore, complete diffusion occurs by the end of the $(q + 1)^{st}$ round in the elastic version. □

3.3 Comparison of the Elastic Network to an Unbalanced Feistel Network

The elastic network is similar to an unbalanced Feistel network. One question that arises is why an unbalanced Feistel network cannot be used instead of the elastic network? An unbalanced Feistel provides the benefit of being its own inverse, with the round keys used in reverse so the round function does not have to be invertible; whereas this is not true of the elastic network. I compare the elastic network to an unbalanced Feistel network and explain why an unbalanced Feistel network does not possess all the properties required to create a variable-length block cipher from any existing block cipher.

While the two networks may appear similar, it is not feasible to use an unbalanced Feistel network to create elastic block ciphers. Figure 3.2 shows the structure of an unbalanced Feistel network compared to the elastic network. In a (balanced) Feistel network, the block is split into two components of equal length; whereas, in an unbalanced Feistel network

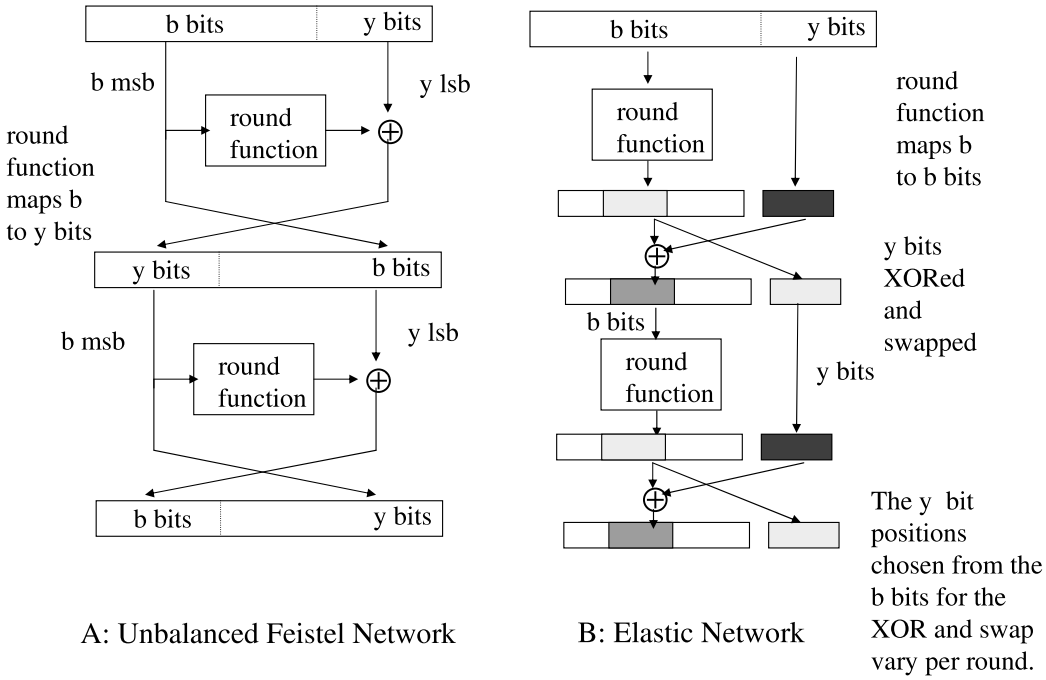

Figure 3.2: Unbalanced Feistel Network Compared to Elastic Network

the components do not have the same length and the lengths of the round function's input differs from the length of its output. The elastic network also involves splitting the block into two components, applying the round function to one component then XORing and swapping bits between the components to form the input to the next round. However, the elastic network differs from an unbalanced Feistel network in several ways.

1. The round function of the elastic network must be invertible; whereas, the round function of the unbalanced Feistel network does not need to be invertible. This is because the structures differ in what bits form the input to the round function. When the original block cipher is a Feistel network, an elastic version is created without requiring the original cipher's round function be invertible by using a complete cycle of the Feistel network as the round function.

2. In an unbalanced Feistel network the input to round i is XORed with the output of round $i + 1$ to form the input to round $i + 2$. In the elastic network, bits from the

outputs of rounds i and $i+1$ are XORed when forming the input to round $i+2$.

3. The round function maps b bits to b bits in the elastic network and maps b bits to y bits in the unbalanced Feistel network. This alone does not prevent an unbalanced Feistel network from being used with the round function of an existing block cipher that maps b bits to b bits because y bits can be chosen from the output of the round function when $y \leq b$.

4. $y \leq b$ in the elastic network. Whereas, an unbalanced Feistel network places no restriction on the length of y in relation to b.

5. The most important difference is that the unbalanced Feistel network provides poor diffusion to the extent that, for r rounds and a $(b+y)$-bit block, $b-y(r-1)$ bits of input appear in the output. Therefore, when encrypting data, part of the plaintext appears in the ciphertext when $y(r-1) < b$. In contrast, the elastic network guarantees complete diffusion in at most one more round than the original cipher. Even when r is large enough to prevent input bits from appearing in the output for all y, where $0 < y \leq b$, an unbalanced Feistel network provides no guarantee on the rate of diffusion. Instead, the rate of diffusion depends on the specific round function. This is due to the second and third items.

It is the last item that prevents an unbalanced Feistel network from being used to convert existing block ciphers to variable-length block ciphers. Obviously, when $y(r-1) < b$, such a cipher is insecure. Even if the number of rounds is set so $y(r-1) \geq b$, an unbalanced Fiestel network is not suitable for creating a variable-length block cipher by inserting the round function of an existing block cipher into the network. In order to (attempt) to use an unbalanced Feistel network with a round function of an existing block cipher that takes a b-bit input to create a variable-length block cipher, the block will be divided into b-bit and y-bit portions where $y \leq b$ and y bits will be selected from the round function's output to use in the XOR. However, this can result in poor diffusion. The round functions of block ciphers used in practice do not provide complete diffusion in a single round (which is one reason for multiple rounds). I consider what happens in an unbalanced Feistel network when $y < b$ and if all input bits to the round function do not impact all output bits. If one

of the b bits, let q be the position of this bit, input to the round function does not impact the bit positions that are involved in the XOR with the y bits and the bit in position q only influences bits in the rightmost y bits of the output, then the bit in position q going into round i will have no influence in the $(i+1)^{st}$ round. In fact, the round function in an unbalanced Feistel network must be defined very carefully; otherwise, it is possible for certain bits to have no impact on the other bits over several rounds. It is precisely this reason why an unbalanced Feistel network cannot be used to generically create variable-length block ciphers from existing ciphers by using an unmodified round function in the same manner as the elastic block cipher algorithm. I require a network structure that allows "plugging in" the round function from any existing block cipher and viewing the round function as a black box while at the same time providing the same level of security as the original cipher (in that the elastic block cipher is immune to any practical attack that recovers key or round-key bits to which the original cipher is immune). Using an unbalanced Feistel network to create variable-length block ciphers would require analyzing the round function to determine how many rounds are needed for sufficient diffusion and to prevent each type of attack, Therefore, an unbalanced Feistel network does not provide a generic structure for creating variable-length block ciphers from existing block ciphers by inserting the block cipher's round function into the network.

3.4 Elastic Block Cipher Algorithm

The method for converting a fixed-length block cipher into an elastic block cipher involves inserting the block cipher's round function (cycle) into the elastic network. Also, I add (or expand from the original cipher) whitening steps, and I add a key-dependent permutation before the first round and after the last round. The general structure of the method is shown in Figure 3.3. The following notation and terms will be used in the description and analysis of the elastic block cipher:

Notation:

- G denotes any existing block cipher with a fixed-length block size that is structured as a sequence of rounds. By default, any block cipher that is not structured as a

sequence of rounds is viewed as having a single round.

- r denotes the number of rounds (cycles) in G.
- b denotes the block length of the input to G in bits.
- y is an integer in the range $[0, b]$.
- G' denotes the modified G with a $(b + y)$-bit input for any valid value of y. G' will be referred to as the elastic version of G.
- r' denotes the number of rounds in G'.
- The round function of G' will refer to one entire cycle of G, as defined in Section 3.2.

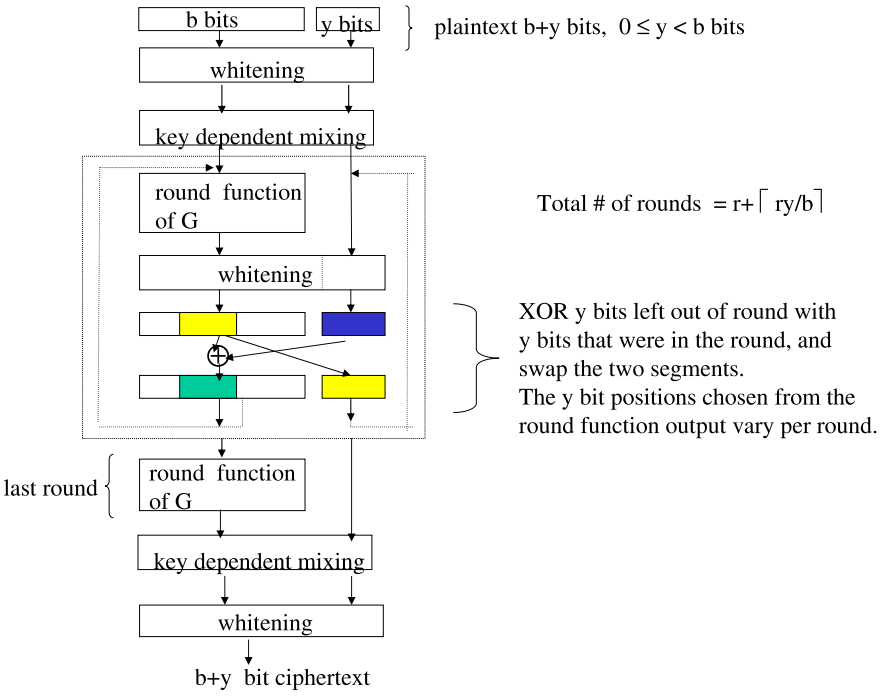

Figure 3.3: Elastic Block Cipher Structure

The process of converting a fixed-length block cipher into an elastic block cipher involves inserting the round function (cycle) of the block cipher into the elastic network, adding initial and final key-dependent permutations, adding or expanding initial and end of round

whitening, and determining the number of rounds required. Given a block cipher G with a b-bit block size, the following modifications are made to G to convert it to its elastic version, G', that can process $b + y$ bits, for $0 \leq y \leq b$.

1. Set the number of rounds, r', such that each of the $b+y$ bits is input to and active in the same number of rounds (cycles) in G' as each of the b bits is in G. $r' = r + \lceil \frac{ry}{b} \rceil$.

2. Apply initial and end of round whitening (XORing with expanded-key bits) to all $b+y$ bits. If G includes these whitening steps, the steps are modified to include all $b + y$ bits. If G does not have these whitening step, the steps are added when creating G'. In either case, additional bits of expanded-key material are required beyond the amount needed for G.

3. Prior to the first round and after the last round, apply a key-dependent mixing step that permutes or mixes the bits in a manner that any individual bit is not guaranteed to be in the rightmost y bits with a probability of 1. The leftmost b bits that are output from the initial mixing step are the input to the first round function. The initial mixing step is between the initial whitening and first round function. The final mixing step is after the last round function and prior to the final whitening.

4. Alternate which y bits are left out of the round function by XORing the y bits left out of the previous round function with y bits from the round function's output, then swap the result with the y bits left out of the previous round. This step is performed after the end of round whitening. Specifically:

   (a) Let Y denote the y bits that were left out of the round function.

   (b) Let X denote some subset of y bits from the round function's output of b bits. A different set of X bits (in terms of position) is selected in each round. How to select X is discussed in Section 3.5.

   (c) Set $Y \leftarrow X \oplus Y$.

   (d) Swap X and Y to form the input to the next round.

This step will be referred to as "swapping" or the "swap step," and may be added to the last round if it is required that all rounds be identical. However, having the swap step after the last round does not provide additional security.

The result, G', is a permutation on $b + y$ bits. Its inverse, the decryption function, consists of the network applied in reverse and the round function replaced by its inverse.

3.5 Explanation of Algorithm

The method is designed for G' to be equivalent to G, with the possible addition of whitening and the key-dependent mixing steps, when the data is an integral number of b-bit blocks, while accommodating a range of b to $2b$-bit blocks. The construction allows the round function of G to be reused and thus builds upon the round function's properties, including its differential and linear bounds. The following is an explanation of why specific steps are included in the construction.

Step 1: Each bit position of the input is required to be active in the same number of rounds in G' as the number of rounds in which each bit is active in G. This requirement allows the computational workload to increase proportionately to the block size while avoiding a reduced round attack on G from being applied to G'. As y increases, the number of rounds increases gradually from $r + 1$ when $0 < \frac{ry}{b} \leq 1$ to $2r$ when $r - 1 < \frac{ry}{b} \leq r$.

Step 2: Whitening is a useful heuristic against attacks that relate the output of a round to the input of the next round. The whitening steps assist in letting rounds work in isolation from each other in that the input to a round is unknown even when given the output of the previous round. In differential cryptanalysis, whitening does not impact the probability of a differential characteristic holding across the rounds of a block cipher because the whitening cancels with itself when computing the XOR of two inputs or outputs of a round. Linear cryptanalysis is an example of an attack whitening helps to prevent. In linear cryptanalysis, linear relationships amongst the plaintext, ciphertext and key bits are now based on the input of the i^{th} round being the output of the $(i-1)^{st}$ round $\oplus$ whitening as opposed to being equal to the output of the $(i-1)^{st}$ round.

Step 3: A key-dependent permutation or mixing of bits prior to the first round when encrypting or decrypting eliminates a one round differential that occurs with a probability of 1. This allows the first round to contribute to preventing a differential attack. The mixing step will need to take less time than a single round; otherwise, an additional round can be added instead to decrease the probability of a specific differential occurring. A trivial mixing that prevents the attacker from knowing with probability 1 which y bits are excluded from the first round is a key-dependent rotation. The implementations of elastic block ciphers described later contain a key-dependent permutation that performed the following operations: If the data block was $8(x1) + x2$ bits where $x1$ and $x2$ are integers and $0 \leq x2 < 8$, the $x1$ leftmost bytes are rotated to left $n1$ bytes where $n1$ is determined from an expanded-key byte. The remaining $x2$ bits (in the case where $b+y$ is not an integral number of bytes) are then swapped with bits from the $(n2)^{th}$ byte of the leftmost $8(x1)$ bits where $n2$ is determined by an expanded-key byte. The key-dependent mixing steps are assumed to be designed in a sensible manner. For example, the inverse of the round function would not be used.

Step 4: $X \oplus Y$ is performed instead of merely swapping X and Y in order to increase the rate of diffusion. If G does not have complete diffusion in one round, then at the end of the first round there is some subset S of bits output from the round that have not been impacted by some of the bits in X. While the bits in Y may impact S in the second round, swapping X and Y would result in the bits in X having no impact in the second round; whereas, swapping X with $X \oplus Y$ will allow the bits in X to impact the second round. Per Claim 3.1, complete diffusion in the elastic version of a block cipher takes at most one more round (cycle) than in the original version. As shown in Chapter 5.2 when proving that there is a direct relationship between the security of G' and the security of G, the relationship is independent of the bit positions involved in the swap step. In practice when implementing elastic block ciphers, I chose to vary the bit positions selected for X to ensure that all bit positions are involved in both the b-bit and y-bit components, as opposed to always selecting the same y positions for use in X. Varying the positions also increases the diffusion amongst the bits, as a bit in position q_1 that is swapped out at the end of round i then is swapped into position q_2 at the end of round $i + 1$, where q_1 and q_2 are in

the leftmost b bits, influences the input to the $(i+1)^{st}$ round function in the q_1^{th} bit due to the XOR and will influence the input to the $(i+2)^{nd}$ round function in the q_2^{th} bit. If all input bits to the round are utilized in the same manner, the bit positions chosen for X can be rotated across the rounds. For example, in AES all bytes are processed by the same functions within the round. In that case, X is formed from consecutive bits starting at position $a_1 + a_2 * i (\text{mod } b)$ in round i for some constants a_1 and a_2. When G is such that the round function only processes a subset of the b bits in each round, the swap step is inserted at the point at which all b bits have been processed by the round function (*i.e.,* after a cycle). For example, when G is a Feistel network the swap step and whitening are added after a complete cycle so a bit participates in the actions applied to each half of the b-bit block once prior to potentially being swapped out regardless of its position. The bit positions chosen to be swapped out after each round are a known part of the algorithm and are not determined by the key, plaintext or ciphertext.

Key Schedule: The options for a key schedule include modifying the key schedule of G to produce additional bytes, increasing the original key length and running the key schedule multiple times, or using an existing efficient stream cipher that is considered secure in practice (this also permits the key schedule to independent of the choice of G). At a minimum, I assume in theory that any expanded-key bits which are external to the round function are independent of expanded-key bits used within the round function and are created independently of the $(b+y)$-bit data block input to the cipher. In all of my implementations of elastic block ciphers, the RC4 stream cipher with the first 512 bytes of output discarded is used as the key schedule. Having one standard key schedule that can output as many expanded-key bits as needed is beneficial because it means only one implementation of a key schedule is necessary regardless of the block cipher and it avoids the need to analyze one key schedule per block cipher for flaws. A stream cipher was chosen to significantly increase the randomness of the expanded-key bits over those produced by existing key schedules. This does incur a performance penalty over existing key schedules, but eliminates certain attacks which arose because of the structure of existing key schedules. I discuss key schedules in more detail in Chapter 8.

Decryption: The inverse of the round function, if it is not its own inverse, must be used for decryption. As explained in Section 3.3, the elastic network is similar to an unbalanced Fiestel network, which does not require the round function be invertible. However, an unbalanced Feistel network could not be used in place of the elastic network. I remind the reader that the swap step is added after a complete cycle when the original cipher is a Feistel network, thus the inverse of the "round" function in the elastic version is merely running the cycle in reverse, as is normally done in any block cipher which is a Feistel network.

3.6 Summary

The elastic network allows for the creation of variable-length block ciphers from existing block ciphers by inserting the round function (cycle) from any existing block cipher into the network. When creating variable-length block ciphers using the elastic network, per round whitening and an intial and final key-dependent permutation are added to improve the security of the resulting cipher, which is referred to as an elastic block cipher.

The elastic version of a b-bit block cipher can process all block sizes in the range of b to $2b$ bits. The number of rounds in the elastic version of a block cipher is determined by the number of rounds in the original cipher and the block size. This results in the computational workload being proportional to the block size. Four examples of elastic block ciphers are described in Chapter 6. As explained in Chapter 5, the security of the elastic version of a cipher against practical, round-key recovery attacks, is directly related to that the security of the original cipher against such attacks.

The elastic network is similar to an unbalanced Feistel network, which could not be used as a generic network for creating the variable-length block ciphers due to its inability to guarantee a sufficient rate of diffusion. Aside from its use for creating variable-length block ciphers for use in practice, the elastic network provides a means for producing variable-length PRPs and SPRPs, as proven in Chapter 4.

Chapter 4

Creation of Variable-Length PRPs and SPRPs

4.1 Overview

Pseudorandom permutations (PRP) and strong pseudorandom permutations (SPRP) are theoretical counterparts to practical block ciphers. Ideally, a block cipher should be a SPRP. It is of interest to determine if the elastic network can create variable-length PRPs and SPRPs from fixed-length permutations, similar to the method for constructing practical elastic block ciphers using the round functions of existing fixed-length block ciphers.

I prove that the elastic network can be used to construct variable-length PRPs and SPRPs from fixed-sized PRPs. I first prove that a three-round elastic network and the inverse of a four-round elastic network are variable-length PRPs when their round functions are independently chosen random permutations. These results allow me to then derive the same result when the round functions are PRPs and to prove that a five-round elastic network is a variable-length SPRP when the round functions are independently chosen PRPs. I show that these are the minimum number of rounds required. It may be possible to relax the requirement that the round functions must independently chosen PRPs in a manner similar to what was done by Noar and Reingold when using Feistel networks to create PRPs and SPRPs from pseudorandom functions (PRF) [NR99]. While I have not determined to what extent the independence of the round functions can be relaxed, I do

prove that at least two of the round functions must differ, except with negligible probability. Specifically, I show that a three-round elastic network and the inverse of a four-round elastic network in which the round functions are identical are not PRPs. These results indicate some independence is required of the round functions.

By using the elastic network with b-bit PRPs as round functions, the resulting variable-length PRPs and SPRPs work on input sizes of $b+y$ bits, where $0 \leq y \leq b$. I can extend the range of the input size by using the CBC-Mask-CBC (CMC) mode of encryption [HR03b]. When the original block cipher is a PRP, the CMC mode creates a multiple-block SPRP, specifically it produces a mb-bit SPRP from a b-bit PRP for an integer $m \geq 2$. By using a PRP in CMC mode for each of the round functions in the elastic network, I am able to create variable-length SPRPs on $mb + y$ bits with $0 \leq y \leq mb$ from PRPs on b bits. I also point out that the elastic network can be used to create variable-length PRPs and SPRPs $b + y$ bits from $\frac{b}{2}$-bit PRFs. This is accomplished by using three-round Feistel networks as the round functions in the elastic network. When the round functions of a Feistel network are independently chosen PRFs, a three-round Feistel network is a PRP [LR88]. Thus, Feistel networks can provide the PRPs to use as the round functions in the elastic network.

4.2 PRP and SPRP Definitions

I remind the reader of the meaning of a PRP and a SPRP. While I am discussing permutations (as opposed to practical block ciphers), I will continue to use the terms "plaintext" and "ciphertext" to refer to the inputs and outputs of the permutation. Querying a permutation is viewed as giving it a plaintext and receiving the ciphertext. Querying the inverse of a permutation is viewed as giving the permutation a ciphertext and receiving the plaintext. I use the following terms in the definitions of a PRP and a SPRP:

- Random permutation: A permutation on b bits that is chosen randomly from all permutations on b bits.
- Let P be a permutations on b bits. P^{-1} denotes its inverse. $P(x)$ is the output of P when given input x.

- Chosen plaintext query: An adversary chooses an input, p_i, to a permutation, P, and receives the output, $c_i = P(p_i)$.
- Chosen ciphertext query: An adversary chooses an input, c_i, to the inverse of a permutation, P^{-1}, and receives the output, $p_i = P^{-1}(c_i)$.
- Chosen plaintext - chosen ciphertext queries: An adversary makes a series of queries to a permutation, P, and its inverse, P^{-1} and receives the outputs.
- Adaptive queries: When making chosen plaintext, chosen ciphertext or chosen plaintext - chosen ciphertext queries to a permutation (and/or its inverse), the queries are said to be adaptive if the adversary making the queries receives the output of the i^{th} query before forming the $(i+1)^{st}$ query and can use the previous i queries and their outputs when forming the $(i+1)^{st}$ query.
- Transcript: a sequence of input/output pairs corresponding to a permutation.

Any type of plaintext and/or ciphertext query is a subset of adaptive chosen plaintext - chosen ciphertext queries.

I now state a property of random permutations regarding the probability of a specific transcript occurring as the result of queries to the permutation. Let $P1$ be a random permutation on b bits and $P1^{-1}$ be its inverse. Let $\{(p_1, c_1), (p_2, c_2)\ ...\ (p_n, c_n)\}$ be n pairs of queries to $P1$ and the resulting output such that $c_i = P1(p_i)$ for $i = 1$ to n and n is polynomially related to b. The sequence of n pairs is a transcript of $P1$. When querying a permutation, queries will not be made for which the answers are already known. Specifically, given $j-1$ pairs, $\{(p_1, c_1), (p_2, c_2)\ ...\ (p_{j-1}, c_{j-1})\}$, if the j^{th} query is a plaintext query, then $p_j \neq p_i$ for $i < j$ and if the j^{th} query is a ciphertext query, then $c_j \neq c_i$ for $i < j$. For any transcript of n such queries to $P1$, the probability of the transcript occurring is $Pr_{RP} = \prod_{i=0}^{n-1} \frac{1}{2^n - i}$. The transcript may have been produced using adaptive chosen plaintext, adaptive chosen ciphertext or adaptive chosen plaintext - chosen ciphertext queries.

Definition 8. *Pseudorandom Permutation (PRP): A permutation, P on b bits is a PRP if it cannot be distinguished from a random permutation on b bits by using polynomially*

many, n, adaptive chosen plaintext or adaptive chosen ciphertext queries (but not both types of queries). Given a transcript, T_P, of n such queries from P, the probability T_P occurs is $\prod_{i=0}^{n-1} \frac{1}{2^n - i} + e$ for negligible e.

Definition 9. *Strong Pseudorandom Permutation (SPRP): A permutation, P on b bits is a SPRP if it cannot be distinguished from a random permutation on b bits by using polynomially many, n, adaptive chosen plaintext - chosen ciphertext queries. Given a transcript, T_P, of n such queries from P, the probability T_P occurs is $\prod_{i=0}^{n-1} \frac{1}{2^n - i} + e$ for negligible e.*

Another way of explaining the concepts of a PRP and a SPRP is to consider the probability with which an adversary can correctly determine whether or not a black box contains a specific permutation or a random permutation on b bits while using only polynomial (in b) many resources. Let P be a permutation on b bits. Given a black box that contains either P (or its inverse) or a random permutation, an adversary makes polynomially many adaptive queries to the black box and receives the outputs of the permutation within the box. If the probability the adversary correctly determines (using polynomial time and memory) the contents of the box is $\frac{1}{2} + e$ for negligible $e \geq 0$, then P is a PRP. In terms of block ciphers, this corresponds to the adversary being able to make either adaptive chosen plaintext queries or adaptive chosen ciphertext queries, but not both, to a black box which contains either the cipher or a random permutation.

A permutation, P, on b bits is a SPRP if it is not possible to distinguish P from a random permutation on b bits in polynomial (in b) time and memory when queries to both the permutation and its inverse are permitted. Given a black box that contains either P or a random permutation, the adversary can make polynomially many queries to the black box where the query indicates whether the permutation or its inverse is to be applied to the b-bit input. The probability the adversary correctly determines (using polynomial time and memory) the contents of the box is $\frac{1}{2} + e$ for negligible $e \geq 0$. In terms of block ciphers, this corresponds to the adversary being able to make both adaptive chosen plaintext - chosen ciphertext queries to a black box which contains either the cipher or a random permutation.

Luby and Rackoff proved that when using round functions which are independently chosen PRFs, three rounds are required in a balanced Feistel network to protect against adaptive chosen plaintext attacks and four rounds are required to protect against adaptive

chosen plaintext - chosen ciphertext attacks [LR88]. Naor and Reingold provided slightly modified constructions which achieve the same resistance to such attacks. The first round of the three-round version is replaced with a permutation, and the first and last rounds of the four-round version are replaced with pair-wise independent permutations [NR99]. An unbalanced Feistel network does not provide a variable-length PRP and SPRP using only three and four rounds, respectively. This is because in a three-round unbalanced Feistel network processing block sizes of $b + y$ bits, where $0 \leq y \leq b$, input bits appear in the output when $0 < 2y < b$, as mentioned in Chapter 3.3. This also prevents a four-round unbalanced Feistel network supporting $(b + y)$-bit block sizes, where $0 \leq y \leq b$, from being a variable-length SPRP.

4.3 Elastic Network: PRP from RPs

4.3.1 Overview

As my first step, I prove two properties that are used as building blocks for later proofs. Specifically, these properties are that a three-round elastic network and the inverse of a four-round elastic network are variable-length PRPs when their round functions are independently chosen random permutations. The proofs provided here can be used with pseudorandom permutations as the round functions instead of random permutations (RP) by taking into account that the probability a given output produced by a round function will vary from the output of a random permutation by some negligble amount. In order to avoid incorporating the negligible difference between a PRP and RP at this point, I prove the case in which the round functions are random permutations is a PRP. By using this result, I can then prove a the same networks are PRPs when the round functions are independently chosen PRPs.

Figure 4.1 shows three-round and four-round elastic networks. I will refer to the components of the network as they are labelled in Figure 4.1. The figure shows the direction of the elastic network used for encryption (three rounds) and for decryption (four rounds). I will refer to these as the encryption direction and the decryption direction.

Before stating the theorems and proofs, I provide a sketch of the proof to assist the

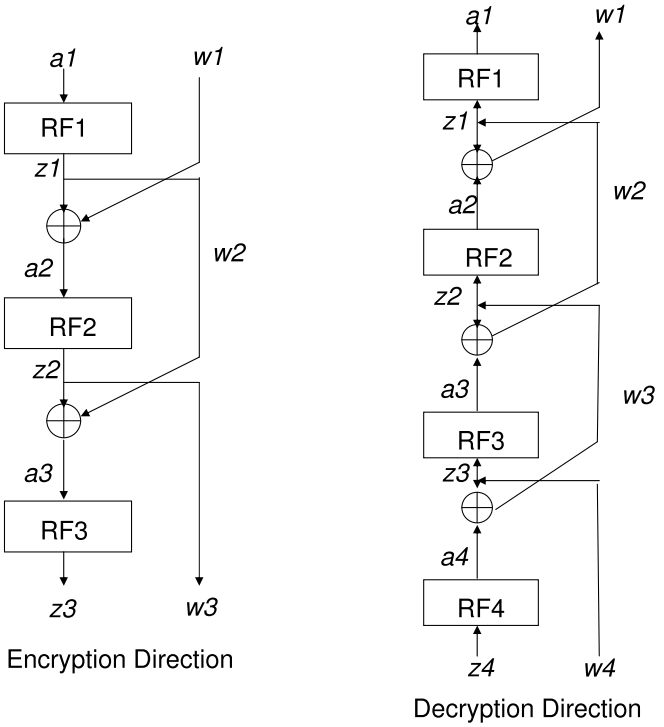

Figure 4.1: Three and Four-Round Elastic Networks

reader in understanding the proof. I show that any change to the plaintext input to a three-round elastic network, G', in which the round functions are independently chosen random permutations changes both the leftmost b bits and the rightmost y bits of the output in such a manner that neither portion of the output can be predicted with non-negligible probability when using polynomially many queries. The difference between the resulting probabilities and the corresponding probabilities from a random permutation on $b + y$ bits is negligible. I show that, for large b, the leftmost b bits of the output occur with probability $2^{-b} \pm e1$ and the rightmost y bits occur with probability $2^{-y} \pm e2$ for negligible $e1, e2$.

I then prove formally that if a three-round elastic network, G', with round functions that are independently chosen random permutations on b bits can be distinguished from a random permutation on $b + y$ bits using polynomially many queries to G', then at least one of the round functions can be distinguished from a random permutation on b bits, which is a contradiction to the fact that I defined G' to use random permutations as the round

functions. Therefore, I conclude that G' is a PRP. I use a black box, $B_{G'}$, that contains either G' or a random permutation on $b + y$ bits. Suppose a distinguisher, D_3, exists that can determine whether or not $B_{G'}$ contains G' using polynomially many adaptive queries to the box. I show that D_3 can be used to create a distinguisher for at least one of the round functions of G' to distinguish the round function from a random permutation on b bits, which is a contradiction to my definition of G' as having random permutations as the round functions. When I say a distinguisher for G' exists, I mean that the distinguisher, using polynomially many adaptive queries in one direction can predict or eliminate a possibility about an additional input/output pair value of the given permutation with greater certainty than that of a random guess. In contrast, with a random permutation, anything beyond the input/output pairs from the queries is known with the same probability as a random guess. I repeat the process for the inversse of a four-round elastic network.

4.3.2 Notation

I refer to the components of the network as they are labelled in Figure 4.1. I use the following notation:

- $b > 0$ is an integer.
- y is an integer such that $0 \leq y \leq b$.
- $X \oplus Y$ where X is a b-bit string and Y is a y-bit string, means the bits of Y are XORed with y specific bits of X and the other $b-y$ bits of X are treated as if they are XORed with 0's. If the resulting string is stored in a variable containing only y bits instead of b bits, the result consists only of the y bits in the positions that involved both X and Y instead of X and the $b - y$ 0's. For example, consider XORing a 2-bit string with a 4-bit string such that the XOR involves the leftmost 2 bits of the 4-bit string. Let $z1$ and $a2$ be 4-bit strings. Let $w1$ and $w2$ be 2-bit strings. If $z1 = 0110$ and $w1 = 11$, $a2 = z1 \oplus w1 = 1010$. $w2 = z1 \oplus w1 = 10$.
- $n > 0$ is an integer that generically represents the number of polynomially many (in terms of the length of the input) queries made to a function.

- $|X|$ is the length, in bits, of X.

I define the following variables and functions as shown in Figure 4.1.

- RFi is the i^{th} round function, for $i = 1, 2, 3, 4$. Any restrictions placed on a RFi will be specified as needed. Each round function is a permutation on b-bits.
- ai is the b-bit input to the i^{th} round function for $i = 1, 2, 3, 4$.
- zi is the b-bit output of the i^{th} round function for $i = 1, 2, 3, 4$.
- wi is the y bits left out of the i^{th} round function for $i = 1, 2, 3, 4$. For any particular elastic network, $w2$ is formed from a fixed set of y-bit positions from $z1$, $w3$ is formed from a fixed set of y-bit positions of $z2$, and $w4$ is formed from a fixed set of y-bit positions of $z3$ (*i.e.,* the positions of the bits taken from $z1$ to form $w1$ do not vary amongst the inputs to a specific three-round elastic network). Likewise, when forming $w2$, $w3$ and $w4$.
- When referring to a specific value for an ai, zi or wi, a subscript will be used. For example, $a1_j$.

4.3.3 G' Probabilities

I first show that any change to the input of a three-round elastic network, G', in which the round functions $(RF1, RF2, RF3)$ are independently chosen random permutations changes both the leftmost b bits and the rightmost y bits of the output in a manner that neither can be predicted with non-negligible probability when using polynomially many inputs. The encryption direction is described first.

When making a series of distinct queries to G' in the encryption direction, I consider how the input and outputs of each round in the j^{th} query compares to those of any previous query. Let $(a1_i, w1_i), (a1_j, w1_j)$ be two inputs where $i < j$ and all inputs and outputs of the previous $j - 1$ queries are known. Unless otherwise stated, when two b-bit values are said not to be equal, it means, given one of the values, the other can be any one of the remaining $2^b - 1$ values. With j such values in which no two are equal, the j^{th} value can take on any one of the remaining $2^b - j + 1$ values. I define x_{val} to be the number of times the specific

value val has occurred in the previous $j-1$ queries. For example, x_{w2_i} is the number of times the value of $w2_i$ has occurred in the previous $j-1$ queries. When randomly selecting b-bit values from all 2^b-bit values without replacement, the probability of a specific value occurring on the j^{th} selection is $\frac{1}{2^b-j+1}$. For a fixed subset of y of the b bits, the probability a specific y-bit value, val, occurs in the j^{th} selection is $\frac{2^{b-y}-x_{val}}{2^b-j+1}$. The inputs to G' must fall into one of the following three cases. For each case, I indicate whether or not the input and output of each round can be equal across two inputs to G'.

Case 1: $a1_j \neq a1_i$ and $w1_j = w1_i$

- $z1_j \neq z1_i$
- $a2_j \neq a2_i$
- $z2_j \neq z2_i$
- $w2_j = w2_i$ with probability $Pr1 = \frac{2^{b-y}-x_{w2_i}}{2^b-j+1}$. Equality holds if the y bits selected from the output of the first round function are identical in $z1_i$ and $z1_j$.

  Case 1a: $w2_j = w2_i$

  - $a3_j \neq a3_i$
  - $z3_j \neq z3_i$
  - $w3_j = w3_i$ with probability $Pr2 = \frac{2^{b-y}-x_{w3_i}}{2^b-j+1}$.

  Case 1b: $w2_j \neq w2_i$

  - If $a3_j = a3_i$, $z2_j$ must differ from $z2_i$ only in the y bits involved in the swap step and the XOR with the $w2$ values results in these y bits being identical. This requires a specific value for $z2_j$. The probability is $Pr3 = \frac{1}{2^b-j+1}$. Then $w3_j \neq w3_i$ and $z3_j = z3_i$.
  - If $a3_j \neq a3_i$ then $z3_j \neq z3_i$. $w3_j = w3_i$ with probability $Pr2$.

Case 2: $a1_j = a1_i$ and $w1_j \neq w1_i$

- $z1_j = z1_i$

- $w2_j = w2_i$
- $a2_j \neq a2_i$
- $z2_j \neq z2_i$
- $a3_j \neq a3_i$
- $z3_j \neq z3_i$
- $w3_j = w3_i$ with probability $Pr2$.

Case 3: $a1_j \neq a1_i$ and $w1_j \neq w1_i$

- $z1_j \neq z1_i$
- $w2_j = w2_i$ with probability $Pr1$.
- **Case 3a:** $a2_j = a2_i$
  - For $a2_j = a2_i$ to occur, $z1_i$ must differ from $z1_j$ only in the y bits involved in the swap step and the XOR with the $w1$ values results in these y bits being identical in $z1_i$ and $z1_j$. This requires a specific value for $z1_j$, which occurs with probability $Pr3$.
  - $z2_j = z2_i$
  - $w3_j = w3_i$
  - $a3_j \neq a3_i$
  - $z3_j \neq z3_i$

  Case 3b: $a2_j \neq a2_i$

  - $z2_j \neq z2_i$
  - For $a3_j = a3_i$ to occur, $z2_j$ must differ from $z2_i$ only in the y bits involved in the swap step and the XOR with the $w2$ values results in these y bits being identical. This requires a specific value for $z2_j$, thus the probability is $Pr3$. This also means $w2_j \neq w2_i$, $z3_j = z3_i$ and $w3_j \neq w3_i$.

– If $a3_j \neq a3_i$ then $z3_j \neq z3_i$. $w3_j = w3_i$ with probability $Pr2$. $a3_j \neq a3_i$ can occur when $w2_j \neq w2_i$ and when $w2_j = w2_i$.

If $(z3_j, w3_j)$ is the j^{th} output of a random permutation on $b+y$ bits, the probability the leftmost b bits equal the leftmost b bits of a previous output $(z3_i, w3_i)$ is $Pr(z3_j = z3_i) = \frac{2^y - x_{z3_i}}{2^{b+y}-j+1}$ where $1 \leq i < j$ and x_{z3_i} is the number of times the value $z3_i$ has appeared in the first $j-1$ queries to the random permutation. The probability the rightmost y bits of the j^{th} output equal those of a previous output is $Pr(w3_j = w3_i) = \frac{2^b - x_{w3_i}}{2^{b+y}-j+1}$ where $1 \leq i < j$ and x_{w3_i} is the number of times the value $w3_i$ has appeared in the first $j-1$ queries. When using polynomially many queries (in terms of $b+y$) to the random permutation and b is large enough to prevent an exhaustive search, the probability that two outputs have the same leftmost b bits is negligible. The probability a specific value of $w3$ occurs on the j^{th} query is $2^{-y} + e$ where e is the difference between 2^{-y} and $\frac{2^b - x_{w3_i}}{2^{b+y}-j+1}$. Therefore, e is negligible for large b.

Given the three classifications of inputs to G', the two probabilities, $Pr(z3_j = z3_i)$ and $Pr(w3_j = w3_i)$, for G' differ from those of a random permutation on $b+y$ bits by a negligible amount. In G', $Pr(z3_j = z3_i) \leq \frac{1}{2^b-j+1}$. Therefore, on polynomially many queries and large b, there is negligible chance of the leftmost b bits repeating. In G', $Pr(w3_j = w3_i) \leq \frac{2^{b-y} - x_{w3_i}}{2^b-j+1}$. The probability a specific value occurs on the j^{th} query is $2^{-y} + e'$ where e' is the difference between 2^{-y} and $\frac{2^{b-y} - x_{w3_i}}{2^b-j+1}$. Therefore, e' is negligible for large b.

I also point out that the probability of the input to either the second or third round function of G' being identical across two queries is negligible. In each case, either $a2_j \neq a2_i$ with probability 1 or $a2_j = a2_i$ with probability $\frac{1}{2^b-j+1}$. In each case, either $a3_j \neq a3_i$ with probability 1 or $a3_j = a3_i$ with probability $\frac{1}{2^b-j+1}$. In no case can both $a2_j = a2_i$ and $a3_j = a3_i$. Therefore, when making n queries to G' in the encryption direction, the inputs to the second round function can be considered to be n distinct values, except with negligible probabilty. Likewise for the inputs to the third round function.

4.3.4 A PRP from a Three-Round Elastic Network

I now prove that if a three-round elastic network, G', in the encryption direction with round functions that are independently chosen random permutations on b bits can be distinguished from a random permutation on $b + y$ bits, then at least one of the round functions can be distinguished from a random permutation on b bits, which is a contradiction. Therefore, I conclude that G' is a PRP when only plaintext queries are allowed to G'.

Theorem 4.1. *A three-round elastic network, G' on $b+y$ bits in which the round functions are independently chosen random permutations on b bits is a pseudorandom permutation on $b + y$ bits, where $0 \leq y \leq b$, in the encryption direction.*

Before beginning the proof, I define the following notation and terms for use in the proofs to both Theorem 4.1 and Theorem 4.2.

- $B_{G'}$ is a black box that contains either G' or a random permutation on $b + y$ bits.
- $(a1_i, w1_i)$ is an input to $B_{G'}$. $|a1_i| = b$ and $|w1_i| = y$ as defined previously.
- $(z3_i, w3_i)$ is the output of $B_{G'}$ corresponding to the input $(a1_i, w1_i)$. $|z3_i| = b$ and $|w3_i| = y$ as defined previously.
- D_3 is a distinguisher for G', meaning D_3 can determine whether or not $B_{G'}$ contains G' with probability $\frac{1}{2} + \alpha$ for non-negligible α, $0 < \alpha \leq \frac{1}{2}$ when using only polynomially (in $b + y$) many resources. Let D_3 return a 1 if it thinks $B_{G'}$ contains G' and a 0 otherwise. D_3 makes adaptive chosen plaintext or adaptive chosen ciphertext queries, but not both.
- $S1 = \{(a1_i, w1_i)\}$ and $S2 = \{(z3_i, w3_i)\}$, for $i = 1$ to n are the sets of n inputs and outputs D_3 uses to distinguish G' from a random permutation. When D_3 works by making queries to $B_{G'}$ in the encryption direction, $S1$ contains the inputs and $S2$ contains the resulting outputs. When D_3 works by making queries to $B_{G'}$ in the decryption direction, $S2$ contains the inputs and $S1$ contains the resulting outputs.
- B_{RFi} is a black box that contains either the i^{th} round function, RFi, of G' or a random permutation on b bits, for $i = 1, 2, 3$.

- $B_{RFi}(X)$ is the output of B_{RFi} when given input X.
- $B_{RFi}^{-1}(X)$ is the inverse of $B_{RFi}(X)$. *i.e.*, the inverse of whatever permutation is in B_{RFi} is applied to X.
- D_{RFi} is a distinguisher for RFi, meaning D_{RFi} can determine whether or not B_{RFi} contains RFi with probability $\frac{1}{2}+\alpha$ for non-negligible $\alpha, 0 < \alpha \leq \frac{1}{2}$ using polynomially (in $b + y$) resources. D_{RFi} uses either adaptive chosen plaintext or adaptive chosen ciphertext queries, but not both.
- $X \in \{0,1\}^m$ refers to selecting any m bit string as the value of X. X does not have to be selected randomly, the m bits can be chosen by any method.
- $X \leftarrow \{0,1\}^m$ refers to setting X to be a randomly chosen m bit string.
- "plaintext query" refers to a query to G' in the encryption direction and "ciphertext query" refers to a query to G' in the decryption direction (a query to G'^{-1}).

I note that the bit positions used in the swap steps in G' are not secret and this information can be used by any distinguisher. I define the following functions corresponding to the swap steps for use by the distinguishers:

- Let $Fi(X,Y)$ be a function that takes a b-bit input, X, and a y-bit input, Y, and returns the pair (Z,W) where Z is a b-bit string and W is a y-bit string. Fi replaces the y bits of X with the y bits of Y such that the bits in X which are replaced are in the same positions as the bits from the output of the i^{th} round function that are involved in the swap step after the i^{th} round of G'. Fi returns the updated X value in Z and returns a bit string, W, that contains the y bits of X that were removed from X XORed with the y bits inserted into X. $Fi(X,Y)$ computes the inverse of the i^{th} swap step in the elastic network.
- Let $FYi(X)$ be a function that takes a b-bit input X and returns the y bits that are in the same bit positions used to create wi from $z(i-1)$ in G'.
- Let Oi be an oracle that contains the i^{th} round function, RFi of G'. Oi^{-1} will refer to an oracle containing RFi^{-1}.

When I say a distinguisher, D_3, exists for G', I mean that the distinguisher, using polynomially many adaptive queries in the encryption direction or decryption direction (but not both directions) can predict or eliminate a possibility about an additional input/output pair of the given permutation with greater certainty than that of a random guess. In contrast, in a random permutation, nothing beyond the input/output pairs from the queries is known better than a random guess. I also note that allowing polynomially many oracle accesses to other random permutations that are not the permutation in question does not help a distinguisher learn about the actual permutation in question due to the statistical independence between any other random permutation and the permutation in question. Polynomial many, n, queries are made to $B_{G'}$ where $B_{G'}$ contains a random permutation or G' in the encryption direction. $S1$ contains the inputs and $S2$ contains the outputs.

Proof. I now prove Theorem 4.1. If D_3, a distinguisher for G' in the encryption direction, exists, D_3 must fall into one of the following categories:

- Category I: D_3 does not use the $z3$ portion of the output in its decision. The only part of the output used is the $w3$ portion. This means that given the n input/output pairs in $S1$ and $S2$, D_3 never uses the $z3$ portion from any of the pairs in $S2$.
- Category II: D_3 does not use the $w3$ portion of the output in its decision. The only part of the output used is the $z3$ portion. This means that given the n input/output pairs in $S1$ and $S2$, D_3 never uses the $w3$ portion from any of the pairs in $S2$.
- Category III: D_3 uses both the $z3$ and $w3$ portion of the outputs in its decision. This means that given n input/output pairs in $S1$ and $S2$, D_3 uses the $z3$ portion of the output from at least one of of the pairs in $S2$ and uses the $w3$ portion from at least one of the pairs in $S2$. Without using both portions, D_3 fails to distinguish the elastic network from a RP.

In each category, there are no restrictions on what portions of the inputs, $\{(a1_i, w1_i)\}$, are used. For each of the categories, I will show that the existence of D_3 implies a distinguisher can be formed for either the second or third round function of G', which contradicts the round functions being independently chosen random permutations.

Category I: If D_3 falls into Category I, a distinguisher, D_{RF2}, can be defined for the second round function, $RF2$. Intuitively, D_3 using only the $w3$ portion of the output of G' when $w3$ is from the output of $RF2$ whose inputs cannot be predicted with non-negligible probability implies D_3 can distinguish $RF2$ from a random permutation. The inputs to $RF2$ are distinct except with negligible probability. Therefore, the $w3$ values are distributed as if they are taken from the outputs of distinct queries to $RF2$, except with negligible probability and D_3 cannot rely on being given $w3$ values that were generated from identical inputs to $RF2$. Define D_{RF2} as follows:

Ask D_3 what its first query (input) would be if it was querying $B_{G'}$. Populate $S1$ with this first input, so $(a1_1, w1_1)$ has been chosen and is in $S1$. $S1$ is known to D_{RF2}.

for $i = 1$ to n {

Take $(a1_i, w1_i)$ from $S1$ for use in subsequent steps.

Set $z1_i = O1(a1)$.

Set $z2_i = B_{RF2}(z1_i \oplus w1_i)$.

Set $w3_i = FY3(z2_i)$.

Give $a1_i, w1_i, w3_i$ to D_3.

Add to $S1$ the next input D_3 would use when trying to distinguish D_3, having seen the inputs and partial output of the first i queries. This is $(a1_{i+1}, w1_{i+1})$.

}

Let ans be the value D_3 returns.

Return ans.

The values given to D_3 are the input and rightmost y bits of the output of a three-round elastic network with $RF1$ as the first round function and the contents of B_{RF2} as the second round function. The third round function is irrelevant in this case because D_3 is not using the output of the third round function. The values given to D_3 correspond to those of $S1$ and the $w3_i$ values of $S2$ when D_3 is allowed to make n adaptive chosen plaintext queries to $B_{G'}$. D_3 succeeds with non-negligible probability in determining whether or not it was given the input and partial output of G' implies D_{RF2} will succeed with non-negligible probability in determining if the n $(a2_i, z2_i)$ pairs correspond to the inputs and

outputs of $RF2$. Therefore, D_{RF2} can distinguish the contents of B_{RF2} using the n queries $\{O1(a1_i) \oplus w1_i\}$. B_{RF2}, contradicting the assumption that the second round function is a random permutation.

Category II: If D_3 falls into Category II, a distinguisher, D_{RF3}, can be defined for the third round function, $RF3$. Intuitively, D_3 using only the $z3$ portion of the output of G' when $z3$ is from the output of $RF3$ whose inputs cannot be predicted with non-negligible probability implies D_3 can distinguish $RF3$ from a random permutation. The inputs to $RF3$ are distinct except with negligible probability. Therefore, the $z3$ values are distributed as if they are the outputs of n distinct queries to $RF3$, except with negligible probability and D_3 cannot depend on being given $z3$ values that were generated from identical inputs to $RF3$. Therefore, D_3 using only the input to G' and the $z3$ portion of the output implies D_3 can distinguish $RF3$ from a random permutation.

Define D_{RF3} as follows:

Ask D_3 what its first query (input) would be if it was querying $B_{G'}$. Populate $S1$ with this first input, so $(a1_1, w1_1)$ has been chosen and is in $S1$. $S1$ is known to D_{RF3}.

for $i = 1$ to n {

Take $(a1_i, w1_i)$ from $S1$ for use in subsequent steps.

Set $z1_i = O1(a1_i)$.

Set $z2_i = O2(z1_i \oplus w1_i)$.

Set $w2_i = FY2(z1_i)$.

Set $z3_i = B_{RF3}(z2_i \oplus w2_i)$.

Give $a1_i, w1_i, z3_i$ to D_3.

Add to $S1$ the next input D_3 would use when trying to distinguish D_3, having seen the inputs and partial output of the first i queries. This is $(a1_{i+1}, w1_{i+1})$.

}

Let ans be the value D_3 returns.

Return ans.

The values given to D_3 are the input and leftmost b bits of the output of a three-round

elastic network with $RF1$ as the first round function, $RF2$ as the second round function and the contents of B_{RF3} as the third round function. The values given to D_3 correspond to those of $S1$ and the $z3_i$ values from $S2$ when D_3 is allowed to make n adaptive chosen plaintext queries to $B_{G'}$. D_3 succeeds with non-negligible probability in determining it was given the input and partial output of G' implies D_{RF3} will succeed with non-negligible probability in determining the contents of B_{RF3} by using n queries, $\{O2(O1(a1_i) \oplus w1_i) \oplus F2(O1(a1_i))\}$, contradicting the assumption that the third round function is a random permutation.

Category III: If D_3 falls into Category III, a second version of the D_{RF3} distinguisher I just defined can be created for the third round function, $RF3$. I call this new version D_{RF3v2}. Intuitively, D_3 using both the $z3$ and $w3$ portions of the output of G' when $z3$ is from the output of $RF3$ whose inputs cannot be predicted with non-negligible probability, where $w3$ is from the output of $RF2$ whose inputs cannot be predicted with non-negligble probability and where $w3$ contributes to the formation of the input of $RF3$ (and thus contributes to the input to the permutation that produces $z3$) implies D_3 can distinguish $RF3$ from random. D_3 cannot depend on being given $z3$ and/or $w3$ values that were generated by holding the inputs to $RF2$ and/or $RF3$ constant since this occurs with negligible probability. Therefore, D_3 can be viewed as using some relationship between partial information (*i.e.* $w3$) used in forming the input to $RF3$ and the output (i.e. $z3$) of $RF3$ to distinguish the third round function from a random permutation.

D_{RF3v2} is D_{RF3} with the modification that $w3_i$ is given to D_3 along with $a1_i, w1_i$ and $z3_i$. Define D_{RFv2} as follows:

Ask D_3 what its first query (input) would be if it was querying $B_{G'}$. Populate $S1$ with this first input, so $(a1_1, w1_1)$ has been chosen and is in $S1$. $S1$ is known to D_{RF3}.

for $i = 1$ to n {

Take $(a1_i, w1_i)$ from $S1$ for use in subsequent steps.

Set $z1_i = O1(a1)$.

Set $z2_i = O2(z1_i \oplus w1_i)$.

Set $w2_i = FY2(z1_i)$.

Set $z3_i = B_{RF3}(z2_i \oplus w2_i)$.

Set $w3_i = FY3(z2_i)$.

Give $a1_i, w1_i, z3_i, w3_i$ to D_3.

Add to $S1$ the next input D_3 would use when trying to distinguish D_3, having seen the inputs and output of the first i queries. This is $(a1_{i+1}, w1_{i+1})$.

}

Let ans be the value D_3 returns.

Return ans.

The values given to D_3 are the inputs and outputs of a three-round elastic network with $RF1$ as the first round function, $RF2$ as the second round function and the contents of B_{RF3} as the third round function. The values given to D_3 correspond to those of $S1$ and $S2$ when D_3 is allowed to make n adaptive chosen plaintext queries to $B_{G'}$. D_3 succeeds with non-negligible probability in determining it was given the input and output of G' implies D_{RF3v2} will succeed with non-negligible probability in determining the contents of B_{RF3} by using n queries, $\{O2(O1(a1_i) \oplus w1_i) \oplus F2(O1(a1_i))\}$, contradicting the assumption that the third round function is a random permutation.

For each category, I have shown that D_3 cannot exist. Therefore, a three-round elastic network cannot be distinguished from a PRP by using polynomially many plaintext queries when the round functions are independently chosen random permutations. □

4.3.5 G'^{-1} Probabilities

An elastic network and its inverse are not indentical (unlike a Feistel network); therefore, the inverse must be evaluated separately. As was done for the encryption direction, I show how changing any part of the input to G'^{-1} changes the output of G'^{-1} unpredictably when using polynomially many inputs. A four-round network is used. The notation is the same as that used in the three-round case. Again the input is divided into three cases and each case evaluated.

Case 1: $z4_j = z4_i$ and $w4_j \neq w4_i$

- $a4_j = a4_i$

- $z3_j \neq z3_i$
- $w3_j \neq w3_i$
- $a3_j \neq a3_i$
- $z2_j \neq z2_i$
- $a2_j \neq a2_i$
- $w2_j = w2_i$ if $a3_i = w3_i \oplus a3_j \oplus w3_j$ in the y bits that form $w2_i$ and $w2_j$. This requires y bits of $a3_j$ to be a specific value, which occurs with probability $Pr1 = \frac{2^{b-y} - x_{a3y_i}}{2^b - j + 1}$ where x_{a3y_i} is the number of times the required value occurred in previous queries.
  - Case 1b: $w2_j = w2_i$
    - $z1_j = z1_i$ if $a2_j = a2_i$ in the $b - y$ bits not involved in the swap step. If this occurs, $w1_j \neq w1_i$. $z1_j = z1_i$ with probability $Pr2 = \frac{2^y - x_{a2b_i}}{2^b - j + 1}$. Then $a1_j = a1_i$.
    - If $z1_j \neq z1_i$, $a1_j \neq a1_i$. Then $w1_j = w1_i$ if $a2_j = a2_i$ in the y bits involved in the swap step, which occurs with probability $Pr3 = \frac{2^y - x_{a2y_i}}{2^b - j + 1}$.
  - Case 1b: If $w2_j \neq w2_i$
    - $z1_j \neq z1_i$
    - $a1_j \neq a1_i$.
    - $w1_j = w1_i$ if $a2_j = w2_j \oplus a2_i \oplus w2_i$, which occurs with probability $Pr4 = \frac{2^{b-y} - x_{a2y_i}}{2^b - j + 1}$ where x_{a2y_i} is the number of times the y-bit value $w2_j \oplus a2_i \oplus w2_i$ occurred previously in the y bits of $a2$ involved in the swap.

Case 2: $z4_j \neq z4_i$ and $w4_j = w4_i$

- $a4_j \neq a4_i$
- **Case 2a:** $z3_j = z3_i$
  - This requires $a4_j = a4_i$ in the $b - y$ bits not involved in the swap step. This occurs with probability $Pr5 = \frac{2^y - x_{a4b_i}}{2^b - j + 1}$ where x_{a4b_i} is the number of times the

required value for the $b-y$ bits occurred in previous queries. Note: when $y = b$, $z3 = w4$ and $Pr5 = 1$. $y = b$ means $x_{a4b_i} = j - 1$ because $b - y = 0$, thus every previous query had $a4_j = a4_i$ in the $b - y$ bits.

- $w3_j \neq w3_i$
- $a3_j = a3_i$
- $z2_j \neq z2_i$
- $a2_j \neq a2_i$
- $w2_j \neq w2_i$
- $z1_j \neq z1_i$
- $a1_j \neq a1_i$
- $w1_j = w1_i$ if $a2_j = w2_j \oplus a2_i \oplus w2_i$, which occurs with probability $Pr4$.

- **Case 2b:** $z3_j \neq z3_i$
  - This requires at least one of the $b - y$ bits not involved in the swap step differ between $a4_i$ and $a4_j$. The probability of this occurring is $1 - Pr5$.
  - $a3_j \neq a3_i$
  - $w3_j = w3_i$ if $a4_j = a4_i$ in the y bits involved in the swap step. This occurs with probability $Pr6 = \frac{2^{b-y} - x_{a4y_i}}{2^b - j + 1}$ where x_{a4y_i} is the number of times the required value occurred in previous queries.
    - If $z2_j = z2_i$, then $a2_j = a2_i$ and $w2_j \neq w2_i$. $z2_j = z2_i$ if the $b - y$ bits not involved in the swap step are equal in $a3_j$ and $a3_i$. $z2_j = z2_i$ with probability $Pr7 = \frac{2^y - x_{a3b_i}}{2^b - j + 1}$ where x_{a3b_i} is number of times the $(b - y)$-bit portion of $a3$ occurred previously. $z1_j \neq z1_i$, $a1_j \neq a1_i$ and $w1_j \neq w1_i$.
    - If $z2_j \neq z2_i$ then $a2_j \neq a2_i$.
      - $w2_j = w2_i$ if $a3_j = a3_i$ in the y bits involved in the swap step. The occurs with probability $Pr8 = \frac{2^{b-y} - x_{a3y_i}}{2^b - j + 1}$ where x_{a3y_i} is the number of times the value corresponding to the y bits of $a3_i$ involved in the swap step previously occurred. $z1_j = z1_i$ with probability $Pr2$ then $a1_j = a1_i$ and $w1_j \neq w1_i$. If $z1_j \neq z1_i$, then $w1_j = w1_i$ with $Pr4$.

· If $w2_j \neq w2_i$ then $z1_j \neq z1_i$, $a1_ij \neq a1_j$. $w1_j = w1_i$ with probability $Pr4$.

– If $w3_j \neq w3_i$, which is the case with probability $1 - Pr6$, then $z2_j \neq z2_i$ and $a2_j \neq a2_i$.

* $w2_j = w2_i$ with probability $Pr8$. $w1_j = w1_i$ with probability $Pr3$ and then $z1_j \neq z1_i$ and $a1_j \neq a1_i$. If $w1_j \neq w1_i$ then $z1_j = z1_i$ with probability $Pr2$ and $a1_j = a1_i$.

* If $w2_j \neq w2_i$ then $z1_j \neq z1_i$ and $a1_j \neq a1_i$. $w1_j = w1_i$ with probability $Pr4$.

Case 3: $z4_j \neq z4_i$ and $w4_j \neq w4_i$

- $a4_j \neq a4_i$
- $z3_j \neq z3_i$
- $a3_j \neq a3_i$
- **Case 3a:** $w3_j = w3_i$

  – $w3_j = w3_i$ if $a4_j \oplus w4_j = a4_i \oplus w4_i$ in the y bits involved in the swap step. This occurs with probability $Pr9 = \frac{2^{b-y} - x_{a4y_i}}{2^b - j + 1}$ where x_{a4y_i} is the number of times the y bit value $w4_j \oplus w4_i \oplus a4_i$ occurred previously in $a4$ in the y bits involved in the swap step.

  * If $z2_j = z2_i$, then $a2_j = a2_i$ and $w2_j \neq w2_i$. $z2_j = z2_i$ with probability $Pr7$. Then $w2_j \neq w2_i$, $a2_j = a2_i$, $z1_j \neq z1_i$, $a1_j \neq a1_i$ and $w1_j \neq w1_i$.

  * If $z2_j \neq z2_i$, then $a2_j \neq a2_i$. $w2_j = w2_i$ with probability $Pr8$ If $w2_j = w2_i$ then $z1_j = z1_i$ and $a1_j = a1_i$ if $a2_j = a2_i$ in the $b - y$ bits not in the swap which occurs with probability $Pr2$, in which case $w1_j \neq w1_i$. If $z1_j \neq z1_i$ then $a1_j \neq a1_i$. i$w1_j = w1_i$ with probability $Pr3$. If $w2_j \neq w2_i$ then $z1_j \neq z1_i$ and $a1_j \neq a1_i$, and $w1_j = w1_i$ with probability $Pr4$.

- **Case 3b:** $w3_j \neq w3_i$

- This occurs with probability $1 - Pr6$. $w3_j \neq w3_i$ if the y-bit value $a4_j \oplus w4_j \neq a4_i \oplus w4_i$.
- $z2_j \neq z2_i$
- $a2_j \neq a2_i$
- $w2_j = w2_i$ with probability $Pr10 = \frac{2^{b-y} - x_{a3y_i}}{2^b - j + 1}$ where x_{a3y_i} is the number of times the y-bit value $w3_i \oplus w3_j \oplus a3_i$ occurred previously in the y bits of $a3$ involved in the swap.
  - $z1_j = z1_i$ with probability $Pr2$, then $a1_j = a1_i$ and $w1_j \neq w1_i$.
  - If $z1_j \neq z1_i$ then $a1_j \neq a1_i$. $w1_j = w1_i$ with probability $Pr3$.
- If $w2_j \neq w2_i$
  - $z1_j \neq z1_i$
  - $a1_j \neq a1_i$
  - $w1_j = w1_i$ with probability $Pr4$.

Given the three classifications of inputs to G'^{-1} with four rounds, the two probabilities, $Pr(a1_j = a1_i)$ and $Pr(w1_j = w1_i)$, for G'^{-1} differ from those of a random permutation on $b + y$ bits by a negligible amount. Let x_{val1} and x_{val2} refer generically to values of the form x_{aky_i} and x_{akb_i}, where $k = 2$ or 3. In all cases, $Pr(w1_j = w1_i) < \frac{2^{b-y} - x_{a2y_i}}{2^b - j + 1}$, which is $2^{-y} \pm e$ for negligible e for large b. $Pr(a1_j = a1_i) < \frac{2^{b-y} - x_{val2}}{2^b - j + 1} * \frac{2^y - x_{val3}}{2^b - j + 1}$, which is $\frac{1}{2^b - j + 1} \pm e$ where e is negligible for large b. These values differ by a negligible amount from the values corresponding to a random permutation stated at the end of Section 4.3.3. Specifically, if $(a1_j, w1_j)$ is the j^{th} output of a random permutation on $b + y$ bits, the probability the leftmost b bits equal the leftmost b bits of a previous output $(a1_i, w1_i)$ is $Pr(a1_j = a1_i) = \frac{2^y - x_{a1_i}}{2^{b+y} - j + 1}$ where $1 \leq i < j$ and x_{a1_i} is the number of times the value $a1_i$ has appeared in the first $j-1$ queries to the random permutation. The probability the rightmost y bits of the j^{th} output equal those of a previous output is $Pr(w1_j = w1_i) = \frac{2^b - x_{w1_i}}{2^{b+y} - j + 1}$ where $1 \leq i < j$ and x_{w1_i} is the number of times the value $w1_i$ has appeared in the first $j - 1$ queries. This is $2^{-y} \pm e$ for negligible e for large b. I also point out that both $z3_j = z3_i$ and $z2_j = z2_i$ cannot occur, and both $z2_j = z2_i$ and $z1_j = z1_i$ cannot occur.

4.3.6 A PRP from a Four-Round Elastic Network

By a similar argument to that used for the encryption direction, the inverse of a four-round elastic network, G'^{-1}, whose round functions are independently chosen random permutations cannot be distinguished from a PRP by using polynomially many queries to a black box containing G'^{-1} or a random permutation.

Theorem 4.2. *The inverse of a four-round elastic network, (G'^{-1}), on $b+y$ bits in which the round functions are independently chosen random permutations on b bits is a pseudorandom permutation on $b+y$ bits, where $0 \leq y \leq b$.*

Proof. The notation and terms are the same as used in the proof to Theorem 4.1 unless otherwise stated. Now the black box, $B_{G'}$, will contain G'^{-1} or a random permutation on $b+y$ bits. The categories for the distinguisher are the same as in the three-round case. For two of the categories, three rounds are sufficient for G'^{-1} to be a PRP. I prove these cases first. Then the proof for the third category, which requires four rounds, follows directly. The inputs are of the form $(z3, w3)$ when using three rounds and $(z4, w4)$ when using four rounds. The outputs are of the form $(a1, w1)$. D_3 and D_4 will denote the distinguishers when three and four rounds are under consideration, respectively. When the number of rounds is not specified, D_r will be used to denote either D_3 or D_4. If a distinguisher exists for G'^{-1} it must fall into one of the following three categories:

- Category I: D_r does not use the $a1$ portion of the output in its decision. The only part of the output used is the $w1$ portion. This means that given the n input/output pairs in $S2$ and $S1$, D_r never uses the $a1$ portion from any of the pairs in $S1$.
- Category II: D_r does not use the $w1$ portion of the output in its decision. The only part of the output used is the $a1$ portion. This means that given the n input/output pairs in $S2$ and $S1$, D_r never uses the $w1$ portion from any of the pairs in $S1$.
- Category III: D_r uses both the $a1$ and $w1$ portion of the outputs in its decision. This means that given n input/output pairs in $S2$ and $S1$, D_r uses the $a1$ portion of the output from at least one of them and uses the $w1$ portion from at least one of them. Without using both portions, D_r fails to distinguish the elastic network from a RP.

In each category, there are no restrictions on what portions of the inputs, $\{(z3_i, w3_i)\}$ or $\{(z4_i, w4_i)\}$, are used.

When D_r is restricted to Category II or III, only three rounds are needed for G^{-1} to be a PRP. These two categories will be addressed before category I. Similar to what was done with the encryption direction, D_r can be used to create a distinguisher for one of the round functions. Since the round functions are random permutations, this results in a contradiction; therefore, D_r cannot exist.

Category II: If D_3 falls into Category II, a distinguisher, D_{RF1}, can be defined for the inverse of the first round function of G' (the last round of G'^{-1}). Intuitively, D_3 using only the $a1$ portion of the output of G'^{-1} when $a1$ is from the output of $RF1^{-1}$ whose inputs cannot be predicted with non-negligible probability implies D_3 can distinguish $RF1^{-1}$ from a random permutation. The inputs to $RF1^{-1}$ are distinct except with negligible probability. Therefore, the $a1$ values are distributed as if they are the outputs of n distinct queries to $RF1^{-1}$, except with negligible probability. Therefore, D_3 using only the input to G'^{-1} and the $a1$ portion of the output implies D_3 can distinguish $RF1^{-1}$ from a random permutation.

Define D_{RF1} as follows:

Ask D_3 what its first query (input) would be if it was querying $B_{G'}$. Populate $S2$ with this first input, so $(z3_1, w3_1)$ has been chosen and is in $S2$. $S2$ is known to D_{RF1}.

for $i = 1$ to n {

Take $(z3_i, w3_i)$ from $S2$ for use in subsequent steps.

Set $a3_i = O3^{-1}(z3_i)$.

Set $(z2_i, w2_i) = F2(a3_i, w3_i)$.

Set $a2_i = O2^{-1}(z2_i)$.

Set $(z1_i, w1_i) = F1(a2_i, w2_i)$.

Set $a1_i = B_{RF1}^{-1}(z1_i)$.

Give $a1_i, z3_i, w3_i$ to D_3.

Add to $S2$ the next input D_3 would use when trying to distinguish D_3, having seen the inputs and output of the first i queries. This is $(z3_{i+1}, w3_{i+1})$.

}

Let ans be the value D_3 returns.

D_{RF3v2} returns ans.

The values given to D_3 are the inputs and outputs of the inverse of a three-round elastic network with $RF3$ as the third round function, $RF2$ as the second round function and the contents of B_{RF1} as the first round function. These values correspond to the contents of $S2$ and the $a1_i$ values of $S1$ when D_3 is allowed to make n adaptive chosen plaintext queries to $B_{G'}$. D_3 succeeds with non-negligible probability in determining it was given the input and output of G' implies D_{RF1} will succeed with non-negligible probability in determining the contents of B_{RF1}, contradicting the assumption that the first round function is a random permutation.

Category III: If D_3 falls into Category III, a distinguisher, D_{RF3}, can be defined for the inverse of the first round function, $RF1^{-1}$. Intuitively, D_3 can be viewed as using some relationship between partial information (*i.e.* $w1$) used in forming the input to $RF1^{-1}$ and the output (i.e. $a1$) of $RF1^{-1}$ to distinguish the first round function from a random permutation.

Define D_{RF1v2} to be D_{RF1} with the addition that the $w1_i$ values are also given to D_3.

Ask D_3 what its first query (input) would be if it was querying $B_{G'}$ in the decryption direction. Populate $S2$ with this first input, so $(z3_1, w3_1)$ has been chosen and is in $S2$. $S2$ is known to D_{RF1v2}.

for $i = 1$ to n {

Take $(z3_i, w3_i)$ from $S2$ for use in subsequent steps.

Set $a3_i = O3^{-1}(z3_i)$.

Set $(z2_i, w2_i) = F2(a3_i, w3_i)$.

Set $a2_i = O2^{-1}(z2_i)$.

Set $(z1_i, w1_i) = F1(a2_i, w2_i)$.

Set $a1_i = B_{RF1}^{-1}(z1_i)$.

Give $a1_i, w1_i, z3_i, w3_i$ to D_3.

Add to $S2$ the next input D_3 would use when trying to distinguish

D_3, having seen the inputs and output of the first i queries.

This is $(z3_{i+1}, w3_{i+1})$.

}

Let ans be the value D_3 returns.

Return ans.

The values given to D_3 are the inputs and outputs of the inverse of a three-round elastic network with $RF3$ as the third round function, $RF2$ as the second round function and the contents of B_{RF1} as the first round function. These values correspond to those of $S1$ and $S2$ when D_3 is allowed to make n adaptive chosen plaintext queries to $B_{G'}$. D_3 succeeds with non-negligible probability in determining it was given the input and output of G' implies D_{RF1v2} will succeed with non-negligible probability in determining the contents of B_{RF1}, contradicting the assumption that the first round function is a random permutation.

Category I: The result for this category follows directly from the results for Categories II and III. If D_4 only uses the $w1$ portion of the outputs, since $w1 = w2 \oplus a2$, this implies D_4 is using a combination of $a2$ and $w2$ on which to base its decision. This implies D_4 is a distinguisher for the first three rounds of the network in the decryption direction that falls into Category III because the leftmost b-bit portion ($a2$) and rightmost y-bit portion ($w2$) of the three round output is used.

Assume D_4 exists for the four-round network. D_4 is used to define a distinguisher, D_3, for the three rounds consisting of $RF4^{-4}$ to $RF2^{-2}$, taking inputs $(z4_i, w4_i)$ and producing outputs $(a2_i, w2_i)$. In this case, $B_{G'}$ is a black box containing either G^{-1} with four-rounds or a random permutation on $b + y$ bits. Let B_3 be a black box containing either the three-round elastic network formed from rounds $RF4^{-4}$ to $RF2^{-2}$ or a random permutation on $b + y$ bits.

Define D_3 as follows:

Ask D_4 what its first query (input) would be if it was querying $B_{G'}$ in the decryption direction. Populate $S2$ with this first input, so $(z4_1, w4_1)$ has been chosen and is in $S2$. $S2$ is known to D_3.

for $i = 1$ to n {

Take $(z4_i, w4_i)$ from $S2$ for use in subsequent steps.

Give $(z4_i, w4_i)$ to B_3 and get back $(a2_i, w2_i)$.

Set $w1_i = a2_i \oplus w2_i$.

Give $w1_i, z4_i, w4_i$ to D_4.

Add to $S2$ the next input D_4 would use when trying to distinguish $B_{G'}$, having seen the inputs and output of the first i queries. This is $(z4_{i+1}, w4_{i+1})$.

}

Let ans be the value D_4 returns.

D_3 returns ans.

The values given to D_4 are the inputs and rightmost y bits of the outputs of the inverse of a four-round elastic network. These y bits are formed from both the b-bit and y-bit portions of the output of three rounds. Therefore, by the assumption D_4 exists, D_3 will succeed with non-negligible probability in determining that the $(a2_i, w2_i)$ values were formed from the first three rounds of decryption. This contradicts the previous result from Category III.

For each of the three categories, I have shown D_r cannot exist. Therefore, the inverse of a four-round elastic network is a PRP when the round functions are independently chosen random permutations. □

4.4 Elastic Networks: Counter-Examples

I provide a lower bound on the minimum number of rounds needed in an elastic network to create PRPs and SPRPs by providing examples of when fewer rounds are not PRPs and SPRPs. I also show that a certain level of independence is required between the round functions by considering cases when all of the round functions are identical. First, I show that at least three rounds are needed for an elastic network to be a PRP by proving that a two-round elastic network is not a PRP regardless of the round functions. Second, I show that a three-round elastic network is not a PRP when the round functions are identical. Third, I show that the inverse of a three-round elastic network is not a PRP regardless of

the round functions. Fourth, I show that the inverse of a four-round elastic network is not a PRP when the round functions are identical. Fifth, I show that three and four-round elastic networks are not SPRPs, regardless of the round functions. When proving an elastic network is not a PRP or SPRP under specific conditions on the number of rounds and/or round functions, it is sufficient to provide an example for one block size. All of the counter-examples use a $2b$-bit block size ($y = b$). Each example will not hold with probability 1 when $y < b$.

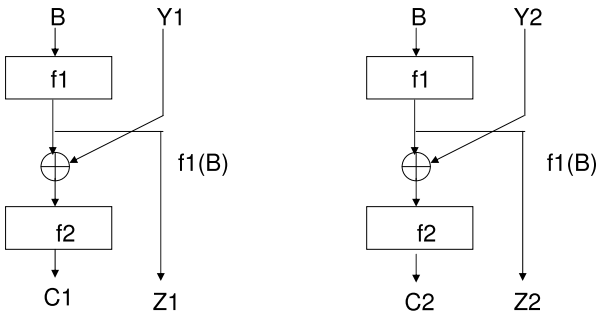

Figure 4.2: Elastic Block Cipher Structure: Two-Round Attack

Claim 4.1. *An elastic network with exactly two rounds is not a PRP.*

Proof. This claim holds regardless of the properties of the round functions. Consider the case where $y = b$. Given two $2b$-bit plaintexts of the form $B||Y1$ and $B||Y2$, let the ciphertexts be denoted by $C1||Z1$ and $C2||Z2$, respectively. As shown in Figure 4.2, $Z1 = Z2$ with probability 1. If the two-round construction was a PRP on $b + y$ bits, then for large b, this equality would occur with probability $2^{-b} \pm e$ for negligible e instead of with probability 1. □

Claim 4.2. *A three-round elastic network is not a PRP when the round functions are identical.*

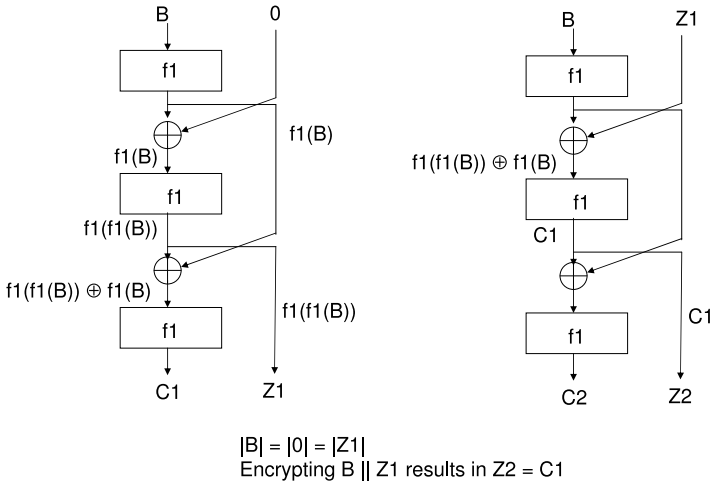

Figure 4.3: Three-Round Elastic Network with Identical Round Functions

Proof. Consider the case shown in Figure 4.3 when $y = b$. Let 0 denote a string of y zeroes. Encrypt $B||0$ and let $C1||Z1$ denote the resulting ciphertext. $Z1 = f1(f1(B))$. $C1 = f1(f1(f1(B)) \oplus f1(B))$. Then encrypt $B||Z1$ and let $C2||Z2$ denote the ciphertext. $Z2 = C1$ with probability 1. If this three-round network was a PRP on $b + y$ bits, then for large b, this equality would occur with probability $2^{-b} \pm e$ for negligible e instead of with probability 1. □

Claim 4.3. *The inverse of a three-round elastic network is not a PRP.*

Proof. This is illustrated in Figure 4.4. The inputs to the round functions are defined in the directions of the arrows in the figure and correspond to the direction of decryption. This claim holds regardless of the properties of the round functions and is due to the fact that, when $y = b$, the input to the inverse of the second round function is known because it is the rightmost y bits. In contrast, in the encryption direction, the XOR after the first round prevents the input to the second round function from being chosen. Let 0 denote a string of b zeroes. When $y = b$, create four $2b$-bit ciphertexts of the form $C1||0$, $C2||0$, $C1||Z$ and $C2||Z$ where $C1 \neq C2$ and $Z \neq 0$. Let the plaintexts be denoted by $B1||Y1$,

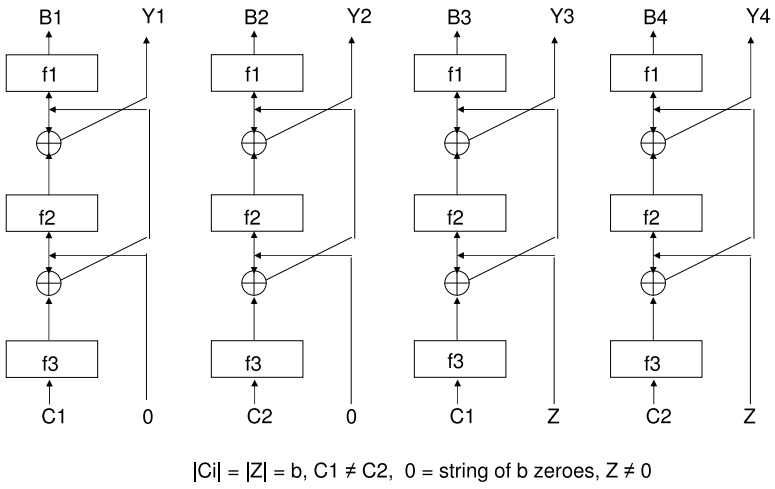

Figure 4.4: Three-Round Elastic Network: Chosen Ciphertext Attack

$B2||Y2$, $B3||Y3$ and $B4||Y4$. Then $Y1 = f2^{-1}(0) \oplus f3^{-1}(C1)$, $Y2 = f2^{-1}(0) \oplus f3^{-1}(C2)$, $Y3 = f2^{-1}(Z) \oplus Z \oplus f3^{-1}(C1)$ and $Y4 = f2^{-1}(Z) \oplus Z \oplus f3^{-1}(C2)$. As a result, $Y1 \oplus Y2 = Y3 \oplus Y4$ with probability 1. If the three-round network was a PRP on $2b$ bits in the decryption direction, then for large b, this equality would occur with probability $2^{-b} \pm e$ for negligible e instead of with probability 1. When $y < b$, the attack does not hold with probability 1 because the input to the second round of decryption contains $b - y$ bits of $f4^{-4}(Ci)$. These $b - y$ bits would have to be equal for $f4^{-4}(C1)$ and $f4^{-4}(C2)$. □

Claim 4.4. *The inverse of a four-round elastic network in which the round functions are identical is not a PRP.*

Proof. Consider the case shown in Figure 4.5 when $y = b$. Let 0 denote a string of b zeroes. Decrypt $0||0$ and let $B1||Y1$ denote the resulting plaintext. $B1 = f1^{-1}(0)$. $Y1 = f1^{-1}(f1^{-1}(0)) = f1^{-1}(B1)$. Decrypt $0||B1$ and let $B2||Y2$ denote the resulting plaintext. $Y2 = f1^{-1}(B1) \oplus f1^{-1}(0) = Y1 \oplus B1$ with probability 1. If the inverse of this four-round network was a PRP on $b+y$ bits, then for large b, this equality would occur with probability $2^{-b} \pm e$ for negligible e instead of with probability 1. □

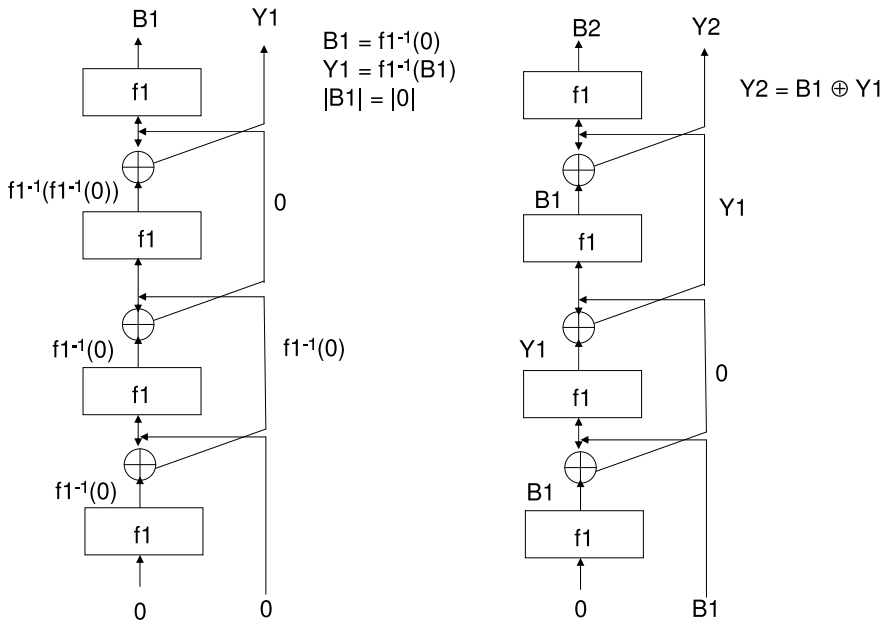

Figure 4.5: Four-Round Elastic Network with Identical Round Functions

Neither a three-round nor a four-round elastic network is a SPRP. In both cases, this can be shown with an adaptive chosen plaintext - chosen ciphertext attack in which two chosen plaintexts are encrypted then two chosen plaintexts formed from the two resulting ciphertexts are decrypted.

Claim 4.5. *A three-round elastic network is not a SPRP when* $b = y$.

Proof. This claim holds regardless of the properties of the round functions. I note that this claim also follows from Claim 4.6. The following sequence of two encryptions and two decryptions can be used to distinguish the three-round elastic network from a SPRP when $b = y$. Each plaintext and ciphertext is of length $2b$, i.e. $|B| = |Bi| = |Yi| = |Ci| = |Zi| = b$ $\forall i$.

Encrypt two plaintexts of the form $B||Y1$ and $B||Y2$. The b-bit portion is constant and the $Yi's$ may be any b bits such that $Y1 \neq Y2$. Let $C1||Z1$ and $C2||Z2$ be the resulting ciphertexts. This is depicted in Figure 4.6.

From the two resulting ciphertexts, form and decrypt the two ciphertexts $C1||(Z1 \oplus Z2)$ and $C2||(Z1 \oplus Z2)$. Let $B3||Y3$ and $B4||Y4$ denote the two resulting plaintexts. This is

depicted in Figure 4.7. $Y3 \oplus Y4 = Z1 \oplus Z2$ with probability 1.

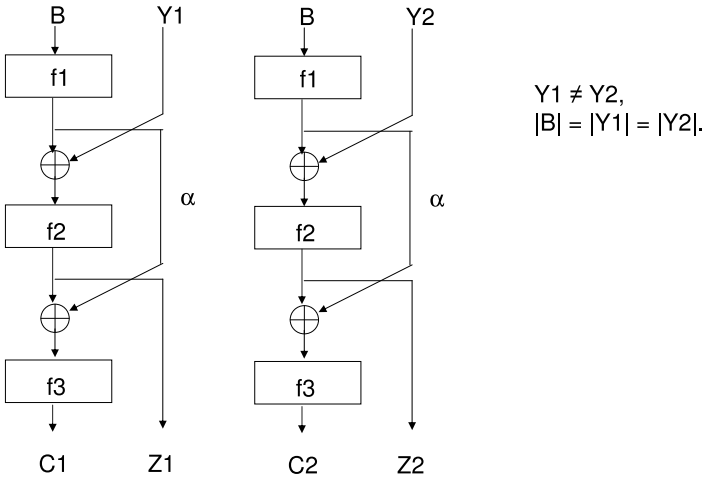

Figure 4.6: Chosen Plaintexts for the Chosen Plaintext - Chosen Ciphertext Attack

The equality is obtained as follows:

- Let $\alpha, \alpha 1$ and $\alpha 2$ denote the bits left out of the second round as shown on the figures.
- Let $\mu = Z1 \oplus Z2$.
- Let ω be the output from the round function in the second round of decryption when the input to the round function is $\mu = Z1 \oplus Z2$ as shown in Figure 4.7.

Notice that:

$$\begin{aligned} \alpha &= Z1 \oplus f3^{-1}(C1) \\ &= Z2 \oplus f3^{-1}(C2) \\ \alpha 1 &= f3^{-1}(C1) \oplus \mu \\ \alpha 2 &= f3^{-1}(C2) \oplus \mu \end{aligned}$$

Expanding u then substituting α in $\alpha 1$ results in:

$$\alpha 1 = f3^{-1}(C1) \oplus Z1 \oplus Z2$$

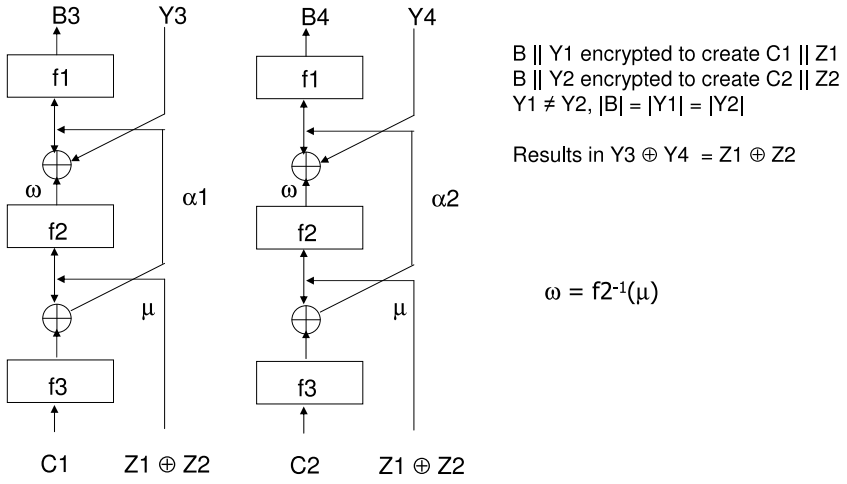

Figure 4.7: Chosen Ciphertexts for the Chosen Plaintext - Chosen Ciphertext Attack

$$= Z2 \oplus \alpha.$$

Likewise,

$$\begin{aligned} \alpha 2 &= f3^{-1}(C2) \oplus Z1 \oplus Z2 \\ &= Z1 \oplus \alpha \end{aligned}$$

Rewriting $Y3$ and $Y4$ in terms of $\alpha 1$ and $\alpha 2$ results in:

$$\begin{aligned} Y3 \oplus Y4 &= \omega \oplus \alpha 1 \oplus \omega \oplus \alpha 2 \\ &= \alpha 1 \oplus \alpha 2 \\ &= Z2 \oplus \alpha \oplus Z1 \oplus \alpha \\ &= Z1 \oplus Z2 \end{aligned}$$

Therefore, with probability 1, $Y3 \oplus Y4 = Z1 \oplus Z2$ in this adaptive chosen plaintext - chosen ciphertext attack. If the elastic network was a SPRP, then for large b, this equality would hold with probability $2^{-b} \pm e$ for negligible e. □

The attack does not hold with probability 1 when $y < b$ because the input to the second round of decryption is y bits of $Z1 \oplus Z2$ and $b - y$ bits from the output of $f3^{-1}$. Since

$C1 \neq C2$, $f3^{-1}(C1) \neq f3^{-1}(C2)$ and the $b - y$ bits are not guaranteed to be equal. A four-round elastic network is also not a SPRP.

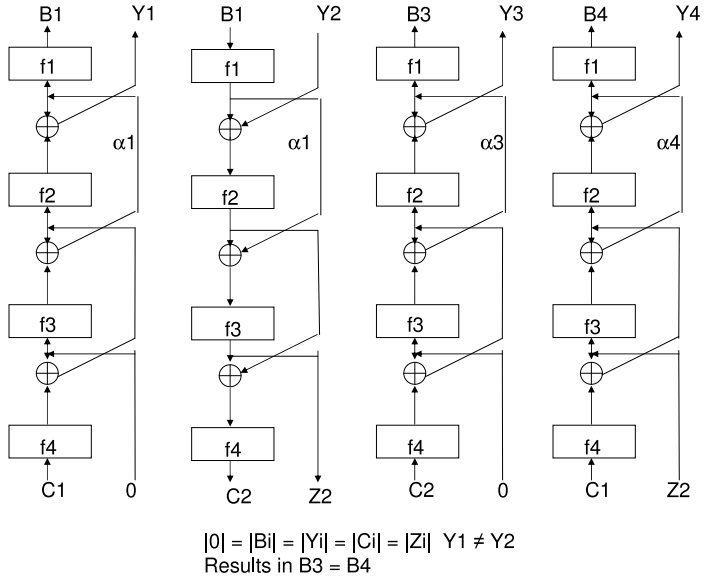

Figure 4.8: Four-Round Elastic Network: Chosen Plaintext - Chosen Ciphertext Attack

Claim 4.6. *A four-round elastic network is not a SPRP when $b = y$.*

Proof. This claim holds regardless of the properties of the round functions and is due to the fact that a three-round elastic network in the decryption direction is not a PRP. In the three round case, using chosen ciphertexts only, a relationship can be pushed through the three rounds of decryption into the right half of the output with probability 1 when $y = b$. In the four round case, the same approach is used in that the halves of two ciphertexts are switched to form to new ciphertexts and push a relationship into the rightmost y bits of the output of the third round. When $y = b$, this becomes the entire input to the round function in the fourth round of decryption. This time, one plaintext must be encrypted to assist in providing the values from which the ciphertexts are formed. The sequence of three decryptions and one encryption shown in Figure 4.8 can be used to distinguish the four-round elastic network from a SPRP when $y = b$. Each plaintext and ciphertext is of length $2b$, i.e. $|B| = |Bi| = |Yi| = |Ci| = |Zi| = b \ \forall i$. Let 0 denote a string of y zeroes.

Decrypt a ciphertext of the form $C1||0$. Let $B1||Y1$ be the resulting plaintext. Encrypt a plaintext of the form $B1||Y2$ with $Y2 \neq Y1$. Let $C2||Z2$ be the resulting ciphertext. The output of the first round function, $\alpha 1$, is identical in both the decryption and encryption. Form two ciphertexts, $C2||0$ and $C1||Z2$, and decrypt them. Let $B3||Y3$ and $B4||Y4$ denote the two resulting plaintexts.

Notice that:

$$\begin{aligned}\alpha 1 &= f4^{-1}(C1) \oplus f3^{-1}(0) \\ &= Z2 \oplus f4^{-1}(C2) \oplus f3^{-1}(Z2) \\ \alpha 3 &= f4^{-1}(C2) \oplus f3^{-1}(0) \\ \alpha 4 &= Z2 \oplus f4^{-1}(C1) \oplus f3^{-1}(Z2)\end{aligned}$$

By rearranging the equations for $\alpha 1$:

$$f4^{-1}(C2) \oplus f3^{-1}(0) = Z2 \oplus f4^{-1}(C1) \oplus f3^{-1}(Z2).$$

Therefore,

$$\alpha 3 = \alpha 4 \text{ and } B3 = B4.$$

$B3 = B4$ with probability 1; whereas, if the network was a SPRP this equality would hold for large b with probability $2^{-b} \pm e$ for negligible e. As in the other examples, this result does not hold with probability 1 when $y < b$. □

4.5 Three-Round Elastic Network: PRP from PRPs

I now prove that a three-round elastic network in the encryption direction is a PRP when the round functions are independently chosen PRPs.

Theorem 4.3. *A three-round elastic network in the encryption direction is a variable-length PRP on $b + y$ bits if the round functions are independently chosen PRPs on b bits and $0 \leq y \leq b$.*

Proof. I consider the relationships between the four versions shown in Figure 4.9 of a three-round elastic network. In each version, the round functions are chosen independently of each other and map a b-bit input to a b-bit output.

I define the following six permutations:

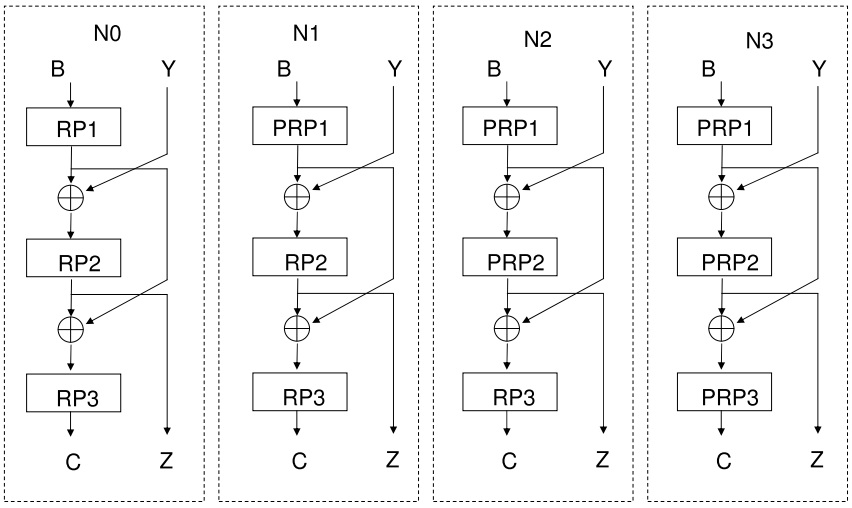

Figure 4.9: Three-Round Networks Consisting of RPs and PRPs

- Let $PRP1, PRP2, PRP3$ be three independently chosen pseudorandom permutations.
- Let $RP1, RP2, RP3$ be three independently chosen random permutations.

Let Ni refer to a three-round elastic network in the encryption direction in which the first i round functions are pseudorandom permutations and the remaining round functions are random permutations, for $i = 0, 1, 2, 3$ defined as follows:

- Network 0 ($N0$): Each round function is a RP. The round functions are $RP1$, $RP2$ and $RP3$.
- Network 1 ($N1$): The first round function is the PRP. The second and third round functions are RPs. The round functions are $PRP1$, $RP2$ and $RP3$.
- Network 2 ($N2$): The first two round functions are PRPs and the third round function is a RP. The round functions are $PRP1$, $PRP2$ and $RP3$.
- Network 3 ($N3$): Each round function is a PRP. The round functions are $PRP1$, $PRP2$ and $PRP3$.

As shown by Theorem 4.1, $N0$ is a PRP. Therefore, if Theorem 4.3 is not true it is possible to distinquish $N3$ from $N0$ with probability $\geq \alpha$ for some non-negligible α where $0 < \alpha \leq 1$. I will show that if $N3$ can be distinquished from random then at least one of $PRP1, PRP2$ and $PRP3$ can be distinguished from random in order to derive a contradiction and thus conclude Theorem 4.3 is true.

Let D be a distinguisher that takes $(b+y)$-bit inputs and runs in polynomial time. D outputs a 1 if it thinks the inputs are the outputs of a random permutation and outputs a 0 otherwise. Let $Pr(Ni)$ be the probability that D outputs a 1 when given polynomially many outputs from Ni. If $N3$ can be distinguished from a random permutation, then $|Pr(N0) - Pr(N3)| \geq \alpha$.

However,

$$|Pr(N0) - Pr(N3)| = |Pr(N0) - Pr(N1) + Pr(N1) - Pr(N2) + Pr(N2) - Pr(N3)|$$
$$\leq |Pr(N0) - Pr(N1)| + |Pr(N1) - Pr(N2)| + |Pr(N2) - Pr(N3)|.$$

Therefore, $\alpha \leq |Pr(N0) - Pr(N1)| + |Pr(N1) - Pr(N2)| + |Pr(N2) - Pr(N3)|$.

This implies at least one term on the right side of the inequality is $\geq \frac{\alpha}{3}$. Therefore, it is possible to distinguish a three-round elastic network in the encryption direction that has i round functions that are pseudorandom permutations and $3-i$ round functions that are random permutations from a three-round elastic network that has $i-1$ round functions that are pseudorandom permutations and $4-i$ round functions that are random permutations with non-negligible probability, where i is at least one value from $\{1,2,3\}$. Therefore, it is possible distinguish between a round function which is a random function and one that is a pseudorandom function with non-negligible probability, contradicting the definition of pseudorandom. □

I note that a four-round elastic network in the encryption direction is also a PRP. This is used later when proving a five-round elastic network is a SPRP.

4.6 Four-Round Elastic Network: PRP from PRPs

I now prove that a four-round elastic network in the decryption direction is a variable-length PRP when the round functions are randomly chosen PRPs.

Theorem 4.4. *The inverse of a four-round elastic network is a variable-length PRP on $b+y$ bits if the round functions are independently chosen PRPs on b bits and $0 \leq y \leq b$.*

Proof. The proof is similar to that for the three round network. This time I consider the relationships between the five networks in the decryption direction instead of four in the encryption direction. In each version, the round functions are chosen independently of each other and map a b-bit input to a b-bit output.

I define the following eight permutations:

- Let $PRP1, PRP2, PRP3, PRP4$ be four independently chosen pseudorandom permutations.
- Let $RP1, RP2, RP3, RP4$ be four independently chosen random permutations.

Let Ni refer to the inverse of a four-round elastic network in which the first i round functions are pseudorandom permutations and the remaining round functions are random permutations, for $i = 0, 1, 2, 3, 4$ defined as follows:

- Network 0 ($N0$): Each round function is a RP. The round functions are $RP1$, $RP2$, $RP3$ and $RP4$.
- Network 1 ($N1$): The first round function is the PRP. The second to fourth round functions are RPs. The round functions are $PRP1$, $RP2$, $RP3$ and $RP4$.
- Network 2 ($N2$): The first two round functions are PRPs and the last two are RPs. The round functions are $PRP1$, $PRP2$, $RP3$ and $RP4$.
- Network 3 ($N3$): The first three round functions are PRPs and the last one is a RP. The round functions are $PRP1$, $PRP2$, $PRP3$ and $RP4$.
- Network 4 ($N4$): Each round function is a PRP. The round functions are $PRP1$, $PRP2$, $PRP3$ and $PRP4$.

As shown by Theorem 4.2, $N0$ is a PRP. Therefore, if Theorem 4.4 is not true it is possible to distinquish $N4$ from $N0$ with probability $\geq \alpha$ for some non-negligible α where $0 < \alpha \leq 1$. I will show that if $N4$ can be distinquished from random then at least one of

$PRP1, PRP2$, $PRP3$ and $PRP4$ can be distinguished from random in order to derive a contradiction and thus conclude Theorem 4.4 is true.

Let D be a distinguisher that takes $(b + y)$-bit inputs and runs in polynomial time. D outputs a 1 if it thinks the inputs are the outputs of a random permutation and outputs a 0 otherwise. Let $Pr(Ni)$ be the probability that D outputs a 1 when given polynomially many outputs from Ni. If $N4$ can be distinguished from a random permutation, then $|Pr(N0) - Pr(N4)| \geq \alpha$.

However,

$|Pr(N0) - Pr(N4)| = |Pr(N0) - Pr(N1) + Pr(N1) - Pr(N2) + Pr(N2) - Pr(N3) + Pr(N3) - Pr(N4)|$

$\leq |Pr(N0) - Pr(N1)| + |Pr(N1) - Pr(N2)| + |Pr(N2) - Pr(N3)| + |Pr(N3 - Pr(N4)|$.

Therefore, $\alpha \leq |Pr(N0) - Pr(N1)| + |Pr(N1) - Pr(N2)| + |Pr(N2) - Pr(N3)| + |Pr(N3) - Pr(N4)|$.

This implies at least one term on the right side of the inequality is $\geq \frac{\alpha}{4}$. Therefore, it is possible to distinguish a four-round elastic network in the decryption direction that has i round functions which are pseudorandom permutations and $4 - i$ round functions that are random permutations from a four-round elastic network that has $i - 1$ round functions that are pseudorandom permutations and $5 - i$ round functions that are random permutations with non-negligible probability, where $i \in \{1, 2, 3, 4\}$. Therefore, it is possible distinguish between a round function which is a random function and one that is a pseudorandom function with non-negligible probability, contradicting the definition of pseudorandom. □

4.7 Five-Round Elastic Network: SPRP from PRPs

I will show that a five-round elastic network in which the round functions are independently chosen pseudorandom permutations is a SPRP. I note that a five-round elastic network consisting of round functions that are independently chosen pseudorandom permutations is a PRP in both the encryption and decryption directions because the last four rounds in either direction is a PRP.

Before stating the theorem regarding the SPRP, I prove a claim. By the definition of

a SPRP, any random permutation is a SPRP. Let $RP1$ and $RP2$ be two independently chosen random permutations, each on n bits. Define $Perm1(x) = RP2(RP1(x))$, where x is of length m. $Perm1$ is a random permutation on n bits and is a SPRP. Now I consider what happens if I use a combination of pseudorandom permutations and permutations in place of RP1 and RP2.

I define the permutations, $P1$, $P2$, $PRP1$ and $PRP2$ to satisfy the following conditions:

- $P1(x)$ and $P2(x)$ are permutations on m bits. $P1$ and $P2$ are independently chosen. $P1 \neq P2$, except with negligible probability. $P1$ is not pseudorandom in that a relationship between some subset of bits in its inputs and outputs that occurs with non-negligible probability is known, but the exact permutation is unknown. Specifically, when given a black box that contains either $P1$ or a random permutation on b bits, it is possible to determine the contents of the box in polynomially many queries. However, when using $P1$ in forming PA as defined below, the exact permutation corresponding to $P1$ is unknown. Likewise for $P2$, which is used to form PB as defined below.

- $PRP1(x)$ and $PRP2(x)$ are pseudorandom permutations on m bits whose indepedence is defined by the independence of $P1$ and $P2$ such that $P2(PRP2(P1(X))) = PRP1^{-1}(X)$.

- $PA(x) = PRP2(P1(x))$

- $PB(x) = PRP1(P2(x))$. Therefore, $PB^{-1} = PA^{-1}$

- $Perm2$ will refer to the permutation corresponding to PA and PB. $Perm2 = PA$ and $Perm2^{-1} = PB$.

It is possible to define $P1, P2, PRP1$ and $PRP2$ that satisfy these constraints. For example, I will later show how a five-round elastic network can be viewed in this manner by defining $P1$ to be the first round, $P2$ to be the inverse of the last round, $PRP2$ to be the last four rounds and $PRP1$ to be the inverse of the first four rounds. $Perm2$ is a pseudorandom permutation on n bits (this is just $PRP2$ and $PRP1$ with the inputs selected by

choosing n bits then applying a permutation, $P1$ or $P2$, to the input before giving it to the pseudorandom permutation).

Claim 4.7. *$Perm2$ is a SPRP.*

Proof. In order for $Perm2$ to be a SPRP it must not be possible to distinguish $Perm2$ from a random permutation on polynomially many (n) queries to PA and its inverse, PB. For simplicity, when I say an adversary is querying $Perm1$ or $Perm2$, I mean the adversary is able to issue queries to both the permutation and its inverse. The adversary does not have direct access to $P1$ and $P2$, meaning the adversary is not able to query $P1$ and use the output as input to $PRP2$ and/or query $P2$ and use the output as input to $PRP1$. The adversary can only give inputs to PA and PB.

- Let (p_i, c_i), for $i = 1$ to n be pairs of m bit strings such that $c_i = PA(p_i)$.
- Let $< +, p_i >$ denote a query to PA using input p_i.
- Let $< -, c_i >$ denote a query to PB using input c_i.
- Let t_i be the output of the i^{th} query. $t_i = c_i$ when the query is $< +, p_i >$ and $t_i = p_i$ when the query is $< -, c_i >$.
- Let $T = (t_1, t_2, ...t_n)$ be the output of n distinct queries to PA. If the i^{th} query is $< +, p_i >$ and the j^{th} query is $< -, c_i >$, $t_j = p_i$ if and only if $t_i = c_j$, for $i \neq j$. Without loss of generality I can assume that if an adversary queries with $< +, p_i >$ that he will not later query with $< -, c_i >$ since he knows the answer will be p_i regardless of whether he is querying $Perm1$ or $Perm2$.
- Let $U = (u_1, u_2, ...u_n)$ be the output of n distinct queries made to $Perm1$.

I will refer to U and T as transcripts of $Perm1$ and $Perm2$, respectively. In order for $Perm2$ to be a SPRP, it must not be possible to distinguish T from U with non-negligible probability. The probability of u_{i+1} ocurring given $(p_1, c_1), (p_2, c_2)...(p_i, c_i)$ is $\frac{1}{2^m - i}$ because $Perm1$ is a random permutation. The probability of a specific U occuring is $Pr_R = \prod_{i=0}^{n-1} \frac{1}{2^m - i}$.

Since PA is a pseudorandom permutation, it is not possible to distinguish the output, t_i, of any single query from the output of a random permutation with non-negligible probability.

For any single query to PA, the output occurs with probability $\frac{1}{2^m} + e$ for some negligible e. When given i queries to PA, the $(i+1)^{st}$ such query produces an output that occurs with probability $\frac{1}{2^m - i} + e_{A_i}$ for negligible e_{A_i}. Likewise, when given i queries to PB, the $(i+1)^{st}$ such query produces an output that occurs with probability $\frac{1}{2^m - i} + e_{B_i}$ for negligible e_{B_i}. Even though PA and PB are inverses of each other, there is no non-negligible relationship between the outputs of PA and PB because these are the outputs of $PRP2$ and $PRP1$, respectively. A transcript of $n1$ distinct queries to PA will occur with probability $(\prod_{i=0}^{n1-1} \frac{1}{2^m - i}) + e_A$ for negligible e_A. A transcript of $n2$ distinct queries to PB will occur with probability $(\prod_{j=0}^{n2-1} \frac{1}{2^m - j}) + e_B$ for negligible e_B.

I consider the probability with which a transcript, T_{PA}, of $n1$ queries to PA occurs and with which a transcript, T_{PB}, of $n2$ queries to PB occurs. Suppose an adversary makes $n1$ queries to PA and that between the queries, the adversary is given (p_l, c_l) pairs that correspond to PA (i.e., the adversary is given extra pairs for which he did not need to expend resources) such that overall, the adversary is given $n2$ such pairs. The adversary will not repeat any query or make a query for which he already been given the outcome. Let na_i be the number of (p_l, c_l) pairs the adversary has been given prior to the $(i+1)^{st}$ query to PA. $na_i \geq na_{i-1}$ for $1 \leq i \leq n1$. T_{PA} occurs with probability $Pr_A = (\prod_{i=0}^{n1-1} \frac{1}{2^m - i - na_i}) + e_{PA}$ for negligible e_{PA}. Suppose an adversary makes $n2$ queries are made to PB and that between the queries, the adversary is given (p_l, c_l) pairs that correspond to PB (i.e., the adversary is given extra pairs for which he did not need to expend resources) such that overall, the adversary is given $n1$ such pairs. The adversary will not repeat any query or make a query for which he already been given the outcome. Let nb_j be the number of (p_l, c_l) pairs the adversary has been given prior to the $(j^+1)st$ query to PB. $nb_j \geq nb_{j-1}$ for $1 \leq j \leq n2$. T_{PB} occurs with probability $Pr_B = (\prod_{j=0}^{n2-1} \frac{1}{2^m - j - nb_j}) + e_{PB}$ for negligible e_{PB}.

When $n = n1 + n2$ queries are made to Perm2 such that $n1$ queries are made to PA and $n2$ are made to PB (the queries can be in any order), the probability of the resulting transcript, T, from Perm2 can be written as the product of Pr_A and Pr_B. Let qB_i be the number of queries made to PB between the i^{th} and $(i+1)^{st}$ queries to PA. Let qA_j be the number of queries made to PA between the j^{th} and $(j+1)^{st}$ queries to PB. By setting $na_i = \sum_{k=0}^{i} qA_k$ and $nb_j = \sum_{k=0}^{j} qB_k$, the probability of T occurring is

$(Pr_A)(PR_B) = ((\prod_{i=0}^{n1-1} \frac{1}{2^m - i - na_i}) + e_{PA}) * ((\prod_{j=0}^{n2-1} \frac{1}{2^m - j - nb_j}) + e_{PB})$

$= (\prod_{i=0}^{n1-1} \frac{1}{2^m - i - na_i}) * (\prod_{j=0}^{n2-1} \frac{1}{2^m - j - nb_j}) + (\prod_{i=0}^{n1-1} \frac{1}{2^m - i - na_i}) * e_{PA} + (\prod_{j=0}^{n2-1} \frac{1}{2^m - j - nb_j}) * e_{PB} + e_{PA} * e_{PB}$.

$= \prod_{i=0}^{n-1} \frac{1}{2^m - i} + e$ for negligible e.

Therefore, it is not possible to distinguish T from U with non-negligible probability. □

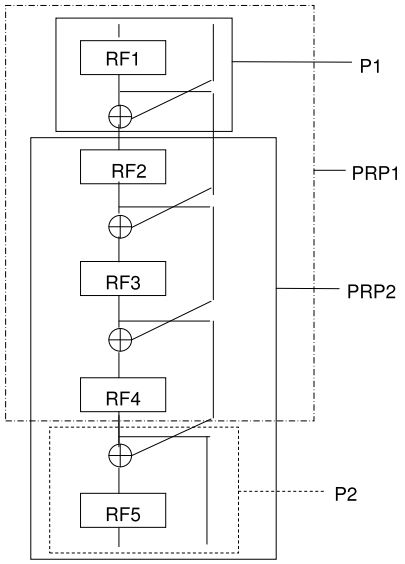

Figure 4.10: Five-Round Elastic Network as Two PRPs and Two Permutations

Theorem 4.5. *A five-round elastic network on $b + y$ bits, $0 \leq y \leq b$, in which each round function is an independently chosen PRP on b bits is a SPRP on $b + y$ bits.*

Proof. G' refers to a five-round elastic network on $b + y$ bits with round functions that are independently chosen PRPs on b bits. G' can be defined in a format consistant with the four permutations used in Claim 4.7: $P1, P2, PRP1, PRP2$. Figure 4.10 shows a five-round elastic network represented in this manner. In the figure, the RFi's are independently chosen pseudorandom permutations.

- Let $P1$ refer to the first round of G', including the swap step.
- Let $P2$ refer to the inverse of the last round of G', including the swap step that precedes the round function. *i.e.* $P2$ is the first round in G'^{-1} (the decryption direction).

- $P1$ and $P2$ are independently chosen permutations, because each RFi is a independently chosen psuedorandom permutations. The exact permutations used for $P1$ and $P2$ are unknown because they involve $RF1$ and $RF4$, respectively. $P1$ and $P2$ are not pseudorandom because they can be distinguished from a random permutation by using queries where the b bit portion of input is held constant and the y-bit portion is varied.

- Let $PRP2$ refer to the last four rounds of G'. $PRP2$ consists of all steps in G' after $P1$.

- Let $PRP1$ refer to the inverse of the first four rounds of G', excluding the swap step after the third round. $PRP1$ consists of all steps in G'^{-1} after $P2$.

$PRP1$ and $PRP2$ are PRPs on $b + y$ bits by Theorems 4.4 and 4.3. $PRP1 \neq PRP2^{-1}$. $P1$ and $P2$ are permutations on $b + y$ bits. By setting $PA = PRP2(P1(x))$ and $PB = PRP1(P2(x))$, $PB = PA^{-1}$. Therefore, by Claim 4.7 G', is a SPRP. □

4.8 $mb + y$ Bit SPRP

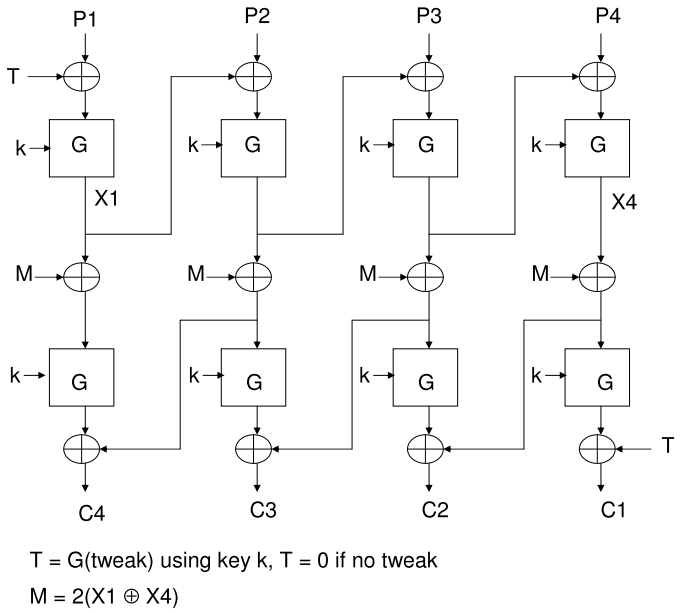

Figure 4.11: CMC Mode for 4b Bits (Halevi and Rogaway)

I can extend the use of the elastic network to create a $(mb + y)$-bit PRP and SPRP, where m is an integer ≥ 1, b is the block size of a block cipher, G, and $0 \leq y \leq mb$. Halevi and Rogaway created a mode of encryption that allows a mb-bit SPRP, for $m \geq 2$, to be created from a block cipher that is a PRP on b bits [HR03b]. The mode is referred to as CBC-Mask-CBC (CMC) and is shown in Figure 4.11.

The advantage an adversary has in distinguishing G in CMC mode from random when querying a black box which contains either G or a random permutation with adaptive chosen plaintext-ciphertext queries can be written in terms of the advantage an adversary has in distinguishing G from random with adaptive chosen plaintext - chosen ciphertext queries (*e.g.* the ability of an adversary to distinguish G in CMC mode from a strong random permutation is quantified in terms of the ability to distinquish G from a strong random permutation). Let $Adv_G(t_G, q_G, in_G + out_G)$ be the advantage of an adversary in distinguishing a block cipher, G, from random in time t_G when using q_G adaptive chosen plaintext - chosen ciphertext queries. in_G is the total number of bits input by the adversary and out_G is the total number of bits output by the adversary. in_G is an integer multiple of the block length b. Let $Adv_{G-CMC}(t_{CMC}, in_G, out_G)$ be the advantage of an adversary in distinguishing G from random in time t_{CMC} when the total number bits input by the adversary is in_G and the total number output by the adversary is out_G. I will abbreviate these advantages as Adv_G and Adv_{G-CMC}. Halevi and Rogaway proved that $Adv_{G-CMC} \leq 2 * Adv_G$ [HR03b].

Suppose G is a PRP. I bound the advantage in distinguishing a five-round elastic network that uses G in CMC mode as the round functions from a five-round elastic network with independently chosen random permutations as the round functions. I use the same method that was used in the proofs to Theorems 4.3 and 4.4. I define the following ten permutations:

- Let $PRP1, PRP2, PRP3, PRP4, PRP5$ be five independently chosen pseudorandom permutations.
- Let $RP1, RP2, RP3, RP4, RP5$ be five independently chosen random permutations.

Let Ni refer to a five-round elastic network in which the first i round functions are pseudorandom permutations and the remaining round functions are random permutations,

for $i = 0, 1, 2, 3, 4, 5$ defined as follows:

- Network 0 ($N0$): Each round function is a RP. The round functions are $RP1$, $RP2$, $RP3$, $RP4$ and $RP5$.
- Network 1 ($N1$): The first round function is the PRP. The second, through fifth round functions are RPs. The round functions are $PRP1$, $RP2$, $RP3$, $RP4$ and $RP5$.
- Network 2 ($N2$): The first two round functions are PRPs, and the third through fifth round functions are RPs. The round functions are $PRP1$, $PRP2$, $RP3$, $RP4$ and $RP5$.
- Network 3 ($N3$): The first three round functions are PRPs and the last two round functions are RPs. The round functions are $PRP1$, $PRP2$, $PRP3$, $RP4$ and $RP5$.
- Network 4 ($N4$): The first four round functions are PRPs and the last round function is a RP. The round functions are $PRP1$, $PRP2$, $PRP3$, $PRP4$ and $RP5$.
- Network 5 ($N5$): Each round function is a PRP. The round functions are $PRP1$, $PRP2$, $PRP3$, $PRP4$ and $PRP5$.

Let each PRP be an instance of G with a specific key. The keys are chosen randomly and independently of each other in order for the PRPs to be chosen independently of each other. The probability that network $N(i+1)$ can be distinguished from network Ni, for $i = 0, 1, 2, 3, 4$ is the probability that one round function consisting of G can be distinguished from a random permutation.

$|Pr(Ni) - Pr(N(i+1))| \leq Adv_G$, for $i = 0, 1, 2, 3, 4$.

$|Pr(N0) - Pr(N5)| =$

$|Pr(N0) - Pr(N1) + Pr(N1) - Pr(N2) + Pr(N2) - Pr(N3) + Pr(N3) - Pr(N4) + Pr(N4) - Pr(N5)|$

$\leq |Pr(N0) - Pr(N1)| + |Pr(N1) - Pr(N2)| + |Pr(N2) - Pr(N3)|$

$+|Pr(N3) - Pr(N4)| + |Pr(N4) - Pr(N5)|$

$\leq 5 * Adv_G$.

If G in CMC mode is used as the PRP instead of G, $|Pr(Ni) - Pr(N(i+1))| \leq 2 * Adv_G$, for $i = 0, 1, 2, 3, 4, 5.$ and $|Pr(N0) - Pr(N5)| \leq 10 * Adv_G$.

When the elastic network has four rounds (*i.e.*, the Ni's each have four rounds and $i = 0, 1, 2, 3, 4)$ then $|Pr(N0) - Pr(N4)| \leq 4 * Adv_G$ when the round functions are instances of G with randomly chosen keys and $\leq 8 * Adv_G$ when the round functions are instances of G with randomly chosen keys in CMC mode. In this case, the four-round elastic network with instances of G in CMC mode is a PRP in either the encryption or decryption direction.

This allows for the creation of variable-length PRPs and SPRPs using G and the elastic network. Inputs range from b to mb for any integer $m \geq 2$ for which $8 * Adv_G$ is negligible when using a four-round elastic network to create a PRP and for which $10 * Adv_G$ is negligible when using a five-round elastic network to create a SPRP. Let $\hat{m_8}$ be the largest value of m for which $8 * Adv_G$ is negligible. and let $\hat{m_{10}}$ be the largest value of m for which $10 * Adv_G$ is negligible. Use G in a four-round elastic network to create PRPs on b to $2b$ bits and use G in CMC mode in a four-round elastic network to create PRPs on $2b + 1$ to $2\hat{m_8}b$ bits. Similarly, variable-length SPRPs using G and a five-round elastic network can be created for inputs of b to $2\hat{m_{10}}b$ bits. Note, I only need G in CMC mode to be a PRP, not a SPRP. Therefore, it may be possible to increase the upper bound on $\hat{m_8}$ and $\hat{m_{10}}$ because Halevi and Rogaway based Adv_G and Adv_{G-CMC} on G being a SPRP as opposed to a PRP [HR03b].

4.9 Variable-Length PRPs and SPRPs Created from Fixed-Length PRFs

The elastic network can be combined with existing methods for creating fixed length PRPs to form variable-length PRPs and SPRPs, as long as the existing method allows for the creation of three or four independently selected PRPs. One such method is the creation of b-bit PRPs from $\frac{b}{2}$-bit PRFs using Feistel networks. Either the method described by Luby and Rackoff using a three-round Feistel network [LR88] or the method described by Naor and Reingold using a b-bit permutation and a two-round Feistel network [NR99] can be used to create the b-bit PRPs needed for the round functions in the elastic network.

For each round of the elastic network, select three $\frac{b}{2}$-bit PRFs independently of each other and independently of the PRFs used in previous rounds. This produces an elastic network with independently selected b-bit PRPs as the round functions and the results for three, four and five-round elastic networks apply, resulting in $(b + y)$-bit PRPs and SPRPs created from $\frac{b}{2}$-bit PRFs. In this case, nine PRFs are needed to created a variable-length PRP in the encryption direction and twelve are needed for the decryption direction. Fifteen PRFs are needed to create the variable-length SPRPs.

A variation of this construction is to replace the first round of each Feistel network with a randomly chosen b-bit permutation. Then, each round function of the elastic network consists of a b-bit permutation followed by a two-round Feistel network. The round function is a PRP, as proven by Naor and Reingold [NR99]. Three b-bit permutations and six $\frac{b}{2}$-bit PRFs, independently chosen, are required to create a three-round elastic network that is a PRP in the encryption direction. These values are four and eight, respectively for the decryption direction. Five b-bit permutations and ten $\frac{b}{2}$-bit PRFs, independently chosen, are required to create a five-round elastic network that is a SPRP.

For both variations, it may be possible to reduce the number of distinct PRFs required by determining the implications of using one or more of the PRFs in multiple rounds. Whether or not the same b-bit permutation can be used in multiple rounds of the second construction has not been investigated.

4.10 Summary

I have proven that a three-round elastic network is a PRP, the inverse of a four-round elastic network is a PRP and a five-round elastic network is a SPRP, when the round functions are independently chosen PRPs. These results allow for the creation of $(b + y)$-bit PRPs and SPRPs from b-bit PRPs, for $0 \leq y \leq b$. I also proved that these are the minimum number of rounds required and that the results do not hold when all of the round functions are identical. By combining the elastic network with the CMC mode of encryption, $2b$ to $2mb$-bit PRPs and SPRPs can be created from b-bit PRPs, for some $m \geq 2$, where m is limited by the (negligible) advantage an adversary has in distinguishing the b-bit PRPs

from a random permutation. By using Feistel networks as the round functions in the elastic network, $(b + y)$-bit PRPs and SPRPs can be created from $\frac{b}{2}$-bit PRFs, for $0 \leq y \leq b$.

Chapter 5

Security Analysis

5.1 Overview

In this chapter I analyze the security of an elastic block cipher independently of the specific block cipher used in the construction. First, I use a reduction to relate the security of the elastic version, G', to that of the original version, G. I prove that G' is secure against any attack that attempts to recover the key or the expanded-key bits if G is secure against the attack, under certain assumptions on the independence of the expanded-key bits in G'. This is accomplished by showing how to convert such an attack on G' to an attack on G. This result is important because it implies G' does not have to be analyzed against any practical attack to which G is immune. My approach is novel because I show how to convert an attack on the variable-length version of a block cipher directly into an attack on the fixed-length version of the block cipher. While Bellare and Rogaway, and Patel, *et al.,* were able to define the security of their variable-length block ciphers described in Chapter 2 in terms of the original, fixed-length, cipher, they did not provide a means by which to convert a practical attack on the variable-length version to an attack on the original cipher. Second, I consider linear cryptanalysis and, more generally, any algebraic attack. While these attacks are covered by the reduction method, I use them to provide an example of how a specific type of attack on G' can be converted into an attack on G. I show how any linear or any algebraic equations relating the plaintext, ciphertext and key bits for G' can be converted into equations for G without decreasing the probability with which a

specific equation holds. Therefore, any linear or algebraic attack on G' can be converted into such an attack on G, implying G' is secure against such attacks if G is secure against such attacks. Third, I discuss side channel attacks and differential fault analysis. Due to the elastic version of a block cipher reusing the round function of the original cipher, any type of side channel attack or differential fault analysis that exploits weaknesses or traits of the round function will also be possible on the elastic version of the cipher. Additional security analysis is provided in Chapter 7 when I discuss differential cryptanalysis and in Chapter 8, where I consider attacks which exploit the key schedules of ciphers and explain why they are not applicable to elastic block ciphers under my assumptions about the pseudorandomness of the expanded-key bits.

5.2 Reduction Between the Original and Elastic Versions of a Cipher

5.2.1 Scope

For any concrete block cipher used in practice (as opposed to a PRP in theory) the cipher cannot be proven secure in a theoretical sense (is not proven to be a PRP or SPRP) but rather is proven secure against known types of attacks. Thus, I can only do the same for the elastic version of such a cipher. In order to provide a general understanding of the security of elastic block ciphers, I provide a method for reducing the security of the elastic version to that of the original version, showing that a security weakness in G' implies a weakness in G. My security analysis of G' exploits the fact that there is an instance of G embedded in G'.

I show how to reduce G' to G in a manner that allows an attack which finds the key or round keys of G' to be used to find the round keys for G. Security against key recovery attacks does not by itself imply security (*e.g.,* the identity function which ignores the key is insecure while key recovery is impossible). However, all concrete attacks against real ciphers (linear, differential, higher order differential, impossible differential, related key attacks, *etc.*) attempt key (or expanded-key) recovery and thus practical block ciphers should be secure against such attacks. I note that if there is a relationship between the plaintext and

ciphertext bits that does not involve the key bits, this relationship would either manifest itself in the results of statistical tests on whatever versions of the block cipher (original and/or elastic) for which the relationship holds, and/or as algebraic equations relating the plaintext and ciphertext. In this latter case, any such equations that hold for the elastic version can be converted into equations for the original cipher as shown in Section 5.3.

In my analysis, I consider G' without the initial and final key-dependent mixing steps. This allows me to focus on the core components of the elastic block cipher algorithm. If present, the mixing steps only serve to increase the security of G' since they prevent an attacker from knowing with probability one which bits are omitted from the first application of the round function. Furthermore, since the mixing steps are added steps (as opposed to modifications to components of G) using key material that is independent of the round and whitening key bits, they do not impact my analysis.

5.2.2 Round-Key Recovery Attack

I use the fact that an instance of G is embedded in G' to create a reduction from G' to G. As a result of this reduction, an attack against G' that allows an attacker to determine some of the round keys implies an attack against G that is polynomially related in resources to the attack on G'. Assuming that G itself is resistant to such attacks, I conclude that G' is also resistant to such attacks. I note that if an attack finds the key as opposed to the expanded-key bits (the round keys) then the attacker can apply the key schedule to the key to obtain the round keys. Therefore, in my analysis, I view any key recovery attack as providing the round keys to the attacker. The reduction requires a set of (plaintext, ciphertext) pairs. This is not considered a limiting factor because in most types of attacks, whether they are known plaintext, chosen plaintext, adaptive chosen plaintext, chosen ciphertext *etc.,* the attacker acquires a set of such pairs.

In order to aid my analysis, I draw attention to the fact that the operations performed in G' on the leftmost b-bit positions in r consecutive rounds is an application of G. This is depicted intuitively in Figure 5.1. This relationship can be used to convert an attack which finds the round keys for G' to an attack which finds the round keys for G. Let G_{rk} denote G using round keys rk. Specifically, if ${G_k}'(p \parallel x) = c \parallel z$, a set of round keys, rk, for G

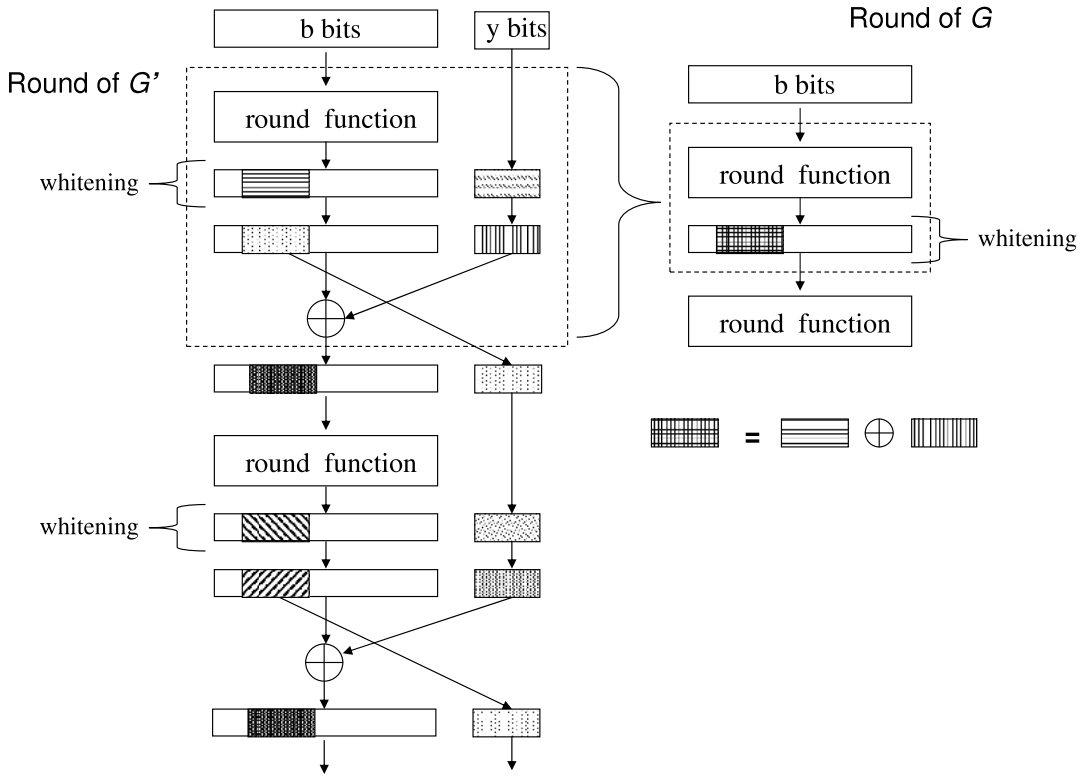

Figure 5.1: G within G'

such that $G_{rk}(p) = c$ can be formed from the round keys and the round outputs in G' by collapsing the end-of-round whitening and swapping steps in G' into a whitening step. The leftmost b bits of the round key for the initial whitening are unchanged, and the rightmost y bits are dropped. The resulting whitening key bits will vary in up to y positions across the (plaintext, ciphertext) pairs due to the previous round's output impacting the end-of-round whitening step. However, it is possible to use these keys to solve for the round keys of G, as shown Sections 5.2.3 and 5.2.4.

I state the following claim to assist the reader in understanding the linkage between G and G'. The claim shows that for any set of (plaintext, ciphertext) pairs encrypted under sets of round keys in G' where the rightmost y bits used for whitening in each round may vary amongst the sets and all other key bits are identical amongst the sets, there exists a corresponding set of (plaintext, ciphertext) pairs for G where the round keys used in G'

for the round function and the leftmost b bits of each whitening step are the same as those used in G, the plaintexts used in G are the leftmost b bits of the plaintexts used in G', and the ciphertexts for G are the leftmost b bits of output of the r^{th} round of G' prior to the swap step.

Claim 5.1. *Let G be a b-bit block cipher and G' be its elastic version. Let $\{(pi, ci)\}$ denote a set of n (plaintext, ciphertext) pairs such that $|pi| = |ci| = b$. Let $b+y$ be the variable block size for G' where $0 \leq y \leq b$. Under the following assumptions regarding the key schedules:*

- *The rightmost y bits of each whitening step in G' can take on any value and are independent of any other expanded-key bits within the round and in other rounds.*
- *No message-related expanded keys. Any expanded-key bits utilized in G depend only on the key and do not vary across plaintext or ciphertext inputs.*
- *Any expanded-key bits used in the round function of the r consecutive rounds of G' can take on the same values as the expanded-key bits used in the round functions of G.*
- *If G contains initial and end of round whitening, any expanded-key bits used for the leftmost b bits of each whitening step in r consecutive rounds of G' can take on the same values as the whitening bits in G.*

if $G_k(pi) = ci$ then there exists n sets of round keys for the first r rounds of G' that are consistent with inputs $pi \parallel w$ producing $ci \parallel vi$ as the output of the r^{th} round prior to the swap at the end of the r^{th} round, for $i = 1$ to n and $|w| = |vi| = y$, such that the leftmost b bits used for whitening in each round are identical across the n sets and any expanded-key bits used internal to the round function are identical across the n sets. There are no restrictions on the vi values.

Proof. Let $rk = \{rk_0, rk_1, ...rk_r\}$ be the set of round keys corresponding to key k for G. rk_0 denotes the key bits used for initial whitening. For (pi, ci), form a set of the first r round keys for G' as follows: Pick a constant string, w, of y bits, such as a string of $0's$. Let $pi \parallel w$ be the input to G'. Let $rki' = \{rki'_0, rki'_1, ...rki'_r\}$ denote the round keys for G' through the

r^{th} round for the pair (pi, ci). Set any bits in rki'_j used internal to the round function to be the same as the corresponding bits in rk_j. Set the leftmost b bits used for whitening in rki'_j to the b bits used for whitening in rk_j. Set the rightmost y bits used for whitening in rki'_j to be the same as the y bits left out of the round function in round j of G'. This is illustrated by Figure 5.2. Notice that the leftmost b bits used for whitening in each round are identical across the n sets, and any bits used internal to the round function are identical across the n sets; specifically, they correspond to rk in each case, and the rightmost y bits used in each whitening step differ based on (pi, ci) across the n sets. The case in which G does not contain whitening steps corresponds to using 0's for the leftmost b bits of each whitening step in G'.

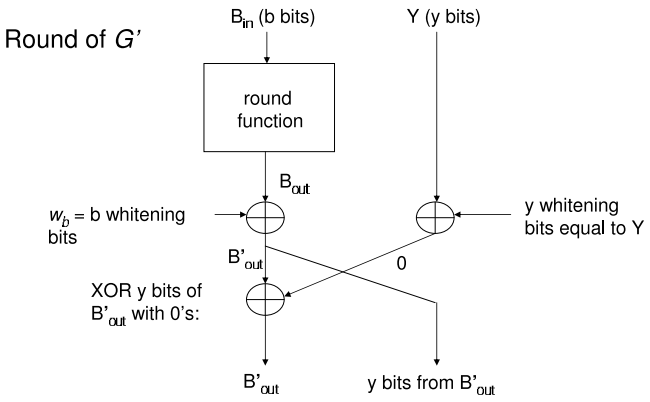

Figure 5.2: Converted Key Unchanged in b Whitening Bits

The operations of G' on the leftmost b bits of rounds 1 through round r, prior to the last swap, are identical to the operations in $G_k(pi)$ because the swap step in G' results in XORing y bits of a round function's output with y $0's$. Therefore, the leftmost b bits output from the r^{th} round prior to the swap in the r^{th} round is ci. Therefore, for $i = 1$ to n there exists a set of round keys, rki' for $G'_{rki'}$ such that $G'(pi)$ produces ci as the leftmost b bits in the r^{th} round prior to the swap step, thus proving Claim 5.1. □

Theorem 5.1. *Given a fixed-length block cipher, G, that works on b-bit blocks and its elastic version, G', that works on $(b+y)$-bit blocks, where $0 \leq y \leq b$, if there exists an attack, $A'_{G'}$, on G' that allows the round keys to be determined for r consecutive rounds of G' using polynomial (in b) time and memory, then there exists an attack on G with r rounds that finds the round keys for G and that uses polynomial (in b) many resources as $A'_{G'}$, assuming:*

- *No message-related expanded keys. Any expanded-key bits utilized in G depend only on the key and do not vary across plaintext or ciphertext inputs.*
- *An attack on r' rounds of G' implies a reduced-round attack on r rounds of G' for $r \leq r'$.*
- *$A'_{G'}$ finds all possible sets of round keys, if more than one set exists.*
- *Any expanded-key bits used in the round function of the r consecutive rounds of G' can take on the same values as the expanded-key bits used in the round functions of G.*
- *If G contains initial and end of round whitening, any expanded-key bits used for the leftmost b bits of each whitening step in r consecutive rounds of G' can take on the same values as the whitening bits in G.*

Proof. Sections 5.2.3 and 5.2.4 contain two different proofs of the theorem. The proof in Section 5.2.3 is applicable to all values of y where $0 \leq y \leq b$ and the proof in Section 5.2.4 works for values of y satisfying $r(y-2) < b$. □

The first assumption is typical of existing block ciphers and is true of the elastic versions of block ciphers, which use a stream cipher as the key schedule. The second assumption is true of block ciphers used in practice. The last two assumptions mean that the key schedule of G' is defined such that a subset of the expanded-key bits can have the same values as if they were generated by the key schedule of G. These assumptions are easily satisfied in practice by using the key schedule of G to generate a subset of the round key bits and a separate algorithm to generate the expanded-key bits required in G' for the additional $r'-r$

rounds and any whitening present in G' that is not present in G. Another option is if the key schedule of G' generates pseudorandom expanded-key bits such that it is possible the expanded-key bits for the round function and leftmost b bits of whitening in r consecutive rounds can take on the same values generated by the key schedule of G. In practice, given an expanded-key, it is feasible to check if the expanded-key adheres to a specific block cipher's key schedule. A subset of the expanded-key bits being tested can be inserted into the key schedule to generate additional key bits which can be checked against the bits in the value being tested. Refer to Chapter 8 for further discussion on the structure of key schedules in existing block ciphers.

The theorem holds by default for the case when $y = 0$, since G' is just G (with the possible addition of whitening which can be set to 0's when applying the attack if G does not contain whitening). I view G as having whitening steps in the proof to Theorem 5.1. This is not an issue for the following reasons. If the attack on G' involves solving for the round key bits directly and allows the bits used in the whitening steps to be set to 0 for bit positions not swapped and to 0 or 1, as necessary, for bit positions swapped, to ensure the whitening on the leftmost b bits is equivalent to XORing with 0, which is the same as having no whitening in G. If the attack on G' finds all possible keys or sets of round keys, the attack must find the key(s) or set(s) of round keys corresponding to round keys that are equivalent to XORing with 0. Setting a subset of bits in each whitening step in G' to 0's is equivalent to using a weaker version of G'. Any attack that works on G' will work on the weaker version. This is merely the case where the attacker knows certain bits of each whitening step are 0's.

I note that Theorem 5.1 only states that an attack on G' can be converted to an attack on G and not the reverse. This is because, in general, the claim that an attack on G can be converted into an attack on G' does not hold. Consider the case when G contains the initial and end of round whitening steps. When $y = 0$, G' is G with the initial and final key-dependent permutations added and the key schedule replaced (such as by a stream cipher). If the attack on G is due to the original key schedule, the attack does not necessarily hold if the key schedule is changed to generate pseudorandom bits when creating G'. For any attack not due to the key schedule, the attack must be such that the addition of the two key-

dependent permuations, the addition or expansion of the whitening steps and the swapping of bits does not result in the attack no longer being applicable or the attack being infeasible computationally in order to say an attack on G implies an attack on G'. In general, the conversion of an attack from G' to G works because there is a decrease in the complexity of the block cipher being attacked when going from G' to G; whereas, the reverse is not true because there is an increase in the complexity of the block cipher when converting G to G'.

To prove Theorem 5.1, I must show for any value of y that if an attack exists on G' it can be converted into an attack on G using polynomial time and memory. I define the steps for converting a round-key recovery attack on G' to an attack on G. I describe two ways of performing the conversion. The first works for any value of y. The second requires fewer computations than the first method, but is only guaranteed to work for small values of y in relation to b. The methods described here treats whitening key bits as if they are pseudorandom in that the whitening key bits can take on any value. In G, if there is a relationship amongst the whitening key bits and/or between whitening key bits and key material used within the round function due to the key schedule of G, such keys will be a subset of all the possible sets of round keys found using the attack on G'. Then the set of round keys that satisfies the key schedule of G can be determined by checking which of the potential keys corresponds to the key schedule. If the number of potential sets of round keys found by the attack on G' is large enough such that it is computationally infeasible to determine which ones adhere to the key schedule of G, then the attack on G' is not computationally feasible. This is because the number of potential sets of round keys it finds for a set of (plaintext, ciphertext) pairs will also be large enough such that it is computationally infeasible for an attacker to determine which set to use to decrypt additional ciphertexts.

In both methods, when I refer to converting the round keys of G' into round keys for G, I mean the following: In round j of G', let b_{jl} denote the l^{th} bit of the b bits output from the round function prior to the end of round whitening. Let kw_{jl} denote the end of round whitening key bit applied to b_{jl}. If b_{jl} is involved in the swap step at the end of round j, let y_{jh} denote the bit from the rightmost y bits with which b_{jl} is swapped and let kw_{jh} denote the whitening key bit applied to y_{jh}. Set the l^{th} whitening bit in round j of

G to $kw_{jl} \oplus kw_{jh} \oplus y_{jh}$ when $j \geq 2$. This conversion of whitening bits is illustrated in the example in Figure 5.3. When $j = 1$, the l^{th} whitening bit is set to $kw_{1l} \oplus kw_{1h} \oplus y_{1h} \oplus kw_0 h$ because the initial whitening is included in the conversion. Set all other key bits used in G (both whitening and any internal to the round function) to be identical to the key bits used in G'. I refer to the initial whitening as round 0. The initial whitening for G' is converted to initial whitening for G by using the leftmost b expanded-key bits of the initial whitening as the initial whitening in G.

5.2.3 Round-Key Recovery Attack: First Method

I describe here a method for converting the attack on G' to an attack on G. Without loss of generality, I use the first r rounds of G' as the r consecutive rounds for which the round keys are found. The attacks are presented in terms of solving for the round keys from the initial whitening to round r, but may also be performed by working from round r back to the initial whitening or by using any consecutive r rounds with whitening applied before the first round as long as the plaintext for G is the leftmost b bits of input to the r rounds and the corresponding ciphertext from G is the leftmost b bits of the output of the r rounds.

This attack runs in quadratic time in the number of rounds of G. The attack, $A'_{G'}$, on G' is used to solve for round keys 0 and 1 for G, then repeatedly solves for one round key of G at a time, using the output of one round of G as partial input to a reduced round version of G', running the attack on G' and converting the 1^{st} round key of G' to the round key for the next round of G. By the second condition in Theorem 5.1, an if an attack on G' with r' rounds exists, then a reduced round attack on G' exists for any number of rounds $< r'$.

Let P be a set of plaintexts and C be a set of ciphertexts. I use the notation $\{(P, C)\}$ to indicate a set of plaintext,ciphertext pairs of the form (pi, ci) with $pi \in P$ and $ci \in C$. Given a set $\{(P^*, C^*)\} = \{(pi^*, ci^*)\}$ of n (plaintext, ciphertext) pairs for G, create a set $\{(P, C)\} = \{(pi^* \parallel 0, ci^* \parallel vi_r)\}$ of n (plaintext, ciphertext) pairs for an r-round version of G'. Note: I only require that the y bits appended to each pi^* when forming $\{(P, C)\}$ be a constant; I choose to use 0. The vi_r values appended to the $ci's$ are arbitrary and do not need to be identical. The r subscript in vi_r denotes the number of rounds. My method runs reduced round attacks on G' and the vi_r's can vary each time. Solve G' for round

keys 0 and 1. By the pseudorandomness of the round keys described in the definition of elastic block ciphers, sets of round keys exist that correspond to $\{(P, C)\}$ and which are identical in at least the first two rounds (the round keys across all n pairs may be identical in additional rounds, but I am only concerned with the first two rounds at this point in the process). Denote these as rk'_0 and rk'_1. Use the leftmost b bits of rk'_0 as round key 0, rk_0, for G. Since the rightmost y bits are identical across all inputs to G', when rk'_1 is converted to a round key for G, the result will be the same across all n elements of $\{(P, C)\}$. Use the converted round key as round key 1, rk_1, for G. For each pi^*, apply the initial whitening and first round of G using the two converted round keys. Let $p1i$ denote the output of the first round of G for $i = 1$ to n. Using a reduced round version of G' with $r - 1$ rounds and the initial whitening removed, set $\{(P, C)\} = \{(p1i \parallel 0, ci^* \parallel vi_{r-1})\}$ and solve for the first round key of G'. As before, convert the resulting round key for the first round to a round key for G. Use the converted round key as the second round key for G. Repeat the process for the remaining rounds of G, each time using the outputs of the last round of G for which the round key has been determined as the inputs to G' and reducing the number of rounds in G' by 1, to sequentially find the round keys for G.

This attack involves applying each round of G to n inputs for a total of rn rounds of G. $\frac{n(r+1)r}{2}$ rounds of G' are computed in the worst case if $A'_{G'}$ requires knowing the output of each round of the reduced round version of G' to find the first round key. r applications of $A'_{G'}$ are needed on the reduced round versions of G'. Let t_A denote the time to run $A'_{G'}$. Let ks_t be the time to check that an expanded-key found by $A'_{G'}$ adheres to the key schedule of G. The time to attack G is $O(nr^2 + rt_A + ks_t)$.

In summary, the attack on G can be written as:

Input $\{(P^*, C^*)\} = \{(pi^*, ci^*)$ for $i = 1$ to $n\}$.

Create $\{(P, C)\} = \{(pi^* \parallel 0, ci^* \parallel vi_r)$ for $i = 1$ to $n\}$ for a r-round version of G',

where the $vi's$ are arbitrary.

Using $A'_{G'}$, solve a r-round version of G' for rk'_0 and rk'_1.

Convert rk'_0 to rk_0 and rk'_1 to rk_1.

Set $p1i =$ first round output of G using rk_0 and rk_1, for $i = 1$ to n.

For $j = 1$ to $r - 1$ {

$\{(P, C)\} = \{(pji \parallel 0, ci^* \parallel vi_{r-j}) \text{ for } i = 1 \text{ to } n\}$.

Solve a $r - j$ reduced round version of G' for the first round key, rk'_1.

Convert rk'_1 to form rk_{j+1}.

$p(j + 1)i =$ output of round $j + 1$ of G on pji using rk_{j+1} for $i = 1$ to n.

}

Another method for proving Theorem 5.1 is presented in the next section. It requires fewer computations than the method just described. When given the round keys for G' which correspond to n (plaintext, ciphertext) pairs, the method described here produces round keys for G which correspond to n (plaintext, ciphertext) pairs. Whereas, the alternative method described in the next section requires $2^{y(r-2)}n$ (plaintext, ciphertext) pairs to guarantee the resulting round keys will correspond to at least n (plaintext, ciphertext) pairs. However, because the second method requires fewer computations, it is useful when y is small relative to b.

5.2.4 Round-Key Recovery Attack: Second Method

My second method for proving Theorem 5.1 requires fewer computations than the first method, but provides rounds keys for a smaller set of (plaintext, ciphertext) pairs. The attack works as follows: Assume there exists a known (plaintext, ciphertext) pair attack on G' which produces the round keys either by finding the original key and then expanding it, or by finding the round keys directly. Using round keys for rounds 0 to r of G', convert the round keys into round keys for G one round at a time. For each round, extract the largest set of (plaintext, ciphertext) pairs used in the attack on G' that have the same converted round key. If there are n_j (plaintext, ciphertext) pairs involved at round j, there will be at least $\frac{n_j}{2^y}$ pairs remaining for which the round keys are consistent after round j. The end result is a set of round keys for G that are consistent with a set of $\frac{n}{2^{y(r-2)}}$ b-bit (plaintext, ciphertext) pairs for G. I then describe how to take a set of (plaintext, ciphertext) pairs for G, convert them into a set of (plaintext, ciphertext) pairs for G' in order to run the attack on G' to find the round keys for G.

Let $\{(P, C)\} = \{(pi \parallel xi, ci \parallel zi)\}$ (for $i = 1$ to n) denote a set of n known $(b + y)$-bit (plaintext, ciphertext) pairs for G', where $|pi| = |ci| = b$ and $|xi| = |zi| = y$.

Let $A_{G'}$ be an attack on G' that finds the key(s) corresponding to $\{(P, C)\}$ in time less than a exhaustive search for the key. Let m denote the number of keys found. Without loss of generality, it is assumed the keys are available in expanded form. Let k_j denote the j^{th} key found by $A_{G'}$. In practice, only one key should be found for any set of (plaintext, ciphertext) pairs.

Let $S = \{ek_j\}$ for $j = 1$ to m be the set of expanded-keys used for whitening for which ek_j is from the expansion of key k_j and $G'_{k_j}(pi \parallel xi) = ci \parallel zi$ for $i = 1$ to n.

Let R_{int} denote any key material utilized within the round function. The values found for such key bits will be the same for G' and G.

Let $\{(P, U)\} = \{(pi||xi, ui||vi)\}$ such that $ui||vi$ is the output of the r^{th} round of G', where $|ui| = b$ and $|vi| = y$.

Let $S' = \{ek'_j \mid ek'_j =$ bits of $ek_j \in S$ corresponding to rounds 0 to r used for whitening $\}$ be the set of expanded-key bits used for whitening in rounds 0 to r of G'.

For each $ek_j \in S'$ and each $(pi \parallel xi, ui \parallel vi) \in \{(P, U)\}$, convert the round keys to round keys for G. Let ek'_{ij} be the converted key corresponding to the i^{th} element of $\{(P, U)\}$ and the j^{th} element of S'. The part of ek'_{ij} corresponding to round 0 will be identical across all elements. When the round keys are converted, at most y bits change in the leftmost b bits. Thus, the resulting round keys for round q, $1 \leq q \leq r$ can be divided for each of the y impacted bits into those that have a 0 in the affected bit and those that have a 1 in the affected bit. For $q = 1$ to r, define S'_{rnd_q} as the maximum-sized set of $ek'_{ij}s$ from $S_{rnd_{q-1}}$ that have identical round key(s) for round q, where $S'_{rnd_0} = S'$. Let $\{(P, U)_{rnd_q}\}$ be the corresponding elements of $\{(P, U)\}$. When forming $\{(P, U)_{rnd_q}\}$, at least $2^{-y}|\{(P, U)_{rnd_{q-1}}\}|$ of the elements from $\{(P, U)_{rnd_{q-1}}\}$ are included.

To illustrate how the sets S'_{rnd_q} and $\{(P, U)_{rnd_q}\}$ are created, consider the example shown in Figure 5.3 where $b = 4$, $y = 2$, and the leftmost 2 bits are swapped with the y bits in the swap step. The round number is q and $\{(P, U)_{rnd_{q-1}}\}$ contains three (plaintext, ciphertext) pairs. Suppose the outputs of the round function in the q^{th} of G' are $100101, 110011$ and 111111 and the whitening bits in the q^{th} round are 011010. The converted round keys corresponding to the three cases are $0110, 1110$ and 1110. Since 1110 occurs in the majority of the cases, set the q^{th} round key of G to 1110. S'_{rnd_q} contains the round keys for rounds 0

to $q-1$ from $S'_{rnd_{q-1}}$ and 0010, and $\{(P,U)_{rnd_q}\}$ contains the second and third (plaintext, ciphertext) pairs from $\{(P,U)_{rnd_{q-1}}\}$.

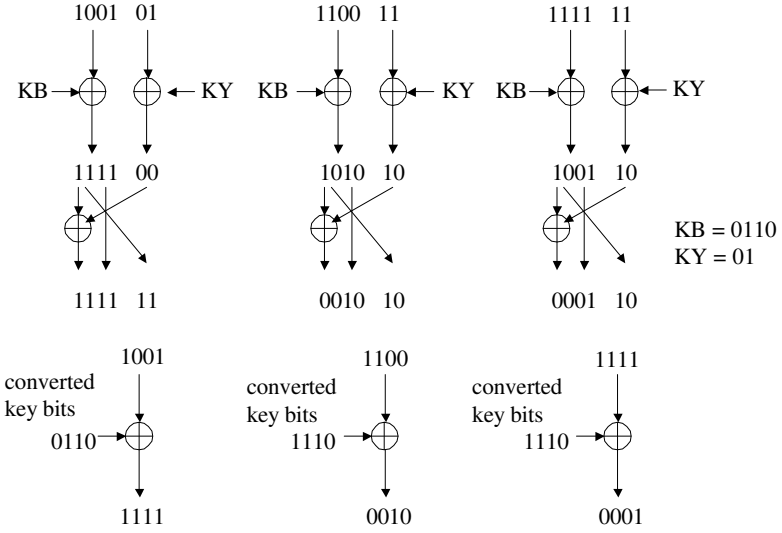

Figure 5.3: Forming S'_{rnd_q}

Let $\{(P,C)_G\} = \{(pi,ci)|(pi \parallel yi, ui \parallel vi) \in \{(P,U)_{rnd_r}\}\}$. $|\{(P,C)_G\}| \geq n/2^{yr}$. $\{(P,C)_G\}$ is a set of (plaintext, ciphertext) pairs for which $G_{rk}(pi) = ci \ \forall \ (pi,ci) \in \{(P,C)_G\}$ with the whitening round keys of $rk \in S'_{rnd_q}$ and any additional key material utilized by the rounds is the same as that for G', namely R_{int}.

To perform the attack on G when given a set of (plaintext, ciphertext) pairs for G, convert the pairs into a set of (plaintext, ciphertext) pairs for G' and find the round keys for G' then for G as follows: Given a set $\{(P^*,C^*)\} = \{(pi^*,ci^*)\}$ for $i = 1$ to n known (plaintext, ciphertext) pairs for G, create the set $\{(P,C)\}$ of (plaintext, ciphertext) pairs to use in the attack on an r-round version of G' by setting $pi \parallel xi = pi^* \parallel 0$ and $ci \parallel zi = (ci^* \parallel zi)$ for $i = 1$ to n. For the set of (P,C) pairs are created, $\{(P,U)\} = \{(pi^* \parallel 0, ci^* \parallel zi)\}$. Apply the attack on G' to solve for the round keys of G' then produce the sets $\{(P,U)_{rnd_r}\}$ and S_{rnd_r}. The sets of round keys in S_{rnd_r} will be consistent with the (plaintext, ciphertext) pairs in $\{(P,U)_{rnd_r}\}$. A set of round keys that adheres to the key schedule of G will be found by Claim 5.1 and the assumption that the attack on G' finds all possible sets of round

keys.

Let t_r be the time to run r rounds of G', t_A be the time to run $A_{G'}$ and m be the number of keys (sets of round keys) found by $A'_{G'}$. In the case of obtaining at least one set $\{(P,U)_{rnd_r}\}$ of size $\geq \frac{n}{2^{yr}}$, the time required beyond t_A consists of: nmt_r time to obtain the outputs of the first r rounds for each $\{(P,U)\}$, $O(nmr)$ time to perform the conversion of the round keys from G' to round keys for G and $O(nmr)$ time to form the S'_{rnd_r} sets. Let ks_t be the time to check that an expanded-key adheres to the key schedule of G. Thus, the additional time required to attack G (beyond the time required to attack G'_{b+y}) is $O(nm(r+t_r)+mks_t)$. The only unknown value is m, the number of keys produced by the attack on G'_{b+1}. If m is large enough, to the extent that it approaches the average number of keys to test in a brute force attack on G', then this contradicts the assumption that an efficient attack exists on G' because the attacker is left with a large set of potential keys for decrypting additional ciphertexts.

I have defined a method which produces a set of at least $\frac{n}{2^{yr}}$ (plaintext, ciphertext) pairs which are consistent with the round keys. This lower bound on the number of plaintext, ciphertext pairs can be slightly increased to $\frac{n}{2^{y(r-2)}}$ by using $(b+y)$-bit plaintexts that are the same in the rightmost y bits (which I did by setting these bits to 0), and by defining the ui values representing the ciphertext output of G in the r^{th} round of G' to be the output of the r^{th} round prior to the swapping step. This will result in $|S'_{rnd_1}| = n$ and $|S'_{rnd_r}| = |S'_{rnd_{r-1}}|$, thus in first and r^{th} rounds the set of (plaintext, ciphertext) pairs is not reduced. The number of (plaintext, ciphertext) pairs produced for G that are consistent with the round keys for G is $\geq \frac{n}{2^{y(r-2)}}$. The number of possible plaintexts for G is 2^b; therefore, it is necessary for $y(r-2) < b$ to use this method. The first proof method overcomes this restriction at the cost of increased computation.

5.3 Linear Cryptanalysis

In order to demonstrate the security of elastic block ciphers against a specific type of attack and provide a more detailed analysis than the general relationship between the security of the elastic and original versions of a cipher presented in Section 5.2, I consider linear

cryptanalysis and differential cryptanalysis of elastic block ciphers. These are two of the most fundamental attacks and bounds for these attacks are typically provided for most block ciphers. In the approach used for the differential cryptanalysis presented in Chapter 7, I analyze specific elastic block ciphers and compute the actual bounds for its differential characteristic probabilities based on the bounds for the original cipher. In this section, I consider linear attacks and algebraic attacks on elastic block ciphers in general. I prove that any practical linear or any algebraic attack on an elastic block cipher, G', can be converted into a polynomial time related attack on the original cipher, G, independently of the specific block cipher used for G.

Linear cryptanalysis involves finding linear equations (equations involving XORs) relating plaintext, ciphertext and key (usually expanded-key) bits that hold with probability $\frac{1}{2} + \alpha$ for non-negligible α, $0 < \alpha \leq \frac{1}{2}$. I note that any equation that holds with probability $< \frac{1}{2}$ can be converted into an equivalent equation that holds with probability $> \frac{1}{2}$. By inserting the bits from known (plaintext, ciphertext) pairs into the equations, the attacker can determine (expanded) key bits based on the values for the key bits most often indicated by the equation. For an equation to be useful in an attack, α must be large enough such that it is computational feasible to obtain and insert enough plaintext, ciphertext pairs into the equation to obtain a statistically significant result. Once a few expanded-key bits are found, it may be possible to insert them into the cipher's key schedule (based on existing key schedules) to find additional expanded-key bits. Or, if the cipher is poorly designed such that most key bits are obtained from the equations, the remaining key bits may be found by an exhaustive search.

I show that a linear relationship across r rounds of G' implies such a relationship across r rounds of G. If any such linear relationship holds with a probability such that fewer than $2^{(b-1)}$ (plaintext, ciphertext) pairs are required for an attack, then G is subject to a linear attack that requires fewer plaintexts, on average, than an exhaustive search over all plaintexts. Whether or not using the equations is computationally feasible depends on number of (plaintext, ciphertext) pairs and the number of equations that must be computed. If at least $2^{(b-1)}$ plaintext, ciphertext pairs are required for an attack on r rounds of G', then either the attack is infeasible on r rounds of G' from a practical perspective or G is

subject to a brute force attack in practice. Note that I am dealing with an attack on only r rounds of G' and the probability of a linear relationship holding across $r' = r + \lceil \frac{ry}{b} \rceil$ rounds of G' will be less than that for r rounds. More specifically, if the attack on G' involves a maximum correlation between plaintext, ciphertext and key bits which occurs with probability $\leq 2^{-b}$ on r rounds (thus requiring in practice $\geq 2^b$ plaintexts), then an attack on $2r$ rounds involves a maximum correlation that occurs with probability $\leq 2^{-2b}$ and requires $> 2^{2b}$ plaintexts. In this case, G' is practically secure against a linear attack when $\lceil \frac{ry}{b} \rceil = r$ regardless of the computational requirements.

Without loss of generality, I assume any linear relationship involves the expanded-key bits as opposed to the original key input to the key schedule. I omit the initial and final key-dependent permutations from my analysis because these permutations do not impact any linear relationship that exists across r rounds of G'. A direct implication of the result is that if G' is subject to an attack using any algebraic equations, as opposed to just linear equations, then so is G. From these results, I can conclude that an elastic version of AES is not subject to a practical linear attack. This is because AES is not subject to such an attack based on the linear trails analysis ([DR99], pages 30-31) and the 128-bit block size and 128-bit key size is not subject to an exhaustive search in practice. Also the elastic version of AES is subject to an algebraic attack only if AES is subject to such an attack.

Claim 5.2. *Given a block cipher G with a block size of b bits and r rounds, and its elastic version G' with a block size of $b + y$ bits for $0 \leq y \leq b$ and r' rounds where $r' = r + \lceil \frac{yr}{b} \rceil$, if G' is subject to a linear attack on r rounds then either G is subject to a linear attack or the resources exist to perform an exhaustive search on G over all plaintexts, under the following assumptions regarding the key schedules:*

- *The rightmost y bits of each whitening step in G' can take on any value and are independent of any other expanded-key bits within the round and in other rounds.*
- *No message-related expanded keys. Any expanded-key bits utilized in G depend only on the key and do not vary across plaintext or ciphertext inputs.*
- *Any expanded-key bits used in the round function of the r consecutive rounds of G' can take on the same values as the expanded-key bits used in the round functions of*

G.

- *If G contains initial and end of round whitening, any expanded-key bits used for the leftmost b bits of each whitening step in r consecutive rounds of G' can take on the same values as the whitening bits in G.*

Proof. I first note that if the linear attack on r rounds of G' requires at least 2^b (plaintext, ciphertext) pairs then either the attack is computationally infeasible or G is insecure independent of the attack (since the attacker has the resources to encrypt 2^b plaintexts). Therefore, it can be assumed that the attack on G' requires $< 2^b$ (plaintext, ciphertext) pairs. To understand how a linear relationship (if one exists) between the plaintext, ciphertext and expanded-key bits is determined for G', I first consider how a linear relationship is derived for a block cipher structured as a series of rounds with block length b and then add the impact of the whitening and swap step to these relationships. I number the rounds from 1 to r. I will refer to any initial whitening step that occurs prior to the first round as round 0 and the round function of round 0 is just the initial whitening. The relationship between the output of the j^{th} round and the input to the $(j+1)^{st}$ round is depicted in Figure 5.4 for both G and G'.

I use the following notation:

- Two bits, $x1$ and $x2$, cancel each other in an equation means $x1 \oplus x2 = 0$ with probability 1.
- Let u_{ji} denote the i^{th} bit of the input to the round function in round j, $1 \leq i \leq b$, $0 \leq j \leq r$.
- Let v_{ji} denote the i^{th} bit of the output from the round function in round j, $1 \leq i \leq b$, $0 \leq j \leq r$.
- Let n_j denote the number of expanded-key bits used in the round function in round j, $0 \leq j \leq r$. This does not include any end of round whitening added to form G', but does include the end of round whitening if it is part of the round function of G (as is the case with AES). The round function in round 0 is the identity function and $n_0 = 0$ if G does not contain initial whitening.

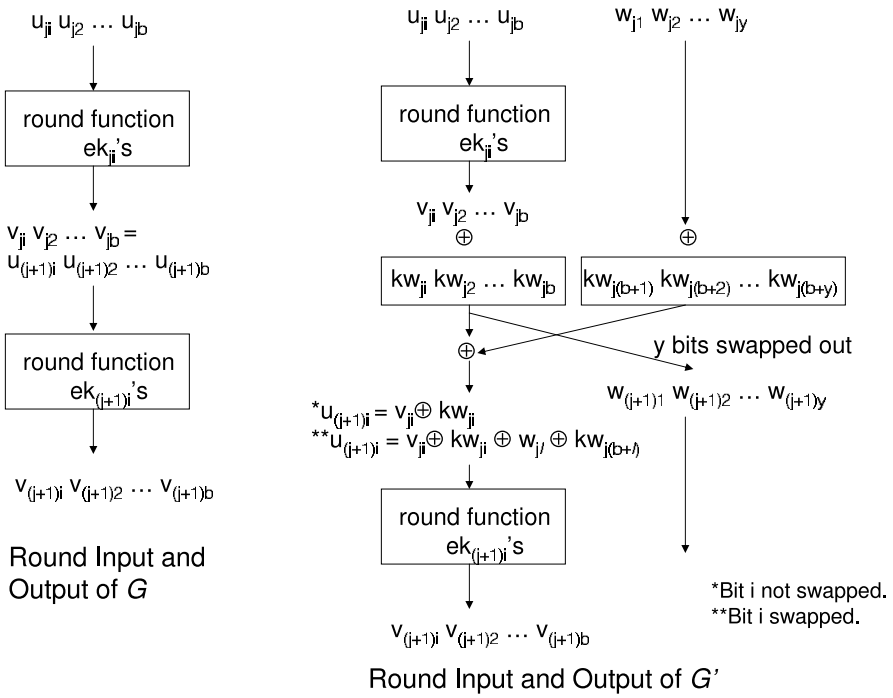

Figure 5.4: Linear Relationship Between Round j's Output and Round $(j+1)$'s Input

- Let ek_{ji} denote the i^{th} expanded-key bit in the round function in round j, $1 \leq i \leq n_j$.
- Let $L_j([u_{j1}, ...u_{jb}] \oplus [v_{j1}, ...v_{jb}] \oplus [ek_{j1}, ...ek_{jn_j}])$ denote the set of linear equations (if any) relating the input, output and round key bits with non-negligible probability for the round function in round j, $0 \leq j \leq r$. I will abbreviate this as L_j. An equation in L_j holds with probability $\frac{1}{2} + \alpha$ for some non-negligible α such that $0 < \alpha \leq \frac{1}{2}$. For example, if $u_{12} \oplus v_{13} \oplus ek_{15} = 0$ with probability 0.75, this equation will be in L_1. Any equation which reflects a negative relationship, meaning the equation holds with probability $\frac{1}{2} - \alpha$, is rewritten as an equation holding with probability $\frac{1}{2} + \alpha$.
- Without loss of generality, the equations in L_j are in reduced form; for example, $u_{j2} \oplus u_{j2} \oplus u_{j2} = 1$ will be reduced to $u_{j2} = 1$.
- Internal variables for r rounds plus any initial whitening (round 0) will refer to the set of u_{ji} for $1 \leq j \leq r$ and v_{ji} for $0 \leq j \leq r-1$, with $1 \leq i \leq b$. *i.e.,* the round

inputs and outputs that are not the plaintext or ciphertext.

A linear relationship across consecutive rounds is obtained by combining the linear equations for each of the rounds, with v_{ji} becoming $u_{(j+1)i}$. A linear relationship exists that involves only plaintext, ciphertext and expanded-key bits if the intermediate round inputs and outputs cancel when combining the per round equations, leaving equation(s) involving only u_{0i}'s, v_{ri}'s and expanded-key bits.
For example, if in G with two rounds:

$$u_{11} \oplus v_{12} = ek_{11}$$

and

$$u_{22} \oplus v_{26} = ek_{23}$$

then since $v_{12} = u_{22}$

$$u_{22} = u_{11} \oplus ek_{11}$$

and

$$u_{11} \oplus ek_{11} \oplus v_{26} = ek_{23}$$

I now consider how the steps between the rounds in G' impact the linear relationships across the rounds.

- Let Y denote the rightmost y bits of the data block for a $(b + y)$-bit data block.
- Let Γ' refer to the set of the equations used in a linear attack on r rounds of G' formed from combining the L_j's for the individual rounds along with the end of round whitening and swap steps.
- Let Γ refer to a set of linear equations for G formed from equations in Γ'.
- Let kw_{ji} denote the i^{th} key bit used for the whitening step added in round j when constructing G', $1 \leq i \leq b + y$ and $1 \leq j \leq r$. $kw_{ji} = 0$ for $1 \leq i \leq b$ if the round function of G includes end of round whitening and $kw_{0i} = 0$ for $1 \leq i \leq b$ if G contains initial whitening because G' does not add whitening to the b bits when it is already present.

- Let w_{jl} denote the l^{th} bit of the Y portion of the data, for $1 \leq l \leq y$ and $2 \leq j \leq r$. $w_{jl} = v_{(j-1)h} \oplus kw_{(j-1)h}$ where $1 \leq h \leq b$ and h is the bit position swapped with bit position l in the previous swap. When $j = 1$, $w_{1l} = w_{0l} \oplus kw_{0(b+l)}$, the initial input bit XORed with the initial whitening applied.

With the addition of the whitening and swap steps, the input to the round function is now defined as:

- $u_{(j+1)i} = v_{ji} \oplus kw_{ji}$ when v_{ji} is not involved in the swap step.
- $u_{(j+1)i} = v_{ji} \oplus kw_{ji} \oplus w_{jl} \oplus kw_{j(b+l)}$ when v_{ji} is involved in the swap step. When $j \geq 2$, this can be written as $u_{(j+1)i} = v_{ji} \oplus kw_{ji} \oplus v_{(j-1)h} \oplus kw_{(j-1)h} \oplus kw_{j(b+l)}$.

Notice that the steps between applications of the round function in G' maintain a linear relationship between the output of one round and the input of the next round.

Recall that the key schedule of G' produces whitening bits which are created independently of the key bits used within the round function (to the extent that the key bits are pseudorandom), and of the round function's input and output. Therefore, these whitening bits will cancel with any v_{ji}, u_{j+1} and/or ek_{ji} with probability $\frac{1}{2} + e$ for negligible e (*i.e.,* there is no discernable relationship between these whitening bits and any of the plaintext, ciphertext and expanded-key bits used internal to the round function by definition of the key schedule). Thus, the kw_{ji}'s added when forming G' will not increase the probability of a linear relationship between plaintext bits, ciphertext bits and expanded-key bits used in the round function. Even if a different key schedule is used that does not guarantee independence amongst the kw_{ji}'s and that results in cancellation among some kw_{ji}'s, this is merely cancelling variables that are not present in the linear equations for the round function and thus will not simplify the equations or increase the probability that an equation holds across r applications of the round function.

Now I assume a set of equations, Γ', exist for G' that contains no internal variables and show how to convert them to a set of equations for G. Given the sets, L_j's, of linear equations for the round function in G', these same sets of equations hold for G because the elastic version does not alter the round function. These equations are combined across rounds as was done for G', except now when forming the input to one round from the output

of the previous round, the impact of the swap step and any whitening added when forming G' is removed as follows:

- Set kw_{ji} to 0 for $0 \leq j \leq r$ and $1 \leq i \leq b$ so these whitening bits are omitted from the resulting equations. This removes any initial and end of round whitening that was added to the leftmost b bits when forming G'. Recall that if G had initial and end of round whitening, it was treated as part of the round function of G and additional whitening on the leftmost b bits in each round was not added when forming G' (*i.e.* kw_{ji} was already 0 in the equations for G' for $0 \leq j \leq r$ and $1 \leq i \leq b$).

- Set $kw_{0(b+l)} = 0$ and $kw_{1(b+l)} = 0$ for $1 \leq l \leq y$. This sets the rightmost y bits of the initial whitening and of the end of round whitening in the first round to 0. By using plaintexts that have the rightmost y bits set to 0, this results in the rightmost y bits in the first round having no impact on the equations.

- Set $kw_{j(b+l)}$ to $v_{(j-1)h}$ for $2 \leq j \leq r-1$ and $1 \leq l \leq y$, where h is the index in the leftmost b bits corresponding to the bit position swapped with the l^{th} bit of the rightmost y bits. This removes the impact of the swap steps by having the rightmost y bits of whitening in each round cancel with the y bits omitted from each round. These settings are needed only on rounds 2 through $r-1$. The output of the r^{th} round function is the ciphertext so the swap step is not applicable after the r^{th} round. Per the previous item, the rightmost y bits in the first round can be set to have no impact on the equations. Each such setting can add an internal variable, $v_{(j-1)h}$, which now equals u_{jh}, to the equations.

These settings result in each input bit to the $(j+1)^{st}$ round function being of the form $u_{(j+1)i} = v_{ji}$ and the impact of any added end of round whitening and the swap step being removed. The equations will combine to form a set of equations, Γ from the equations in Γ' with any kw_{ji}'s which appear in Γ' removed and with at most $(r-2)y$ internal variables added to the equations. Before explaining how these variables can be accommodated, I first state a few additional notes on the resulting equations. The equations in Γ may contain up to y extra plaintext bits and up to y extra ciphertext bits beyond the b-bit block size of G since G' processes $b+y$ bit blocks. The attacker can set these extraneous y plaintext bits

to any value (the whitening bits were set in the conversion based on these plaintext bits being set to 0) and the extra y ciphertext bits are identical to y of the bits output from the next to last round function. For any equation $Eq' \in \Gamma'$ that holds with probability $\frac{1}{2} + \alpha$, the corresponding equation, $Eq \in \Gamma$, formed by removing the $kw'_{ji}s$ from Eq' will also hold with probability $\frac{1}{2} + \alpha$. Furthermore, only variables representing whitening bits present not in G are deleted when converting Γ' to Γ and no equations are added or removed. An equation will not be disappear when removing kw_{ji} variables because that would imply the equation did not involve plaintext and/or ciphertext bits. Since any whitening bits added when forming G' are pseudorandom, there will not be equations in Γ' containing only such whitening bits.

I now address the presence of the internal variables in Γ. Since it was assumed Γ' consists entirely of equations involving only plaintext, ciphertext and expanded-key bits, the removal of the swap step can introduce up to y internal variables, $(v_{ji's})$, per round into the equations. The removal of the swap step impacts $r-2$ rounds, resulting in a maximum of $(r-2)y$ internal variables in the equations in Γ. If equations in Γ' corresponding to some $y > 0$ are converted directly into equations for the original cipher ($y = 0$), this results in at most $2^{(r-2)y}$ possible values to try for the internal variables. However, if Γ' is converted into a set of equations for the cipher corresponding to a $b + y - 1$ blocksize, there are at most 2^{r-2} possible combinations of values for the internal values to try. Let Γ'_{-1} denote this set of equations. Solve the equations, setting the $r-2$ internal variables in the equations to the specific values that resulted in a solution. Then convert the resulting equations to equations for the cipher corresponding to a $b + y - 2$ blocksize. Repeat the processs, setting the $r-2$ internal variables to specific values each time that result in a solution for the specific block size. This removes the internal variables from the equations. Let Γ'_{-n} denote the sets of equations corresponding to the version of G' with a $b + y - n$ block size, where $1 \leq n \leq y$. The set of equations, Γ, used to attack G will be Γ'_{-b}. This results in $\leq \sum_1^y 2^{(r-2)} = y2^{(r-2)}$ possible combinations of the internal variables to try as opposed to $\leq 2^{(r-2)y}$ combinations. Since r is constant (and small in practice) and y is bounded by b, which is constant, the amount of work in converting the attack on G' to an attack on G is polynomial in the time to attack G', specifically, the work is bounded by a constant times the time to attack G'.

For example, in AES the value of $2^{(r-2)}b$ is $256 * 128 = 32768$. The amount of memory required is linear in the amount of memory required to attack G'. In the worst case, a separate amount of memory is required when forming each Γ'_{-n}. Thus, a linear attack on a r-round version of G' that requires less than 2^b (plaintext, ciphertext) pairs implies a linear attack exists on G. □

Claim 5.2 can be applied to algebraic equations in general. An algebraic attack on a block cipher G is defined in the same manner as the linear attack with the modification that the equations can involve any algebraic operations, not just XORs.

Claim 5.3. *Given a block cipher G with a block size of b bits and r rounds, and its elastic version G' with a block size of $b + y$ bits for $0 \leq y \leq b$ and r' rounds where $r' = r + \lceil \frac{yr}{b} \rceil$, if G' is subject to an algebraic attack on r rounds then either G is subject to an algebraic attack or the resources exist to perform an exhaustive search on G over all plaintexts. An algebraic attack refers to an attack involving a set of equations relating the bits of a single plaintext and its corresponding ciphertext, and the expanded-key bits.*

Proof. The proof follows directly from the proof to Claim 5.2 by removing the qualification in Claim 5.2's proof that the equations in the L_j sets are linear. Now Γ' and Γ contain algebraic equations instead of only linear equations. Γ is formed from Γ' exactly as before (the conversion adds only XORs of variables to the equations). Therefore, if an algebraic attack exists on r rounds of G' then an attack exists on G. □

5.4 Side Channel and Differential Fault Analysis Discussion

The elastic version of a block cipher will likely be subject to any side channel and differential fault attacks on the original cipher. These types of attacks involve monitoring and/or physically damaging the device performing the cryptographic operations. Such attacks rely on at least partial access to the component performing the encryption. The first type is passive side channel analysis in which either resources of the device performing the cryptographic operation are monitored or information the device emits is monitored. The second type is differential fault analysis. This involves the introduction of faults into the device and comparing the results of the operation performed with and without the fault.

The idea for these two categories of attacks first appeared in the late 1990's. Side channel attacks involve observing side channels of a device. The type of information monitored includes power consumption, timing characteristics, electromagnetic emanation and acoustic emanation. For example, the time and/or power involved in performing an exponentiation may vary depending on the exponent in a public key algorithm [Koc96]. There are companies, for example Riscure, that sell software for performing power analysis on smart cards. CPU acoustics were found to be of use in attacking RSA [ST04]. In the acoustic attacks, the sound omitted from the CPU can be separated from that of the fan because the fan noise is typically less than 10Khz; whereas, the CPU is above 10Khz. Early work by J. Kelsey, *et al.*, on applying side channel attacks to symmetric key ciphers discusses side channel attacks on block ciphers based on their structure as a number of rounds [KSWH00]. Examples of how certain types of side channel information can be used to attack DES [NIS99a], IDEA [LM91] and RC5 [Riv95] are covered. The side channel attacks attempt to recover internal state information, such as the output from or input to a round in order to determine enough round key bits per round to allow an exhaustive search on the remaining bits. Memory usage has also been shown to be a valuable source of information. D. Osvik, *et al.*, demonstrated how the cache, which is shared memory, contributes to information leakage [OST06]. By having a second process accessing the same memory used by the cipher, enough information is obtained to determine the expanded-key for AES.

Differential fault analysis involves inducing faults into the device performing the cryptographic operation then observing outputs prior to the fault and after the introduction of the fault. For example, by using radiation to damage a device. The concept of fault analysis was first proposed by Boneh, *et al.*, for public key ciphers [BDL97]. The applicability of fault analysis to a symmetric-key cipher was discussed by Biham and Shamir, who described how the round key bits of DES, the standard block cipher at the time, could potentially be recovered through the introduction of faults [BS97]. However, differential fault analysis attacks assume that an attacker is able to introduce faults into a sealed tamper proof device with non-negligible probability that a fault is created in a (unknown) single bit location in one of the registers at some random intermediate stage in the encryption or decryption. As a result, this concept is less practical than side channel analysis.

In the elastic version of a cipher, the length of y and the exact bit positions being swapped can influence the time spent in the swap step. However, y and the bit positions involved in the swap are already known (except y is unknown in the proposed Elastic ECB mode in Chapter 10 that varies y based on key bits). Any side channel or differential fault analysis that works on the round function of the original cipher will work on the elastic version since the round function is not modified. Attacks which alter memory or depend on memory access will also continue to be feasible. The difference is that the key schedule will not be highly structured, assuming a stream cipher is used, resulting in the need for the attack to determine all expanded-key bits instead of relying on using the key schedule to determine additional expanded-key bits from the bits found by the attack.

5.5 Summary

I have proven that the security of the elastic version of a block cipher against key recovery attacks is related, in polynomial time and memory, to the security of the original, fixed-length, version of the block cipher. All practical attacks attempt to recover key or expanded-key bits. If the original version is immune to such attacks, then the elastic version is also immune to such attacks. This result is due to a reduction between the original and elastic versions of the ciphers that allows the round keys for the original version to be derived from the round keys for the elastic version. An attack on the original version of a cipher does not automatically imply an attack exists on the elastic version of the cipher due to the fact that the elastic version involves steps in addition to the steps of the original cipher.

While the general result regarding key recovery attacks covers linear cryptanalysis, I use a separate method to prove that any attack on the elastic version of a block cipher using linear cryptanalysis can be converted into an attack on the original version of the block cipher, using polynomial time and memory. More generally, any algebraic equation relating the plaintext, ciphertext and key or round key bits for the elastic version can be converted into an equation for the original version. Therefore, if the original cipher is immune to any attack based entirely on algebraic equations involving the plaintext, ciphertext and key bits, then the elastic version is also immune to such attacks. I also discussed why any

side channel or differential fault attack on the original cipher is likely to be applicable to the elastic version of the cipher, provided that the attack does not work by finding a few expanded key bits then inserting them into the key schedule to find the remaining expanded key bits.

Chapter 6

Elastic Block Cipher Examples

6.1 Overview

I created four examples of elastic block ciphers from AES, Camellia, MISTY1 and RC6. These examples demonstrate that the method for creating elastic block ciphers is practical and can be applied to existing block ciphers, in general. AES is the winner of NIST's advanced encryption standard competition for block ciphers. RC6 was a finalist in the competition. AES and Camellia are 128-bit block ciphers recommended in the final results of NESSIE's competition for cryptographic algorithms. MISTY1 is NESSIE's recommended 64-bit cipher for existing applications requiring a 64-bit cipher. I chose to create an elastic version of AES because it is a standard. Creating elastic versions of Camellia and MISTY1 allowed me to demonstrate how the method can be applied to block ciphers using a Feistel network. Both Camellia and MISTY1 include other functions in addition to a Feistel network. RC6 is a simple algorithm consisting of few steps compared to the other block ciphers. I use RC6 to demonstrate how to create an elastic version of a block cipher that is not a Feistel network but whose round function processes only segments (one half) of the block. The elastic version of MISTY1 covers block sizes of 64 to 128 bits and the elastic versions of the other three block ciphers each cover block sizes of 128 to 256 bits.

For each of the four elastic block ciphers, I measured the performance of the elastic version to the original version with padding. The performance indicates the rate of encryption. It does not include the time to expand the key, which I measured separately and summarize

in Chapter 8. The relationship between the rates of decryption of the original cipher and that of the elastic version is the same as the relationship for encryption and is not reported (*i.e.,* if the elastic version encrypts 110 blocks in the time it takes the original version to encrypt 100 blocks, it also decrypted 110 blocks in the time it took for the original version to decrypt 100 blocks). This is because the time to execute the decryption function is close (although not always identical) to the time to execute the encryption function in each of the four ciphers. As a result, the operations added when creating the elastic version represented approximately the same percentage of the computations when decrypting as when encrypting a block of data. I include the performance for encrypting each block size that is an integral number of bytes. In the implementations, when the block size is not an integral number of bytes, the fractional byte is stored in a byte and the processing time is the same as if a full byte of data is present; therefore, the time to encrypt $b + y$ bits is the time to encrypt $\lceil \frac{b+y}{8} \rceil$ bytes. It is possible for the computational workload to vary at a more granular level, such as in a hardware implementation.

I performed the statistical tests used by NIST in the AES competition to measure the randomness of a cipher's output. I implemented my own version of the tests in order to accommodate the different block sizes. The description of the statistical tests and the results are in Appendix A. Based on the results, all four constructions of the elastic block ciphers show no signs of any statistical weakness compared to the original ciphers. In the AES competition, finalists passed each test at a rate of 96.33% or higher [NIS00]. The elastic versions of the ciphers also met or exceeded this rate. In the remainder of this chapter, I provide a summary of each elastic block cipher and results from the performance tests. Descriptions of the original ciphers are in appendices C, D, E and F.

6.2 Common Items

Per the elastic block cipher algorithm description, the round function and range for the number of rounds will vary amongst elastic versions of different ciphers. The exact bit positions used in the swap steps may also vary. The other steps will not vary. I describe here implementation details shared by the four examples I created. I remind the reader

that the number of rounds in the elastic version is $r + \lceil \frac{ry}{b} \rceil$ where r is the number of rounds or cycles in the original version of the cipher.

- In the elastic versions of block ciphers, the bits in a block of data are counted from the most significant (leftmost) to the least significant (rightmost). Bits 1 to b become the b-bit portion and bits $b+1$ to $b+y$ become the y-bit portion. For example, in the bit string `0111111`, the first bit is the 0 and the bits numbered two to eight are 1's.
- The mixing step performed a byte or word level rotation combined with a swapping of any fractional byte of data. The amount of the rotation depended on an expanded-key byte. When the block size was not an integral number of bytes or words, the rightmost fractional byte or word was omitted from the rotation and swapped with bits from the rotation's result. A second expanded-key byte determined the byte or word from which bits are swapped with the fractional byte. If the block size is an integral number of bytes or words, this second expanded-key byte is unused.
- RC4 was used for the key schedule in all implementations of the elastic block cipher. The first 512 bytes of RC4's output were discarded [Mir02], then RC4 was run until the required amount of expanded key bytes were obtained. Refer to Chapter 8 for a discussion on key schedules and the benefits using RC4 (or another stream cipher) provides over modifying the cipher's original key schedule when creating the elastic version of a block cipher.
- When comparing the performance of the elastic version of a cipher against the original version with padding, the time to pad the data was not included when measuring the original version's performance.

In the examples, how the bits selected for the swap steps vary slightly among the ciphers. In all cases, the bits swapped out of the b-bit portion at the end of the round are y sequential bits (circling back to the leftmost bit after reaching the rightmost bit). It is the starting position of this sequence that varies. However, as shown in Chapter 5, the exact positions of the bits swapped does not matter in the sense that the elastic version will be secure against any attack that works by recovering key or round key bits if the original cipher is secure against the attack regardless of the bit positions chosen for each swap step.

6.3 Elastic AES

Description

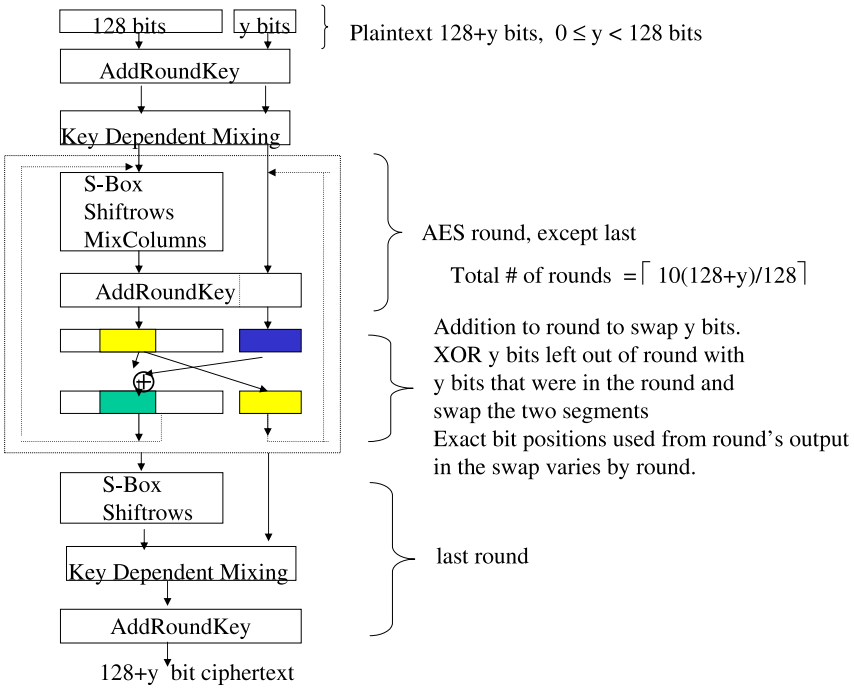

Figure 6.1: Elastic AES

I implemented two elastic versions of AES with 128-bit blocks as the original block size. Elastic AES is shown in Figure 6.1. I refer to the first version, Version I, as the standard textbook definition of AES, with the round function described as the SubBytes, ShiftRows, MixColumns and AddRoundKey steps [NIS01b]. This version is described in Appendix C. Each of the steps of the round function are performed in sequence in Version I. The second version, Version II, combines the SubBytes, ShiftRows and MixColumns steps and performs them as a table lookup. This results in the round function being a series of table lookups and XORs. Version II requires fewer CPU cycles than Version I, at the cost of an increase in memory usage. Version II performs table lookups at the byte level. AES can also be implemented to process 32-bit words, in which case the table entries are 32-bit words. I kept the processing at the byte level because I chose to implement the key-dependent permutations and swap step at the byte level.

The number of rounds, r, in AES for 128-bit blocks is 10; therefore, the number of rounds, r', in the elastic version ranges from 10 (when $y = 0$) to 20 (when $b + y \geq 244$). The application of whitening in the elastic version requires including y extra bits in AES's AddRoundKey step. I implemented the swap step by selecting y sequential bits from the leftmost b bits, wrapping around from the right end to the left as needed. The starting position is varied by moving one byte to the right each round to avoid using the same bit positions in each swap. This avoids any complex selection process for choosing the y bits that would decrease performance.

Performance

I implemented the elastic versions of both Version I and Version II of AES in *C* and compared the performance of the elastic versions to the fixed-sized versions with padding. The elastic versions increase the number of operations beyond the 128-bit versions due to the swap steps, the two key-dependent permutations and the expansion of whitening to cover $128 + y$ bits. In Version I, the elastic version saves processing time over padding. Obviously, as the block size approaches two full blocks, 20 rounds of AES are incurred in the elastic version along with the added steps, which increases the number of operations beyond the 20 rounds of AES that are required when padding the data to two full blocks. Therefore, it is expected that there is no performance benefit when encrypting blocks just under 32 bytes. In Version II, the elastic version does not offer a performance benefit compared to padding. This is because of the simplistic nature of the operations involved (table lookups and XORs) for the round function. Even though there are fewer rounds in the elastic version than with padding, the operations for the swap step and the two key-dependent permutations consume any savings gained from having fewer rounds. However, Version II offers a performance benefit over the black-box approaches described in Chapter 2.

Both the elastic version and regular 128-bit version of AES were run on several processors in Linux and Windows environments to compare their performance. In the tests, the data to be encrypted was viewed as individual $(b + y)$-bit blocks. The elastic version of AES encrypted each block individually with no padding. To encrypt the data with regular AES, the $b + y$ bits were padded to $2b$ bits and encrypted as two b-bit blocks. When measuring

encryption performance (in terms of blocks per second), AES's performance for a single block was based on the time to encrypt 32 bytes, to represent the padding required when using AES for $b + y$ bit blocks. I measured the time to encrypt one million $(128 + y)$-bit blocks, where $y = 8n$ for $n = 1$ to 16, using the elastic version of AES and two million 128-bit blocks using the original version of AES. As explained previously, the performance of the elastic version adjusts per byte size increments as opposed per bit. Figure 6.2 summarizes the results from the following three cases. The values used to generate the graph are in Table B.1 in Appendix B.

- Case 1: A *C* implementation of Version I tested on a 1.3 Ghz Pentium 4 processor with 512MB RAM running Windows XP.
- Case 2: A *C* implementation of Version I tested on a 2.8 Ghz Pentium 4 processor with 1GB RAM running Redhat Linux 2.4.22.
- Case 3: A *C* implementation of Version II tested on a 2.8 Ghz Pentium 4 processor with 1GB RAM running Redhat Linux 2.4.22.

In the first trial, the number of $(b + y)$-bit blocks the elastic version can encrypt per second ranges from 190% of the number of $2b$-bit blocks AES can encrypt per second when $y = 1$ to 100% when $y = 97$. Then the elastic version's performance decreased gradually to a low of 83% of AES's rate when $y = 128$. In the second trial, the values ranged from 186% to 69% of AES's rate, with the elastic version becoming slower than the fixed-length version when $y = 73$. The third trial used Version II of AES and the elastic version was slower than the fixed-sized version with padding for all block sizes.

In contrast to the elastic version of AES, the computational workload does not vary by block size in the methods by Bellare and Rogaway and by Patel, *et al.*, described in Chapter 2. Bellare and Rogaway's method requires slightly more than twice the work of using fixed-sized, 128-bit blocks for any $(b + y)$-bit block size where $0 < y \leq b$. Patel's method requires two full applications of the block cipher plus the cost of a hash function. I compared Bellare and Rogaway's method and Patel's method to AES with padding on the Pentium 4 processor used in cases 2 and 3. I used SHA-256 [NIS02] as the hash function

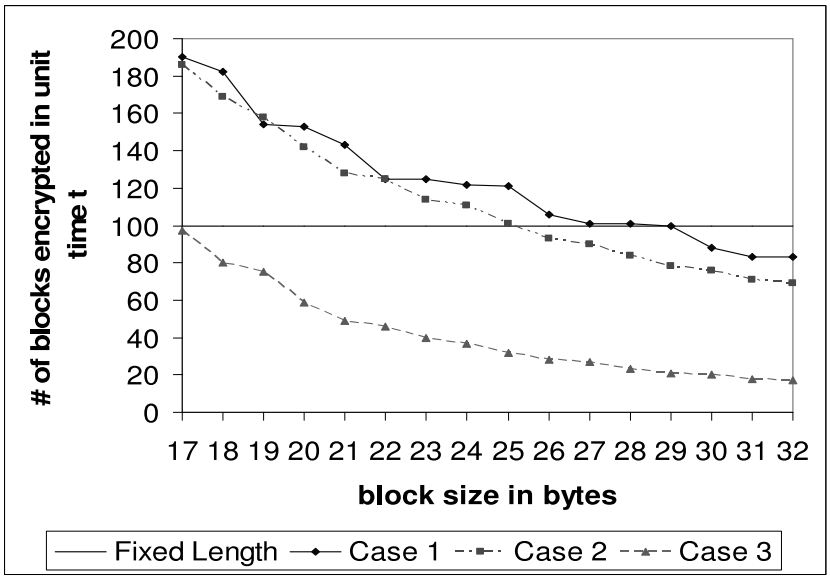

Figure 6.2: Normalized Number of Blocks Encrypted by Elastic AES in Unit Time (Regular AES = 100)

in Patel's method. Bellare and Rogaway's method encrypted between 49 and 50 $(b + y)$-bit blocks in the same amount of time AES with padding encrypted 100 blocks, for both Version I and II of AES. Patel's method encrypted 96 $(b + y)$-bit blocks in the time it took Version I of AES to encrypt 100 blocks, and encrypted 18 $(b + y)$-bit blocks in the time it took Version II of AES to encrypt 100 blocks. When using Version I of AES, elastic AES is computationally more efficient than both Bellare and Rogaway's method and Patel's method. When using Version II of AES, elastic AES is computationally more efficient than Bellare and Rogaway's method for block sizes up to 21 bytes in length, and is more efficient than Patel's method for block sizes less than 31 bytes and is as efficient as Patel's method for block sizes between 31 and 32 bytes.

6.4 Elastic Camellia

Description

Camellia process 128-bit blocks and is a Feistel network with additional steps [AIK+00].

A description of Camellia is in Appendix D. A function, referred to as the FL function, is applied after every three cycles in the Feistel network, except after the last three cycles. FL is applied to the left half and its inverse is applied to right half of the 128 bits. Camellia contains initial and final whitening steps. Creating the elastic version requires expanding these two whitening steps and adding end of round whitening steps to all the other rounds. I also added the initial and final key-dependent permutations and the swap steps. In the elastic version, a cycle of Camellia is used as the round function and the FL function is applied after every three rounds. The round function of the elastic version is shown in Figure 6.3. The data is processed as bytes. The swap step was implemented by altering the starting positions between the left and right halves of the b-bit portion. Camellia has 9 cycles. The number of rounds in the elastic version ranges from 9 to 18, where the round function is a cycle of Camellia.

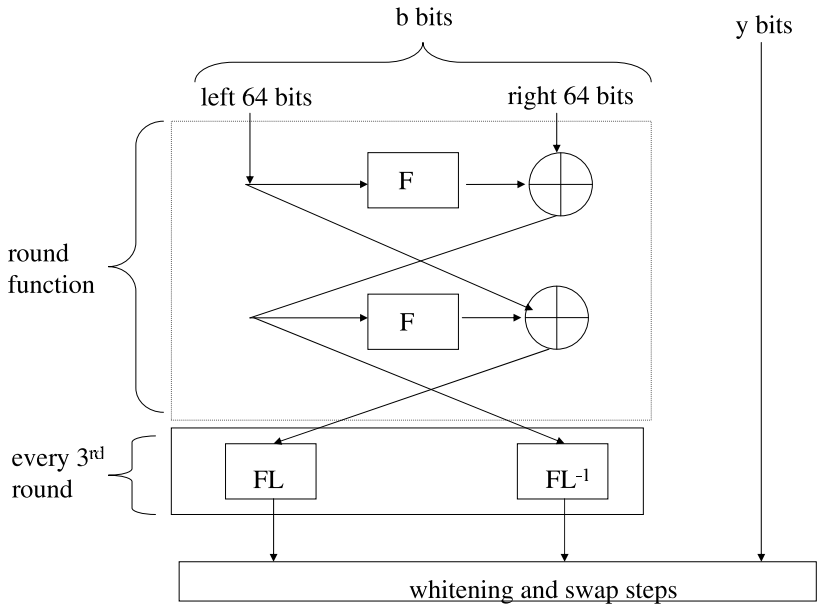

Figure 6.3: Round Function for Elastic Camellia

Performance

Unlike the other three ciphers, there was no performance gain when using elastic Camellia compared to the original Camellia with padding. When processing 128-bit blocks, there

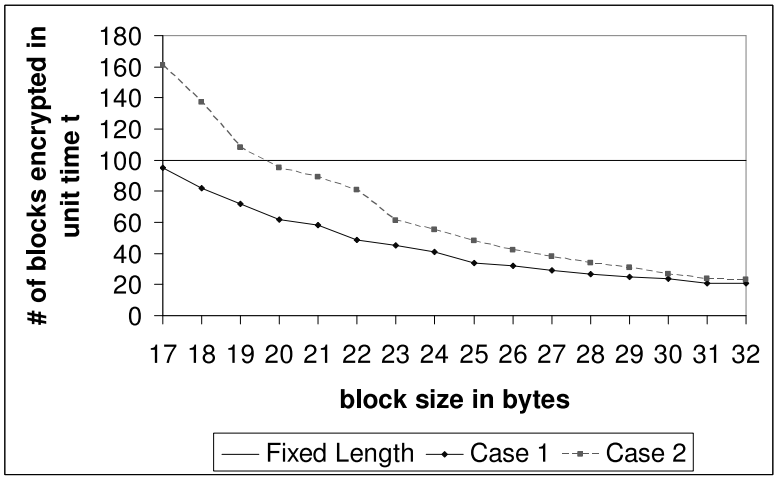

Figure 6.4: Normalized Number of Blocks Encrypted by Elastic Camellia in Unit Time (Regular Camellia = 100)

are nine cycles in the Feistel network, and there are applications of the FL function after the third and sixth cycles. I also measured the performance without the initial and and final permutations Removing these two steps results in a performance benefit when encrypting blocks that are one to three bytes over the normal 16 byte block size with the elastic version compared to using the original version with padding. Results for the following two cases are shown in Figure 6.4:

- Case 1: A C implementation of elastic Camellia with all steps.
- Case 2: A C implementation of elastic Camellia without the initial and final key-dependent permutations.

Results for both cases are from tests on a 2.8 Ghz Pentium 4 processor with 1GB RAM running Redhat Linux 2.4.22. The values used to generate the graph are in Table B.2 in Appendix B. By using a lower bound of twice the work of padding for Bellare and Rogaway's method, elastic Camellia with the key-dependent permutations provides a performance benefit for block sizes up to 22 bytes and the version without the key-dependent

permutations provides a performance benefit for block sizes in the range of 9 to 25 bytes compared to Bellare and Rogaway's method. Patel's method using SHA-256 as the hash function encrypted 61 $(b + y)$-bit blocks, $0 < y \leq b$, in the time it took Camellia with padding to encrypt 100 blocks. Elastic Camellia is more efficient than Patel's method for block sizes up to 21 bytes and 23 bytes, respectively, for the two cases.

6.5 Elastic MISTY1

Description

MISTY1 is a 64-bit block cipher structured as a Feistel network with an additional function, called the FL function (not to be confused with the FL function from Camellia), applied once per cycle [Mat00a]. MISTY1 does not contain whitening steps. A cycle from MISTY1 is used as the round function in the elastic version. Creating the elastic version involved adding the whitening steps, the initial and final key dependent permutations and the swapping of bits after each cycle. The description provided here summarize the general structure of MISTY1. Refer to Appendix E for the definitions of the individual components. While the number of rounds is not fixed, four cycles are recommended [NES03] and is the number upon which I based the number of rounds in the elastic version.

Notation:

- L_i and R_i denote the left and right halves of output, respectively, of the i^{th} round after the halves are switched with $i = 0$ denoting the input to round 1.
- $FL(x, KL)$ and $F0(x, K0, KI)$ are functions taking bit string x and key material $KL, K0, KI$.
- FL_i and $F0_i$ denote the i^{th} occurrence of FL and $F0$, respectively.
- KL, KI and $K0$ denote subkeys from the expanded-key material, with a subscript of i denoting the i^{th} component.

In the fixed-length version of MISTY1, the output of odd numbered rounds is defined by:

$R_i = FL_i(L_{i-1}, KL_i)$

$L_i = FL_{i+1}(R_{i-1}, KL_{i+1}) \oplus F0_i(R_i, K0_i, KI_i)$

The output of even numbered rounds is defined by:

$R_i = L_{i-1}$

$L_i = R_{i-1} \oplus F0_i(R_i, K0_i, KI_i)$

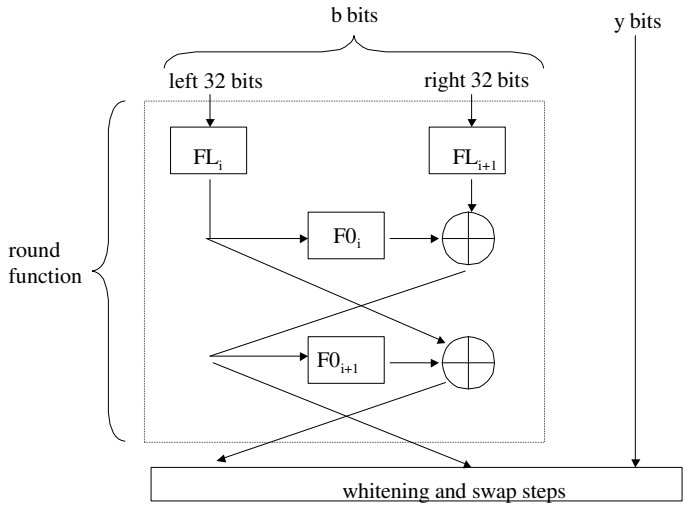

Figure 6.5: Round Function for Elastic MISTY1

When creating the elastic version by positioning the swap step after a cycle, every bit ends up in the left half and right half of input to the round function at least the same number of times as occurs in regular MISTY1. The round function in the elastic version is shown in Figure 6.5. The data is processed as 32-bit words. The bit positions from a round's output that are involved in the swap step will vary across each round to avoid some bit positions from being swapped every round while others are never involved in the swap. I chose to alternate the starting position for the swap between the left and right halves of the round's output and, within each halve, rotate the starting position one word each time. Regardless of where a bit was positioned when swapped out and where it is when it is swapped back in, the bit will end up in the left and right halves of the original round function the same number of times as in regular MISTY1. Furthermore, since the swap XORs the bits being swapped out with those being swapped in, the bits swapped out continue to influence the next round in the same positions they would have influenced had they not been swapped out.

Performance

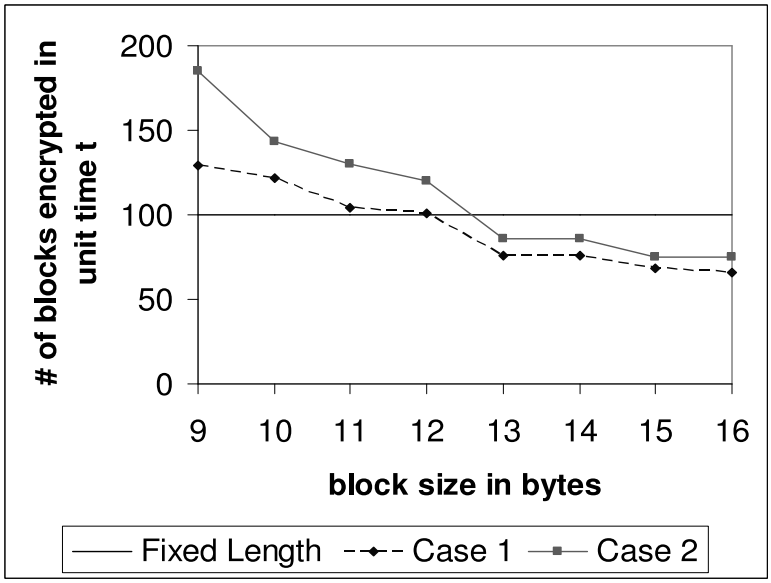

Figure 6.6: Normalized Number of Blocks Encrypted by Elastic MISTY1 In Unit Time (Regular MISTY1 = 100)

I implemented elastic versions, with and without the key-dependent permutations, and the regular version of MISTY1 in *C*. The elastic versions accommodate block sizes of 64 to 128 bits. Case 1 refers to the version with the key-dependent permutations and Case 2 refers to the version without the key-dependent permutations. The elastic versions increased the number of operations beyond the 64-bit version of MISTY1 due to the whitening, the swap steps and, in one version, the key-dependent permutations.

Both the elastic and regular versions of MISTY1 were tested in Redhat Linux and Windows XP environments with Intel Centrino, Pentium 3 and Pentium 4 processors varying from 1Ghz to 2.8Ghz to compare their encryption and decryption rates. The results in terms of how large the block size can be in the elastic version while remaining more efficient than padding a second block and encrypting two blocks in MISTY1 are consistent across the environments.

The results shown in Figure 6.6 are from trials on a 2.8Ghz Pentium 4 processor with

1GB RAM running Redhat Linux 2.4.22. The values used to generate the graph are in Table B.3 in Appendix B. The elastic version of MISTY1 provides a performance benefit compared to padding for blocks that are one to four bytes over the 8-byte block size that MISTY1 processes. The benefit increases significantly in Case 2 compared to Case 1 for block sizes that are up to one additional byte over MISTY1's 8-byte block size. The performance benefit from removing the initial and final key permutations decreases as the block size increases because they represent an increasingly smaller portion of the operations as more rounds are added. In both cases, the elastic version provides a performance benefit when compared to Bellare and Rogaway's method based on a lower bound of twice the work of padding for Bellare and Rogaway's method. Patel's method with SHA-256 as the hash function encrypted 51 $(b + y)$-bit, $0 < y \leq b$, in the time it took MISTY1 with padding to encrypt 100 blocks using padding. Both cases of the elastic version of MISTY1 encrypt at a faster rate than Patel's method for all blocks between 8 and 16 bytes.

6.6 Elastic RC6

Description

RC6 is an example of a 128-bit block cipher whose round function processes only a segment of the data block, but it is not a Feistel network. Appendix F contains a description of RC6. Creating the elastic version of RC6 involved adding the swap steps, whitening steps, and the initial and final key dependent permutations to RC6. RC6 divides the 128-bit data block into four 32-bit words, which I will refer to as ABCD. Each word is processed differently in the round function. A and C are updated by the round function based on the values of B and D. At the end of the round the words, A and C have expanded-key bits added to them then all the words are rotated to the left one word. Before the first round of RC6, B and D have expanded-key bits added to them and after the last round, A and C have expanded-key bits added to them. The addition of expanded-key bits to a word is a type of whitening. Since this "whitening" does not cover the entire data block and is not the same as performing whitening by XORing data with expanded-key bits, I view this addition as a step in the round function and not as whitening that should be

expanded to all $b + y$ bits when forming the elastic version. A sequence of four applications of the round function of RC6 is a cycle (the point at which each word has had the entire sequence of operations from the round function performed on it) This cycle becomes the round function in the elastic version of RC6. Therefore, the end of round whitening and swap step are added after every four original rounds of RC6 as shown in Figure 6.7. The number of cycles in RC6 for 128-bit blocks is 5 (20 applications of the round function). The number of rounds in the elastic version ranges from 5 to 10 (20 to 40 applications of the round function). The swap step was implemented with the starting position rotating to the right one word each round.

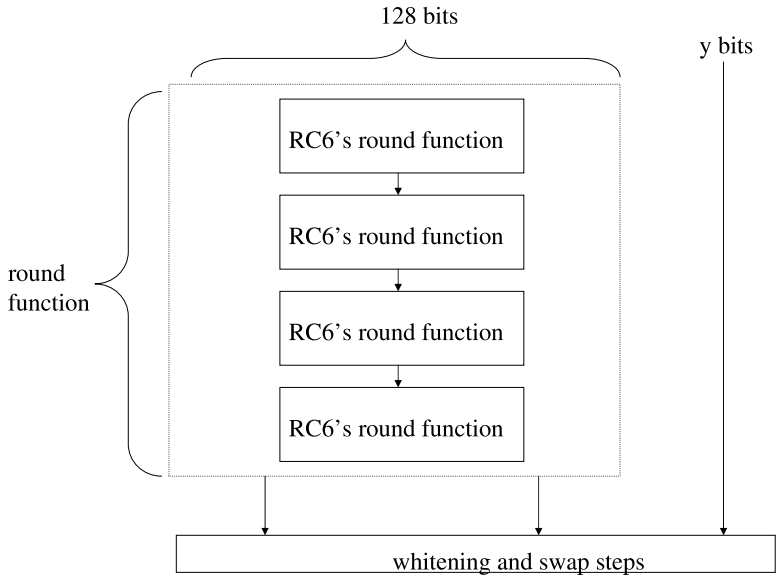

Figure 6.7: Round Function for Elastic RC6

Performance

The performance was tested on a 3Ghz Pentium 4 processor with 1GB of RAM running Redhat Linux 2.4.22. The elastic version provides a performance benefit compared to padding for blocks of under 21 bytes in length. The results are shown in Figure 6.8. The values used to generate the graph are in Table B.4 in Appendix B. Using a lower bound of twice the work of padding for Bellare and Rogaway's method, the elastic version of RC6 provides a performance benefit for blocks under 30 bytes in length when compared to Bel-

lare and Rogaway's method. Patel's method with SHA-256 as the hash function encrypted 52 blocks $(b + y)$-bit blocks, $0 < y \leq b$, in the time it took RC6 with padding to encrypt 100 blocks, making elastic RC6 more efficient than Patel's method for block sizes up to 29 bytes.

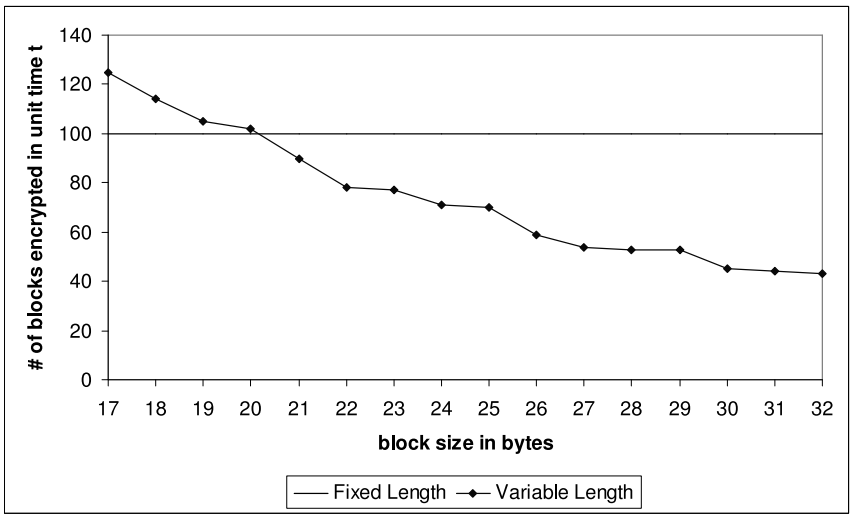

Figure 6.8: Normalized Number of Blocks Encrypted by Elastic RC6 in Unit Time (Regular RC6 = 100)

6.7 Summary

The constructions of the elastic versions of AES, Camellia, MISTY1 and RC6 illustrate how to apply the method for creating variable-length block ciphers. By applying the statistical tests used in NIST's AES competition, I conclude that there is no obvious flaw in the design because the level of randomness of the ciphertext produced by each of the elastic versions is consistent with the level required in the AES competition. The workload of the elastic version of a cipher is proportional to the block size, with the number of rounds increasing as the block size increases. This offers a performance benefit over previous methods that treated the block cipher as a black box and apply it multiple times, which results in the

amount of computation being fixed for any size block between one and two times the original block size instead of being proportional to the block size.

The performance benefit from using the elastic version of a block cipher for encrypting one plus a fractional block of data instead of padding the data to two full blocks depends on the original cipher and the exact implementation. The percent of overhead involved in adding the swap steps, whitening and two key dependent permutations varies based on the number of operations and exact implementation of the original cipher. For AES, whose block size is 16 bytes, there is a significant performance benefit when using the elastic version to encrypt blocks up to 25 bytes in length using an implementation of AES that requires little memory; whereas, there is no performance benefit when using a memory intensive implementation that consists entirely of table lookups and XORs. For Camellia, whose block size is 16 bytes, there is a performance benefit when using the elastic version for block sizes up to 19 bytes in length when the initial and final key-dependent permutations are not included. For MISTY1, whose block size is 8 bytes, there is a performance benefit when using the elastic version for block sizes up to 12 bytes. For RC6 with a block size of 16 bytes, there is a performance benefit when using the elastic version for blocks up to 20 bytes in length.

Chapter 7

Differential Cryptanalysis

7.1 Overview

In this chapter I discuss differential cryptanalysis of elastic block ciphers. While differential cryptanalysis is covered under Theorem 5.1, the results regarding the security of elastic block ciphers presented in Chapter 5 do not include examples of cryptanalysis for specific instantiations of elastic block ciphers. In order to provide an example of concrete cryptanalysis, I consider how the conversion of a block cipher to its elastic form impacts differential cryptanalysis. I define a general method for bounding the probability a differential characteristic occurs in the elastic version of a cipher when given the bound for a single round of the original cipher. I illustrate the method on the elastic versions of AES and MISTY1 described in Chapter 6.

Differential cryptanalysis was introduced by Biham and Shamir in their analysis of DES [BS93]. It works by determining a sequence of differences computed using the XOR of pair of inputs and outputs of each round, as shown in Figure 7.1. I will refer to any individual difference as a delta, indicated by the symbol Δ. The sequence of delta inputs and outputs of the rounds is called a differential characteristic. Specifically, let $(P1, C1)$ and $(P2, C2)$ be two (plaintext, ciphertext) pairs for a block cipher with r rounds. $\Delta P = P1 \oplus P2$ and $\Delta C = C1 \oplus C2$. Let λ_{ij} refer to the delta input to round j and let λ_{oj} refer to the delta output of round j. $\lambda_{i1} = \Delta P$. $\lambda_{or} = \Delta C$. Let pr_j be the probability λ_{oj} occurs given λ_{ij}. Let $\Omega = (\lambda_{i1}, \lambda_{o1}, \lambda_{i2}, \lambda_{o2} ... \lambda_{ir}, \lambda_{or})$. The probability Ω ocurrs is $\prod_{j=1}^{j=r} pr_j$.

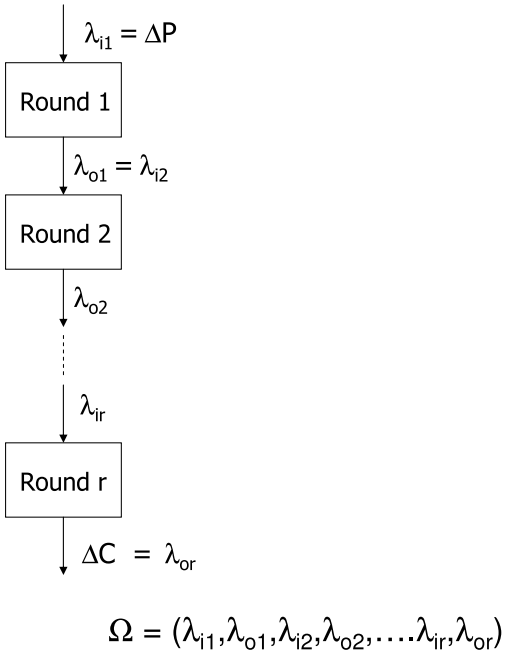

Figure 7.1: Differential Characteristic

If a differential characteristic occurs with high enough probability, a round key recovery attack may be possible. Given a set of (plaintext, ciphertext) pairs matching the ΔP and ΔC of the characteristic, the delta inputs and outputs of the internal rounds are assumed to match those of the differential characteristics when encrypting or decrypting the plaintexts or ciphertexts. Typically starting with the first or last round, round key bits impacting the deltas are set to correspond the values needed to produce the delta. As the key bits for a round are determined, the number of rounds is reduced by one and the process is repeated, using one less round. If the block size is b bits, a differential characteristic must hold with probability $> 2^{-b}$ for an attack to even be possible. Otherwise, the number of (plaintext, ciphertext) pairs required to perform the attack is $\geq 2^b$. Therefore, when proving a differential attack is not possible on a block cipher, it is sufficient to show that no differential characteristic occurs with probability $\leq 2^{-b}$. A characteristic that occurs with probability $> 2^{-b}$ does not automatically imply a differential attack is possible. The probability of the differential characteristic must be large enough to allow the result to be

both statistically relevant and reduce the number of (plaintext, ciphertext) pairs needed to a quantity for which the memory and CPU requirements are feasible.

The method I use to bound the probabilities of differential characteristics for an elastic block cipher involves defining states representing which bytes in the differential input to a round have a non-zero delta and tracking what sequences of states the cipher can be potentially pass through over a number of rounds. Using this method and differential bounds for the round function of the original cipher, I can derive an upper bound on differential characteristics for the elastic version of a cipher. I exclude the initial and final key-dependent mixing steps from my analysis because they will only reduce the probability of any specific differential characteristic occurring. I analyzed the elastic versions of AES and MISTY1 described in Chapter 6 in this manner to bound the probability of a differential characteristic occurring across all rounds of the elastic versions.

7.2 General Observation

The first observation I make regarding differential cryptanalysis of elastic block ciphers is that, unlike linear cryptanalysis where the equations for the elastic version, G', of a block cipher can be converted directly into equations for the original cipher G, a differential characteristic for G' cannot be converted directly into a differential characteristic for G except for one special case.

I use the following notation when describing a differential characteristic of an elastic block cipher.

- ΔY_i is the XOR of two y-bit segments for round i.
- ΔBin_i is the XOR of two b-bit segments input to the round function in round i.
- $\Delta Bout_i$ is the XOR of two b-bit segments output from the round function in round i.
- A b-bit value formed from the XOR of a b-bit value and a y-bit value, where $y \leq b$, refers to the b-bit result when the y bits are XORed with a subset of y bits of the b bits and the remaining $b - y$ bits are unchanged.

- Forming ΔY_{i+1} from $\Delta Bout_i$ refers to setting ΔY_i to the y bits from $\Delta Bout_i$ that are in the bit positions involved in the swap step after round i.
- ΔY, ΔBin and $\Delta Bout$ without a subscript of i refers to a specific delta independently of the round.

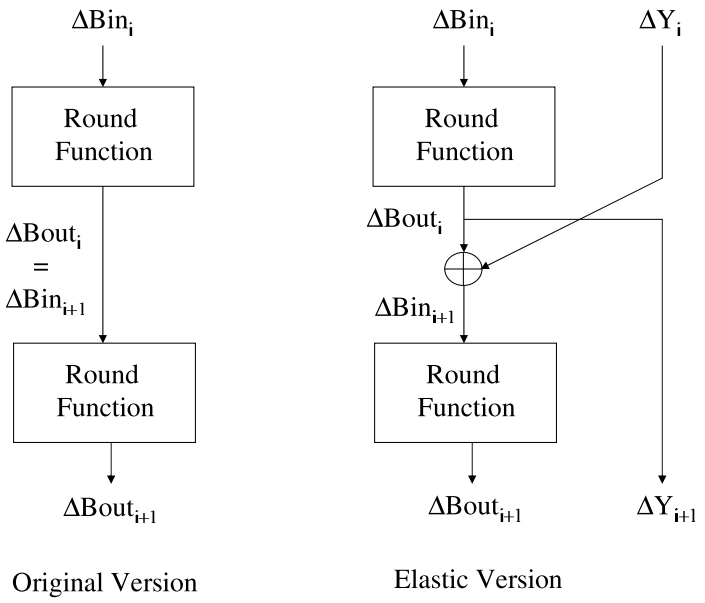

Figure 7.2: Two-Round Differential in Original and Elastic Versions of a Cipher

In the elastic version of a cipher, ΔBin_{i+1} is determined by $\Delta Bout_i$ and ΔY_i. If $\Delta Y_i \neq 0$ then $\Delta Bin_{i+1} \neq \Delta Bout_i$; whereas, $\Delta Bin_{i+1} = \Delta Bout_i$ in the original block cipher. This is shown in Figure 7.2. Therefore, a sequence of deltas ocurring across multiple rounds in the elastic version will not hold across the original version unless $\Delta Y_i = 0$ for r sequential rounds. For the same sequence of ΔBin_i values to occur in the original version, ΔBin_i must produce the ΔBin_{i+1} from the elastic version. There is no reason this will occur with the same probabilities corresponding to the ΔBin_i values producing the $\Delta Bout_i$ values in the elastic version unless the ΔY_i values are all zero.

Now consider the special case where r consecutive ΔY_i's are 0. If a differential charac-

teristic occurs in the elastic version, G', of a block cipher that contains r consecutive rounds with $\Delta Y_i = 0$ and this characteristic can be used to attack G', then it can be used to attack G. Let Ω' be the characteristic corresponding to the ΔBin_i values and $\Delta Bout_i$ values for the r consecutive rounds each with $\Delta Y_i = 0$. Ω' is also a characteristic for the r rounds of G. Ω' must hold with probability $> 2^{-b-y}$ to be used in an attack on G'. If Ω' holds with probability $2^{-\alpha} > 2^{-b}$, then it can be used to attack G directly, provided the probability is large enough that it is computationally feasible to encrypt $O(2^{\alpha})$ plaintexts. If it holds with probability $2^{-\alpha}$ such that $2^{-b} > 2^{-\alpha} > 2^{-b-y}$, it can be used to attack G as follows: Using an r round version of G' and (plaintext, ciphertext) pairs consistent with the delta input and delta output of Ω' by setting the leftmost b bits to be consistent with Ω' and the rightmost Y bits equal to 0. Then apply the attack on G' to find the round keys for the r rounds and use these as the round keys for the r rounds of G. However, if this is computationally feasible, it implies it is feasible to encrypt or decrypt 2^{b+y} data blocks, which means it is computationally feasible to encrypt 2^b plaintexts with G. Thus G is insecure because given a ciphertext, C, an attacker can ask for all 2^b plaintexts be encrypted with the same key (which is unknown) used to generate C and see which plaintext produces C. As an estimate of the probability of r consecutive rounds having $\Delta Y = 0$, consider what happens if the y bits left out of each round in G' take on any of the possible 2^y values with equal probability. Then, ignoring the differential for the b-bit portions of each round's input and output, a case where $\Delta Y_i = 0$ for r consecutive rounds may be found for small values of y and r. If each ΔY_i occurs with probability 2^{-y}, then the probability that $\Delta Y_i = 0$ in r consecutive rounds is 2^{-yr}. For example, in MISTY1 $r = 4$ (MISTY1 contains four cycles and a cycle is used as the round function in the elastic version). When $y = 1$, the probability of r consecutive ΔY's being zero is $\frac{1}{16}$.

7.3 State Transition Method

The general method I use is the tracking of states through the rounds of an elastic block cipher. I devise a method for categorizing the impact of the swapping of bits between rounds on the differentials entering a round. I combine the impact of the swap step with the upper

bound on the probability a differential characteristic occurs in a single application of the round function (from available analysis on the original version of the cipher) to determine an upper bound the probability of a differential characteristic across multiple rounds. By obtaining a bound, α, on the probability across n rounds, the probability across r' rounds can be bounded by $\alpha^{\lfloor \frac{r'}{n} \rfloor}$.

In the case of elastic AES, I view the $(b+y)$-bit data block entering a round as a b-bit segment and a y-bit segment. Three main states are defined:

- $\Delta Bin = 0$ and $\Delta Y \neq 0$.
- $\Delta Bin \neq 0$ and $\Delta Y = 0$.
- $\Delta Bin \neq 0$ and $\Delta Y \neq 0$.

The state in which $\Delta Bin = 0$ and $\Delta Yin = 0$ is not of interest because, given a non-zero delta input to the cipher, a delta of zero across all $b+y$ bits cannot occur. Within a main state, the number of bytes for which the delta is non-zero are counted. For example, if the input to the third round has a ΔBin that is 1 in the 2^{nd} and 18^{th} bit positions and is zero in all other bits, then there are two bytes with non-zero deltas in ΔBin. Tracking of states between rounds involves determining what $\Delta Bin||\Delta Y$ can result for the $(i+1)^{st}$ round based on the delta in the i^{th} round. For example, if $\Delta Bin = 0$ and $\Delta Y \neq 0$ in the input to round i, then $\Delta Bin \neq 0$ and $\Delta Y = 0$ in round $i+1$. This is because the delta output of the i^{th} round function wil be zero, then the non-zero ΔY will be swapped into the b-bit portion input to the $(i+1)^{st}$ round and a delta of zero will be swapped out to form the ΔY for the $(i+1)^{st}$ round.

When the original cipher is a Feistel network (or is a Feistel network with additional steps as in the case of MISTY1), the ΔBin portion is viewed as a left half (ΔLin) and right half (ΔRin). The main states are:

- $\Delta Lin = 0$, $\Delta Rin = 0$, $\Delta Y \neq 0$.
- $\Delta Lin \neq 0$, $\Delta Rin = 0$, $\Delta Y = 0$.
- $\Delta Lin = 0$, $\Delta Rin \neq 0$, $\Delta Y = 0$.

- $\Delta Lin \neq 0$, $\Delta Rin \neq 0$, $\Delta Y = 0$.
- $\Delta Lin = 0$, $\Delta Rin \neq 0$, $\Delta Y \neq 0$.
- $\Delta Lin \neq 0$, $\Delta Rin = 0$, $\Delta Y \neq 0$.
- $\Delta Lin \neq 0$, $\Delta Rin \neq 0$, $\Delta Y \neq 0$.

Using the states, an upper bound (which is not necessarily a tight upper bound) can be determined for the probability of a differential characteristic for r' rounds of the elastic block cipher. The probability of a differential characteristic occurring for a single application of the round function from existing analysis of the original cipher and the possible ΔB or $\Delta L||\Delta R$ values entering the round function in each round are used to bound the probability for each round. The possible ΔB or $\Delta L||\Delta R$ and ΔY values in a round determine the possible input states to the next round.

7.4 Elastic AES Differential Bounds

I show that a differential attack on an elastic version of AES is impossible for block sizes of $128 + y$ bits where $0 \leq y \leq 128$. My intent here is to show how the probabilities of the differential characteristics can be calculated or bounded in the elastic version of a block cipher when given the upper bound on the probability a differential characteristic for one application of the round function. My analysis is performed using a computer model that tracks how many bytes contain a non-zero differential characteristic in each round. This allows me to form an upper bound on the probability a differential characteristic occurs across all rounds of the cipher.

The variable block size and the swap step significantly increase the number of cases to explore when determining the probability of a differential characteristic compared to that of the fixed-length version of a block cipher. This is the reason why I had to find a new approach to modelling the differentials instead of using the differential trails approach used on AES [DR02]. My analysis results in the following claim.

Claim 7.1. *For the elastic version of AES described in Chapter 6 accepting block sizes of* $128 + y$ *bits for* $0 \leq y \leq 128$*, the probability of a differential characteristic occurring is* $\leq 2^{-128-y}$.

I use the remainder of this section to explain the computer model and how the result in Claim 7.1 was obtained. I note that my analysis is general in terms of block size but only considers a single method for selecting the bits to swap after each round as opposed to all possible ways of selecting y bits from 128 bits. The method used in the computer model can be applied to any choice of bits to swap, but it is computationally infeasible to include in one model all $2^{y(r'-1)}$ possible ways of selecting the bits to swap in the first $r' - 1$ rounds (recall that the swap step adds no value after the last round and thus can be omitted from round r'). Therefore, I modelled the swap step to correspond to how I implemented it in the software version of elastic AES. In the implementation, the swap is performed by selecting y consecutive bits from the round function's output to swap with the y bits left out of the round function and the starting position of the bits selected rotates to the right one byte each round. Also, I omit the initial and final key-dependent permutations from the model for simplicity since they will only decrease the probability of any differential characteristic. I first explain how the model is designed and then follow with specific results.

7.4.1 Model Description

I describe here how I modelled the elastic version of AES and the impacts of each step on the differential characteristic. The following terms and notation are used:

- delta refers to the difference between two bit strings. Given two data blocks, a delta byte means there is a **non-zero** difference between the two bytes in a specific position in the two data blocks.
- *exp*: The probability a specific differential characteristic occurs (across some number of rounds) is 2^{-exp} where *exp* is a non-negative integer. If $exp \geq b + y$ for all differentials across r' rounds in the elastic version of a block cipher then the elastic version is immune to a differential attack regardless of the attacker's resources because more than 2^{b+y} (plaintext, ciphertext) pairs are required for an attack.

- $B||Y$ indicates the $(b+y)$-bit plaintext input to the elastic block cipher. For the elastic version of AES, B refers to the leftmost 16-byte portion of the data block input to AES's round function and Y refers to the rightmost y bits of the data block.
- ΔBin and ΔY are as defined in Section 7.2.
- ΔB refers to the 16-byte delta in the B portion of the data block without specifying whether or not it is the input or output of a round function. *i.e.* ΔB can indicate ΔBin or $\Delta Bout$.
- $|\Delta Bin_i|$ and $|\Delta Y_i|$ refer to the number of delta bytes in ΔBin_i and ΔY_i, respectively.

In the model, the ΔB portion of the data is viewed as a 4-by-4 matrix of bytes. Refer to Figure 7.3. In this representation, the bytes are numbered from 1 to 16 as opposed to being referred to by the row and column number as done in the description of AES in FIPS197 [NIS01b]. The model tracks the number of bytes which differ per column of this matrix at the start of each application of the round function. I refer to this number as the column delta. For example, if bytes in the 1^{st},2^{nd} and 4^{th} rows of the second column contain a delta, the delta for column two is 3. I track the delta at the column level instead of the exact byte position due to the number of cases that would need to be considered if every byte is tracked. The lower bound of a probability for a differential characteristic is determined by adding up the number of delta bytes entering each application of AES's round function and multiplying the result by 6 to obtain a lower bound for exp. The multiplication by 6 is due to the fact that the probability a specific difference in two one-byte inputs to the S-Box produces a specific difference in the two outputs of the S-Box is 2^{-6} or 2^{-7}, depending on the exact byte values ([DR02] pages 205-206). The ShiftRows and MixColumns steps in AES's round function, and the swap step and end of round whitening transform the differential in a fixed manner in that the specific input differential results in a specific output differential with probability 1 for each step. Therefore, the differential probability for a single round is $\leq 2^{-exp}$ where $exp = 6 * |\Delta Bin|$.

The bytes in the ΔY portion of the data are also viewed as a 4-by-4 matrix as shown in Figure 7.3, with the matrix being filled in one row at a time. For example, if Y contains 7 bytes, there will be three columns containing two bytes each and one column containing one byte. The last byte is a partial byte if Y is not an integral number of bytes. In elastic

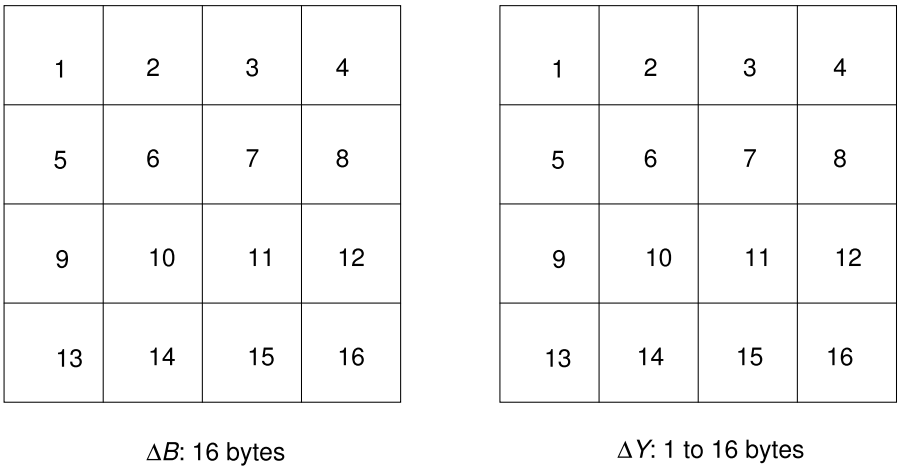

Figure 7.3: Elastic AES Data Block as 4x4 Matrices

AES, the swap step swaps Y with y bits in $\lceil \frac{y}{8} \rceil$ sequential bytes from B, moving the starting position one byte to the right in B each round and wrapping around to the beginning of B as needed. Therefore, when computing the differentials, a column of ΔY is swapped with a column of ΔB. If the last byte in ΔY is a partial byte, the bits are swapped with the most significant bits of the corresponding byte in ΔB.

The column deltas are tracked through each step of the round function and through the swap step, with all possible resulting cases explored recursively. The following is an overview of how the tracking is performed. An input state consisting of the number of delta bytes per column for both ΔBin_1 and ΔY_1 along with the size of Y in bytes are input to the model. The model determines all possible states of $\Delta Bout_1$ resulting from AES's round function then, for each of these states, determines all possible states of $\Delta Bin_2 || \Delta Y_2$ resulting from the swap step. These states become inputs to the next round. The specific impacts of each step are modelled as follows:

AES's Round Function:

- SubBytes: While it is the S-Box that produces the 2^{-6} bound for the round function, the S-Box has no impact on the size of the column delta. This is because the model is

not tracking the actual delta value but only the number of bytes with a delta. Each delta byte input to the SubBytes step produces a delta byte in the output of this step and does not alter the position of the delta byte.

- ShiftRows: This step causes each byte in the 2^{nd} through 4^{th} rows to change columns. Since the number of delta bytes per column and not their exact positions are tracked (*i.e.* the exact row number is not tracked), this step leads to multiple cases to investigate. For example, if one column has a single delta byte and the other three columns have zero deltas, the single delta byte may be in any of the four columns after ShiftRows is applied because the row number is not tracked.
- MixColumns: Given the number of delta bytes in a column when entering MixColumns, all possible states that can result are investigated. If there are x delta bytes in a column at the start of MixColumns, there can be anywhere from $5-x$ to 4 delta bytes in the column at the end of MixColumns because the specific delta values input to MixColumns are not known. For example, two delta bytes in a column will result in either 3 or 4 delta bytes in the output column.
- AddRoundKey: The whitening steps have no impact on the number of and positions of the delta bytes. Any whitening step cancels with itself when determing the difference between two inputs or between two outputs of a round.

Swap Step:

The impact of XORing a column of ΔY with a column of ΔB and swapping the results is determined. Given that the specific positions of the delta bytes are not tracked, the swap step may lead to more than one state and all possible resulting states are investigated. For example, suppose a column of ΔB has a delta of 2 after AES's round function is applied, the swap impacts all bytes and the corresponding column in ΔY has a delta of 1. Then the resulting column in ΔB can have a delta of 1,2 or 3 bytes and the resulting column in ΔY will have a delta of 2 bytes. This is because the entire two byte delta in ΔB is swapped out into the next ΔY. The single delta byte in ΔY may be XORed with one of the delta bytes in ΔB during the swap step and result in either a delta or no delta in this byte, or the single delta byte in ΔY may be XORed with a non-delta byte in ΔB. A second example illustrates the cases where the swap does not impact all bytes of the column of ΔB (this

will happen to one or more columns of ΔB when $y \leq 120$). If only one byte of a column in ΔB is impacted by the swap, the column in ΔB has a delta of 2 and the byte from ΔY being swapped into the column is a delta byte, the resulting columns in ΔB and ΔY may have deltas of 1 and 1, 2 and 1, or 3 and 0 bytes, respectively. Since the exact delta values are not known, the model assumes that any delta byte in ΔY which is XORed with a delta byte in ΔB may or may not produce a zero (a non-delta byte), resulting in multiple cases to explore.

Even with the multiple cases to investigate in ShiftRows, MixColumns and the swap step, the amount of computation required is lower than what is required to track the exact position of each delta byte. The multiple cases to investigate due to MixColumns can be avoided only if the exact delta for each byte along with each delta byte's exact position was tracked, which requires computing all 2^{128+y} differentials. As a result of tracking the deltas only at the column level, the model will investigate states which cannot be reached. The inclusion of these unattainable states is not an issue because they can only lower *exp* and thus reduce the tightness of the bound.

7.4.2 Results

I ran the model for each case where Y contains an integral number of bytes from 1 to 16. These cases cover all values of y for $0 < y \leq 128$. Since I am only tracking how many byte positions involved non-zero deltas, the lower bound for $y = 8x$ where x is an integer such that $1 \leq x \leq 16$ covers the cases of y such that $8(x-1) < y \leq 8x$. For example, the lower bound for when Y contains exactly one byte ($y = 8$) is also the lower bound for values of y in the range of 1 to 7 because this range of y influences exactly one byte in B during each of the swap steps.

For each case, the model was run through three rounds for all possible input states except for the state involving $\Delta Bin_1 = 0$ because this produces a first round probability of 1 and involves the same analysis as the cases where $\Delta Bin_1 \neq 0$ and $\Delta Y_1 = 0$, only with one less round. States producing a three round bound which did not exclude the possibility of a differential attack were traced through subsequent rounds with the number of rounds depending on the exact size of Y and the probability produced after each round.

An example of the my analysis is provided in Section 7.4.3. For each case (value of Y tested), I found that the total *exp* across $r' - 1$ rounds is at least $128 + y$. Recall that r' depends on the size of Y. Therefore, the probability of a specific differential characteristic occurring is $\leq 2^{-128-y}$. The probability was computed across $r' - 1$ rounds instead of r' rounds to cover the case of the first round producing a differential characteristic with a probability of 1 when $\Delta Bin_1 = 0$ and $\Delta Y_1 \neq 0$ due to the key-dependent permutations being omitted.

The model was run for only three rounds for all input states because of the time required to compute the trace for subsequent rounds for all possible input states. Performing the trace over four or five rounds resulted in the program running up to three hours in some cases. While this is not an infeasible amount of time for a few cases, it is infeasible when testing all cases. By viewing a byte has having either a zero or non-zero delta, there are 2^{32} possible combinations when dealing with a 32-byte block.

Claim 7.2. *In the elastic version of AES with the block size ranging from 129 to 224 bits, any three round differential characteristic occurs with probability* $\leq 2^{-30}$.

Proof. For the cases where Y contains at most 12 bytes, the total *exp* across three rounds is ≥ 30. This was determined via the model for the cases in which $\Delta Bin_1 \neq 0$ (although it can easily be determined manually). In the case where $\Delta Bin_1 = 0$, *exp* is also at least 30 for the first three rounds regardless of the size of Y. This is because the $\Delta Bin_2 = \Delta Y_1$. If $1 \leq |\Delta Y_i| < 5$ and $\Delta Bin_1 = 0$, the delta output from the second application of the round function will be at least $5 - |\Delta Y_1|$ due to the ShiftRows and MixColumns states in the round function, and $|\Delta Bin_2| + |\Delta Bin_3| \geq 5$. If $|\Delta Y_1| \geq 5$, then $|\Delta Bin_2| \geq 5$. Therefore, the total *exp* across rounds two and three is at least 30 $(6 * (|\Delta Bin_2| + |\Delta Bin_3|))$. □

For block sizes between 225 and 256 bits, I obtain the following result:

Claim 7.3. *In the elastic version of AES with the block size ranging from 225 to 256 bits, any three round differential characteristic occurs either with probability* $\leq 2^{-30}$ *or with probability* 2^{-12}. *If the three round probability is* 2^{-12}, *the five round probability is* 2^{-132}.

Proof. For the block sizes where Y involves 13 or more bytes, the model produces a three round $exp \geq 30$ except in the scenario depicted in Figure 7.4. In this exception case, a three

round $exp = 12$ is possible, but leads to a five round $exp = 132$. This five round exp is large enough such that when it is combined with the other possible three to five round exp values it produces a total $exp \geq 128 + y$ for $r' - 1$ rounds. The three round $exp = 12$ can be produced by having a single byte delta in the 4^{th} row of the 4-by-4 matrix representing B. Let a denote the value of this single byte delta. Let a' be the value after SubBytes is applied. This will produce delta of 4, d, in a single column as a result of MixColumns. Suppose Y contains a delta which is entirely in the column which will be swapped with the 4-byte delta output from the round function. Also assume ΔY and d are identical in the three bytes in rows 1 to 3, and produce a delta byte $= a$ in the byte in the 4^{th} row. Furthermore, assume the two bytes producing this difference of a are such that the SubBytes step will produce the same delta, a', as it did in the first round. This results in the input to the second round function involving a single delta byte with value a which is identical to the delta input to the first round function, with the exception that it will be in a different column. Since a is in the fourth row, it will be rotated 3 columns to the left during ShiftRows. The starting position of the swap step moves one byte to the right each round. In the second round, ΔY contains the 4 bytes swapped out of the first round function's output. These 4 bytes are in a single column whose value is $a', a', 3a', 2a'$. The delta output from the second round function will also be a single column containing $a', a', 3a', 2a'$. Thus the swap step after the second round will result in no delta input to the third round function and a ΔY of 4 bytes in a single column. Therefore, a 4-byte delta in a single column is input to the 4^{th} round, Y contains no delta in the 4^{th} round, a 16-byte delta is output from the 4^{th} round and a 16-byte delta is input to the 5^{th} round.

The exception case occurs only when Y is at least 13 bytes because it requires that Y contains enough bytes to impact an entire column of B in the swap step. 12 bytes in Y populate three rows entirely (3 bytes in every column). □

7.4.3 Example: Differential Bounds for the Elastic Version of AES with 152-Bit Block Size

As an example of how the output of the model was used to calculate the differential bounds for the elastic version of AES, I include here the results from the case of Y containing 3

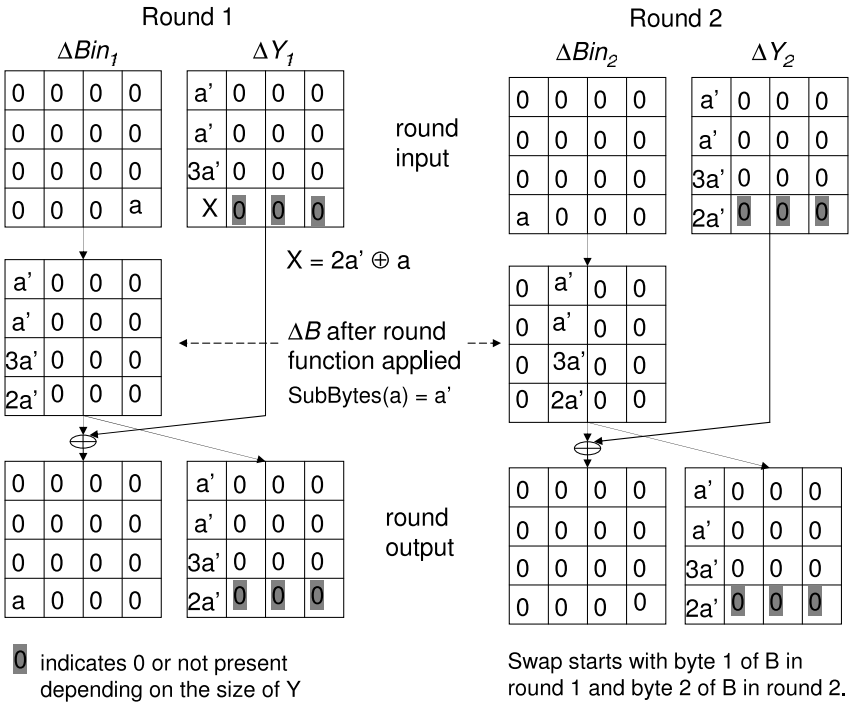

Figure 7.4: Elastic AES Differential: Case with Three-Round $exp = 12$ and Five-Round $exp = 132$

bytes. When Y is 3 bytes, the total block size is 152 bits (19 bytes) and twelve rounds are applied. I compute the bound for eleven rounds to accommodate the case when $\Delta Bin_1 = 0$. Notice that a differential attack is not possible if each series of three rounds has a $exp \geq 54$ because this produces a $exp \geq 162$ across nine rounds. States producing a three round bound ≤ 48 were evaluated through four or five rounds as needed. The resulting bounds for exp are indicated in Table 7.1. For simplicity, I do not list cases in the table with a three round $exp \geq 48$ and a four round $exp \geq 152$, which precludes a differential attack.

If a total $exp \geq 152$ occurs in eleven or fewer rounds, a differential attack cannot occur. To bound the value of exp, I consider what happens based on the number of three round sequences that have an $exp \geq 54$ and fill in the remaining rounds with the values from Table 7.1. Specifically, I view the eleven rounds as three, four and five round sequences, such as (three rounds, four rounds, four rounds). As I previously stated, three sequences of three rounds with $exp \geq 54$ in each sequence eliminates a differential attack. If two

3 Round *exp*	4 Round *exp*	minimum 5 Round *exp*
30	54	144
42	60	126
42	84	132
42	108	not computed
48	66	132
≥ 54	not computed	not computed

Table 7.1: Elastic AES Differential: Minimum *exp* Across 4 and 5 Rounds when Y is 3 Bytes

series of three rounds have an $exp \geq 54$, the minimum *exp* for four rounds in the remaining five rounds is 54; therefore, the total *exp* across ten rounds is ≥ 162. If one series of three rounds has an $exp \geq 54$, there are two remaining series of four rounds, each with an $exp \geq 54$ and the total *exp* across eleven rounds is ≥ 162. If each series of three rounds has a $exp \leq 48$, then there is either a five round $exp \geq 126$, in which case any other four or five round *exp* from Table 7.1 will produce a nine or ten round $exp \geq 234$, or there is a four round $exp \geq 108$ and any two three-round *exp* values combined with the 108 will produce a $exp \geq 168$ in ten rounds. In all cases, the total *exp* exceeds 152. Thus the probability of a specific differential characteristic occurring is $< 2^{-152}$ when Y is 3 bytes, implying a differential attack is not possible for a 152-bit block size.

7.5 Elastic MISTY1 Differential Bounds

In MISTY1, an upper bound of 2^{-56} on the probability a differential characteristic occurs was derived for 4 cycles of the 64-bit version [Mat96]. This is a result of a bound of 2^{-14} per cycle due to the $F0$ function. Using an analysis of state transitions and only the bound for the $F0$ function, I derive an upper bound on the elastic version of MISTY1 of $2^{-14(r'-1)}$, where r' is the number of rounds (cycles of MISTY1) in the elastic version. The bounds for each block size are shown in Table 7.2. This bound is not tight and does not by itself eliminate the possibility of a differential attack (either in MISTY1 or the elastic version).

However, the state transition analysis does reduce the number of state sequences that need to be investigated to tighten the bound over r' rounds. The bound of $2^{-14(r'-1)}$ also allows the potential contribution needed from the initial and final key-dependent mixing steps in preventing differential attacks to be determined. I evaluate the elastic version of MISTY1 with the swap step defined to have the starting position of the y bits selected to be swapped out at the end of a round alternating between the left and right halves.

Rounds r'	Block Size	Bound $14(r'-1)$
5	65 to 80	2^{-56}
6	81 to 96	2^{-70}
7	97 to 112	2^{-84}
8	113 to 128	2^{-98}

Table 7.2: Elastic MISTY1 Differential Bounds Based on $F0$ Function

Notation:

- FL and $F0$ refer to the functions within a cycle of MISTY1.
- $b+y$ is the variable-length block. $b = 64$, is the block length in MISTY1. $0 \leq y \leq 64$.
- A single round will be defined as the application of FL to each half of the leftmost 64 bits of data and two applications of $F0$. The swap step will occur after the second application of $F0$ as indicated in Figure 7.5.
- Round input and output refers to the inputs and outputs of MISTY1's round function unless otherwise stated.
- Δ indicates the difference between two bit strings.
- ΔLin and ΔRin refer to the Δ for the left and right halves of input to the round function, respectively.
- $\Delta Lout$ and $\Delta Rout$ refer to the Δ for the left and right halves of output from the round function prior to the swap.

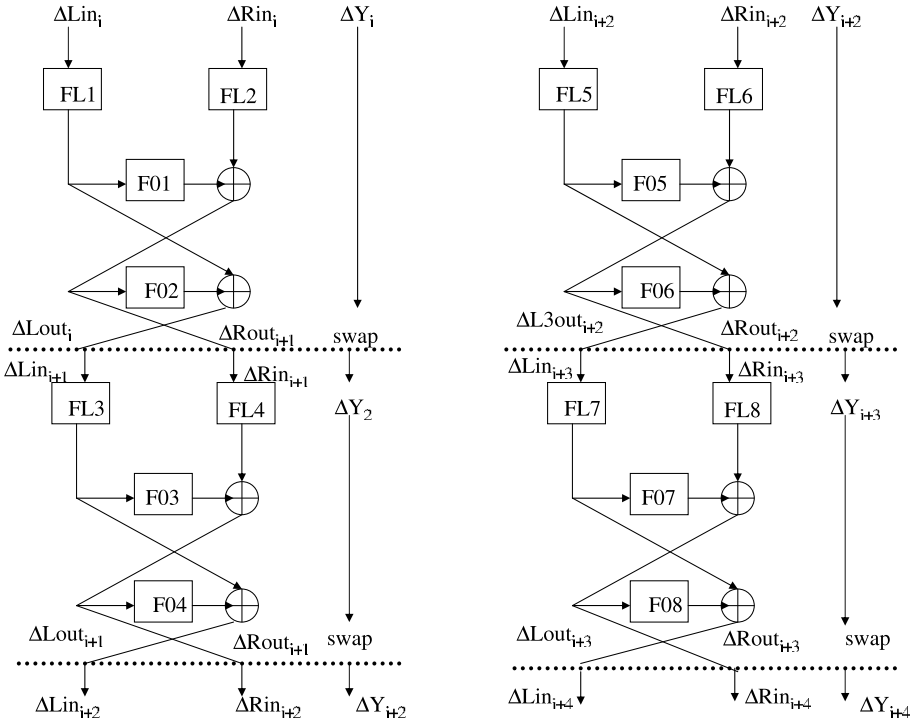

Figure 7.5: Differentials for Elastic MISTY1

- ΔY refers to the Δ for the y bits left out of a round.
- A subscript of i after any Δ indicates the i^{th} round.
- exp is used as it was in the description of the elastic AES differential cryptanalysis. It is the absolute value of the exponent in the probability that a differential characteristic occurs, *i.e.* 2^{-exp}.
- $\Delta F0(\Delta X)$ for $\Delta X = x1 \oplus x2$ refers to $F0(x1) \oplus F0(x2)$.
- $\Delta FL(\Delta X)$ for $\Delta X = x1 \oplus x2$ refers to $FL(x1) \oplus FL(x2)$.
- $||$ indicates concatenation. For example, $\Delta Lin||\Delta Rin$ is the Δ of the leftmost b bits input to a round.

Figure 7.5 shows four consecutive rounds of elastic MISTY1 with the differentials for the inputs and outputs of each round labelled. The applications of the FL and $F0$ functions

are numbered in order to be easily referenced later.

Assumptions:

- The probabilities for per round differential characteristics in the elastic version of MISTY1 are independent of other rounds. Specifically, the probability $F0$ produces a specific differential in round $i+1$ is independent of the differential produced by $F0$ in round i. This assumption is a result of the expanded key bits being pseudorandom. In the original version of MISTY1, the expanded key bits have a known relationship to each other and this assumption does not hold.

Facts:

- Fact 1: The probability a specific differential characteristic occurs as the output of $F0$ is $\leq 2^{-14}$ [Mat96].
- Fact 2: FL does not influence the probability of a differential characteristic [Mat97].
- Fact 3: A non-zero differential in one half of the round's input impacts both halves of the round's output. This follows directly from the definition of MISTY1.
- Fact 4: If $\Delta L_i = 0$ and $\Delta R_i \neq 0$, the probability of a differential characteristic occurring in round i is $\leq 2^{-14}$ because there is a non-zero delta input only to the second $F0$ in the round.
- Fact 5: If $\Delta L_i \neq 0$ and $\Delta R_i = 0$, the probability of a differential characteristic occurring in round i is $\leq 2^{-28}$ because there is a non-zero delta input to both $F0's$ in the round.
- Fact 6: The probability of a differential characteristic occurring in MISTY1 with 4 cycles is $\leq 2^{-56}$ [Mat97].

Claim 7.4. *In the elastic version of MISTY1 as described in Chapter 6, the probability a differential characteristic occurs* $\leq 2^{-14(r'-1)}$. *where* r' *is the number of rounds in the elastic version.*

Proof. I derive the bound using the following steps. First, I define the possible states of ΔL, ΔR and ΔY. There are seven possible states consisting of all combinations of each of the

three components being zero or non-zero, excluding the state in which all are zero. These are the states listed in Section 7.3. Second, I determine the possible transitions between states and the bound on the value of the *exp* for each state and transition. Figure 7.6 shows the states and possible transitions between states. Third, I determine what sequence of states are possible that involve multiple occurrences of state I. The bound is derived by proving that between any two occurrences of state I, at least one state with an $exp \geq 28$ must occur. In the second step of my analysis, I do not determine whether it is possible for the y bits swapped in to result in a delta of zero in either the left or right half of input to the round function, but instead determine bounds assuming this could occur. The bounds are based entirely on whether or not a non-zero delta occurs in the input to one or both applications of the $F0$ function within the round. The bound is $\leq 2^{-28}$ when the delta is non-zero to both applications of $F0$, and is $\leq 2^{-14}$ when there is a possibility that the delta is non-zero to exactly one application of $F0$. When the $\Delta Lin||\Delta Rin = 0$ for a round, the differential does not change in that round. When $\Delta Lin||\Delta Rin \neq 0$, a delta of zero can occur in the input to exactly one of applications of $F0$ for the following reasons:

- $\Delta Lin = 0$, in which case the input to the first $F0$ has a delta of zero and the input to the second $F0$ has a non-zero delta.
- $\Delta Lin \neq 0$, $\Delta Rin \neq 0$ and the XOR after the first application of $F0$ produces a delta of zero, *i.e.*, $\Delta F0(\Delta FL(\Delta Lin)) = \Delta FL(Rin)$. This results in a delta of zero input to the second application of $F0$.

I also note that if the inputs to both applications of $F0$ in a round have a non-zero delta, there is the possibility that the XOR after the second application of $F0$ results in $\Delta Lout = 0$. This requires $\Delta F0$ output from the second application of $F0$ being equal to $\Delta FL(\Delta Lin)$.

The possible transitions between the states shown in Figure 7.6 are indicated in Table 7.4. In Figure 7.6, the value next to each state is the upper bound on the probability any specific differential occurs when considering only the current state and the next state. The sequence of states prior to the current state and the exact block size are not considered at this point in the analysis. Taking these into consideration lowers the bound for some transitions and eliminates other transitions, as seen later in the analysis. In two cases, the

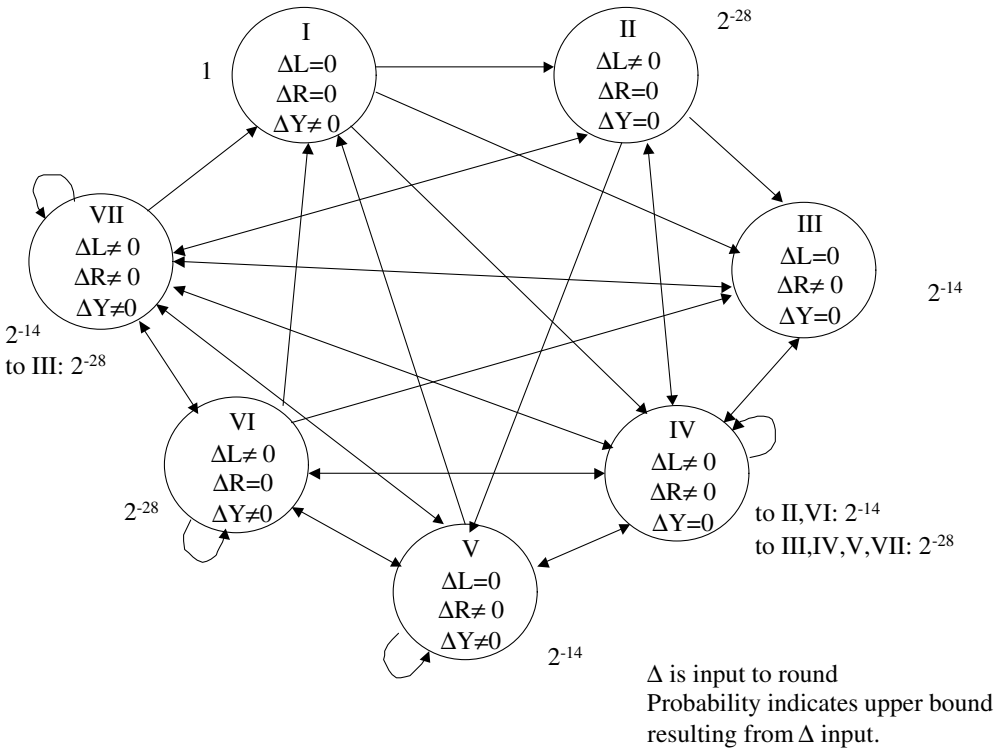

Figure 7.6: Elastic MISTY1 Differential States

bound is lower if a specific next state occurs and is listed separately along with the corresponding next state(s). For example, state VII has an upper bound of 2^{-14} except when the next state is state III, in which case the bound is 2^{-28}. These bounds are summarized in Table 7.3.

The state transitions assume all outcomes are possible in theory. Specifically, in cases where $\Delta Lout \neq 0$ and $\Delta Rout \neq 0$, I make no assumptions on the bits swapped and assume it is possible that the y bits swapped from the round output can produce $\Delta Y = 0$ (except in the case where $y = 64$ since then either all of $\Delta Lout$ or all of $\Delta Rout$ becomes part of ΔY which I omit indicating since it only serves to reduce the bounds for one specific value of y). I note that as y approaches 64, it is unlikely that the majority of bits in $\Delta Lout \parallel \Delta Rout$ are 0, as required to obtain $\Delta Y = 0$ entering the next round. As y increases, it is also unlikely that a non-zero ΔY will result in $\Delta L = 0$ and/or $\Delta R = 0$ in the input to the next round.

State	Bound
I	1
II	$\leq 2^{-28}$
III	$\leq 2^{-14}$
IV	$\leq 2^{-14}$ to states II,VI $\leq 2^{-28}$ to states III,IV,V,VII
V	$\leq 2^{-14}$
VI	$\leq 2^{-28}$
VII	$\leq 2^{-28}$ to state III $\leq 2^{-14}$ to states I,II,IV,V,VI,VII

Table 7.3: Elastic MISTY1 Differential Probabilities per State

State I is the only state for which a differential holds through the round with probability 1.

In the third part of my analysis, I derive the bounds in Table 7.2 by determining what sequences of states are possible. This is done in a series of steps. From the state transition diagram, at least two states other than state I must occur between any two occurences of state I.

In order to show that the total exp across r' rounds is $\geq 14(r'-1)$ for $5 \leq r' \leq 8$, I need to bound how many times state I can occur because state I is the only state with an $exp < 14$ and determine the total exp across the states between any two occurrences of state I. If state I does not occur in r' rounds, the total exp across all r' rounds is at least $14r'$. If state I occurs exactly once in r' rounds, the total exp across all r' rounds is at least $14(r'-1)$. In order to achieve a lower bound of $14(r'-1)$ for any r' rounds with two or more occurrences of state I, it is necessary to show that at least one state between any two occurrences of state I must have an $exp \geq 28$.

Using only the state transition diagram, the following sequences are potential three state sequences that begin with state I:

- Case A. I,II,X for X $\in$ {III,IV,V,VII}
- Case B. I,III,IV

From State	To State
I	II,III,IV
II	III,IV,V,VII
III	IV,VII
IV	II,III,IV,V,VI,VII
V	I,IV,V,VI,VII
VI	I,III,IV,V,VI,VII
VII	all

Table 7.4: Elastic MISTY1 Differential State Transitions

- Case C. I,III,VII
- Case D. I,IV,X for X $\in$ {II,III,IV,V,VI,VII}

In Case A, state II has an $exp \geq 28$. In Case D, for each X, either state IV and/or state X has an $exp \geq 28$. In Case B, I need to consider one more state to guarantee some $exp \geq 28$ occurs. Adding one more state to the sequence results in a sequence of the form I,III,IV, X for X $\in$ {II,III,IV,V,VI,VII}. For each X, either state IV and/or state X has an $exp \geq 28$. Therefore, only Case C needs to be investigated further to conclude that a state with an $exp \geq 28$ occurs before the next (if any) occurrence of state I.

I extend the sequence in Case C by one state and consider the sequence I,III,VII,X for X $\in$ {I,II,III,IV,V,VI,VII}. If X is state II, III, IV or VI then a state with $exp \geq 28$ occurs before state I can potentially occur again. This is because state II and VI both have $exp \geq 28$ and state VI or the state that follows it will have an $exp \geq 28$. If X is state III, state VII will have an $exp \geq 28$. This leaves the following sequences to be investigated:

- Case C1: I,III,VII,I
- Case C2: I,III,VII,V
- Case C3: I,III,VII,VII

I will show that each case cannot occur if each occurrence of state VII has an $exp < 28$ (*i.e.* the bound on exp is ≥ 14 as opposed to ≥ 28). I note that it may not even be possible for

each case to occur even if state VII occurs with an $exp \geq 28$; however, to prove the claim, I only need to show at least one state has an $exp \geq 28$ in each sequence.

Using Figure 7.5 to represent the four states, the following values can be assigned to the inputs and outputs of the first three rounds:
Round 1 corresponds to state I:

- $\Delta Lin_i = 0$
- $\Delta Rin_i = 0$
- $\Delta Y_i \neq 0$
- $\Delta Lout_i = 0$
- $\Delta Rout_i = 0$

The swap at the end of round 1 must move any non-zero portion of ΔY_i into ΔRin_{i+1}.
Round 2 corresponds to state III:

- $\Delta Lin_{i+1} = 0$
- $\Delta Rin_{i+1} \neq 0$
- $\Delta Y_{i+1} = 0$
- $\Delta Lout_{i+1} = F04(FL4(\Delta Rin_{i+1}))$
- $\Delta Rout_{i+1} = FL4(\Delta Rin_{i+1})$

Because I defined the swap step such that the starting position of the y bits swapped out after each round alternates between the left and right halves of the round function's output, at least some of $\Delta Lout_{i+1}$ will form ΔY_{i+2}.
Round 3 corresponds to state VII with $exp < 28$:

- $\Delta Lin_{i+2} = \Delta Lout_{i+1} = \Delta F04(\Delta FL4(\Delta Rin_{i+1}))$
- $\Delta Rin_{i+2} = \Delta Rout_{i+1} = \Delta FL4(\Delta Rin_{i+1})$
- $\Delta Y_{i+2} \neq 0$ and contains at least part of $\Delta Lout_{i+1}$

- $\Delta Lout_{i+2} = \Delta FL5(\Delta F04(\Delta FL4(\Delta Rin_{i+1})))$
- $\Delta Rout_{i+2} = 0$

In order for state VII to have an $exp < 28$, the output of $F05$ XORed with the output of $FL6$ must produce an delta of zero entering $F06$. This results in the $\Delta Lout_{i+2}$ and $\Delta Rout_{i+2}$ values indicated. The swap at the end of round 3 will impact at least the right half, resulting in $\Delta Rin_{i+3} \neq 0$. Therefore, state VII with an $exp < 28$ cannot transition to state I in this sequence and Case C1 cannot occur.

The next round also cannot be state V because the swap will not produce a $\Delta Lin_{i+3} = 0$. If the span of ΔY_{i+2} is not large enough to impact ΔLin_{i+3} then $\Delta Lin_{i+3} \neq 0$ and state V cannot occur. If the span of ΔY_{i+2} is large enough to impact ΔLin_{i+3}, then $\Delta Lin_{i+3} = \Delta Y_{i+2} \oplus \Delta Lout_{i+2} \neq 0$. This is because ΔY_{i+2} will contain some or all of $\Delta Rout_{i+1}$, which equals $\Delta FL4(\Delta Rin_i + 1)$. It is this value that will be XORed with $\Delta Lout_{i+2}$ to form ΔLin_{i+3}. However, $\Delta Lout_{i+2} = \Delta FL5(\Delta F04(\Delta FL4(\Delta Rin_{i+1})))$. Since $F0$ influences the probability of the differential and FL does not, $\Delta Lout_{i+2} \oplus \Delta Rout_{i+1} = \Delta FL5(\Delta F04(\Delta FL4(\Delta Rin_{i+1}))) \oplus \Delta FL4(\Delta Rin_{i+1}) \neq 0$. Therefore, Case C2 cannot occur.

So far I have shown cases C1 and C2 cannot occur. This leaves Case C3 (round four is state VII) to consider.

Round 4 corresponds to state VII with $exp < 28$:

- $\Delta Lin_{i+3} \neq 0$
- $\Delta Rin_{i+3} \neq 0$
- $\Delta Y_{i+3} \neq 0$ and its span must be large enough to contain at least part of $\Delta Rout_{i+2}$, which is 0 from the output of round 3.
- $\Delta Lout_{i+3} = FL7(\Delta Lin_{i+3})$
- $\Delta Rout_{i+3} = 0$

At the end of round 3, $\Delta Rout_{i+2} = 0$ if $exp < 28$ in round 3. This results in the part of ΔY_{i+3} that impacts the left half in the swap at the end of round 4 being 0. Since $\Delta Lout_{i+3} \neq 0$, the swap step at the end of round 4 cannot lead to state I. In general, any

sequence of n state VII's for $n \geq 2$, cannot lead directly back to state I if each occurrence of state VII has $exp < 28$ because $\Delta Lout_n \neq 0$ and ΔY_n is 0 (due to $\Delta Rout_{n-1} = 0$ in the part that will impact the ΔL portion of the n^{th} swap. As a result, $\Delta Lin_{n+1} \neq 0$ and the $(n+1)^{st}$ state cannot be state I. Therefore, state VII must transition to some state X $\in \{II, III, IV, V, VI\}$ before state I can occur and the analysis from Cases A,B,C and D applies (from the point after state I), which implies a state with an $exp \geq 28$ must occur at some point in the sequence before state I can be reached.

Overall, I have shown that for any sequence of states involving two occurrences of state I, at least one state between the occurrences of state I must have an $exp \geq 28$. Since any state X other than state I has an $exp \geq 14$, the total exp in any sequence of r' states is $\geq 14(r'-1)$, where $5 \leq r \leq 8$ in the elastic version of MISTY1 for block sizes in the range of 65 to 128 bits. □

7.6 Summary

When performing differential cryptanalysis on an elastic block cipher, the differential bound for the round function is unchanged from the original version of the cipher because the round function in unaltered. The swapping of bits between rounds impacts the sequence of differentials entering the series of rounds by altering the output of the i^{th} application of the round function before it is input to the $(i+1)^{st}$ application of the round function. The bound for the round function and the impact of the swap step can be combined to bound the probability a differential charactertistic occurs in the elastic version of a block cipher. This is accomplished by defining states representing whether or not there is a non-zero differential in the b-bit portion or y-bit portion of the round's input, then determining what states may potentially occur as input to each round as a result of the swap step. The possible state sequences in the elastic version of the cipher are combined with the probabilities a differential characteristic occurs in one round of the original cipher to bound the probability of a differential characteristic across r' rounds of the elastic version of the cipher.

I used the elastic versions of AES and MISTY implemented as described in Chapter 6

to illustrate this method. I used a computer model to assist with determining the possible sequence of states in the elastic version of AES. The resulting bounds show that a differential attack on the elastic version of AES is impossible. I note that the upper bound for a differential characteristic on AES precludes a differential attack on AES. In contrast, the existing bounds for MISTY1's round function do not prove a differential attack is impossible. As a result, when using the existing bound for a differential occurring in one round of MISTY1 combined with the possible state transitions in the elastic version, I can only derive an equivalent bound for the elastic version. Specifically I show that, on average, each round contributes a factor that is $\leq 2^{-14}$ to the probability a differential characteristic occurs across all rounds in the elastic version of MISTY1 (which is the same result obtained for MISTY1). Thus, the bound is not sufficient to conclude from it alone that a differential attack on elastic MISTY1 is impossible, just as it does not prove a differential attack on MISTY1 is impossible.

Chapter 8

Key Schedules

8.1 Overview

In this chapter I discuss the choice of key schedules for elastic block ciphers along with general key schedule requirements for any block cipher. In practice, every block cipher includes its own key schedule, which typically is designed with a focus on performance and little concern about the lack of pseudorandomness in the expanded-key bits. This tendency in key schedule design results in key schedules contributing to attacks and forces applications supporting multiple block ciphers to support a separate key schedule for each cipher. When creating elastic block ciphers, I wanted to avoid these disadvantages of existing key schedules. Therefore, I required a key schedule that is independent of the block cipher and that generates pseudorandom expanded keys (or close to pseudorandom) while adhering to a performance bound. Existing stream ciphers are potential candidates for satisfying these requirements. I used RC4 as the key schedule in the elastic block ciphers to illustrate the concept of a generic key schedule satisfying these requirements. I explain how increasing the randomness of expanded-key bits beyond that found in existing block ciphers contributes to the security of block ciphers.

The key schedule for an elastic version of a block cipher has to generate more expanded-key bits than the key schedule of the original block cipher. Additional key bits are needed due to the expansion or addition of whitening steps, the two key-dependent mixing steps and the increase in the number of rounds. As a result, I needed to consider how to create

the key schedules of elastic block ciphers. Options include modifying the original cipher's key schedule to generate additional bits, running the original key schedule multiple times or replacing the key schedule entirely. The first option is unattractive because it requires modifying each cipher's key schedule when creating the elastic versions of ciphers. Unlike the encryption algorithms of block ciphers which follow a somewhat generic structure by being a series of rounds, key schedules vary extensively in their structures. This makes it unlikely a general method can be devised for modifying existing key schedules to generate additional bits. One solution is to use the existing key schedule followed by another algorithm (that could be used for all block ciphers) that generates all additional key bits needed by the elastic version. However, there is no point in keeping the original key schedule in this case because the common algorithm can be used to generate all expanded-key bits, which is my third option. The second option is simplier than the first, but either requires a longer secret key or a way of generating additional "secret" keys from the original key to use in the additional runs of the key schedule. The third option is the simplest of the options because a stream cipher can be used to generate as many key bits as needed and can be used for any block cipher. By using this option, I am able to satisfy the requirements I placed on key schedules in general.

8.2 Key Schedule Requirements

There are two features I required of the key schedule for elastic block ciphers which are not characteristic of the key schedules of existing, fixed-length block ciphers. First, I wanted the key schedule to be a stand-alone algorithm that is usable by any block cipher. This eliminates the need to create a new key schedule for every block cipher, whether it is a fixed-length or variable-length block cipher. Second, I wanted the expanded-key bits to be pseudorandom (or as close to pseudorandom as practical). This requirement is to prevent attacks that exploit the lack of randomness in the output of existing key schedules. In order to satisfy the second requirement, the key expansion will be slower than that in the original block ciphers; however, the performance must still be reasonable. For example, a key schedule which satisfies the first two requirements but which takes a minute to run is

unacceptable. Therefore, I placed a third requirement on the key schedule to bound the performance. The key expansion rate for elastic block ciphers should be a small multiple of the key expansion rate of a standard block cipher. I use AES's key expansion rate as the reference point. As shown later, using RC4 as a key schedule for elastic block ciphers results in a key expansion time of approximately six to eight times that of AES's key schedule for the four example elastic block ciphers created.

The first requirement makes it easier to implement block ciphers because there is no need to support multiple key schedules in applications that must support a set of ciphers. This is especially beneficial to hardware implementations by allowing shared hardware across all the block ciphers supported. Furthermore, the elastic version of a block cipher requires more expanded-key bits than the fixed-length version and the number of expanded key bits needed will depend on the block size. Having a single algorithm that generates as many expanded-key bits as needed avoids modifying each cipher's key schedule when creating the elastic version to create additional key bits based on the block size.

The second requirement is because, ideally, expanded-key bits should be pseudorandom. This contributes to the security of a block cipher by eliminating attacks based on the relationship between expanded-key bits. If an attacker discovers a few expanded-key bits, the attacker should not be able to easily determine other expanded-key bits from them. However, this is not the case in existing block ciphers. Refer to appendices C, D, E, and F for the key schedules of AES, Camellia, MISTY1 and RC6. For example, in AES's key schedule, knowing two expanded-key bits located four bytes apart can immediately expose another key bit. In both MISTY1 and Camellia, the same expanded-key bit is used in multiple locations. All practical attacks based on relationships between the inputs and outputs of the cipher and or individual rounds attempt to recover the round key bits as opposed to directly recovering the original key. Differential fault analysis and side channel attacks also attempt to find expanded-key bits. While practical attacks attempt to recover as many expanded-key bits as possible, they depend on the fact that additional expanded key bits can be determined once a few are found because it is unlikely enough expanded-key bits can be found by using only an attack (and not the key schedule) to allow an exhaustive search on the remaining expanded-key bits. Any 16-byte block cipher that includes initial

and end of round whitening has at least $128(r+1)$ expanded-key bits over r rounds. AES has 1408 expanded-key bits for a 128-bit block. An attack would have to find 1280 of these bits just to reach the point of being equivalent to an exhaustive search on the original 128-bit key if it wasn't for the lack of randomness in AES's expanded key. Linear cryptanalysis (or any algebraic cryptanalysis) may only find equations relating plaintext, ciphertext and a few round key bits. Differential cryptanalysis may only produce a differential that allows recovery of a few round key bits. Furthermore, there are attacks (related key attacks) that arose due to the lack of randonmess in the expanded key. Therefore, eliminating the ability to derive many expanded-key bits from a few known expanded-key bits is beneficial. The second requirement is consistent with the definition of a strong key schedule defined by Knudsen [Knu94], which requires a key schedule with the following two properties:

- Given any s bits of the round keys, it is "hard" to find any of the remaining expanded-key bits from the known s bits.
- Given a relationship between two secret keys, it is "difficult" to predict the relationship between any of the round keys in the expansion of the two secret keys.

Knudsen's definition states that the terms "hard" and "difficult" are to be replaced by more precise definitions depending on the application.

The third requirement is to insure some reasonable performance of the key schedule. The main reason existing key schedules are not designed to produce pseudorandom bits is not because of a lack of algorithms (any existing stream cipher is an improvement over the current key schedules in terms of randomness) but because of performance considerations. Key schedules are designed to be fast at the cost of security. The rate at which a key can be expanded is an issue because of the need to quickly re-key a cipher in certain circumstances. The time to key a cipher is not an issue when key expansion occurs infrequently, such as an application involving a user accessing a database encrypted with one key. At most, the key may be expanded each time the user initiates a session with the database (if the expanded key is not stored) and when the key is changed. In contrast, in network applications the key may need to change frequently. The specific circumstances determine if the key schedule for the block cipher is a significant factor impacting performance. For example,

key agility is an issue when a hardware device is processing multiple streams of data, each using a different key. Key expansion is a performance concern if the expanded keys cannot be stored in the device but instead the device stores the secret keys and expands a key each time it switches what data stream is being processed. Additional concerns on the key agility of block ciphers are discussed in public comments to the AES selection process [WSB00]. In circumstances involving re-keying, the key expansion is less of a concern if the time to expand a key is negligible compared to the time to establish a new secret key. For example, the IETF recommendation for SSH is to re-key a block cipher after every gigabyte or once per hour, which ever occurs first [YL06]. A generalization of this recommends re-keying after every $2^{b/4}$ bytes, where b is the block length of the cipher [BKN06]. The recommendations are due to the probability that identical blocks with the same sequence number will occur in the ciphertext after this many bytes and potentially allow blocks to be replayed or reordered. Furthermore, continuous use of a single key with streaming data readily provides an attacker with enough data blocks for attacks (if any attacks exist) [AB00]. However, in such circumstances, the block cipher's key schedule is not the factor determining the re-keying rate. In protocols that create a key for a block cipher by establishing a shared secret key using public key cryptography, such as Diffie-Hellman or RSA, the public key portion is the limiting factor when re-keying the block cipher.

8.3 Randomness of Key Schedules

Here I discuss the randomness of existing key schedules compared to that of RC4. Refer to Appendix G for the definition of RC4. I chose to use RC4 for the key schedule in the four examples of elastic block ciphers that I created because it demonstrates that the three requirements can be satisfied with a simple, existing algorithm. Other stream ciphers can be used in place of RC4. A fourth benefit of RC4 is that is allows any key of length up to 256 bytes to be expanded, in contrast to block ciphers' key schedules that support a few key sizes, such as 128, 192 and 256 bits. While the existing key schedules are faster than RC4 (refer to Section 8.5), their expanded-key bits are highly predictable when given

just a few expanded-key bits, with the possible exception of RC6. I note that the output of RC4 (a single key stream) can be distinguished from random, but doing so requires a number of bytes well beyond the number of bytes needed for an expanded key in any practical cipher. One test by Golic uses a linear sequential circuit approximation (LSCA) method and requires $64^8/225$ (over 2^{40}) bytes [Gol97]. Another method by Fluhrer and McGrew requires $2^{30.6}$ bytes [FM00]. These methods only distinguish the RC4 key stream from random. They do not allow an attacker to recreate portions of the key stream. There are also weaknesses in the beginning of the key stream. A second byte of all 0's occurs with probability 2^{-7} instead of 2^{-8} [MS01] and, for n key bytes used as the seed ($n \leq 256$), there is a weakness that exists in the first n key stream bytes [FMS01]. Therefore, the first 256 bytes are discarded to cover all sizes of n (with a recommendation that 512 bytes be discarded) [Mir02]. Currently there is no known attack that can generate a portion of the key stream when given a portion of the key stream. This implies there is no known method by which an attacker can generate additional expanded-key bits from known expanded-key bits. If there are x expanded-key bits generated by RC4 (after the first 512 bytes are discarded), as far as an attacker is concerned, the expanded key may be any of the 2^x possible bit strings. I tested the randomness of RC4's output with a subset of the statistical tests used by NIST for testing the randomness of a block cipher's output (some of the tests are not applicable to a stream cipher and thus were not used). The results are included in Appendix A.

In contrast to RC4 and any other stream cipher used in practice, the key schedules of AES, Camellia and MISTY1 generate expanded keys that can be distinguished from random bits. If x is the number of expanded-key bits, there are significantly fewer than 2^x possible expanded keys when using the key schedules of existing block ciphers. In AES, each expanded-key byte is a combination of two other expanded-key bytes. When designing AES, Daemen and Rijmen noted the benefit of pseudorandom key bits, but stated that they took a "less ambitious" approach focused on avoiding symmetry between rounds and attacks due to related keys because "All other attacks are supposed to be prevented by the rounds of the block cipher." [DR02], page 77. In Camellia, there is a large overlap amongst the round keys. In MISTY1, the same expanded-key bits are used in multiple locations

within the cipher. In RC6, it is more difficult to determine key bits from other expanded-key bits compared to AES, Camellia and MISTY1. Each original key byte is altered with an addition and a rotation. The resulting byte is then added to a previous expanded-key byte and a constant to create the next expanded-key byte.

8.4 Key Schedules' Contributions to Attacks

The lack of pseudorandomness in existing key schedules may result in specific attacks aimed at exploiting the key schedules. Related key attacks (such as the slide attack) are examples of such attacks. Related key attacks involve encrypting data with two related keys where the attacker gets to specify the relationship between the (expanded) keys but not the exact keys [CPQ02]. For example, in the slide attack, two keys, $k1$ and $k2$, are chosen such that the expansion of $k2$ consists of $k1$'s expansion slide forward one round. In a differential related key attack, the attacker specifies $k1 \oplus k2$ or the XOR of their expansions. While the attacker, in theory, can request two related expanded keys be used, this will not convey any information about the actual values of $k1$ and $k2$ when using a pseudorandom bit generator as the key schedule. Given an expanded key, each value of $k1$ may have produced it with a probability of approximately $\frac{1}{|k1|}$. A related key attack where the relationship is between the expanded keys (such as a slide attack) becomes meaningless. In order for someone to grant an attacker's request to encrypt plaintexts with the related expanded keys, the expanded keys are not created per the key schedule because there is no way to determine a $k1$ and $k2$ that can be expanded to produce the desired relationship. Thus, encrypting plaintexts in this manner is an artificial exercise. If the relationship between $k1$ and $k2$ is specified and the expanded-key bits are pseudorandom, the resulting expanded keys for both keys will still be pseudorandom and thus will have no discernable relationship between them. Therefore, specifying the relationship between $k1$ and $k2$ will not assist the attacker if the key schedule generates pseudorandom expanded-key bits.

In general, the usefulness of related key attacks and specifically the slide attack, is questionable because, while the key schedules of existing block ciphers are obviously not pseudorandom functions, more recent ones do include steps that prevent two keys from

producing expanded keys where one is the other slide forward one round. For example, the application of rounds of encryption using constants as keys in Camellia's key schedule removes any obvious relationships between two expanded keys. AES's key schedule allows for the initial whitening of one expanded key to be the first round's end of round whitening created from another key by applying the key schedule to a key, $k1$, and setting $k2$ to be the second 128 bits generated from expanding $k1$. Therefore, the "slide" works for one round. The XOR of every 4^{th} word with a constant that varies per round prevents the slide from holding across more rounds. The use of the S-Box in AES's key schedule assists in eliminating any obvious relationship between two expanded keys when the XOR of the 128-bit keys, $k1 \oplus k2$, is specified. RC6's and MISTY1's key schedules also prevent obvious relationships between two expanded keys, including sliding the keys one round, and when expanding $k1$ and $k2$ in the case where $k1 \oplus k2$ is known.

As previously mentioned, using pseudorandom expanded-key bits assists in preventing any attack that attempts to recover expanded-key bits. An attack will likely find only a few expanded-key bits (unless the block cipher has a serious flaw) and rely at least partially on the key schedule to fill in missing bits. Not being able to obtain additional expanded-key bits from a few known expanded-key bits decreases the possibility of a successful attack based on the recovery of round key bits.

8.5 Key Schedule Performance

While the use of RC4 as the key schedule in the elastic block cipher examples satisfies the requirements of being usable by any block cipher and in creating pseudorandom expanded keys, it comes with the cost of being slower than existing key schedules. The following performance measurements are from RC4, and the key schedules of AES, Camellia, MISTY1 and RC6. All conditional statements in AES, Camellia, MISTY1 and RC6 that are present for accommodating key sizes aside from 128 bit keys were removed. For example, AES's key schedule supports 128, 192 and 256 bit keys. The conditional statements were eliminated so they did not impact the time to expand the 128 bit keys. Including them (as would be the case in implementations that support multiple key sizes) will slightly decrease the

performance. The tests were run on a 2.8 Ghz processor with 1MB RAM running the Redhat Linux operating system, version 2.4.22.

The times provided are the times to create 1 million expanded keys. In the tests, a fixed 128-bit key was used and expanded 1 million times by each key schedule. Each test was run ten times and the average of the times is listed.

Cipher	Time to Expand 1 Million Keys (in Seconds)
AES	9.025
Camellia	1.428
Misty1	0.457
RC6	11.259

Table 8.1: Performance of Original Key Schedules

When using the key schedules of the four fixed-length block ciphers, the expanded key corresponded to the fixed sized block of $b = 128$ bits in the case of AES, Camellia and RC6, and to a block of $b = 64$ bits in the case of MISTY1. Three expanded key sizes were considered when using RC4 as the key schedule and the elastic versions of the block ciphers. The sizes correspond to the number of expanded-key bits needed when encrypting $b + y$ bit blocks for $y \in \{0, 8, b\}$. In the elastic block cipher, two expanded-key bytes were used in each of the two key-dependent permutations (for a total of 32 bits). When $y = 0$, the number of expanded-key bits needed is 32 more than that needed for the original cipher plus any whitening added that was not present in the original cipher. $y = 0$ provides a comparison between using RC4 and the original key schedules when working on the fixed-length b-bit block (with the minor overhead of creating four additional bytes with RC4 for the two permutations). $y = 8$ shows the impact of adding one additional round key and an extra byte of whitening per round compared to when $y = 0$. $y = b$ shows the impact of doubling both the number of round keys and the number of expanded-key bits used for whitening compared to when $y = 0$.

Table 8.2 shows the number of expanded-key bytes needed for each of the four block

ciphers for each block size tested. In all calculations, the first term is the number of whitening bytes and the "+4" is for the key-dependent permutations. Let r be the number of rounds in the original version of the cipher, let r' be the number of rounds in the elastic version. Elastic AES requires expanded-key bits only for the whitening and key dependent mixing steps, which equates to $(r'+1)(b+y)+32$ bits. Elastic RC6 requires $(r'+1)(b+y)+64r'+32+128$ expanded-key bits. The 64 bits per round are used internal to the round function. The 128 bits are for the initial and final "addition" steps in RC6. The 32 bits are for the key-dependent permutations. Aside from the whitening and key-dependent permutations, MISTY1 and Camellia require expanded-key bytes within the round function as well as in their FL functions (both have a function called "FL", these are not identical). Each round in the elastic versions of Camellia and MISTY1 corresponds to a cycle in the fixed-length version. The elastic versions will require $(r'+1)(b+y)+32$ expanded-key bits for the whitening steps and key-dependent mixing steps, plus the key bytes needed for the round function and FL function in each. In Camellia, the FL function is applied after every third cycle except for the last cycle. 16 bytes of expanded key are required for each such cycle (two applications of the FL function, one per half). Each cycle involves two applications of the F function, which takes an 8-byte key each time. In MISTY1, the F0 function takes 14 bytes and is applied twice per cycle. (FO takes one 8-byte and one 6-byte key for a total of 14 bytes). The FL function takes 4 bytes and is applied twice per cycle, plus at the end of the last round. Therefore, the number of expanded-key bytes needed for the F0 and FL functions in MISTY1 is 8 plus 36 times the number of cycles.

Three variations of the elastic block cipher's key schedule (*i.e.* RC4) were tested to measure the impact of discarding the first 512 bytes and the impact of re-initializing RC4's "S" array. The rates for the key expansion are shown in Table 8.3.

- Case 1: The first 512 bytes are discarded and the "S" array is re-initialized for each expanded key.
- Case 2: The first 512 bytes are discarded once. The "S" array is not re-initialized for each expanded key (*i.e.,* one continuous key stream is generated).
- Case 3: The first 512 bytes are not discarded. The "S" array is re-initialized each time.

Cipher	Block Size in Bytes	# of Rounds (or Cycles)	Calculation	# of Expanded-Key Bytes
AES	16	10	16*11 + 4	180
AES	17	11	17*12 + 4	208
AES	32	20	21*32 + 4	676
Camellia	16	9	16*10 + 4 + 9*16 + 2*16	340
Camellia	17	10	17*11 + 4 + 10*16 + 2*16	383
Camellia	32	18	32*19 + 4 + 18*16 + 5*16	980
MISTY1	8	4	8*5 + 4 + 4*36 + 8	196
MISTY1	9	5	9*6 + 4 + 5*36 + 8	246
MISTY1	16	8	16*9 + 4 + 8*36 + 8	444
RC6	16	20	16*21 + 20*8 + 16 + 4	516
RC6	17	21	17*22 + 21*8 + 16 + 4	562
RC6	32	40	32*41 + 40*8 + 16 + 4	1652

Table 8.2: Number of Expanded-Key Bytes Needed

Case 1 represents how RC4 is used as the key schedule in the implementations of the elastic block ciphers. Case 2 reflects the use of RC4 as a key stream generator, outputting a continous stream of bytes that are used as expanded keys without re-seeding RC4. This is not realistic of how RC4 would be used as a key schedule but indicates the best rate possible, with the overhead of discarding the first 512 bytes once. Case 3 is intended for comparison to Case 1 and indicates the overhead due to discarding the first 512 bytes.

I use AES's key schedule as a reference point because, since it is a standard, the time it takes for re-keying AES is sufficient in practice. I remind the reader that the time given for AES's key schedule is for a version in which the conditional statements used for supporting 192 and 256-bit key sizes were removed. I compare the time it takes to generate the expanded-key bytes in the elastic versions of the ciphers for b-bit blocks ($y = 0$) to that of the original cipher's key schedule. Let t_i be the time it takes to expand a key in

Cipher	Expanded Key Bytes	Case 1	Case 2	Case 3
AES	180	53.632	9.628	25.468
AES	208	55.069	11.117	26.964
AES	676	80.202	36.113	52.033
Camellia	340	62.179	18.197	34.048
Camellia	383	65.298	21.368	37.246
Camellia	980	96.224	52.447	68.345
MISTY1	196	54.494	10.504	27.028
MISTY1	246	57.173	13.173	28.880
MISTY1	444	67.732	23.785	38.966
RC6	516	70.763	26.704	42.602
RC6	562	73.257	29.172	46.081
RC6	1652	131.53	87.402	103.397

Table 8.3: Key Expansion Times in Seconds Using RC4

the original cipher, for i = 1,2,3,4 corresponding to AES, Camellia, MISTY1 and RC6, respectively. The key expansion rates for the four elastic block ciphers for b-bit blocks are shown in Table 8.4.

I note that Camellia and MISTY1 have the fastest key schedules of the four ciphers and also require the most expanded-key bits, thus resulting in the elastic version's key schedule appearing to be significantly slower. However, Camellia and MISTY1 have the key schedules with the least amount of randomness of the four ciphers due to reusing expanded-key bits in multiple locations. When the RC4 based key schedule is compared to AES's key schedule, the rate is $6.89t_1$ for elastic Camellia and $6.09t_1$ for elastic MISTY1. The rate for Elastic RC6 is $7.84t_1$. Overall, the time to generate the expanded key when encrypting 16-byte blocks of data is just under six to just under eight times the speed of AES's key schedule.

Elastic Cipher	Block Size	Case 1 Rate vs Original Cipher's Rate	Case 1 Rate vs AES Rate
AES	16	$5.94t_1$	$5.94t_1$
Camellia	16	$43.54t_2$	$6.89t_1$
MISTY1	8	$119.24t_3$	$6.09t_1$
RC6	16	$6.29t_4$	$7.84t_1$

Table 8.4: Key Expansion Rates

8.6 Summary

Overall, creating a generic, standalone key schedule for block ciphers that outputs (almost) pseudorandom bits offers advantages over existing key schedules. Only one key schedule implementation is required in applications supporting multiple block ciphers. Pseudorandom key bits enhance the security of block ciphers. Using an existing stream cipher is one option for creating such a key schedule. RC4 was used in the four elastic block ciphers. It serves as a generic key schedule that generates pseudorandom bytes (when limited to the number of bytes required in any expanded key) while performing at a rate which is a small multiple of AES's key schedule rate. The one disadvantage of a generic key schedule is that if a weakness is discovered in the key schedule that can be exploited independently of what block cipher is being used, then any block cipher using the key schedule will be impacted. However, having one key schedule decreases the likeliness of overlooked design flaws and implementation errors compared to when multiple key schedules are required.

Chapter 9

Application: Database Encryption

9.1 Overview

One application of elastic block ciphers is the encryption of databases. Using variable-length blocks can reduce the amount of ciphertext by eliminating padding when encrypting a database. Due to data storage being inexpensive, in general, reducing the size of an encrypted database may not be viewed as a significant benefit in terms of required storage capacity. However, using an elastic block cipher that is computationally faster than a fixed-length block cipher with padding can reduce querying times by reducing the time required for decryption. Also, reducing the database size reduces backup and retrieval times. The level at which encryption is performed on a database depends on the application and the implementation. Obviously, encrypting the entire database as a single file is not an appropriate approach because the entire file must be decrypted when querying or updating the database. Encryption must be done at a more granular level to allow for efficient processing of queries and updates. This may involve encrypting at the row, column or field level. When a query is performed, only the relevant rows, columns or fields need to be decrypted in order to return the result. When updating an entry, only the row, column or field being updated needs to be decrypted, altered then encrypted. Another approach is to encrypt segments of tables. The segment containing the data being accessed is determined and only the segment decrypted as opposed to decrypting an entire table. This approach is beneficial compared to encrypting an entire table as a single component when the query

can be isolated to a specific segment.

Encrypting a database at the row, column or field level with a fixed-length block cipher can easily increase the amount of padding required compared to encrypting entire tables or the entire database as a single component. This is because a database is, as one would expect, designed to fit the application. The type of data stored, the table layouts and size of fields are based on the application, not around the need to break data into 16-byte segments in order to encrypt it. As examples, consider databases a consumer queries when accessing online banking, brokerage and credit card accounts, and databases storing customer information for online retailers. The fields are seldom an integral number of 16 bytes. The online transaction history for Mastercard customers with consumer accounts consisted of the following information in 2005: account number: 16 bytes; sales date: 10 bytes; post date: 10 bytes; description: maximum of 46 bytes; amount: 8 bytes. If this is a row of a database, each row is 90 bytes if the description field is a fixed 46 bytes. Therefore, 6 bytes of padding are required when using a 16-byte block cipher and encrypting at the row level. As a result, 6.25% of the ciphertext is due to padding. If the fields are encrypted, an elastic block cipher eliminates 2 bytes per row by eliminating the padding in the description field if it is a fixed-length field and eliminates between 0 to 15 bytes per row if it is a variable-length field. Typical customer information stored with any account includes name, address and phone number. Table 9.1 indicates the length in characters for such fields in a database from a Schwab brokerage account and a Mastercard credit card account in 2005.

When a 128-bit block cipher is used and individual fields are encrypted, only one of the values greater than 16 bytes (the street address field in the credit card information) is an integral number of 16 bytes. Using an elastic block cipher with a range of 16 to 32 bytes will eliminate the padding required for the street address (for the brokerage account), city, and email address fields. While the savings obtained using an elastic block cipher are small per individual record in a table, the amount is significant when considering millions of records and numerous other fields storing information for holdings and transactions. In the next section, I consider one database example in detail and compute the amount of padding avoided by using an elastic block cipher in place of a fixed-length block cipher when encrypting at the row level and when encrypting at the field level.

	Length in Characters	
Field	Brokerage	Credit Card
street address	62	48
city	31	20
state	2	2
zip code	9	9
phone number	10	10
email address	31	42
data of birth	10	10
account number	9	16

Table 9.1: Sample Database Field Sizes

9.2 Example: Online Bookstore

In order to illustrate the space savings obtained by using a variable-length block cipher to eliminate padding, I consider a database for an online bookstore from a database benchmark [Tra02]. The benchmark provides specifications of database layouts and sizes for different types of applications. The tables in the example are representative of the information Barnes and Noble, and Amazon.com maintain. The tables described here are those containing customer information, including contact and credit card information, and customer orders. There are separate tables in the benchmark for the merchandise information that is displayed to customers shopping on the web site. These tables are not described here because they do not contain confidential information and can be stored unencrypted.

There are eight tables containing information about the customers and their orders. The following is a list of the tables along with their sizes and fields. The size of each field in bytes is listed after each field name. An '*' after a field indicates a variable-length field, in which case the size listed is the maximum length allowed. All other fields are fixed-length. The sum of bytes per field is calculated using the maximum value for variable-length fields. The number of entries in each table is indicated by a size factor that is a constant times the scale factor (sf) or item scale factor (isf). The size of a table is the number of bytes per

row multiplied by the size factor. The first field in each table is the key.

- address table:
  - size factor: 2sf
  - bytes per row: 154
  - fields: id 10, street1* 40, street2* 40, city* 30, state* 20, zip* 10, country 4
- author table:
  - size factor: isf/4
  - bytes per row: 580
  - fields: author id 10, first name* 20, last name* 20, middle name* 20, date of birth 10, bio* 500
- cc xacts table:
  - size factor: 0.9sf
  - bytes per row: 123
  - fields: order id 10, credit card type* 10, credit card number 16, name* 31, expiration date 10, auth id 15, amount 8, auth date and time 19, country 4
- country table:
  - size factor: 92 entries
  - bytes per row: 45
  - fields: id 4, name* 15, exchange 8, currency* 18
- customer table:
  - size factor: sf
  - bytes per row: 248

  - fields: id 10, user name* 20, password* 20, first name* 15, last name* 15, address id 10, phone* 16, email* 50, since date 10, last visit date 10, login date and time 19, expiration date and time 19, discount 8, balance 8, ytd payment 8, birthdate 10

- item table:
  - size factor: isf
  - bytes per row: 851
  - fields: id 10, title* 60, author id 10, publication date 10, publisher* 60, subject* 60, description* 500, related item id 1 to 5: 10 bytes each (5 fields), pointer to image 4, availability 10, suggested retail price 8, cost 8, stock 4, isbn 13, page 4, backing* 15, dimensions 25

- orders table:
  - size factor: 0.9sf
  - bytes per row: 127
  - fields: id 10, customer id of order 10, date 19, subtotal 8, tax 8, total 8, ship type* 10, ship date 19, billing address id 10, ship address id 10, status* 15

- order line table:
  - size factor: 2.7sf
  - bytes per row: 23
  - fields: id 3, order id 10, unique item id 10

I calculated the amount of padding that is required when encrypting at the row level and at the field level. The computations are based on the number of bytes per field only and exclude any formatting information a particular database utility may add when storing the data. The maximum length is used for variable-length fields. For example, the street address has a maximum allowed length of 40 bytes. If a customer's street address is under 40 bytes, 40 bytes are still allocated for the entry. When using fixed-length fields, the

field separation can be based on byte position as opposed to formating information or field deliminators. Using fixed-length fields assists in limiting information leakage based on the field length. Consider the city field in the address table. Even if padding is used to reach 32 bytes, the attacker knows the city name is in the first 30 bytes. However, if the city name is encrypted based on its actual length instead of the maximum field length of 30 bytes, then any city name less than or equal to 16 bytes will be encrypted as 16 bytes, which allows the attacker to narrow down the possible candidates for the city. For the variable-length fields, if the field entries are encrypted based on the actual length of the entry instead of the maximum length, the elastic block cipher may or may not reduce the size of the ciphertext for an individual entry in comparison to the ciphertext resulting from a fixed-length block cipher. A 34-byte value stored in a field with a maximum length of 40 bytes will be padded to 48 bytes when using a fixed-sized block and encrypting at the field level. In this case, the elastic block cipher will produce only 34 bytes of ciphertext for a savings of 14 bytes. If the entry was 32 bytes, then it would not require padding and the elastic block cipher will provide no space savings for this entry.

table name	multiple of scale factor	size in bytes	padding in bytes	total size without padding	total padding
address	2sf	154	6	308sf	12sf
author	isf/4	580	12	145isf	3isf
cc xacts	0.9sf	123	5	110.7sf	4.5sf
country	92 entries	45	3	4140	276
customer	sf	248	4	248sf	8sf
item	isf	851	13	851isf	13isf
orders	0.9sf	127	1	114.3sf	0.9sf
order line	2.7sf	134	10	361.8sf	27sf

Table 9.2: Padding per Table When Encrypting at the Row Level

Table 9.2 indicates the amount of padding required per table when encrypting individual

table	scale factor	size in bytes	required padding fields < 16 bytes	avoidable padding fields > 16 bytes	total bytes includes all padding	total bytes of avoidable padding	percent avoidable padding
address	2sf	154	24	30	416sf	60sf	14.42%
author	isf/4	580	12	48	640isf	48isf	7.50%
cc xacts	0.9sf	123	31	14	151.2sf	12.6sf	8.33%
country	92 entries	45	21	14	7360	1288	17.50%
customer	sf	248	56	64	368sf	64sf	17.39%
item	isf	851	110	31	992isf	31isf	3.13%
orders	0.9sf	127	55	26	187.2sf	23.4sf	12.50%
order line	2.7sf	134	46	12	518.4sf	32.4sf	6.25%

Table 9.3: Padding per Table when Encrypting at the Field Level

rows with a fixed-length block cipher using 16-byte blocks. This padding can be avoided if an elastic block cipher is used. The benchmark uses 10,000, 100,000, 1 million, 10 million as test values for the sf and isf parameters. The size of a table is the number of bytes per row times the scale factor. For example, the address table contains 2sf * 154 bytes of data when there is no padding and 2sf * 160 bytes with padding. When encrypting at the row level, the total bytes across all tables without padding is 1642.8sf + 4140 + 996isf. The amount of padding is 52.4sf + 276 + 16isf. The percent of ciphertext that is padding will range between approximately 1.61% when isf is significantly larger than sf and 3.19% when sf is the dominant value. If the number of items for sale (the isf value) is 1 million, the percent of ciphertext that is padding for 10,000, 100,000 and 1 million customers is 1.61%, 1.80% and 2.53%, respectively. As a point of reference, Amazon.com had 33.8 million customers in 2002 [GLS02]. Using 34 million customers for Amazon and 1 million items, the percent of ciphertext that is padding is 3.05%.

Table 9.3 indicates the amount of padding required per table when encrypting individual

fields with a 16-byte block cipher. All fields under 16 bytes are padded to 16 bytes regardless of whether a fixed-length or elastic block cipher is used. The amount of padding per table due to fields under 16 bytes is indicated in the "required padding" column. All fields over 16 bytes are padded when using a fixed-length version of the block cipher. The amount of such padding per table entry is indicated in the "avoidable padding" column. The avoidable padding is eliminated when using a variable-length block cipher. The last column indicates the percent of ciphertext when using a 16-byte fixed-length block cipher that is avoidable padding. The country, customer and address tables have the highest percentage of avoidable padding. Of all the tables, the customer table contributes the largest amount of avoidable padding bytes when using the sf and isf values in the benchmark.

The total size of all the tables when padding per field is 1640sf + 1152isf + 7360 bytes. 192.4sf + 43isf + 1288 bytes of this total are avoidable padding. When padding per field, the percent of ciphertext across all the table that is avoidable padding will range from 3.73% when isf dominants sf to 11.73% when sf dominants isf. When isf is 1 million, the percent of ciphertext that is avoidable padding for 10,000, 100,000 and 1 million customers is 3.85%, 4.73% and 8.43%, respectively. Using 34 million customers and 1 million items, the percent of ciphertext that is avoidable padding is 11.57%.

9.3 Summary

In real applications, databases are not (and should not have to be) designed around a single fixed value in order to conform to the block size of current block ciphers. Encrypting a database at a more granular level then the entire database or table level can reduce the time to perform queries and updates, but increases the amount of padding required due to the data being treated as smaller segments, each of which must be padded to an integral of the block size when using a fixed-length block cipher. Using an elastic block cipher can reduce or eliminate padding. I used a database from a benchmark for online bookstores to illustrate the percentage of ciphertext that can be eliminated per table and over all tables when an elastic block cipher is used in place of a 16-byte block cipher.

Chapter 10

Application: Modes of Encryption

10.1 Overview

An elastic block cipher can be used in existing modes of encryption in two ways. The first option is to use the block size of the original, fixed-length block cipher for all blocks except the last block, then use a variable-length block at the end to avoid padding. A second option is to use a block size different from the fixed-length block cipher for all blocks, with the size of the last block set to avoid padding. When using an existing mode, the only benefit the elastic version of a cipher provides is the elimination of padding; it does not eliminate any existing attack against the mode. For short segments of data between one and two blocks, an elastic block ciphers allows all of the bits to be encrypted as a single block, avoiding the need to use a mode of encryption and creating a stronger binding across the ciphertext bits compared to the ciphertext produced by a mode of encryption. For example, when encrypting a 256-bit key with a 128-bit block cipher, the entire key can be encrypted as a single plaintext block with the elastic version of the cipher instead of as two 128-bit blocks.

The elastic block cipher structure also allows for new modes of encryption. As examples of how support for variable-length blocks can be used to create modes of encryption, I provide a sketch of two new modes: Elastic Chaining mode and Elastic Electronic Code Book mode. Each mode is suitable for certain types of applications, and has advantages and disadvantages compared to existing modes. Both modes are intended as initial ideas for future work.

10.2 Existing Modes and Attacks

Before describing the new modes, I provide a summary of the most common modes of encryption. I list the modes, their strengths and their weaknesses. The weaknesses stated are based on using the mode by itself. Combining a message authentication code (MAC) with a specific mode can eliminate the possibility that ciphertext is altered and not detected.

Electronic Code Book (ECB)

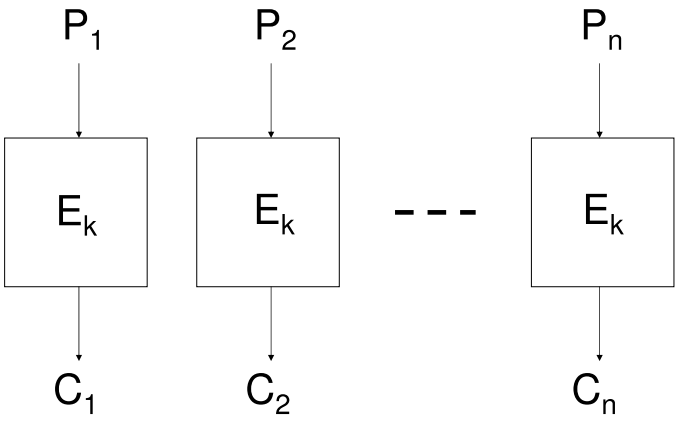

Figure 10.1: ECB Encryption Mode

ECB, shown in Figure 10.1, is basic block-by-block encryption. The plaintext is divided into b-bit blocks, with padding at the end if needed, and each block is encrypted individually. The mode allows data blocks to be encrypted in parallel and any individual ciphertext block to be decrypted without decrypting other blocks. This mode is not recommended for use because identical plaintext blocks encrypt to the same ciphertext, allowing for patterns and identical blocks to be detected, and the ciphertext can be rearranged or have blocks removed or replaced with other ciphertext blocks to alter the message. Altering a ciphertext block will alter only the corresponding plaintext block. As shown in the tests described in Section 10.4, patterns are not likely to occur in files containing content where strings of bytes may be repeated but do not occur on block boundaries. For example, files containing articles written in English, such as the content of this thesis. Certain words occur freuqently in this

thesis, but few are 16 bytes long and those that are 16 bytes are unlikely to align perfectly on block boundaries. Even digital images of scenes that appear to have repeated bytes, such as a picture of grass, will not cause patterns due to compression and subtle changes in color between pixels. In contrast, highly structured files, such as system and email logs, will produce patterns. Images with large areas of a single shade stored without compression, such as a cartoon character on an all-white background, will also have noticable patterns. A final note is that regardless of what method of encryption is used, if multiple files containing identical first blocks are encrypted with the same key, the pattern will be detectable with all of the methods described here. For example, multiple postscript files with the same formatting information in the first block will produce the same first ciphertext block.

Cipher Block Chaining Mode (CBC)

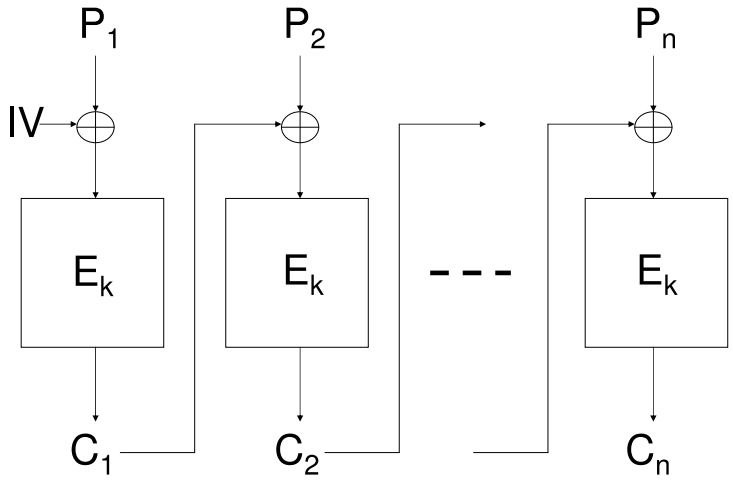

Figure 10.2: CBC Encryption Mode

CBC is shown in Figure 10.2. The plaintext is divided into b-bit blocks. When encrypting the blocks, the ciphertext generated by the i^{th} block is XORed with the $(i+1)^{st}$ plaintext block and the result of the XOR encrypted. An initialization vector (IV) is XORed with the first plaintext block. Any individual ciphertext block can be decrypted by applying the decryption function and XORing the result with the previous ciphertext block to obtain the plaintext.

CBC is not subject to ECB's flaws of pattern detection and ease with which messages can be modified by inserting, swapping or removing blocks. Identical plaintext blocks will not encrypt to the same ciphertext block, except with probability 2^{-b}. Changing the i^{th} ciphertext block will result in both the i^{th} and $(i+1)^{st}$ plaintext blocks being altered when decrypting. Removing a ciphertext block will result in the i^{th} plaintext block missing and the $(i+1)^{st}$ plaintext block being garbled.

CBC is subject to block-wise adaptive attacks [JMV02]. In such an attack, the attacker sends two $2b$-bit messages, $M1$ and $M2$, to be encrypted and receives the ciphertext of one message. The attacker then sends a message $M3$ to be encrypted by first sending an arbitrary block and receiving the ciphertext. The attacker generates the next block of $M3$ by XORing the first block from the first ciphertext received, the ciphertext block just received from $M3$ and the second block of $M1$. This block is then encrypted; if the ciphertext matches the second block of the first ciphertext, the first message encrypted was M1, else it was M2. Specifically,

- $M1, M2, M3$ are three distinct plaintexts.
- $Mi[j]$ is the j^{th} block of Mi with $j = 1$ denoting the first block.
- $Cx =$ the encryption of whichever of $M1$ or $M2$ was encrypted.
- $C3 =$ the encryption of $M3$.
- $Ci[j]$ is the j^{th} block of Ci with $j = 1$ denoting the first block.
- $M3[2] = Cx[1] \oplus C3[1] \oplus M1[2]$

When $M3[2]$ is encrypted, $C3[1]$ is XORed with $M3[2]$ then encrypted, resulting in the encryption of $Cx[1] \oplus M1[2]$. Therefore, $C3[2] = Cx[2]$ if Cx is the encryption of $M1$ and $C3[2] \neq Cx[2]$ if Cx is the encryption of $M2$.

A splicing attack is also possible on CBC with minor distortion of the plaintext. When two ciphertexts are spliced together, only the first plaintext block of the second portion is altered. Attacks which produce only minor distortion of the plaintext are a concern because one or two garbled plaintext blocks can easily go unnoticed if no other means of detecting

changes is used, such as a MAC. For example, a few pixels of an image, or one line or symbol in a figure being garbled can easily go unnoticed by the user.

Output Feedback Mode (OFB)

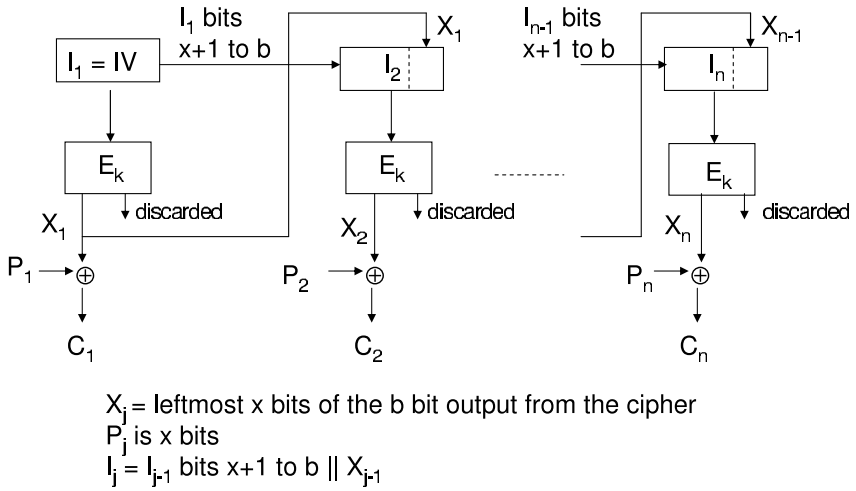

Figure 10.3: OFB Encryption Mode

OFB, shown in Figure 10.3, uses a block cipher to build a stream cipher. In OFB, an IV is encrypted. x bits from the IV are used in the key stream. The remaining $b - x$ bits are discarded. The x bits are also appended to the IV and the last b bits of the result are used as the next input to the block cipher. The process is repeated, each time taking x bits from output of the block cipher to add to the key stream, and forming the next input to the block cipher from the current input and x bits of the output. The key stream bits are XORed with the plaintext. When using OFB, the ciphertext blocks must be decrypted sequentially from the beginning in order to allow the key stream to be regenerated.

By creating a stream cipher, the key stream can be computed in advance or as needed. Altering a bit in the ciphertext will only alter the corresponding plaintext bit. Removing a ciphertext bit will result in the key stream being out of sync with the data stream when decrypting, garbling all subsequent plaintext, unless a synchronization method is used and the missing bit is viewed as lost. Inserting ciphertext also will not work because the data

will be out of sync with the key stream and all blocks from the point of modification onward will be garbled when decrypted. Splicing is only possible if the ciphertexts being spliced together were created using the same IV and key (which should not be done since this would produce the same key stream), and the splice occurs at the i^{th} block of the first ciphertext and $i + 1^{st}$ block of the second ciphertext so the key stream is unaltered. As with any stream cipher, if both the plaintext and ciphertext are known by the attacker, the attacker can compute the key stream by XORing the plaintext and ciphertext and thus has the information necessary to modify the ciphertext such that it decrypts to a different meaningful plaintext. Pattern detection is not possible because identical plaintext blocks will not encrypt to identical ciphertext blocks.

Cipher Feedback Mode (CFB)

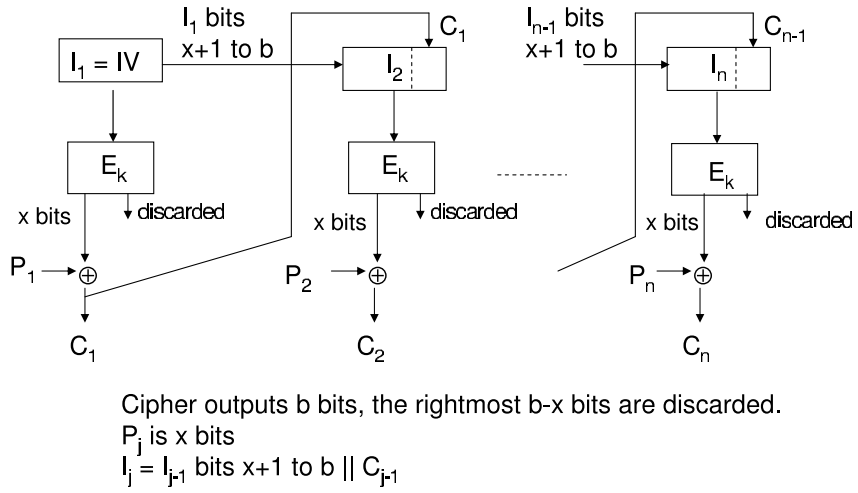

Figure 10.4: CFB Encryption Mode

CFB, shown in Figure 10.4, is similar in design to OFB. Instead of using x bits from the block cipher's output to form the next input to the block cipher, x bits from the ciphertext (the result of the XOR with the plaintext) are used. This prevents the key stream from being computed in advance since now x plaintext bits must be XORed with x key stream bits

before the next x-bit segment of key stream can be computed. In CFB mode it is possible to start decryption at any point in the ciphertext as long as the necessary ciphertext bits prior to the starting point are available to enable the key stream to be computed.

If any bit in a x-bit segment of the ciphertext is altered, the corresponding bit in the plaintext will be altered as well as the next x-bit segment of the plaintext. This makes a single bit change in the ciphertext more likely to be detected than in OFB, where only one bit in the plaintext is impacted. If a bit is removed from the ciphertext in CFB, the number of impacted plaintext blocks depends on x because the values encrypted to produce the key stream are now altered for however many blocks use the missing ciphertext bit as input to the encryption function. The ciphertext cannot be rearranged without garbling the plaintext. Splicing of ciphertext will produce several garbled blocks because the values encrypted depend on a series of previous ciphertext blocks. Like OFB, pattern detection is not an issue.

Counter Mode (CTR)

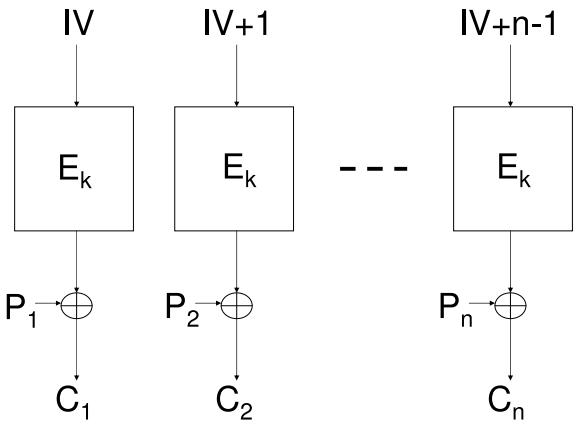

Figure 10.5: CTR Encryption Mode

CTR, shown in Figure 10.5, also creates a stream cipher from the block cipher. An IV is used as input to the block cipher and the output is used as the key stream. The IV is repeatedly incremented to form the next input to the block cipher. CTR mode allows individual ciphertext blocks to be decrypted. Splicing can easily be done if the same IV

was used to generate the two ciphertexts being spliced together, in which case the i^{th} block of one ciphertext can replace the i^{th} block of the second ciphertext. However this would imply the same key stream was used to encrypt different plaintexts. If $\mathrm{IV}_1 + i = \mathrm{IV}_2 + j$ for some i and j when $\mathrm{IV}_1 \neq \mathrm{IV}_2$, then the keystreams will match from the i^{th} and j^{th} blocks onward and splicing is possible. Removing or inserting ciphertext will cause the key stream and data stream to be out of synch, garbling all subsequent plaintext until the two streams can be synchronized.

10.3 Elastic Chaining Mode

The first new mode is a method of chaining blocks. It is depicted in Figure 10.6. This mode is useful in applications where the entity decrypting the ciphertext should or can decrypt starting at the last block. For example, when decrypting a file or segments of a database, decryption can start at the last block. This mode would not be applicable to real time streaming data which has to be decrypted in the order in which it is received. Given a block cipher that operates on b-bit blocks, the elastic version can be applied to encrypt $(b+y)$-bit blocks, where y bits are taken from the previous ciphertext block and prepended to the next plaintext block. The output consists of the leftmost b bits from each ciphertext block for all but the last block and the entire ciphertext of the last block. The first block to be encrypted can consist of b plaintext bits with a y-bit IV prepended to it, $b+y$ plaintext bits, or contain only b plaintext bits. Overall, the ciphertext will be at most y bits longer than the plaintext. If the plaintext is not an integral number of b-bit blocks, the last block may be shorter than $b + y$ bits. When the plaintext is not an integral number of b-bit blocks, the mode can be implemented without padding the last block; whereas, using the non-elastic version of the block cipher would require padding and also produce a ciphertext longer than the plaintext. The performance of the mode depends on the size of y because the number of rounds applied per block increases as y increases. For a block cipher with r rounds, nr rounds are computed to encrypt n b-bit blocks with ECB, CBC or CTR mode. The number of rounds using elastic chaining will range from $n(r+1)$ when $y = 1$ to $2nr$ when $\lceil \frac{ry}{b} \rceil = r$.

The mode resembles CBC but instead of XORing the i^{th} ciphertext block with the $(i+1)^{st}$ plaintext block, bits from the i^{th} ciphertext block are concatenated with the $(i+1)^{st}$ plaintext block. This concatentation creates a stronger binding between the i^{th} and $(i+1)^{st}$ blocks compared to that created by the XOR used in CBC mode. The stronger binding is achieved by increasing the work per block from the number of rounds required for b bits to the number required for $b + y$ bits, while the number of blocks is unchanged.

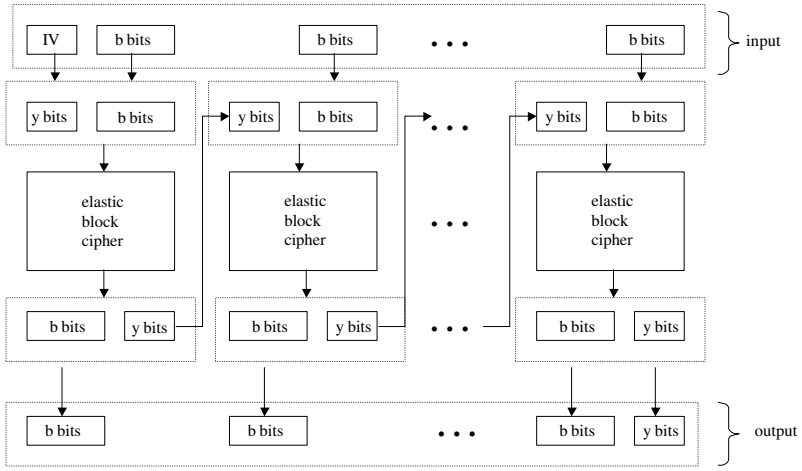

Figure 10.6: Elastic Chaining Mode

The ciphertext can be decrypted by decrypting the last block, concatenating the y bits from the plaintext block with the previous ciphertext block, and then decrypting the next block. When using an IV with the first block, the IV is not needed for decryption; however, having it available for decryption provides a type of integrity check in that the first y bits of the resulting plaintext can be verified against the IV.

The mode allows for variations. These include altering which positions the y bits from the previous ciphertext block are inserted into the current plaintext block. Instead of prepending the y bits to the next plaintext block, they could be appended or inserted amongst the b bits as either y consecutive or nonconsecutive bits. The size of y can also vary between blocks, possibly based on the key value.

This mode offers several security benefits because, even if the plaintext is known, an attacker does not know the actual $(b + y)$-bit block being encrypted. If y varies per block based on key material, the attacker does not even know the length of each block being encrypted. Incorporating the previous ciphertext block into the current plaintext block when encrypting will hide plaintext patterns. In the way the mode is depicted in Figure 10.6, a single bit toggled in the ciphertext is detectable because it will garble all plaintext prior to and including the altered block. In order to insert or splice together ciphertext blocks, the inserted ciphertext block must decrypt to a plaintext which produces the same leftmost y bits as the original ciphertext block; otherwise, all plaintext blocks prior to this one will be garbled, resulting in a much more noticeable impact than the single garbled block produced by a splicing attack on CBC. Block-wise adaptive attacks are prevented because there is no need for the device performing the encryption to output the last y bits of each ciphertext block, except for the last block. This prevents the attacker from knowing the actual block being encrypted because the attacker only gets to choose b bits of the $b + y$ bit block and block-wise adaptive attacks depend on the attacker knowing the exact plaintext.

To prepend blocks to the ciphertext, the attacker must be able to insert a ciphertext block that, when prepended to the leftmost y bits of the original first plaintext block will decrypt to some meaningful plaintext. Since these y bits are the IV, if the IV is not secret, the attacker will know what the y bits are and needs to find b bits that can be prepended to the y bits. However, notice that the attacker does not have a library of (plaintext, ciphertext) pairs from which to search for a possible b-bit value to prepend to the IV unless the entire plaintext is one block, in which case the mode is not necessary. The attacker will not have $(b + y)$-bit (plaintext, ciphertext) pairs from the (input, output) pairs of data encrypted with this mode because the leftmost y bits of the ciphertext are not included in the output except for the last block and the $b + y$ input to the last block is not known.

Appending blocks requires the attacker append blocks of ciphertext which decrypt to a plaintext whose leftmost y bits are the same as the last y bits of the original ciphertext. In both cases, the smaller y is, the more likely it is that the attacker can form meaningful blocks to prepend or append, since there are only 2^y values to try. If y or the bit positions

used for the y bit vary per block based on the key, an attacker will need to try all values of y and possible positions for the y bits.

It is not possible to rearrange ciphertext blocks without garbling the plaintext because y bits from each plaintext block are used to decrypt the previous plaintext block. In order to swap ciphertext block i with ciphertext block j, the attacker has to find a ciphertext block in position i which, when prepended to the leftmost y bits from the $(j+1)^{st}$ plaintext block, will decrypt to a plaintext block whose leftmost y bits are the same as the y bits appended to the $(j-1)^{st}$ ciphertext block during decryption. Likewise, the j^{th} block must be such that when it is prepended to the leftmost y bits from the $(i+1)^{st}$ plaintext block, will decrypt to a plaintext block whose leftmost y bits are the same as the y bits appended to the $(i-1)^{st}$ ciphertext block during decryption. Furthermore, because the recipient of the ciphertext does not receive the rightmost y bits of each block except for the last block. Thus the attacker does not even know all of the ciphertext bits used to decrypt a given block of plaintext when trying to determine what ciphertext blocks can be rearranged without garbling the message.

10.4 Elastic ECB Mode

10.4.1 Description

A second new mode is a possible alternative to ECB which offers some protection against pattern detection in and alterations of the ciphertext. The mode is shown in Figure 10.7. The data is encrypted as in ECB mode, but the block size varies per block based on the key. The i^{th} block is of length $b + y_i$ where y_i is based on key bits. The i^{th} block can be decrypted without decrypting any other block by determining its starting position and length from the key. If the key bits are sufficiently random, y_i will be uniformly random within $[0, b]$. Another option is to use key bits to set the first block's length then set each subsequent block size based on bits from the previous ciphertext block, although this will not allow the block lengths to be set in advance. The i^{th} block can start at any position in the range $b(i-1)+1$ and $2b(i-1)+1$, with an average of $\frac{3b(i-1)+2}{2}$. For a plaintext pattern to show up in the ciphertext, the starting position of the block (which is now random) and

y_i would have to match the starting position of the plaintext pattern and its length in the file. This method does not work for all cases because there are $b + 1$ possible block sizes (b to $2b$) if all values of y are used and $\frac{b}{8} + 1$ possible block sizes if the block size must be an integral number of bytes. If the file is large enough and has a significant number of repeated entries, ciphertext repetitions will occur. The degenerate case is a file consisting entirely of the same byte value repeated, in which case there will be $\frac{b}{8}$ distinct ciphertext blocks if y is restricted to being a multiple of 8.

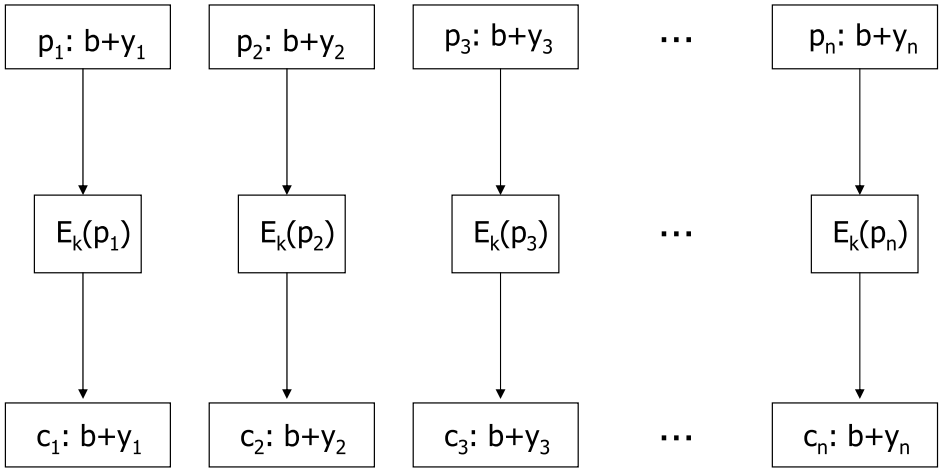

Figure 10.7: Elastic ECB Encryption Mode: Block Size Varies per Block

Replacing individual blocks without garbling the plaintext is possible if the attacker can determine the start and end position of the individual blocks where the modification will occur. Any block being replaced will have to be replaced with a block of the same length; otherwise, the block and all subsequent plaintext blocks will be garbled. The probability of an attacker determining the start and end of the i^{th} block is $\frac{1}{b^2(i-1)}$. (The probability of guessing the start position of the i^{th} block is $\frac{1}{b(i-1)}$ and the probability of guessing the length, y_i, of the i^{th} block is $\frac{1}{b}$.)

Splicing is even more difficult than replacing individual blocks. If two ciphertexts, $C1$

and $C2$ are being spliced together, the individual block lengths of the result must be the same as the lengths corresponding to the key. If a block is removed, the block boundaries for all subsequent blocks will not correspond to the boundaries used in encryption and the remaining plaintext will be garbled. A block-wise adaptive attack is not applicable because such an attack requires a constant block size.

Elastic ECB mode is not aimed at applications where one plaintext block is processed before receiving the next block, but instead aimed at applications where at least two b-bit blocks are available when encrypting all but the last block so the block size can be varied, with y set to any value in the range of 0 to b. Elastic ECB mode may require a greater amount of computation than ECB due to the need to compute the y_i's from the key and due to varying the block length. Overall encryption time compared to ECB may or may not increase because the longer block lengths will result in fewer blocks to encrypt. The total number of rounds required of the elastic ECB mode to encrypt nb bits, for some integer $n > 0$, will depend on the n, b and y_i values.

10.4.2 Experiments

To illustrate how elastic ECB mode reduces patterns, the number of times two or more identical blocks occur within a file was determined when using 16-byte blocks and (16+Y)-byte blocks, where Y is an integer between 0 and 16 that varies per block. These tests also demonstrate that ECB mode is not likely to produce patterns in the ciphertext when encrypting files that are not formatted or structured. For example, English text is unlikely to have repeated phrases that align on 128-bit block boundaries. In contrast, patterns are likely to appear in the ciphertext in structured files where the format of the content results in patterns, such as email logs. Three categories of plaintext were tested:

1. Files where 128-bit repetitions in plaintext were not generally expected. These files consisted of news articles from the NY Times, the text (but not the formatting) of various research papers, and the content of emails. While repetitions of specific words occur in the text, for example the word "the" frequently appears, repeated words or phrases constituting a single 128-bit block seldom appeared. Each file contained at least 1600 bytes to allow for a minimum of 100 fixed sized blocks. The largest file

contained 16,000 bytes.

2. Files where repetitions of plaintext that encompassed one or more blocks were expected, but were spread relatively far apart and were not necessarily expected to be on block boundaries. These files consisted of tex files for research papers containing formatting commands and JAVA programs containing repeated variables and function calls of at least 16 bytes. For example,

   - `"\vspace{-16pt}  " , " \begin{tabular} "`,
   - `"Section ~\ref{subsec"` .
   - `"new BufferedReader("` and
   - `"System.err.println("`

   A space or newline at the end of the vspace and begin commands fills in the 16^{th} byte. Each file contained at least 1,600 bytes to allow for 100 fixed-sized blocks. The largest file contained 16,000 bytes.

3. Files where repetitions of plaintext were frequent but not intentionally aligned on 16-byte boundaries. These files consisted of emails, email logs and a log of visitors (IP addresses, and related information) to a web site. The email consisted of a unix email file involving emails between three people. The first 160,000 bytes were used. The emails were generally short and included forwarded emails. No attached files were included.

   The email log contained the email header information from emails sent to one user. The following are sample entries:

```
From owner-cryptography+dcook=cs.columbia.edu@metzdowd.com  Mon Mar
28 15:14:58 2005
 Subject: Re: NSA warned Bush it needed to monitor networks
  Folder: /home/dcook/.mailspool/dcook

From dcook@cs.columbia.edu  Sun May  1 00:41:38 2005
```

```
 Subject: CVS commit: EBC
  Folder: /home/dcook/.mailspool/dcook

From mice@cs.columbia.edu  Fri May  6 16:12:11 2005
 Subject: Your spring 2005 review
  Folder: /home/dcook/.mailspool/dcook

From angelos@cs.columbia.edu  Fri Dec  9 00:23:08 2005
 Subject: Autoreply... [Re: CVS commit: BOOKCG]
  Folder: /home/dcook/.mailspool/dcook
```

A log of "visitors", excluding web crawlers, to my home page was tested. The log contained minor HTML formatting around the entries. The following are sample entries:

```
<br> Server:magnum.cs.columbia.eduAddress:128.59.16.117
Name:cpe-66-108-54-146.nyc.res.rr.comAddress:66.108.54.146
index.shtml  07/02/05 00:24:03  http://nsl.cs.columbia.edu

<br> Server:orion.cs.columbia.eduAddress:128.59.16.21
Name:chatter.cs.Virginia.EDUAddress:128.143.136.192
index.shtml  07/16/05 03:20:24  http://nsl.cs.columbia.edu

<br> Server:lion.cs.columbia.eduAddress:128.59.16.120
Name:vinci2.cs.umass.eduAddress:128.119.246.71
index.shtml  08/01/05 14:10:44

<br> Server:play.cs.columbia.eduAddress:128.59.21.100
Name:nefeli.cs.columbia.eduAddress:128.59.23.88
index.shtml  09/27/05 21:26:59
http://www.cs.columbia.edu/~dcook/dcookres.shtml
```

Each log file contained at least 160,000 bytes.

Using 50 files from the first category, each file was divided into 16-byte blocks and checked for multiple occurences of the same block. The number of blocks per file ranged from 100 to 1,000. No identical blocks were found. The files were then broken into (16+Y)-byte blocks where $0 \leq Y \leq 16$ and each Y was chosen using the SecureRandom function in JAVA. This corresponds to variable-length blocks adhering to byte boundaries, which is the most likely way an elastic block cipher would be used on files. No identical blocks were found.

The same tests were repeated with 50 files from the second category. Again no identical blocks were found. Even though there were repeated sequences of 16 characters within the files, they did not align on the block boundaries.

The same tests were repeated on the files from the third category. Matches between blocks were found. A match means that a block is identical to a previous block.

- Mail file with 10,000 16-byte blocks: The file contained 37 emails, most of which included forwarded emails and email chains.

  The most common patterns were:

  - 286 occurrences of `"@cs.columbia.edu"`
  - 151 occurrences of `"dcook@cs.columbia.edu"`
  - 73 occurrences of `"angelos@cs.columbia.edu"`
  - 65 occurrences of `"disco.cs.columbia.edu"`
  - 62 occurrence of `"lion.cs.columbia.edu"`
  - 57 ocurrences of `"moti@cs.columbia.edu"`
  - 48 occurrences of `"version=TLSv1/SSLv3`, of which 37 were identical for an entire line with text indicating DES-CBC3 was used and 11 were identical for the rest of the line with text indicating AES was used.
  - 38 occurrences of `"MIME-Version: 1.0"`

  - 37 occurrences of `"Message-ID: <"`. Although this string is only 13 bytes, if the message id number which followed started with the same digits, a match could occur.
  - 33 ocurrences of `"Content-Type: TEXT/PLAIN; charset=US-ASCII"`

  The email addresses were preceded by `"From: "`, `"To: "`, `"cc:"`, or `"Return-Path:"`. In the From, To and cc lines, the name of the user may appear followed by the email address enclosed in `< >`.

  When using ECB mode with 16-byte blocks, 1362 matches involving 399 blocks ocurred. Each pattern involved 2 to 28 identical blocks. There were 451 cases with exactly two identical blocks, 156 cases with 3 identical blocks, 63 cases of 4 identical blocks, 30 cases of each 5 identical blocks, 11 cases of 6 identical blocks, 8 cases of 7 identical blocks, 1 case each of 8,10,11,17,18,21,23 and 28 identical blocks. There were 4 cases of 9 identical blocks and 2 cases of 13 identical blocks.

- Mail file with 160,000 bytes and variable block size:

  The file was tested 10 times with the block sizes set randomly per block. The number of blocks ranged from 6792 to 6845. The number of matches found ranged from 36 to 58 across the trials. In all cases, pairs of identical blocks accounted for most of the matches. Cases of three identical blocks occurred in two trials. The trial with 58 matches had 6845 blocks with 52 pairs of identical blocks and three triples of identical blocks. The number of matches in the trials were 36, 40, 41, 44, 46, 48, 51, 53, 55 and 58. This corresponds to at least a 96% decrease in the number of matches compared to using ECB mode with 16-byte blocks.

- Mail log with 10,000 16-byte blocks:

  There were 1030 entries in the file. The most common patterns were:

  - 834 occurrences of `"/home/dcook/.mailspool/dcook"`,
  - 190 occurrences of `"/home/dcook/mail/spam"`.
  - 109 occurrences of `"From owner-cryptography@metzdowd.com"`,

  - 69 occurrences of "From angelos@cs.columbia",
  - 34 occurrences of "From moti@cs.columbia.edu",
  - 34 occurrences of "From dcook@cs.columbia.edu",

  Using ECB mode with 16-byte blocks, there were 3438 matches. For a given pattern, the number of identical blocks found ranged from 2 to 73. 368 cases involved exactly two identical blocks, 110 cases involved 3 identical blocks, 52 cases involved 4 identical blocks, 33 cases involved 5 identical blocks, 21 cases involved 6 identical blocks. There were 89 other cases involving 7 to 73 identical blocks.

- Mail log with 160,000 bytes and variable block sizes:

  When varying the block size, matches still occur, but were significantly reduced. The maximum number of blocks matching a given pattern was also significantly reduced, with at most 7 identical blocks ocurring. The file was tested ten times with the block sizes set randomly per block. Of the ten trials, the one with the most matches had 645 matches in 6814 blocks. Of these 645 matches, 178 cases involved 2 identical blocks, 94 cases involved 3 identical blocks, 51 cases involved 4 identical blocks, 22 cases involved 5 identical blocks, 4 cases involved 6 identical blocks and 3 cases involved 7 identical blocks. For each trial, there was at least an 81% reduction in the number of matches compared to ECB mode with 16 byte blocks.

- Web log with 10,000 16 bytes:

  There were 986 entries with the HTML command "
" appearing before each entry. The most common patterns were:

  - 848 occurrences of ".cs.columbia.edu"
  - 295 occurrences of "nsl.cs.columbia.edu"
  - 170 occurrences of "http://www.cs.columbia/~dcook"
  - 65 occurrences of "disco.cs.columbia.edu"
  - 61 occurrences of "play.cs.columbia.edu"
  - 41 occurrences of "bear.cs.columbia.edu"

When using ECB mode with 16-byte blocks there were 3870 matches. There were 379 cases involving 2 identical blocks, 112 cases involving 3 identical blocks, 69 cases involving 4 identical blocks, 56 cases involving 5 identical blocks, 42 cases involving 6 identical blocks, 26 cases involving 7 identical blocks and 25 cases involving 8 identical blocks. There were 92 cases involving 9 to 75 identical blocks. Two cases involved 75 identical blocks.

- Web log with 160,000 bytes and variable sized blocks:

  Varying the block size significantly reduced the number of patterns. The file was tested ten times with the block size set randomly per block. In the worst case, 6819 blocks resulted with 511 matches. The number of identical blocks for a given pattern ranged from 2 to 8. There were 196 cases with 2 identical blocks, 55 cases with 3 identical blocks, 29 cases with 4 identical blocks, 15 cases with 5 identical blocks, 4 cases with 6 identical blocks, 4 cases with 7 identical blocks and 2 cases with 8 identical blocks. In each of the ten trials, there was at least an 87% percent reduction in the number of matches compared to ECB mode with 16-byte blocks.

10.4.3 Conclusions

The experiments demonstrate that the appearance of patterns in the ciphertext when encrypting with ECB mode depends on the type of file and the content. When encrypting English text, it seems unlikely that repeated phrases will align on 128-bit block boundaries. As a result, no patterns appeared in the ciphertext when encrypting news articles, and the content of research papers and emails with both the fixed-length and elastic versions of AES (or any 16-byte block cipher). Even files with moderate occurrences of repeated strings, such as tex files, did not have the repeated strings align on block boundaries. In the files where the structure did result in patterns aligned on block boundaries, varying the block size signficantly reduced the number of repeated ciphertext blocks.

10.5 Summary

The ability to encrypt variable-length blocks allows new modes of encryption to be designed. I described two ways of using variable-length blocks to create new modes. The first mode involves chaining blocks in a manner such that bits from the i^{th} ciphertext block become part of the $(i+1)^{st}$ plaintext block. When encrypting a sequence of blocks, y bits from the previous ciphertext block are prepended to the current plaintext block to form a $(b+y)$-bit block. This mode prevents the block-wise adaptive attacks that CBC is subject to and, compared to CBC, results in more garbled plaintext blocks when attempting to splice or otherwise alter ciphertext blocks. The second mode is ECB with the block size varying across the blocks. Expanded-key bits can be used to set each block's size. Varying the block size significantly reduces the probability that patterns are detected, even in highly repetitious data. Furthermore, insertion, removal or rearrangement of blocks requires determining the start position and length of the blocks. These proposals are intended as initial concepts to demonstrate additional potential uses of elastic block ciphers and require further analysis.

Chapter 11

Conclusions

11.1 Summary

I devised a solution for how to design a variable-length block cipher that is both secure and computationally efficient. Existing block ciphers used in practice support only one or a few block sizes. As a result, when encrypting data that is not an integral number of blocks, the fractional block is padded to a full block. This padding results in computational and memory overheads that can impact performance in certain applications. While there have been previous proposals for variable-length block ciphers, each is either computationally inefficient compared to my scheme or has been proven to be insecure. Previous proposals fall into two catgeories: black box designs where an existing block cipher is applied multiple times or ad-hoc designs. A related problem is how to construct variable-length PRPs and SPRPs, which represent the ideal security for a block cipher. The structure I devised for creating practical variable-length block ciphers allows for the creation of variable-length PRPs and variable-length SPRPs.

I defined a method for converting any existing block cipher that works on a fixed-length, b-bit block into a block cipher that works on any block size in the range of b to $2b$ bits. The resulting cipher is referred to as an elastic block cipher. The method is a combination of the black box and design from scratch approaches, with the round function being treated as a black box instead of the entire original cipher and operations added between rounds. Elastic block ciphers eliminate space overhead due to padding and allow

the computational workload of encrypting data to be proportional to the block size. In contrast, both padding a fractional block to a full block and the use of ciphertext stealing result in two full applications of a fixed-length block cipher when encrypting $b + y$ bits for $0 < y < b$. Relevant applications where space savings is valuable include the encryption of databases and the encryption of internet traffic.

I created a basic structure, called the elastic network, for building elastic block ciphers. I proved that the elastic network allows for the creation of variable-length PRPs and SPRPs in the range of b to $2b$ bits from independently chosen b-bit PRPs. Three rounds of the elastic network in the encryption direction and four in the decryption direction are needed to create a variable-length PRP and five rounds are needed to create a variable-length SPRP. By using three-round Fiestel networks as the round functions, the variable-length PRPs and SPRPs can be created from $\frac{b}{2}$-bit PRFs. By combining the elastic network with the CMC encryption mode, $2b$-bit to $2mb$-bit PRPs and SPRPs can be created from b-bit PRPs, for an integer $m \geq 2$ that is dependent on the b-bit PRPs.

My method for creating elastic block ciphers involves inserting the round function of an existing block cipher into the elastic network. This results in steps being added between the rounds of the original block cipher and allows the properties of the round function to be maintained when creating the elastic version. I demonstrated the method by creating elastic versions of AES, Camellia, MISTY1 and RC6. The performance benefit of using an elastic version in place of the original cipher with padding varied across the four examples. The performance benefit is the most significant for one version of AES where almost twice the number of 17-byte blocks were encrypted with the elastic version of AES compared to padding the blocks to 32 bytes each, with the benefit decreasing as the block size approached two full blocks. The elastic versions of MISTY1 and RC6 both provided faster encryption rates compared to the original versions with padding for block sizes up to 4 bytes over the original block size. There was no performance benefit to using the elastic version of AES in a second representation where the round function is implemented entirely as table lookups and XORs. There was also no performance benefit to using the elastic version of Camellia unless the initial and final key dependent permutations are omitted. I illustrated the space savings an elastic block cipher can provide when encrypting a database by computing the

amount of padding that can be eliminated when encrypting a sample database storing customer information for an online bookstore.

The elastic network and the reuse of the round function allows the security of the elastic version of a cipher to be related directly to the security of the original cipher. By forming a reduction from the elastic version of a cipher to the original version of a cipher, I proved that an elastic version of a cipher is secure against round-key recovery attacks if the original cipher is secure against such attacks. This eliminates the need to analyze an elastic version of a block cipher against these types of attacks if the original cipher is secure against such attacks. Independent of this general method, I showed that if linear cryptanalysis (which is also covered by the result for round-key recovery attacks) is possible on the elastic version of a cipher then a linear attack also exists on the original cipher. I extended this result to any attack using algebraic equations relating the plaintext, ciphertext and expanded-key bits. In order to provide a concrete example of cryptanalysis, I considered differential cryptanalysis (which is also covered under the round-key recovery attacks) on the elastic versions of AES and MISTY1. Using a state transition method and the properties of the round functions, I derived upper bounds on the probability of a differential characteristic occurring in the elastic versions.

My work also demonstrates the concept of having a generic key schedule. This allows a single implementation of a key schedule to be in applications supporting multiple block ciphers, which is a benefit in hardware implementations. I explained why it is advantageous to produce pseudorandom expanded-key bits. The lack of independence among expanded-key bits produced by existing key schedules increases the potential for an attack on the cipher. I used RC4 as the key schedule in the four elastic block ciphers I implemented to illustrate the concept of a generic key schedule with pseudorandom output.

Finally, I described how the elastic block cipher construction can be used to define new modes of encryption. I described two potential new modes based on the ability to use varying block sizes and provided a preliminary analysis for both of the modes. The first mode is a method of chaining blocks with bits of the i^{th} ciphertext block prepended to the $(i+1)^{st}$ plaintext block. The second mode is a variation of ECB mode with variable block sizes.

11.2 Future Work

There are aspects of the elastic block cipher work where further analysis can be performed and possible extensions to the work. Areas for additional research include the following items.

1. Additional work on potential uses of elastic block ciphers and the elastic network is of value. One potential use of the elastic network is in the design of a hash function.

2. Additional analysis of the proposed encryption modes is needed. Designing other new modes of encryption based on the elastic network is also of interest.

3. The current creation of variable-length PRPs and SPRPs from PRFs requires using a three-round Feistel network as the round function. An open question is if the elastic network can be used to create PRPs from PRFs without using a Feistel network or by combining the elastic network and Feistel network in a manner that requires fewer rounds of either the Feistel network or of the elastic network.

4. PRPs in CMC mode were used as the round functions of the elastic network when forming $2b$ to $2mb$-bit PRPs and SPRPs. It may be possible to combine the elastic network and CMC mode more efficiently than this to create a PRP.

5. It may be possible to increase the limit on the number of blocks used in CMC mode when creating the variable-length PRPs. The current bound corresponds to the case where CMC mode produces a SPRP; whereas, the round function in the elastic network only has to be a PRP.

6. Further work can be performed on implementations of elastic block ciphers. Optimizations may be possible to the software versions of the elastic block ciphers. While preliminary analysis indicates the elastic versions of block ciphers can be implemented in FPGAs, the present work does not include hardware implementations of elastic block ciphers.

7. The suitability of additional stream ciphers as candidates for a generic key schedule can be determined.

In summary, I have created a method for converting existing block ciphers to variable-length block ciphers and for creating variable-length PRPs and SPRPs. Additional uses of variable-length block ciphers are being considered, and improvements to the elastic block ciphers and variable-length PRPs and SPRPs presented are the subject of future work.

Bibliography

[AB99] J.H. An and M. Bellare. Constructing VIL-MACs from FIL-MACs: Message Authentication Under Weakened Assumptions. In *Proceedings of Advances in Cryptology - Crypto 1999, LNCS 1666, Springer-Verlag*, 1999.

[AB00] M. Abdalla and M. Bellare. Increasing the Lifetime of a Key: A Comparative Analysis of the Security of Re-Keying Techniques. In *Proceedings of Advances in Cryptology, ASIACRYPT, LNCS 1976, Springer-Verlag*, 2000.

[ABK98] R. Anderson, E. Biham, and L. Knudsen. Serpent. `http://www.cl.cam.ac.uk/~rja14/serpent.html`, 1998.

[AIK+00] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita. Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms - Design and Analysis. In *Proceedings of Selected Areas in Cryptography, LNCS 2012, Springer-Verlag*, pages 39–56, 2000.

[BCK96] M. Bellare, R. Canetti, and H. Krawczyk. Pseudorandom Functions Re-Visited: The Cascade Construction and its Concrete Security. In *Proceedings of Foundations of Computer Science, IEEE*, 1996.

[BDL97] Boneh, Demillo, and Lipton. On the Importance of Checking Cryptgraphic Protocols for Faults. In *Proceedings of Advances in Cryptology - Eurocrypt, LNCS 1233, Springer-Verlag*, pages 37–51, 1997.

[Ber99] D. Bernstein. How to Stretch Random Functions: The Security of Protected Counter Sums. In *Journal of Cryptology, Vol. 12(3), Springer-Verlag*, pages 185–192, 1999.

[Bih93] E. Biham. New Types of Cryptanalytic Attacks Using Related Keys. In *Proceedings of Advances in Cryptology - Eurocrypt 1993, LNCS 0765, Springer-Verlag*, 1993.

[BKN06] M. Bellare, T. Kohno, and C. Namprempre. IETF RFC 4344 The Secure Shell (SSH) Transport Layer Encryption Modes. `http://www.ietf.org/rfc/rfc4344.txt`, 2006.

[BR99] M. Bellare and P. Rogaway. On the Construction of Variable Length-Input Ciphers. In *Proceedings of Fast Software Encryption, LNCS 1636, Springer-Verlag*, 1999.

[BR00] J. Black and P. Rogaway. CBC MACs for Arbitrary-Length: The Three-Key Constructions. In *Proceedings of Advances in Cryptology - Crypto 2000, LNCS 1880, Springer-Verlag*, 2000.

[BS93] E. Biham and A. Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer-Verlag, New York, 1993.

[BS97] E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. Computer Science Technical Report CS0910, Technion, 1997.

[Cop99] Coppersmith, *et al.,*. The MARS Cipher. `http://www.research.ibm.com/security/mars.html`, 1999.

[CPQ02] M. Ciet, G. Piret, and J. Quisquater. Related-Key and Slide Attacks: Analysis, Connections and Improvements, Extended Abstract. UCL Crypto Group Technical Report, 2002.

[DR99] J. Daemen and V. Rijmen. The Rijndael Block Cipher Version 2.0. Amended AES Proposal to NIST, March 1999.

[DR02] J. Daemon and V. Rijmen. *The Design of Rijndael: AES the Advanced Encryption Standard*. Springer-Verlag, Berlin, 2002.

[FM00] S. Fluhrer and D. McGrew. Statistical Analysis of the Alleged RC4 Keystream Generator. In *Proceedings of Fast Software Encryption 2000, LNCS 1978, Springer-Verlag*, 2000.

[FMS01] S. Fluhrer, I. Mantin, and A. Shamir. Weaknesses in the Key Scheduling Algorithm of RC4. In *Proceedings of Selected Areas of Cryptography, LNCS 2259, Springer-Verlag*, 2001.

[GLS02] S. Gupta, D. Lehmann, and J. Stuart. Valuing Customers. Manuscript, Columbia Center for Excellence in E-business, August 2002.

[Gol97] J. Golic. Linear Statistical Weakness of the Alleged RC4 Keystream Generator. In *Proceedings of Advances in Cryptology, Eurocrypt 1997, LNCS 1233, Springer-Verlag*, 1997.

[HR03a] S. Halevi and P. Rogaway. A Parallelizable Enciphering Mode. Cryptology ePrint Archive, Report 2003/147, 2003. `http://eprint.iacr.org/`.

[HR03b] S. Halevi and P. Rogaway. A Tweakable Enciphering Mode. In *Proceedings of Advances in Cryptology - Crypto 2003, LNCS 2729, Springer-Verlag*, 2003.

[HWKS98] C. Hall, D. Wagner, J. Kelsey, and B. Schneier. Building PRFs from PRPs. In *Proceedings of Advances in Cryptology - Crypto 1998, LNCS 1462, Springer-Verlag*, pages 370–389, 1998.

[Inf03] Information Technology Security Center of Japan. CRYPTEC. `http://www.ipa.go.jp/security/enc/CRYPTREC/fy13/cryptrec20010629e.html`, 2003.

[JMV02] A. Joux, G. Martinet, and F. Valette. Blockwise-Adaptive Attackers: Revisiting the (In)Security of Some Provably Secure Encryption Models. In *Proceedings of Advances in Cryptology - Crypto 2002, LNCS 2442, Springer-Verlag*, 2002.

[Knu94] L. Knudsen. Block Ciphers - Analysis, Design and Applications, Ph.D. Thesis. http://www2.mat.dtu.dk/people/Lars.R.Knudsen, 1994.

[Knu95] L. Knudsen. Truncated and Higher Order Differentials. In *Proceedings of Fast Software Encryption 1994, LNCS 1008, Springer-Verlag*, pages 196–211, 1995.

[Koc96] P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS and Other Systems. In *Proceedings of Advances in Cryptology - Crypto 1996, LNCS 1109, Springer-Verlag*, pages 104–113, 1996.

[KSWH00] J. Kelsey, B. Schneier, D. Wagner, and C. Hall. Side Channel Cryptanalysis of Product Ciphers. *Journal of Computer Security*, 8(2-3):141–158, 2000.

[Lab93] RSA Laboratories. *PKCS #7: Cryptographic Message Syntax Standard, Version 1.5*, November 1993.

[LM91] X. Lai and J. Massey. A Proposal for a New Block Encryption Standard. In *Proceedings of Advances in Cryptology - Eurocrypt 1990, LNCS 473, Springer-Verlag*, pages 389–404, 1991.

[LR88] M. Luby and C. Rackoff. How to Construct Pseudorandom Permutations from Pseudorandom Functions. *Siam Journal of Computing*, 17(2), April 1988.

[Mat93] M. Matsui. Linear Cryptanalysis Method for DES Cipher. In *Proceedings of Advances in Cryptology - Eurocrypt 1993, LNCS 0765, Springer-Verlag*, 1993.

[Mat96] M. Matsui. New Structure of Block Ciphers with Provable Security Against Differential and Linear Cryptanalysis. In *Proceedings of Fast Software Encryption 1996, LNCS 1039, Springer-Verlag*, pages 205–218, 1996.

[Mat97] M. Matsui. New Block Encryption Algorithm MISTY. In *Proceedings of Fast Software Encryption 1997, LNCS 1267, Springer-Verlag*, pages 54–68, 1997.

[Mat00a] M. Matsui. Specification of MISTY1 - a 64-bit Block Cipher. Manuscript, Mitsubishi Electric Corporation, September 2000.

[Mat00b] M. Matsui. Specification of MISTY1 - a 64-bit Block Cipher, September 18, 2000.

[Mir02] I. Mironov. (Not So) Random Shuffles of RC4. In *Proceedings of Advances in Cryptology - Crypto 2002, LNCS 2442, Springer-Verlag*, 2002.

[MOV] A. Menezes, P. Van Oorschot, and S. Vanstone. *The Handbook of Applied Cryptography*. CRC Press.

[MS01] I. Mantin and A. Shamir. A Practical Attack on Broadcast RC4. In *Proceedings of Fast Software Encryption 2001, Springer-Verlag*, 2001.

[NES00] NESSIE. NESSIE Call for Cryptographic Primitives, Version 2.2. `https://www.cosic.esat.kuleuven.ac.be/nessie/call`, March, 8 2000.

[NES03] NESSIE. NESSIE Security Report, Version 2. `https://www.cosic.esat.kuleuven.ac.be/nessie`, February 2003.

[NIS99a] NIST. FIPS 46-3 Data Encryption Standard (DES), 1999.

[NIS99b] NIST. Status Report on the First Round of the Development of the Advanced Encryption Standard, Oct 1 1999.

[NIS00] NIST. Randomness Testing of the Advanced Encryption Standard Finalist Candidates, March 2000.

[NIS01a] NIST. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, NIST Special Publication 800-22. `csrc.nist.gov/publications/nistir`, 2001.

[NIS01b] NIST. FIPS 197 Advanced Encryption Standard (AES), 2001.

[NIS01c] NIST. SP800-38A Recommendation for Block Cipher Modes of Operation, December 2001.

[NIS02] NIST. FIPS 180-2 Secure Hash Standard, 2002.

[NIS04] NIST. SP800-38C Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality, May 2004.

[NR99] M. Noar and O. Reingold. On the Construction of Pseudo-random Permutations: Luby-Rackoff Revisited. In *Journal of Cryptology*, volume 12, pages 29–66, 1999.

[OSK+01] K. Ohkuma, H. Schimizu, S. Kawamura, F. Sano, H. Muratani, and M. Motoyam. Hierocrypt-3. `http://www.toshiba.co.jp/rdc/security/hierocrypt/index.htm`, 2001.

[OST06] D. Osvik, A. Shamir, and E. Tromer. Cache Attacks and Countermeasures: The Case of AES. In *Proceedings of RSA Conference Cryptographers Track (CT-RSA)*, 2006.

[PRS04] S. Patel, Z. Ramzan, and G. Sundaram. Efficient Constructions of Variable-Input-Length Block Ciphers. In *Proceedings of Selected Areas in Cryptography 2004, LNCS 3357, Springer-Verlag*, 2004.

[ran] random.org. `http://wwww.random.org/files`.

[Ree92] J. Reeds. III,. Cryptosystem for Cellular Telephony. US Patent 5,159,634, 1992.

[Riv95] R Rivest. The RC5 Encryption Algorithm. *CryptoBytes*, 1(1), 1995.

[Riv96] R. Rivest. RC4. in Applied Cryptography by B. Schneier, John Wiley and Sons, New York,, 1996.

[RRSY98] Rivest, Robshaw, Sidney, and Yin. RC6 Block Cipher. `http://www.rsa.security.com/rsalabs/rc6`, 1998.

[Sch98] R. Schroeppel. Hasty Pudding Cipher. `http://www.cs.arizona.edu/rcs/hpc`, 1998.

[Sha49] C. Shannon. Communication Theory of Secrecy. *Bell System Technical Journal*, 28(4):656–715, 1949.

[SK96] B. Schneier and J. Kelsey. Unbalanced Feistel Networks and Block Cipher Design. In *Proceedings of Fast Software Encryption 1996, LNCS 1039, Springer-Verlag*, 1996.

[SKW+98] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. Twofish. `http://www.schneier.com/twofish.html`, 1998.

[ST04] A. Shamir and E. Tromer. Acoustic Cryptanalysis On Nosy People and Noisy Machines. Eurocrypt rump session presentation, 2004.

[SYY+01] T. Shimoyama, H. Yanami, K. Yokoyama, M. Takenaka, K. Itoh, J. Yajima, N. Torii, and H. Tanaka. The Block Cipher SC2000. In *Proceedings of Fast Software Encryption 2001, LNCS 2355, Springer-Verlag*, pages 326–340, 2001.

[TKMN00] Y. Tsunoo, H. Kubo, H. Miyauchi, and K. Nakamura. A New 128-bit Block Cipher CIPHERUNICORN-A. ISEC2000-5 , Technical Report of IEICE, The Institute of Electronics, Information and Communication Engineers, 2000.

[Tra02] Transaction Processing Performance Council. TPC Benchmark (Web Commerce) Specification, Version 1.8. `http://http://www.tpc.org`, February 2002.

[WSB00] D. Whiting, B. Schneier, and S. Bellovin. AES Key Agility Issues in High-Speed IPsec Implementations. http://csrc.nist.gov/Crypto Toolkit/aes/round2/pubcmnts.htm, 2000.

[WSK97] D. Wagner, B. Schneier, and J. Kelsey. Cryptanalysis of the Cellular Message Encryption Algorithm. `http://www.schneier.com/paper-cmea.html`, 1997.

[YL06] T. Ylonen and C. Lonvick. IETF RFC 4253 The Secure Shell (SSH) Transport Layer Protocol. `http://www.ietf.org/rfc/rfc4253.txt`, 2006.

Appendix A

Randomness Test Results

A.1 Overview

There are statistical tests for measuring the randomness within a string of bits. Such tests can be applied to ciphertext produced by a block cipher to assist in determining if there are any obvious weaknesses with the cipher. I applied the tests used by NIST on the AES candidates to the four elastic block ciphers described in Chapter 6. This appendix contains a brief summary of the tests, and the results for the original and elastic versions of AES, Camellia, MISTY1 and RC6. In addition, the key stream produced by RC4 was tested to determine the randomness of the expanded-key bits. The test results for the elastic versions of all four ciphers indicate no obvious lack of randomness in the ciphertext. The test results for RC4 indicate RC4 is a feasible option for use as a key schedule that will generate expanded-key bits that can be considered pseudorandom when compared to the output of the key schedules of the four ciphers. Refer to NIST's special publication 800-22 [NIS01a] for a complete description of the tests and to the NIST report entitled "Randomness Testing of the Advanced Encryption Standard Finalist Candidates" [NIS00] for a complete description of the data sets. The data sets defined by NIST included the use of 128, 192 and 256-bit keys. All of the elastic block ciphers use RC4 with a 128-bit key; therefore, the data descriptions provided below include only 128-bit keys. Due to the varying block sizes, I was not able to reuse the library available from NIST to conduct the tests. Instead, my own code and some functions from the GNU gsl scientific library version

1.5 were used. When testing the elastic versions, every $b + y$ bit block size where y is an integral of 8 and $b < b + y < 2b$ were tested. I also tested two block sizes that were not an integral number of bytes. These were 129 and 171 bit blocks for the elastic versions of AES, Camellia and RC6, and 69 and 75 bit blocks for elastic MISTY1. Each data set required either an initial set of random plaintexts or random keys. I created these random bit strings by extracting bits from files of random bits available from random.org [ran].

A.2 Test Descriptions

Sixteen tests were performed on eight sets of data for each cipher. In the following descriptions, "bit sequence" refers to the entire string of bits tested.

1. Frequency (Monobit): This determines if the proportions of 0's and 1's in the bit sequence are close enough to $\frac{1}{2}$.

2. Frequency within a Block: This is the Frequency test applied to fixed-sized blocks within the bit sequence. It is the same as the first test when the number of blocks is 1.

3. Runs: The number of runs (a sequence of all 0's or all 1's) in the bit sequence is determined.

4. Longest Run of Ones within a Block: The longest run of 1's within a block is determined.

5. Binary Matrix Rank: 32-by-32 matrices are created from the bit sequence and their ranks computed. This determines if there is any linear dependence among fixed-length segments of bits within the sequence.

6. Discrete Fourier Transform: This test determines if there are repetitive patterns in the bit sequence.

7. Non-overlapping Template Matching: This test counts the number of times a m-bit pattern occurs in the bit sequence using a sliding window. The window slides 1 bit when a match does not occur and slides m bits when a match occurs so a bit will be

involved in at most one match for a given pattern. m was set to 9 to be consistent with the value NIST used and all 9-bit patterns were tested.

8. Overlapping Template Matching: This is the same as the previous test except that the window always slides 1 bit. Therefore, a bit may be involved in more than one match for a given pattern.

9. Maurer's Universal Statistical: This determines if the bit sequence can be compressed based on the number of bits between occurrences of a pattern.

10. Lempel-Ziv Compression: This determines how much a bit sequence can be compressed based on the number of distinct patterns.

11. Linear Complexity: The Berlekamp-Massey algorithm [MOV] is applied to a 1000 bit sequence to determine a linear feedback shift register (LFSR) that produces the sequence. The length of the LFRS indicates if the sequence is sufficiently random.

12. Serial: The number of times each 2^m bit pattern occurs is determined, for some integer m.

13. Approximate Entropy: The number of times each 2^m and each 2^{m+1} bit pattern is determined, for some integer m.

14. Cumulative Sums: The cumulative sum of the bits is computed for each position in the sequence. The sum is computed by adding -1 for each bit that is 0 and adding 1 for each bit that is 1.

15. Random Excursions: Using cumulative sums, the number of times the sum crosses zero is determined.

16. Random Excursions Variant: Using cumulative sums, the number of times the sum is a particular value is determined.

The data sets involved 128-bit keys and $(b + y)$-bit plaintexts. Unless otherwise stated, ECB mode is used when encrypting the plaintexts. (Note: these data sets do not apply when testing RC4's output. The data used when testing RC4 is described with the RC4

test results.) In the NIST tests, the last four data sets included additional data formed by using plaintexts or keys containing exactly two 0's (low density case) or exactly two 1's (high density case). Only single 0's and single 1's were used when testing the elastic block ciphers due to the volume of data and the overall level of randomness illustrated by the data sets across all the other tests (indicating there was no need to continue testing). The eight data sets are:

1. Plaintext Avalanche: The key is a fixed random value. Random plaintexts are used. The data tested is the XOR of the encrypted plaintext and the encryption of the plaintext with the i^{th} bit flipped. This is repeated for $i = 1$ to $b + y$ and for all plaintexts.

2. Key Avalanche: The plaintext consists of all zeroes. Random keys are used. The data tested is the XOR of the plaintext encrypted with a random key and the plaintext encrypted with the random key with the i^{th} bit flipped. This is repeated for $i = 1$ to 128 and for all keys.

3. Plaintext-Ciphertext Correlation: Random keys and random plaintexts were used. The data tested consisted of the ciphertext XORed with the plaintext, for all plaintexts and all keys.

4. CBC Mode: Random keys, random plaintexts and an IV of all 0's are used. For each key, the plaintexts are encrypted using CBC mode.

5. Low Density Plaintext: Random keys are used. For each key, a plaintext block of all 0's and every plaintext block containing exactly one 1 are encrypted.

6. Low Density Keys: Random plaintext blocks are used. Each plaintext is encrypted with a key of all 0's and every key containing a single 1.

7. High Density Plaintext: This is the same as the low density plaintexts except plaintexts of all 1's and plaintexts with a single 0 are used instead of all 0's and a single 1.

8. High Density Keys: This is the same as the low density keys except keys of all 1's and keys with a single 0 are used instead of all 0's and a single 1.

A.3 Block Ciphers: Test Results

The test results are summarized in the tables in the remainder of this appendix. In each test, one or more P-values are calculated. A sample passes the test if the computed P-value(s) is greater than 0.01. The P-value is the probability of the bit sequence observed in the sample occurring in a sequence of random bits. The smaller the P-value, the less likely the sample is random. The results for the four elastic block ciphers are consistent with the percentage of samples from the AES finalists passing the tests in NIST's AES competition. For the elastic versions of the ciphers, the percentage of samples passing were consistent across all block sizes and data sets; therefore, the range (minimum and maximum percent) of samples passing across all block sizes and data sets is provided as a summary as opposed to listing the percentage of samples passing per block size per data set per test. The number of samples per test ranged from 100 to 300, depending on the amount of data required. The sample sizes are provided in the tables with the test results. I also tested the original versions of the ciphers and provide the results for comparison. The sample size used in each test on the original ciphers is the same as sample size used for the elastic versions. Due to the smaller number of data set - test combinations (there is only one block size to test with the original version), I report the results for each data set for each test on the original versions of the ciphers.

Notes:

1. In cases where a set failed test 5, it was due to the number of matrices having full rank exceeding what is expected as opposed to matrices of low rank existing within the data. Thus, the failures were not due to the existence of linear relationships among the ciphertext bits.

2. The results reported for test 7 is the percent of the 512 9-bit patterns passed by a sample. For example, if the minimum and maximum values reported are 96.68% and 98.05%, respectively, then each of the 100 samples passed between 495 and 502 of the

patterns. The total number of cases passing (out of 512 patterns times the number of samples) exceeds 96.33% for each block size and data set.

3. Test 8 is a template matching test. In the test descriptions provided by NIST, the criteria for passing the test is defined for the 9-bit template of all 1's [NIS01a]; therefore, I ran the test with this 9-bit template.

4. Test 15 computes 8 probabilities and test 16 computes 18 probabilities. For both of these tests, I marked a sample as failing if one or more of the probabilities did not meet the required threshold as opposed to reporting the percentage that passed out of all cases over all samples, which exceeds 96.33% for all ciphers tested and all data sets. In most cases where a sample failed test 15, it was due to exactly one of the 8 values failing. In most cases where a sample failed test 16, it was due to one to three of the 18 values failing. In both tests, no particular probability was responsible for the majority of the failures.

5. In the general summary of the test results for the AES finalists, it is stated that tests which resulted in a pass rate of $< 96.33\%$ were subject to further analysis [NIS00]. Therefore, I consider an elastic version of a block cipher to exhibit sufficient randomness if at least 96.33% of the samples passed each test. The results I report for tests 15 and 16 are not covered by the 96.33% threshold because I report the range of the 8 or 18 cases passed per sample as opposed to over all samples. The results I report for test 7 also are not covered by the 96.33% threshold because I report the range of cases out of 512 passing per sample as opposed to the total percent passed over all samples. Because the results for each of these three tests exceeded the threshold if the percent passed is counted over all cases over all samples, I choose to report the minimum and maximum number of cases passing for a single sample to show if every single sample was passing every case or if there were some samples failing multiple cases.

A.4 RC4: Test Results

RC4 was used as the key schedule in each of the elastic block ciphers. The randomness of RC4's output using random keys and related keys was tested. In all tests, the first 512 bytes of the key stream were discarded, as was done when using RC4 for the key schedule. The following three data sets were tested:

1. The RC4 key stream in general was tested using 300 random keys. For each random key, 1 million bytes of key stream were generated.

2. For each of 300 random keys, the 128 keys corresponding to flipping one bit of the original key were created. For each key, 10,000 key stream bytes were generated and XORed with 10,000 bytes generated using the original key (the key prior to flipping one bit).

3. For each 128 bit key consisting of one 1 and 127 0's, 1 million key stream bytes were generated.

4. For each 128 bit key consisting of one 0 and 127 1's, 1 million key stream bytes were generated.

The random keys were the same 300 128-bit random keys used when testing the block ciphers. The related keys (flipping one bit, a single 1 and a single 0) are the same related keys used in some of the data sets defined for the block cipher. The number of key stream bytes generated was set to provide enough data for at least 100 samples for each test except for test 7, which involved 10 samples each tested with 512 different patterns.

For the first data set, each case that failed test 15 or test 16 was due to one pattern failing and there was no individual pattern that caused the failures. For data sets 2, 3 and 4, most failures in tests 15 and 16 were caused by a sample failing a single pattern. No individual pattern was causing the failures. A few samples failed multiple patterns: for data set 2, test 16: four samples failed two patterns each. For data set 3, test 15, one sample failed 2 patterns. For data set 3, test 16, two samples failed 4 patterns and one sample failed 3 patterns. For data set 4, test 15: two samples failed 2 patterns and one sample failed 3 patterns. For data set 4, test 16: three samples failed 2 patterns, and there was

one case each where a sample failed 4,6 and 8 patterns. While no individual pattern was responsible for the failures, in the cases of the 6 and 8 patterns failing, these were the last 6 and first 8 patterns that failed.

	Data Set							
Test	1	2	3	4	5	6	7	8
1	99.33	98.00	99.00	99.67	98.33	99.33	99.00	99.67
2	100	97.00	98.33	98.67	99.00	99.00	99.00	98.33
3	98.33	99.33	100	98.67	99.00	99.67	97.33	98.67
4	99.00	99.67	99.67	98.33	98.67	99.33	99.67	100
5	99.00	100	99.00	99.00	97.00	98.00	98.00	96.00
6	100	100	100	100	100	100	100	100
7 min	95.31	94.92	94.33	94.92	96.09	94.92	94.72	96.87
max	96.67	97.65	97.46	97.26	97.26	97.65	96.87	97.07
8	100	100	100	100	100	100	100	100
9	100	100	100	100	95	100	100	100
10	100	100	100	100	100	100	100	100
11	100	100	100	100	100	100	100	100
12	98.33	96.67	98.67	99.00	98.33	98.33	99.67	98.33
13	98.00	99.00	98.67	100	98.67	98.33	99.33	99.00
14	100	100	100	100	95	100	100	100
15	80.00	100	80.00	100	80.00	80.00	60.00	100
16	80.00	100	100	100	80.00	100	100	100

Table A.1: AES with 128-bit Block Size: Percent of Samples Passing Tests

Refer to the notes in Section A.3 for the meaning of the results for tests 7,15 and 16.

Test	Min	Max	Notes
1	97.33	99.67	300 samples
2	97.33	100	300 samples
3	97.33	100	300 samples
4	97.00	100	300 samples
5	95.00	100	100 samples of 38 matrices
6	100	100	300 samples
7	93.35	98.43	100 samples, 512 9-bit patterns
8	100	100	100 samples
9	97.00	100	100 samples of 2,068,400 bits
10	100	100	100 samples of 200,000 bits
11	100	100	100 samples
12	96.33	100	300 samples of 10,000 bits, 3 bit patterns
13	97.00	100	300 samples of 10,000 bits, 3 bit patterns
14	100	100	100 samples of 100 bits
15	40.00	100	100 samples of 1 million bits, 8 cases
16	60.00	100	100 samples of 1 million bits, 18 cases

Table A.2: Elastic AES: Percent of Samples Passing Over All Data Sets and Tests
Refer to the notes in Section A.3 for the meaning of the results for tests 7,15 and 16.

	Data Set							
Test	1	2	3	4	5	6	7	8
1	99.33	98.33	99.67	98.33	99.67	99.00	98.67	99.00
2	99.33	99.67	98.00	99.67	100	98.67	99.33	98.33
3	99.00	100	98.33	100	99.33	98.67	98.67	98.67
4	98.00	99.00	99.33	99.67	99.00	99.00	100	98.67
5	99.00	99.00	99.00	100	100	100	100	100
6	100	100	99.67	100	100	100	100	100
7 min	95.50	94.92	96.48	96.67	96.48	96.09	95.31	97.07
max	97.26	96.87	97.26	98.04	96.67	96.87	96.87	97.65
8	100	100	100	100	100	100	100	100
9	100	100	100	100	100	100	100	100
10	100	100	100	100	100	100	100	100
11	100	100	100	100	100	100	100	100
12	99.33	98.00	98.33	99.33	99.33	99.00	98.33	98.33
13	99.00	99.67	99.33	99.67	99.67	98.67	99.00	100
14	100	100	100	100	95	100	100	100
15	80.00	100	100	100	100	80.00	100	100
16	60.00	100	100	100	100	100	80.00	100

Table A.3: Camellia with 128-bit Block Size: Percent of Samples Passing Tests
Refer to the notes in Section A.3 for the meaning of the results for tests 7,15 and 16.

Test	Min	Max	Notes
1	95.00	100	300 samples
2	96.00	100	300 samples
3	97.00	100	300 samples
4	98.00	100	300 samples
5	97.00	100	100 samples of 38 matrices
6	100	100	300 samples
7	93.94	98.05	100 samples, 512 9-bit patterns
8	100	100	100 samples
9	97.00	100	100 samples of 2,068,400 bits
10	100	100	100 samples of 200,000 bits
11	100	100	100 samples
12	97.00	100	300 samples of 10,000 bits, 3 bit patterns
13	97.33	100	300 samples of 10,000 bits, 3 bit patterns
14	100	100	100 samples of 100 bits
15	40.00	100	100 samples of 1 million bits, 8 cases
16	40.00	100	100 samples of 1 million bits, 18 cases

Table A.4: Elastic Camellia: Percent of Samples Passing Over All Data Sets and Tests
Refer to the notes in Section A.3 for the meaning of the results for tests 7,15 and 16.

	Data Set							
Test	1	2	3	4	5	6	7	8
1	99.00	99.00	99.00	98.00	99.00	98.00	99.00	99.00
2	100	100	99.00	100	99.00	97.00	100	100
3	99.00	100	98.00	99.00	99.00	100	98.00	99.00
4	100	100	100	98.00	98.00	100	100	100
5	100	100	100	96.67	100	96.67	100	100
6	100	100	100	100	95	100	100	100
7 min	96.29	96.48	96.28	96.09	97.65	95.31	95.51	97.66
max	97.46	98.43	98.05	97.66	98.43	97.65	97.65	98.05
8	100	100	100	100	100	100	100	100
9	97.00	98.00	100	100	98.00	100	97.00	100
10	100	100	100	100	100	100	100	100
11	100	100	100	100	100	100	100	100
12	100	98.00	99.00	97.00	97.00	97.00	97.00	98.00
13	100	100	100	97.00	98.00	99.00	97.00	99.00
14	100	100	100	100	95.00	100	100	100
15	80	100	100	100	95.00	100	100	100
16	100	100	50.00	50.00	100	100	100	100

Table A.5: MISTY1 with 64-bit Block Size: Percent of Samples Passing Tests
Refer to the notes in Section A.3 for the meaning of the results for tests 7,15 and 16.

Test	Min	Max	Notes
1	97.00	99.67	300 samples
2	98.33	100	300 samples
3	97.67	100	300 samples
4	97.67	100	300 samples
5	97.00	100	100 samples
6	99.67	100	300 samples
7	98.04	98.43	100 samples, 512 9-bit patterns
8	100	100	100 samples
9	98.00	100	100 samples of 2,068,400 bits
10	100	100	100 samples of 200,000 bits
11	100	100	100 samples
12	96.00	100	150 samples of 10,000 bits, 3 bit patterns
13	98.00	100	150 samples of 10,000 bits, 3 bit patterns
14	100	100	100 samples of 100 bits
15	50.00	100	100 samples of 1 million bits, 8 cases
16	50.00	100	100 samples of 1 million bits, 18 cases

Table A.6: Elastic MISTY1: Percent of Samples Passing Over All Data Sets and Tests
Refer to the notes in Section A.3 for the meaning of the results for tests 7,15 and 16.

	Data Set							
Test	1	2	3	4	5	6	7	8
1	99.67	99.67	98.33	98.33	99.33	99.33	99.33	99.00
2	99.33	99.00	99.00	98.67	99.67	99.00	99.33	99.00
3	98.33	98.33	99.33	99.33	99.33	100	99.33	99.67
4	99.67	99.00	99.00	99.33	99.00	99.67	99.00	99.00
5	100	100	98.00	100	98.00	98.00	98.00	99.00
6	100	100	100	100	100	100	100	100
7 min	95.70	96.09	95.51	96.09	97.66	95.11	97.66	95.51
max	97.27	97.46	97.66	96.48	97.66	97.27	97.66	97.27
8	100	100	100	100	95	100	100	100
9	98.00	100	100	100	100	100	100	100
10	100	100	100	100	100	100	100	100
11	100	100	100	100	100	100	100	100
12	98.33	99.67	97.33	98.00	98.33	98.33	98.33	98.67
13	98.33	99.00	99.33	99.00	99.00	99.00	99.00	99.33
14	100	100	100	100	100	100	100	100
15	100	80.00	80.00	100	100	80.00	100	60.00
16	100	100	60.00	100	80.00	80.00	80.00	60.00

Table A.7: RC6 with 128-bit Block Size: Percent of Samples Passing Tests

Refer to the notes in Section A.3 for the meaning of the results for tests 7,15 and 16.

Test	Min	Max	Notes
1	97.00	100	300 samples
2	95.00	100	300 samples
3	96.00	100	300 samples
4	98.00	100	300 samples
5	96.00	100	100 samples of 38 matrices
6	100	100	300 samples
7	92.58	98.05	100 samples, 512 9-bit patterns
8	100	100	100 samples
9	98.00	100	100 samples of 2,068,400 bits
10	100	100	100 samples of 200,000 bits
11	100	100	100 samples
12	96.67	100	300 samples of 10,000 bits, 3 bit patterns
13	97.67	100	300 samples of 10,000 bits, 3 bit patterns
14	100	100	100 samples of 100 bits
15	40.00	100	100 samples of 1 million bits, 8 cases
16	60.00	100	100 samples of 1 million bits, 18 cases

Table A.8: Elastic RC6: Percent of Samples Passing Over All Data Sets and Tests
Refer to the notes in Section A.3 for the meaning of the results for tests 7,15 and 16.

	Data Set			
Test	1	2	3	4
1	100	100	100	100
2	100.00	99.00	100	100
3	98.00	98.00	99.00	95.00
4	99.00	99.00	99.00	99.00
5	99.00	100	99.00	99.00
6	100	100	100	100
7	96.09 to 97.66	95.70 to 97.27	95.31 to 97.66	96.09 to 98.05
8	100	100	100	100
9	99.00	94.00	100	99.00
10	100	100	100	100
11	100	100	100	100
12	98.00	98.00	98.00	98.00
13	98.00	98.00	98.00	99.00
14	100.00	100	100	100
15	93.00	94.00	90.00	90.00
16	97.00	91.00	94.00	92.00

Table A.9: Expanded-Key Bytes from RC4: Percent of Samples Passing Tests

Refer to the notes in Section A.3 for the meaning of the results for tests 7,15 and 16.

Appendix B

Performance Results

The tables contained within this appendix summarize the performance results for the elastic versions of AES, Camellia, MISTY1 and RC6. These are the values used to generate the graphs in Chapter 6.

Number of Bytes	Number of Rounds	Number of Blocks Encrypted Case 1	Number of Blocks Encrypted Case 2	Number of Blocks Encrypted Case 3
17	11	190	186	97
18	12	182	169	80
19	12	154	158	75
20	13	153	142	59
21	14	143	128	49
22	14	125	125	46
23	15	125	114	40
24	15	122	111	37
25	16	121	101	32
26	17	106	93	28
27	17	101	90	27
28	18	101	84	23
29	19	100	78	21
30	19	88	76	20
31	20	88	71	18
32	20	83	69	17

Table B.1: Normalized Number of Blocks Encrypted by Elastic AES in Unit Time (Regular AES = 100)

Number of Bytes	Number of Rounds	Number of Blocks Encrypted Case 1	Number of Blocks Encrypted Case 2
17	10	95	161
18	11	82	137
19	11	72	108
20	12	62	95
21	12	58	89
22	13	49	81
23	13	45	61
24	14	41	55
25	15	34	48
26	15	32	42
27	16	29	38
28	16	27	34
29	17	25	31
30	17	24	27
31	18	21	24
32	18	21	23

Table B.2: Normalized Number of Blocks Encrypted by Elastic Camellia in Unit Time (Regular Camellia = 100)

Number of Bytes	Number of Rounds	Number of Blocks Encrypted Case 1	Number of Blocks Encrypted Case 2
9	5	129	185
10	5	122	143
11	6	104	130
12	6	101	120
13	7	76	86
14	7	76	86
15	8	68	75
16	8	66	75

Table B.3: Normalized Number of Blocks Encrypted by Elastic MISTY1 In Unit Time (Regular MISTY1 = 100)

Number of Bytes	Number of Rounds	Number of Blocks Encrypted
17	6	125
18	6	114
19	6	105
20	7	102
21	7	90
22	7	78
23	8	77
24	8	71
25	8	70
26	9	59
27	9	54
28	9	53
29	10	53
30	10	45
31	10	44
32	10	43

Table B.4: Normalized Number of Blocks Encrypted by Elastic RC6 in Unit Time (Regular RC6 = 100)

Appendix C

AES

C.1 Encryption and Decryption

The following is a description of the AES block cipher for 128-bit blocks and 128-bit keys as described in FIPS 197 [NIS01b]. The steps described in FIPS 197 can be rearranged and combined to produce a representation in which the round function consists entirely of table lookups and XORs. This later version results in significantly faster encryption and decryption rates compared to the an implementation following the steps in FIPS 197, but requires additional memory to store the tables.

For 128-bit blocks and 128-bit keys, the AES round function for encryption is typically described with data represented as a 4x4-byte matrix, A, with each entry containing one byte.

$$A = \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}$$

The round function is applied ten times. The data is XORed with key bits prior to the first round. A round of encryption consists of the following steps:

(I) SubBytes (S-Box applied to each entry)
ShiftRows (bytes within each row of A are shifted 0 to 3 columns)
MixColumns (a matrix multiplication; absent in the last round)
AddRoundKey (A is XORed with a round key)

The round function for decryption applies AddRoundKey then the inverse functions for MixColumns, ShiftRows and SubBytes.

Encryption is defined as follows:

```
AddRoundKey
for (i=1; i < 10; ++i) {
    SubBytes
    ShiftRows
    MixColumns
    AddRoundKey
}
SubBytes
ShiftRows
AddRoundKey
```

In the SubBytes step, each byte is used as in index into a table and the byte is replaced with the table entry. In block ciphers, tables used for such substiutions are referred to as S-Boxes. Tables C.1 and C.2 contain the S-Boxes for AES encryption and decryption, respectively. The table lookup is performed by viewing the byte as two 4-bit values, with the leftmost 4 bits used as the row index and the rightmost 4 bits used as the column index. For example, 0x29 uses the row corresponding to 2 and column corresponding to 9. 0x29 is replaced with 0xa5 when encrypting and 0x4c when decrypting.

In the ShiftRows step, the entries in the i^{th} row of the matrix A are rotated i positions to the left, for $i = 0$ to 3, when encrypting. Specifically, the matrix A

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Table C.1: AES's S-Box for Encryption

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}$$

becomes

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{11} & a_{12} & a_{13} & a_{10} \\ a_{22} & a_{23} & a_{20} & a_{21} \\ a_{33} & a_{30} & a_{31} & a_{32} \end{pmatrix}$$

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Table C.2: AES's S-Box for Decryption

When decrypting, the rotation is reversed.

The MixColumns step consists of the matrix multiplication $M_e * A$ when encrypting where M_e is the constant matrix:

$$M_e = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$$

MixColumns uses the inverse of M_e when decrypting and consists of the matrix multiplication $M_d * A$ where M_d is the constant matrix:

$$M_d = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix}$$

A faster implementation for environments with sufficient memory operates on 32-bit words and reduces the AES round function described in (I) to four table lookups and four XORs. If A denotes a 4x4 matrix input to the round, $a_{i,j}$ denotes the i^{th} row and j^{th} column of A, $j - x$ is computed modulo 4, and Tk are tables with 256 32-bit entries, the round function is reduced to the form:

(II) $$A'_j = T0[a_{0,j}] \oplus T1[a_{1,j-1}] \oplus T2[a_{2,j-2}] \oplus T3[a_{3,j-3}] \oplus RoundKey$$

where A'_j denotes the j^{th} column of the round's output. Refer to pages 58–59 of *The Design of Rijndael* [DR02] for a complete description and the derivation of this version. The entries in the tables in (II) are concatenations of 1, 2, and 3 times the S-Box entries. This version is due to the fact that the order of the SubBytes and ShiftRows steps can be switched and the MixColumn step can be viewed as the linear combination of four column vectors, which is actually a linear combination of the S-Box entries.

C.2 Key Schedule

AES's key schedule expands the key to eleven 128-bit round keys used for the AddRoundKey steps (10 rounds plus the initial AddRoundKey). Each round key is viewed as four 32-bit words. The key schedule creates the eleven round keys as an array of forty-four 32-bit words. The 128-bit key is split into four 32-bit words to form the first four array entries. Each remaining word is formed by XORing two previous words or by performing an S-Box lookup on a previous word then XORing it with a constant and a previous word. The following is pseudo code for the key schedule when using 128-bit keys. Refer to FIPS 197

[NIS01b] for a general description of the key schedule that processes 128, 196 and 256-bit keys.

- EK is the array of 32-bit words containing the expanded key.
- x is a word
- concat(a,b,c,d) indicates the concatenation of the inputs to form a single bit-string of abcd.
- x <<< 8 means to rotate x to the left by 8 bits
- C is an array of constants.
- SubWord(x) applies the S-Box used in round function to each byte of the word x.
- C = [0x01000000,0x02000000,0x04000000,0x08000000,0x10000000,
  0x20000000,0x40000000,0x80000000,0x1b000000,0x36000000]

```
/* Place the 128-bit key in the first 4 entries of EK */
for (i=0; i < 4; ++i) {
    EK[i] = concat(K[4*i] K[4*i+1] K[4*i+2] K[4*i+3])
}
/* The first word of each remaining round key is formed from the XOR
of a S-Box entry, a constant and a previous word.
The second to fourth words of each remaining round key is the XOR
of two previous words. */
for (i=4; i < 44; ++i) {
    x = EK[i-1]
    if (i mod 4 == 0) {
      x = SubWord((x <<< 8)) XOR C[i/4]
    }
    EK[i] = EK[i-4] XOR x
}
```

Appendix D

Camellia

D.1 Encryption and Decryption

The following summarizes the Camellia block cipher [AIK+00]. The cipher is a Feistel network with the addition of a function applied to each half of the data after every six rounds, excluding the last round (thus the function is applied after the 6^{th} and 12^{th} rounds). Eighteen rounds are used. The block size is 128 bits. There is initial and final whitening (XOR with key bits).
Notation:

- L_i and R_i denote the left and right halves of output, respectively, of the i^{th} round. $i = 0$ denotes the input to round 1.
- $FL(x, kl)$ is a function taking input x and key material kl.
- $FL^{-1}(x, kl)$ is the inverse of FL.
- $F(x, k)$ is a function taking 64 bits bit string x and 64 bits of key material k.
- k and kl denote subkeys from the expanded key material, with a subscript of i denoting the i^{th} component.
- $\cup$ is the bitwise OR operator.
- $\cap$ is the bitwise AND operator.

- When a string of bytes is divided into individual bytes, the bytes are numbered from left to right.
- $<<< z$ is a left rotation by z bits.
- $<< z$ and $>> z$ are shifts to the left and right, respectively, by z bits.

All rounds except for rounds 6 and 12 are defined by:

$L_i = R_{i-1} \oplus F(L_{i-1}, k_i)$

$R_i = L_{i-1}$

The input to round 1 has whitening applied first. The output of round 18 has whitening applied.

The outputs of rounds 6 and 12 is defined by creating L_i and R_i as in the other rounds then applying the function FL to each:

$L_i = FL(R_{i-1} \oplus F(L_{i-1}, k_i), kl_{(2i/6)-1})$.

$R_i = FL^{-1}(L_{i-1}, kl_{2i/6})$.

F Function:

The round funtion, F, is defined as: $F(x, k) = P(S(x \oplus k))$, where S is a S-Box. P is a function that XORs bytes of its 8-byte input to form an 8-byte output. The bytes of the output are formed by XORing the following bytes of input:

Output Byte	Input Bytes XORed
1	1,3,4,6,7,8
2	1,2,4,5,7,8
3	1,2,3,5,6,8
4	2,3,4,5,6,7
5	1,2,6,7,8
6	2,3,5,7,8
7	3,4,5,6,8
8	1,4,5,6,7

Table D.1: Camellia's P Function

The substitution performed by S is done by viewing the data as 8 bytes and using one of four S-Boxes, $(S1, S2, S3, S4)$, on each byte. Bytes 1 and 8 have S1 applied, bytes 2 and 5 have S2 applied, bytes 3 and 6 have S3 applied, and bytes 4 and 7 have S4 applied. The four S-Boxes are created from one S-Box as follows:

```
S =
[112,130, 44,236,179, 39,192,229,228,133, 87, 53,234, 12,174, 65,
 35,239,107,147, 69, 25,165, 33,237, 14, 79, 78, 29,101,146,189,
134,184,175,143,124,235, 31,206, 62, 48,220, 95, 94,197, 11, 26,
166,225, 57,202,213, 71, 93, 61,217,  1, 90,214, 81, 86,108, 77,
139, 13,154,102,251,204,176, 45,116, 18, 43, 32,240,177,132,153,
223, 76,203,194, 52,126,118,  5,109,183,169, 49,209, 23,  4,215,
 20, 88, 58, 97,222, 27, 17, 28, 50, 15,156, 22, 83, 24,242, 34,
254, 68,207,178,195,181,122,145, 36,  8,232,168, 96,252,105, 80,
170,208,160,125,161,137, 98,151, 84, 91, 30,149,224,255,100,210,
 16,196,  0, 72,163,247,117,219,138,  3,230,218,  9, 63,221,148,
135, 92,131,  2,205, 74,144, 51,115,103,246,243,157,127,191,226,
 82,155,216, 38,200, 55,198, 59,129,150,111, 75, 19,190, 99, 46,
233,121,167,140,159,110,188,142, 41,245,249,182, 47,253,180, 89,
120,152,  6,106,231, 70,113,186,212, 37,171, 66,136,162,141,250,
114,  7,185, 85,248,238,172, 10, 54, 73, 42,104, 60, 56,241,164,
 64, 40,211,123,187,201, 67,193, 21,227,173,244,119,199,128,158]


For i = 0 to 255:
S1[i] = S[i]
S2[i] = (S[i] >> 7 XOR S[i] << 1) & 0xff
S3[i] = (S[i] >> 1 XOR S[i] << 7) & 0xff
S4[i] = S[((i) << 1 XOR i >> 7) & 0xff]
```

FL Function:

The FL function takes a 64-bit input and 64 bits of expanded-key bits. Let X_L and

X_R denote the left and right halves of the input, respectively and Y_L and Y_R denote the left and right halves of the output, respectively. Let kl_L and kl_R denote the left and right halves of the 64 key bits.

FL is defined as:

$$Y_R = ((X_L \cap kl_L) <<< 1) \oplus X_R$$

$$Y_L = (Y_R \cup kl_R) \oplus X_L$$

FL^{-1} is:

$$X_L = (Y_R \cup kl_R) \oplus Y_L$$

$$X_R = ((X_L \cap KL_L) <<< 1) \oplus Y_R$$

D.2 Key Schedule

Camellia's key schedule is defined as follows for 128-bit keys. Camellia also supports 192-bit and 256-bit keys. Refer to Camellia's specifications [AIK+00] for a full description. Let K be the original 128-bit key. Encrypt K with two rounds of Camellia using constants as the key. The first two constants are from Table D.2. XOR the result with K and encrypt it two rounds using the third and fourth constants from the table. Call the result KA. The constants values used for the round keys are:

Number	Value
1	0xA09E667F3BCC908B
2	0xB67AE8584CAA73B2
3	0xC6EF372FE94F82BE
4	0x54FF53A5F1D36F1C

Table D.2: Camellia's Constants Used in the Key Expansion for 128-bit Keys

The subkeys for the rounds and whitening are then created from K and KA as shown in Table D.3. A subscript of L or R indicates the left or right 64 bits of a 128-bit value. The number after 'F' indicates the round.

Key	Value
initial whitening	K
F 1	KA_L
F 2	KA_R
F 3	$(K <<< 15)_L$
F 4	$(K <<< 15)_R$
F 5	$(KA <<< 15)_L$
F 6	$(KA <<< 15)_R$
FL	$(KA <<< 30)_L$
FL^{-1}	$(KA <<< 30)_R$
F 7	$(K <<< 45)_L$
F 8	$(K <<< 45)_R$
F 9	$(KA <<< 45)_L$
F 10	$(K <<< 60)_R$
F 11	$(KA <<< 60)_L$
F 12	$(KA <<< 60)_R$
FL	$(K <<< 77)_L$
FL^{-1}	$(K <<< 77)_R$
F 13	$(K <<< 94)_L$
F 14	$(K <<< 94)_R$
F 15	$(KA <<< 94)_L$
F 16	$(KA <<< 94)_R$
F 17	$(K <<< 111_L$
F 18	$(K <<< 111)_R$
final whitening	$K_A <<< 111$

Table D.3: Camellia's Key Expansion

Appendix E

MISTY1

E.1 Encryption and Decryption

The following summarizes the MISTY1 block cipher [Mat97, Mat00a]. The cipher is a Feistel network with the addition of a function applied to each half of the data at the start of the odd numbered rounds, thus the round function differs between even and odd numbered rounds. While the number of rounds is not fixed, eight rounds are recommended [NES03]. The block size is 64 bits.
Notation:

- L_i and R_i denote the left and right halves of output, respectively, of the i^{th} round after the halves are switched with $i = 0$ denoting the input to round 1.
- $FL(x, KL)$ and $F0(x, K0, KI)$ are functions taking bit string x and key material $KL, K0, KI$.
- FL_i and $F0_i$ denote the i^{th} occurrence of FL and $F0$, respectively.
- KL, KI and $K0$ denote subkeys from the expanded-key material, with a subscript of i denoting the i^{th} component.
- $\cup$ is the bitwise OR operator.
- $\cap$ is the bitwise AND operator.

The output of odd numbered rounds is defined by:

$R_i = FL_i(L_{i-1}, KL_i)$

$L_i = FL_{i+1}(R_{i-1}, KL_{i+1}) \oplus F0_i(R_i, K0_i, KI_i)$

The output of even numbered rounds is defined by:

$R_i = L_{i-1}$

$L_i = R_{i-1} \oplus F0_i(R_i, K0_i, KI_i)$

FL Function:

The FL function takes a 32-bit input and 32 bits of expanded-key bits. Let X_L and X_R denote the left and right halves of the input, respectively. Let KL_{iL} and KL_{iR} denote the left and right halves of the 32 key bits. The index i refers to the component.

$Y_R = (X_L \cap KL_{iL}) \oplus X_R$

$Y_L = (Y_R \cup KL_{iR}) \oplus X_L$

The 32-bit output is $Y_L||Y_R$.

The inverse of FL is used in decryption and is defined by

$X_L = (Y_R \cup KL_{iR}) \oplus Y_L$

$X_R = (X_L \cap KL_{iL}) \oplus Y_R$

The 32-bit output is $X_L||X_R$.

$F0$ Function:

The $F0$ function takes a 32-bit input, a 64-bit key and a 48-bit key (the keys are from the expanded key bits). Let L_0 and R_0 denote the left and right halves of the input, respectively. Let KO_i be the 64-bit key and KI_i be the 48-bit key. KO_i and KI_i are each divided into 16-bit segments. KO_ij and KI_ij denote the j^{th} 16-bit segment of KO_i and KI_i, respectively.

For $(j = 1; j <= 3; ++j)$ {

$R_j = FI(L_{j-1} \oplus KO_{ij}, KI_{ij}) \oplus R_{j-1}$

$L_j = R_{j-1}$

}

The value $(L_3 \oplus KO_i4)||R_3$ is returned.

FI Function:

The FI function is defined as follows:

It takes a 16-bit input, X_j, and a 16-bit key, KI_{ij}. $X_j = L_{0(9)}||R_{0(7)}$ where (9) and (7) indicates 9 and 7 bits, respectively. $KI_{ij} = KI_{ijL(7)}||KI_{ijR(9)}$.

$S7$ and $S9$ are two S-Boxes mapping 7 and 9-bit inputs to 7 and 9-bit outputs, respectively.

$ZE(x)$ is a function that takes a 7-bit input, x and adds two 0's as the most significant bits.

$TR(x)$ is a function that takes a 9-bit input, x, and discards the two most significant bits.

$L_{1(7)} = R_{0(7)}$

$R_{1(9)} = S9(L_{0(9)} \oplus ZE(R_{0(7)})$

$L_{2(9)} = R_{1(9)} \oplus KI_{ijR(9)}$

$R_{2(7)} = S7(L_{1(7)}) \oplus TR(R_{1(9)}) \oplus KI_{ijL(7)}$

$L_{3(7)} = R_{2(7)}$

$R_{3(9)} = S9(L_{2(9)}) \oplus ZE(R_{2(7)})$

FI returns $L_{3(7)}||R_{3(9)}$.

S-Boxes:

$S7$:

```
[
0x1b,0x32,0x33,0x5a,0x3b,0x10,0x17,0x54,0x5b,0x1a,0x72,0x73,0x6b,0x2c,
0x66,0x49,
0x1f,0x24,0x13,0x6c,0x37,0x2e,0x3f,0x4a,0x5d,0x0f,0x40,0x56,0x25,0x51,
0x1c,0x04,
0x0b,0x46,0x20,0x0d,0x7b,0x35,0x44,0x42,0x2b,0x1e,0x41,0x14,0x4b,0x79,
0x15,0x6f,
0x0e,0x55,0x09,0x36,0x74,0x0c,0x67,0x53,0x28,0x0a,0x7e,0x38,0x02,0x07,
0x60,0x29,
0x19,0x12,0x65,0x2f,0x30,0x39,0x08,0x68,0x5f,0x78,0x2a,0x4c,0x64,0x45,
0x75,0x3d,
```

```
0x59,0x48,0x03,0x57,0x7c,0x4f,0x62,0x3c,0x1d,0x21,0x5e,0x27,0x6a,0x70,
0x4d,0x3a,
0x01,0x6d,0x6e,0x63,0x18,0x77,0x23,0x05,0x26,0x76,0x00,0x31,0x2d,0x7a,
0x7f,0x61,
0x50,0x22,0x11,0x06,0x47,0x16,0x52,0x4e,0x71,0x3e,0x69,0x43,0x34,0x5c,
0x58,0x7d]
```

$S9$:

```
[
0x1c3,0x0cb,0x153,0x19f,0x1e3,0x0e9,0x0fb,0x035,0x181,0x0b9,0x117,0x1eb,
0x133,0x009,0x02d,0x0d3,
0x0c7,0x14a,0x037,0x07e,0x0eb,0x164,0x193,0x1d8,0x0a3,0x11e,0x055,0x02c,
0x01d, 0x1a2, 0x163, 0x118,
0x14b,0x152,0x1d2,0x00f,0x02b,0x030,0x13a,0x0e5,0x111,0x138,0x18e,0x063,
0x0e3,0x0c8,0x1f4,0x01b,
0x001,0x09d,0x0f8,0x1a0,0x16d,0x1f3,0x01c,0x146,0x07d,0x0d1,0x082,0x1ea,
0x183,0x12d,0x0f4,0x19e,
0x1d3,0x0dd,0x1e2,0x128,0x1e0,0x0ec,0x059,0x091,0x011,0x12f,0x026,0x0dc,
0x0b0,0x18c,0x10f,0x1f7,
0x0e7,0x16c,0x0b6,0x0f9,0x0d8,0x151,0x101,0x14c,0x103,0x0b8,0x154,0x12b,
0x1ae,0x017,0x071,0x00c,
0x047,0x058,0x07f,0x1a4,0x134,0x129,0x084,0x15d,0x19d,0x1b2,0x1a3,0x048,
0x07c,0x051,0x1ca,0x023,
0x13d,0x1a7,0x165,0x03b,0x042,0x0da,0x192,0x0ce,0x0c1,0x06b,0x09f,0x1f1,
0x12c,0x184,0x0fa,0x196,
0x1e1,0x169,0x17d,0x031,0x180,0x10a,0x094,0x1da,0x186,0x13e,0x11c,0x060,
0x175,0x1cf,0x067,0x119,
0x065,0x068,0x099,0x150,0x008,0x007,0x17c,0x0b7,0x024,0x019,0x0de,0x127,
0x0db,0x0e4,0x1a9,0x052,
0x109,0x090,0x19c,0x1c1,0x028,0x1b3,0x135,0x16a,0x176,0x0df,0x1e5,0x188,
0x0c5,0x16e,0x1de,0x1b1,
```

```
0x0c3,0x1df,0x036,0x0ee,0x1ee,0x0f0,0x093,0x049,0x09a,0x1b6,0x069,0x081,
0x125,0x00b,0x05e,0x0b4,
0x149,0x1c7,0x174,0x03e,0x13b,0x1b7,0x08e,0x1c6,0x0ae,0x010,0x095,0x1ef,
0x04e,0x0f2,0x1fd,0x085,
0x0fd,0x0f6,0x0a0,0x16f,0x083,0x08a,0x156,0x09b,0x13c,0x107,0x167,0x098,
0x1d0,0x1e9,0x003,0x1fe,
0x0bd,0x122,0x089,0x0d2,0x18f,0x012,0x033,0x06a,0x142,0x0ed,0x170,0x11b,
0x0e2,0x14f,0x158,0x131,
0x147,0x05d,0x113,0x1cd,0x079,0x161,0x1a5,0x179,0x09e,0x1b4,0x0cc,0x022,
0x132,0x01a,0x0e8,0x004,
0x187,0x1ed,0x197,0x039,0x1bf,0x1d7,0x027,0x18b,0x0c6,0x09c,0x0d0,0x14e,
0x06c,0x034,0x1f2,0x06e,
0x0ca,0x025,0x0ba,0x191,0x0fe,0x013,0x106,0x02f,0x1ad,0x172,0x1db,0x0c0,
0x10b,0x1d6,0x0f5,0x1ec,
0x10d,0x076,0x114,0x1ab,0x075,0x10c,0x1e4,0x159,0x054,0x11f,0x04b,0x0c4,
0x1be,0x0f7,0x029,0x0a4,
0x00e,0x1f0,0x077,0x04d,0x17a,0x086,0x08b,0x0b3,0x171,0x0bf,0x10e,0x104,
0x097,0x15b,0x160,0x168,
0x0d7,0x0bb,0x066,0x1ce,0x0fc,0x092,0x1c5,0x06f,0x016,0x04a,0x0a1,0x139,
0x0af,0x0f1,0x190,0x00a,
0x1aa,0x143,0x17b,0x056,0x18d,0x166,0x0d4,0x1fb,0x14d,0x194,0x19a,0x087,
0x1f8,0x123,0x0a7,0x1b8,
0x141,0x03c,0x1f9,0x140,0x02a,0x155,0x11a,0x1a1,0x198,0x0d5,0x126,0x1af,
0x061,0x12e,0x157,0x1dc,
0x072,0x18a,0x0aa,0x096,0x115,0x0ef,0x045,0x07b,0x08d,0x145,0x053,0x05f,
0x178,0x0b2,0x02e,0x020,
0x1d5,0x03f,0x1c9,0x1e7,0x1ac,0x044,0x038,0x014,0x0b1,0x16b,0x0ab,0x0b5,
0x05a,0x182,0x1c8,0x1d4,
0x018,0x177,0x064,0x0cf,0x06d,0x100,0x199,0x130,0x15a,0x005,0x120,0x1bb,
0x1bd,0x0e0,0x04f,0x0d6,
```

```
0x13f,0x1c4,0x12a,0x015,0x006,0x0ff,0x19b,0x0a6,0x043,0x088,0x050,0x15f,
0x1e8,0x121,0x073,0x17e,
0x0bc,0x0c2,0x0c9,0x173,0x189,0x1f5,0x074,0x1cc,0x1e6,0x1a8,0x195,0x01f,
0x041,0x00d,0x1ba,0x032,
0x03d,0x1d1,0x080,0x0a8,0x057,0x1b9,0x162,0x148,0x0d9,0x105,0x062,0x07a,
0x021,0x1ff,0x112,0x108,
0x1c0,0x0a9,0x11d,0x1b0,0x1a6,0x0cd,0x0f3,0x05c,0x102,0x05b,0x1d9,0x144,
0x1f6,0x0ad,0x0a5,0x03a,
0x1cb,0x136,0x17f,0x046,0x0e1,0x01e,0x1dd,0x0e6,0x137,0x1fa,0x185,0x08c,
0x08f,0x040,0x1b5,0x0be,
0x078,0x000,0x0ac,0x110,0x15e,0x124,0x002,0x1bc,0x0a2,0x0ea,0x070,0x1fc,
0x116,0x15c,0x04c,0x1c2]
```

E.2 Key Schedule

MISTY1's key schedule is defined as follows:

One 128-bit key is divided into eight 16-bit values. Let K_i be the i^{th} 16-bit portion. In the following, $i = i - 8$ for $i > 8$. Create eight 16-bit values using the K_i's and the FI function: $K'_i = FI(K_i, K_{i+1})$ with K_{8+1} set to K_1.

$KO_{i1} = K_i$

$KO_{i2} = K_{i+2}$

$KO_{i3} = K_{i+7}$

$KO_{i4} = K_{i+4}$

$KI_{i1} = K'_{i+5}$

$KI_{i2} = K'_{i+1}$

$KI_{i3} = K'_{i+3}$

$KL_{iL} = K_{(i+1)/2}$ when i is odd and $K'_{i/2+2}$ when i is even.

$KL_{iR} = K'_{(i+1)/2+6}$ when i is odd and $K_{i/2+4}$ when i is even.

Appendix F

RC6

F.1 Encryption and Decryption

RC6 [RRSY98] is defined to operate on blocks of size $4w$ where w is the word size, and where the key size and number of rounds are parameters. The following is a description of RC6's encryption function when operating on 128-bit blocks ($w = 32$) and 128-bit keys with 20 rounds. The 128-bit block of data is broken into four 32-bit words. These will be referred to as A,B,C and D. The number of rounds is indicated by r. The array, S, contains the expanded key. Each entry is a 32-bit word. x <<< y means to rotate x to the left by y. The rotation amount is $log_2(w)$, which is 5 for 128-bit blocks. All arithmetic is performed modulo 2^w unless otherwise noted.

When encrypting, the B and D portions have key material added to them. They are then used to create the values t and u, which are XORed with the A and C portions, and as the amount to rotate A and C. The new A and C values then have key material added to them. At the end of the round, the A,C portions are swapped with the B, D portions by rotating the entire block one word. After the last round, the A and C portions have key material added to them. Decryption reverses the operations of the encryption function.

RC6's Encryption Function:

B = B + S[0]

D = D + S[1]

```
for (i=0; i < 20; ++i) {
    t = (B * (2*B+1)) <<< 5
    u = (D * (2*D+1)) <<< 5
    A = ((A XOR t) <<< u) + S[2i]
    C = ((C XOR u) <<< t) + S[2i+1]
    (A,B,C,D) = (B,C,D,A)
}
A = A + S[2r+2]
C = C + S[2r+3]
```

RC6's Decryption Function:

```
A = A - S[2r+3]
C = C - S[2r+2]
for (i=0; i < 20; ++i) {
    (A,B,C,D) = (D,A,B,C)
    t = (B * (2*B+1)) <<< 5
    u = (D * (2*D+1)) <<< 5
    A = ((A - S[2i]) >>> u) XOR t
    C = ((C - S[2i+1]) >>> t) XOR u
}
B = B - S[0]
D = D - S[1]
```

F.2 Key Schedule

When using 32-bit words, a $4c$-byte key is loaded into an array of words, L[0, ... c-1]. The 32-bit round keys are placed in the array S[0, ... 2r+3], where r is the number of rounds and is set to 20 within the scope of this work.

For $w = 32$, P_w = B7E15163 and Q_w = 9E3779B9

S[0] = P_w
for (i=0; i <= 2r+3; ++i) {
    S[i] = S[i-1] + Q_w
}
a = b = i = j = 0
v = 3*max{c,2r+4}
for (s = 0; s <= v; ++s) {
    S[i] = (S[i] + a + b) <<< 3
    a = S[i]
    L[j] = (L[j] + a + b) <<< (a+b)
    b = L[j]
    i = (i+1) mod (2r+4)
    j = (j+1) mod c
}

Note: The P_w and Q_w values are described as "somewhat arbitrary, and other values could be chosen to given 'custom' or proprietary versions of RC6" in the specification of RC6 [RRSY98].

Appendix G

RC4

The following is pseudo code for RC4 [Riv96]. The secret key is contained in the 256-byte array K. S is a 256-byte array initialized with values 0 to 255. len is the number of key stream bytes needed.

```
/* Initialize S */
for (i=0; i < 256; ++i) {
    S[i] = i
}
/* Incorporate K into S */
j = 0
for (i = 0; i < 256; ++i) {
    j = (j + S[i] + K[i]) mod 256
    swap(S[i],S[j])
}

/* Generate the key stream */
i = 0
j = 0
cnt = 0
while (cnt < len) {
```

```
        i = (i + 1) mod 256
        j = (j + S[i]) mod 256
        swap(S[i],S[j])

        /* output a byte to the key stream */
        output(S[(S[i] + S[j]) mod 256])
        cnt = cnt + 1
    }
```