Case studies in symmetric key cryptography

Case Studies in Symmetric Key Cryptography

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr.ir. J.C. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op donderdag 8 september 2005 om 16.00 uur

door

Scott Patrick Contini

geboren te Grosse Pointe, Verenigde Staten

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. A.K. Lenstra
en
prof.dr.ir. H.C.A. van Tilborg

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Contini, Scott

Case Studies in Symmetric Key Cryptography / by Scott Contini
– Eindhoven : Technische Universiteit Eindhoven, 2003.
Proefschrift. – ISBN 90-386-0782-2
NUR 919
Subject headings : cryptology / block cipher / hash function / differential cryptanalysis
2000 Mathematics Subject Classification : 94A60, 94C10, 94A55, 62P99, 11T71

Printed by Eindhoven University Press
Cover by JWL Producties

Preface

This thesis is about two symmetric key cryptographic functions of real-world significance – the RC6 block cipher and the SecurID hash function. RC6 was a widely acclaimed finalist candidate for the Advanced Encryption Standard. It is owned and promoted by RSA Security, and therefore is likely to be used within millions of products worldwide. SecurID is an authentication hardware token also owned by RSA Security. It is known to be used by over 13 million people in over 80 countries throughout the world.

Our work focuses on the security of RC6 and the lack of security of SecurID. The RC6 block cipher was designed by cryptographic experts and supported by extensive public security analysis, part of which is the subject of the second chapter of this work. More specifically, we demonstrate the resistance of RC6 to differential cryptanalysis, which is a major portion of the complete security analysis of RC6. In contrast, the SecurID hash function was designed in secret and kept secret for many years. After the device was reverse engineered and source code was published on the web, security weaknesses soon began to appear. Our third chapter explains our results on practical attacks which can defeat the security of SecurID.

Contents

Chapter 1

Introduction

As we begin the 21^{st} century, we find secure communication a necessary part of our everyday lives. A few examples include ecommerce, stock trading from handheld devices, phone calls or facsimiles involving private or proprietary information, and secure money transfers or withdrawals throughout the world. This has all become possible because of the astonishing evolution of cryptography over the past couple of decades.

Originally cryptography dealt with the science of secret writing, whence its name is derived. It is well known to have been used for thousands of years, but only since the age of the digital computer has it really been embraced by the scientific community. Today, researchers like to describe it as "communication in the presence of an adversary," to reflect the fact that the field now involves much more than just encryption.

The first major step in the development of modern cryptography was Shannon's concept of perfect secrecy based upon an information theoretic model [39]. Unfortunately, this model is not suitable for most real world applications: to attain perfect secrecy one must have a secret encryption key that is as long as the message to be encrypted. Thus, in order to gain any headway for the common use of cryptography, it was necessary to relax the security requirements. Naturally, this led to a complexity theoretic model of security: the use of functions that are easy to compute but hard to invert, unless a secret piece of "trapdoor" information is known (the key). But even if we could construct such functions, we cannot have secure communication unless both parties had a copy of the secret key. This seemed paradoxical at first: if the parties had a way to securely exchange a secret key, then why do they need cryptography in the first place? They should simply exchange the message in the same way that they could exchange the secret key. Fortunately, it is not a paradox. In 1976, Whit Diffie and Martin Hellman provided a mathematical solution in their paper "New Directions in Cryptography" [13], which is widely accepted as the most important paper in the history of the field. They also introduced the concept of public key cryptography: a system where every user would have a public key for encryption and a private key for decryption. Each user's public key

would be made known to everybody, so anybody could use it for encrypting a secret message to that person. Yet only he who owns the corresponding private key would be able to decrypt it. For this to work, it must be the case that nobody can derive the private key from the public, except the person who created the key pair in the first place. Furthermore, by reversing the roles of the public key and the private, a user could create a "digital signature" on a digital document. Digital signatures essentially provide a mechanism for digital contracts.

Shortly after Diffie and Hellman's publication, Ronald Rivest, Adi Shamir, and Len Adleman introduced the RSA public key cryptosystem [36]. RSA is an elegant solution based upon the difficulty of factoring large integers. Their cryptosystem rightfully received an enormous amount of attention, having a huge impact on the field. For instance, Rivest started a company known as RSA Data Security which was very successful. RSA Data Security was able to convince numerous corporations of the value of their ideas, especially the potential for expanding their businesses on the world wide web. Hence, the RSA brand name became widely known, even to people without technical expertise. At the same time, the RSA cryptosystem attracted a lot of attention from top mathematicians, especially number theorists. They found that some rather advanced mathematical ideas had real-world value, and therefore they were able to get increased funding for doing mathematical research that had cryptographic implications.

As RSA became more and more widely used, many new cryptographic research concepts arose. For example, even though RSA is based upon the difficulty of factoring, it is not provably as strong. In fact, the ordinary textbook description of RSA is breakable in several attack scenarios, such as existential forgeries for digital signatures as well as many other little tricks. To prevent such attacks, "padding" strategies were developed. Originally the padding methods were heuristic methods of defeating known attacks, but later provable security reductions came about. At the same time, other public key cryptosystems with provable security properties were being invented.

Today, public key cryptography with provable properties is widely used. However, public key cryptography by itself is not the solution to all secure communication needs. The main problem is that it tends to be very slow, so one does not want to encrypt (or sign) an enormous amount of data using it alone. Instead, public key methods are generally used in combination with traditional, symmetric key methods since they are much faster. A common scenario is that the public key method will be used to secretly exchange a random value, and the random value will be used as the secret key for a symmetric encryption method.

The style of research in symmetric key cryptography has lagged behind that of public key cryptography, the reason being the performance requirements. Nobody has found an *efficient* way of doing symmetric key cryptography (in the complexity theoretic model) which is provably as secure as some assumed hard mathematical problem. For this reason, the best we can do is design heuristic methods that are secure against all types of attacks that have been discovered in the history of the subject. The rule of thumb is that a symmetric key method can begin to be trusted

after a lot of experts have studied it and gained confidence in it. For now, this seems to be working, and we can also say that the converse is true: systems which have not been widely studied by the experts are almost always insecure. But, it should be evident that the discovery of an efficient, provably secure symmetric key method will result in a great breakthrough in modern cryptography. We hope to see such a discovery in the near future.

1.1 Our Research

Two of the most important primitives in symmetric key cryptography are block ciphers and hash functions. Informally, block ciphers break messages into fixed length "blocks" and encrypt them one at a time in a way that depends upon a secret key. Their primary purpose is to provide secrecy of data, assuming that the secret key is available only to the intended parties. Hash functions are functions that map an arbitrary length input to a fixed length output. Hash functions may be used for many purposes, but here we shall be concerned with only keyed hash functions for the application of authentication. This thesis is about the security analysis of the block cipher RC6 [34] and the keyed hash function used in the SecurID authentication device. Both of these are of real-world significance. RC6 was a widely acclaimed finalist candidate for the Advanced Encryption Standard (AES) and SecurID is the most popular authentication device of its type in the world – used by over 13 million people.

In our research, we wear two different faces. First, in the security analysis of RC6, we worked with the designers of the ciphers. Our goal is therefore to give arguments justifying the security. In the analysis of SecurID, we take the role of the attacker who is trying to demonstrate security weaknesses. The two different scenarios lead to two different styles of analysis. For one who wants to promote a security algorithm, he must give very convincing arguments that even the most powerful adversaries should not be able to break it. In this situation, one must consider the most general attacks possible, since the cryptanalyst will always seek a new approach that the designer had failed to consider. On the other hand, the publisher of a cryptanalysis weakness has a narrower task. He only needs to show *one* specific weakness that the designer had failed to recognize. Often times the weakness is very small and may have no real-world implication. Regardless, such a result is welcomed and celebrated in the cryptographic community, since the nature of the science demands skepticism.

In the following two subsections, we give brief, formal treatment of block ciphers and keyed hash functions. We also talk at a very high-level about our results on RC6 and SecurID.

1.2 Block Ciphers and RC6

Formally, a block cipher is a family of bijective functions that map from a fixed size message space (plaintext space) M to a ciphertext space C of the same size. The functions are parameterized by a key from a fixed size key space K. Without loss of generality, we will assume the sizes are measured in bits: an $n-$bit message space and an $h-$bit key space. In this case, we say the cipher has a *block size* of n. Given a key $k \in K$ and an $n-$bit message $m \in M$, the encryption is denoted $c = E_k(m)$. Similarly, the decryption is denoted $m = D_k(c)$, where $D_k = E_k^{-1}$.

The adversary (attacker) has the goal of trying to decipher the ciphertext without legitimately being given the cryptographic key k. Ideally, he would like to determine k so he can read any ciphertext that is sent. On the other hand, it is often sufficient for the attacker to attain less lofty goals, such as finding a relationship between the plaintext and the ciphertext. For example, if the attacker knows beforehand that the plaintext is one of a small set of values, like "yes" or "no", then finding such a relation may allow him to determine which value was sent without knowing the secret key. Therefore, a good cipher should leak no information about the message.

There are a few models in which the attacker may operate. We assume that all plaintext-ciphertext pairs are derived from a fixed key.

- A *ciphertext only attack* is when the attacker tries to decrypt a set of ciphertexts without knowing any of the corresponding plaintexts beforehand.
- A *known plaintext attack* is when the attacker knows some of the plaintexts corresponding to the set of ciphertexts, and tries to use this extra information to decrypt the remaining ciphertext.
- A *chosen plaintext attack* is when the attacker is able to find the encryptions of plaintexts of his choice in order to aid him in decrypting a set of ciphertexts that he is given beforehand.
- A *chosen ciphertext attack* is when the attacker is able to decrypt an arbitrary set of ciphertexts of his choice, and will later use this information to decrypt other ciphertexts.

Chosen plaintext and chosen ciphertext attacks are sometimes grouped together as *chosen text attacks*. They are the most powerful attacks an adversary can mount.

The strength of a cipher against chosen text attacks is in fact very important. There are at least two reasons. The first is these type of attacks can usually be converted into other types of attacks under weaker models such as known text attacks [4]. The second is that in some cases, these type of attacks are indeed realistic. For example, consider cable set-top box security, where every user has a set-top box with a secret key embedded inside. If the user pays their bills, their set-top box is enabled to decrypt the programs that are constantly sent over the network. A dishonest user could disconnect his box from the network and then send in his own pseudo ciphertexts into the box, thus getting the corresponding plaintexts.

So he is performing a real chosen ciphertext attack. If this process allows him to derive the key, he could manufacture pirate hardware decoders which bypass the security mechanisms installed by the manufacturer. Anybody who purchases the pirate decoder from the dishonest user would then be able to watch cable programs for free indefinitely.

Note that with the various models, there are several ways we can measure the complexity of an attack: in terms of time, memory, and known plaintext or chosen text requirements. We generally consider the most dominant of these values when assessing its feasibility.

For a cipher with an $h-$bit key space, key recovery can be done by exhaustive search which takes 2^h encryption/decryption operations in the worst case, or 2^{h-1} on average. If an attack exists which is faster than that while not requiring excessive other resources, then the cipher is considered to be broken at least in a theoretical sense. In some situations, it may be easier to attack the cipher when certain keys are used but not others. If a portion of w keys have the property that the corresponding encryptions can be attacked more than w times faster than exhaustive search, then we say the cipher has *weak keys*. Weak keys are also considered a theoretical break of the cipher.

For a cipher with $n-$bit block length, one can always find the encryption of all 2^n possible input blocks and store the result. Then, all future encryptions can be found by simply doing a table lookup. Thus, this type of attack does not require knowing the secret key. Its storage requirements are 2^n plaintext-ciphertext pairs. If an attack is found which requires less storage (while not requiring excessive other resources), then again the cipher is considered broken.

Chosen text attacks tend to require all three types of resources: storage, computation time, and of course chosen texts. Often, the number of chosen texts is the dominant resource. We will see examples of this in our research.

The complete security analysis of RC6 considered many different types of attacks in different attack models. The research reported in this thesis only includes the information which the author of this document actively participated in: the resistance to differential cryptanalysis. Differential cryptanalysis is the most powerful chosen text attack known to date. A brief overview of differential cryptanalysis is given in Section 2.1.1. Our security analysis is given in Chapter 2. The very detailed analysis ultimately argues that differential attacks of up to 16 (out of 20) rounds are far out of reach due to the number of chosen text requirements. There have been no improvements upon our differential analysis to date.

1.3 Keyed Hash Functions and SecurID

Hash functions map an input of arbitrary length to a fixed length output. They may or may not involve a cryptographic key. In our case, we are interested in keyed hash functions, or more specifically *message authentication codes* (MACs). The primary purpose of a MAC is to authenticate the origin and the integrity of

a message. Formally, a MAC is defined as a family of functions h_k parameterized by a key k that map an input x to a fixed size output. The output is known as the MAC-value. The main security property it should provide is, for any set of inputs and their corresponding MAC-values $(x_1, h_k(x_1)), \ldots, (x_\ell, h_k(x_\ell))$, it should be computationally infeasible to find $(\tilde{x}, h_k(\tilde{x}))$ for any $\tilde{x} \neq x_i \, (1 \leq i \leq \ell)$ assuming the key k is not known beforehand [28]. Similar to block ciphers, we may consider known text or chosen text attack models (see Section 1.2). Moreover, we may also measure the attacks complexity in terms of time, memory, and known plaintext or chosen text requirements.

The SecurID authentication device[1] actually uses a MAC that maps a 64-bit input to a pair of 6- or 8-digit numbers via a 64-bit key. Technically, this falls a bit short of the definition of MAC or hash function, since the input size is fixed. However, it is trivial to extend any such function to a proper definition by using the Merkle-Damgård construction. The construction involves iterating a *compression function* enough times to cover the length of the message, as well as including message padding. The SecurID function (to be described in Section 3.1) would serve as the compression function. For exact details on the construction, which are not relevant to our present research, we refer the reader to [28]. Despite the fact that the SecurID function falls a bit short of the proper definitions, it is commonly referred to as the *SecurID hash function*. We shall also use this misnomer hereafter.

The SecurID hash function was developed within a corporation known as Security Dynamics. For many years, the algorithm had been kept secret. Security Dynamics insisted that the security of the algorithm did not depend upon its secrecy, and that the algorithm had been analyzed extensively by customers, including many cryptographic experts [26]. They justified keeping their algorithm secret by insisting that their early customers would only buy it under this condition. Towards the end of 2001, the SecurID was reverse-engineered and a software implementation was posted on the web [40]. Once the cryptographic community studied it, security weaknesses were readily published [5, 7, 11].

The known weaknesses exploit the fact that collisions happen far too often. Here, a collision means two inputs x_1 and x_2 such that $h_k(x_1) = h_k(x_2)$ where h_k is the SecurID hash function using fixed key k. Since the block size is only 64-bits, one would expect collisions to happen after about 2^{32} operations according to the birthday paradox. But the SecurID has collisions in an expected $\approx 2^{19}$ operations [5].

Unlike our analysis of RC6 where we try to show resistance against the most powerful types of attacks, our analysis of SecurID will be restricted to attacks on the way the device is used in practice. Thus, we acknowledge that the function was not intended to be a general MAC which can be used for numerous applications. Instead, the MAC is used to hash only consecutive time values (after the time is expanded to a 64-bit value in a way described in Section 3.1.1), and only at a rate of about one per minute. So, for this application chosen text attacks do not seem

[1] A description of the hardware device and how it is used in practice is given in Chapter 3.

to be possible[2].

We note that SecurID devices are only intended to be used for a few years, partly due to limited battery lifetime. However, at the pace of one hash per minute, collisions are expected to happen in about one year. A decent portion of devices will have collisions much sooner.

After a collision happens, the device is vulnerable to an off-line key search. For a device with a 64-bit secret key, finding the key can be done in 2^{64} hash (MAC) operations, which is impractical today except for an extremely powerful adversary. However, using knowledge of the collision, the attack can be sped up to about 2^{45} operations in most cases by the techniques of [5] and our improvements. Moreover, using additional information that an attacker would likely have, the search can be sped up even more. We demonstrate how this can be done, and estimate that an attacker would probably be able to find the key in about 2^{40} operations. This makes the attack quite practical to a typical adversary.

Only recently has the company, now known as RSA Security[3] acknowledged the problems [27]. In fact, they claim that they became aware of some of the problems a few years earlier and began to phase out the devices with the old hash algorithm. The newer devices use a function based upon the AES, which has been widely analyzed by expert cryptographers.

Our cryptanalysis results are given in Chapter 3.

[2]The only way a chosen text attack would be possible is to open it and replace the clock with a circuit that feeds in inputs of the attacker's choice. However, the devices are tamper-resistant: if one attempts to open it, it will zero-out the secret key and hence will no longer be able to perform its function of authentication.

[3]Security Dynamics bought out the company RSA Data Security 1996, and later changed their name to RSA Security.

Chapter 2

Differential Cryptanalysis Applied to RC6

In this chapter, we describe the security analysis of the block cipher RC6 [34] with respect to differential cryptanalysis. The work summarized here is part of a more comprehensive analysis, published in [8, 9, 10]. The topics which are not included from the referenced publications are those which the author of this document had little to no involvement with during the research phase. Of those topics, the resistance of RC6 to linear cryptanalysis is the most significant and substantial.

There is a long history that motivates the RC6 block cipher and its analysis, so we start from the beginning. In 1976, the National Institute of Standards and Technology (NIST) made a call to the public for an encryption algorithm to protect sensitive but unclassified electronic data. A submission from IBM was adopted by NIST, which became known as the *Data Encryption Standard* (DES) [31]. DES is a Feistel cipher with 64-bit block size and a 56-bit key. From the beginning, the key size of DES was suspected of being small enough so that an organization with enough funding and resources could exhaustively search the key space and break it [14]. As time progressed, it became increasingly apparent that such an attack was indeed realistic, and the effect of Moore's law was reducing the funding and resource requirements necessary to carry it out.

In 1993, NIST added a statement to the DES specification foreshadowing the possibility of a new algorithm within the not too distant future. Consequently, many people began designing new block ciphers to accomplish this purpose. Among the designs that appeared was Ron Rivest's simple and elegant RC5 [33] block cipher, which was put into products by RSA Data Security, and before long it became widely used. It also became the subject of much security analysis.

In January of 1997, NIST made the official announcement for the development of a new encryption algorithm, to be called the *Advanced Encryption Standard* (AES), for the protection of sensitive government information well into the 21st century [32]. As in the case of DES, the public was invited to submit designs for that cipher. The announcement came approximately one year and a half before the insecurity of DES

was proven beyond all doubt: A team of privacy advocates and cryptographers spent approximately US$250,000 to build a real DES cracker machine [15]. The machine was used to solve an RSA secret key DES challenge, which it succeeded in doing in 56 hours.

The RC5 cipher was not ideally suited for submission to NIST as a possible AES candidate. It did not perfectly match NIST's requirements, and there was room for improvement of the security based upon what had been learned from public analysis. Hence, the cipher RC6 was born, which became the submission of RSA Laboratories. A total of twenty one algorithms were submitted, but six were discarded immediately because they were incomplete. The fifteen complete submissions were eventually narrowed down to five finalists which were: MARS, RC6, Rijndael, Serpent, and Twofish. After more analysis and much public feedback, Rijndael emerged as the winner.

All of the finalists and some of the other candidates that did not make the list of finalists are widely believed to provide adequate security. What made the difference was other factors such as speed, hardware requirements, adequacy for very constrained environments, simplicity of design, etc.... RC6 was criticized for its adequacy in extremely constrained environments and its gate count in hardware. Whether these criticisms are justified is a subject of debate [35] which we do not intend to argue here. But, we will mention that it is commonly agreed that RC6 had the best performance on the target platform (Pentium Pro) and many other modern platforms, that it had the most in-depth and accurate security analysis[1], and that it had the most elegant and simple design.

Since the design and analysis of RC6 was based upon what was learned from RC5, it is only appropriate to first talk about RC5. In doing so, we also summarize the strategy of differential cryptanalysis.

2.1 RC5 and Differential Cryptanalysis

RC5 is a parameterized algorithm allowing variable word size w, a variable number of rounds r, and a variable number of bytes b for the secret key. It encrypts $2w-$bit plaintexts into $2w-$bit ciphertexts. A specific choice of parameters for RC5 is designated by RC5-$w/r/b$. The permissible values for w, r, and b are as follows:

w	The word size may be 16, 32, or 64-bits.
r	The number of rounds may be any value between 0 and 255 inclusive.
b	The number of bytes in the secret key may be any value between 0 and 255 inclusive.

Obviously, some of choices for parameters do not provide much security, such as a small number of rounds or a small key size. However, the flexibility in choices facilitates security analysis, especially in terms of being able to experiment with

[1] At the time of writing, it is the only AES finalist for which nobody improved upon the analysis given by the original authors.

Encryption with RC5-$w/r/b$

Input:	Plaintext stored in two $w-$bit input registers A, B. Number of rounds r. Subkeys $S[0], \ldots, S[2r+1]$, each $w-$bits.
Output:	Ciphertext stored in A, B.
Procedure:	$A = A + S[0]$ $B = B + S[1]$ **for** $i = 1$ **to** r **do** $A = ((A \oplus B) \lll B) + S[2i]$ $B = ((B \oplus A) \lll A) + S[2i+1]$

Table 2.1: RC5 Encryption

smaller versions of the design. For real world applications, a proposed choice of paramaters is $w = 32$, $r = 12$, and $b = 16$ [33].

The first step in RC5 is a one-time key expansion algorithm that takes the user entered $b-$byte key and expands it into a sequence of $2(r+1)$ subkeys of w bits each, denoted $S[i]$ for $0 \leq i < 2(r+1)$. The key expansion algorithm is given in Appendix 3.12. The exact details are not so important for our analysis, especially since no weaknesses have ever been found with this algorithm.

The RC5 encryption function is shown in Table 2.1. It uses three operations:

$a + b$	Integer addition modulo 2^w.
$a \oplus b$	Bitwise exclusive-or.
$a \lll b$	Left rotation of the $w-$bit word a by the amount given in the least significant $\lg(w)$ bits of b.

The notation 'lg' means $\log_2$. The decryption algorithm is easily derived from the encryption. It uses integer subtraction in place of integer addition, and right rotations in place of left rotations.

The use of the data-dependent rotates is intended to provide the most critical security component of RC5. As we shall see, differential cryptanalysis attempts have focused on maintaining invariant rotates for a pair of inputs that pass through the function. We shall give some analytic justification for this in Section 2.3.

Note that the second line in the inner loop of RC5 is the same as the first, except with A and B swapped. Thus, an alternate description of RC5 uses $2r$ rounds, where the inner loops consists of $A = ((A \oplus B) \lll B) + S[i]$ followed by a swap of A and B. According to this description, the inner loop is called a *half-round*. A diagram of RC5 in this form is shown in Figure 2.1.

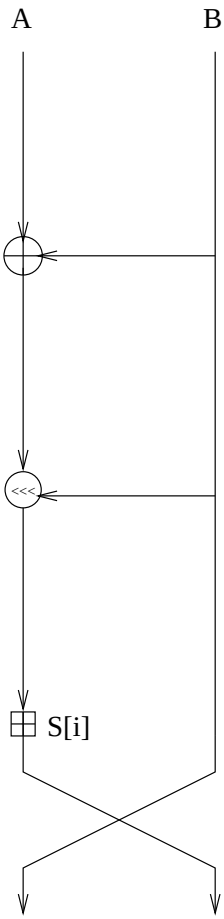

Figure 2.1: Diagram of an RC5 half-round.

2.1.1 Introduction to Differential Cryptanalysis

Differential Cryptanalysis is a chosen text attack that was first published in 1991 by Biham and Shamir [2]. There is considerable evidence that the designers of DES knew of it beforehand, but were not able to reveal information about it, as requested by the NSA (see [12] for evidence). For example, the choices of S-boxes in DES have made it immune to being practically threatened by the attack, whereas many other block ciphers were not so fortunate.

Differential cryptanalysis has probably been the single most influential discovery that has effected the design of modern block ciphers. The one other potential rival discovery is linear cryptanalysis [25], which has been more effective against DES, but less effective against most other block ciphers. No new block cipher proposal is taken seriously until considerable evidence has been given that it is resistant to both types of attacks.

Let P_1 and P_2 be two closely related plaintexts, and define their difference P' with respect to some cipher operation. Usually the difference is defined with respect to exclusive-or, i.e. $P' = P_1 \oplus P_2$, but other differences should be considered as well. The strategy of differential cryptanalysis is based upon the evolution of differences as the two plaintexts pass through the cipher. If certain differences are more likely to happen than is expected from a random permutation, then a key recovery algorithm may be possible.

Let X_1 and X_2 denote the two texts at the beginning of any round of the cipher, and X' their difference. Accordingly, we denote Y_1, Y_2, and Y' the outputs of the round. Depending upon the round function, certain difference pairs (X', Y') may be more likely than others, when evaluated over all possible X_1, Y_1, X_2, Y_2, and subkeys. For each such pair, we associate a probability which can be computed or approximated by analytical or numerical means. By stringing these difference pairs together from round to round, we get a *characteristic* that has an associated probability p. Each characteristic requires that the output difference from one round is the input difference to the next. Assuming the probabities across rounds can be modelled as independent events, the probability p is expected to be the product of the difference probabilities from round to round. In general, this assumption may not be true, but if the cipher is well designed, we expect that it accurately approximates the real probability. As we will later see (Section 2.4.6), it is very important to verify these probabilities via computer experiments.

Consider a characteristic that is defined from the input of the cipher to the penultimate round of a block cipher. We will refer to the output difference of the penultimate round as the characteristic's *final difference*. If the probability p of the characteristic is sufficiently large, then we may be able to recover the final round subkey using the algorithm sketched below. By iteratively applying the same algorithm, we should be able to recover all previous subkeys as well, which is effectively as good as knowing the original user chosen key. The subkey recovery algorithm works as follows:

1. Encrypt a large number of plaintext pairs having the same input difference of

the characteristic.

2. For each resulting ciphertext pair, determine the final round subkey(s) (if any) that will transform the characteristic's final difference into the ciphertext difference that was observed. Keep a counter for each subkey possibility that occurs.

3. After enough inputs, the correct subkey is expected to have a significantly higher count than all others, and therefore the attacker can identify it. In some cases, the attacker may only be able to narrow down the possibilities for the actual subkey to a small group of candidates.

A pair of plaintexts and the corresponding final differences that match the characteristic exactly is called a *right pair*. Each right pair increases the count of the correct subkey by 1. On the other hand, the procedure often increments counters of wrong candidate subkeys. The wrong candidates are expected to be approximately uniformly distributed, whereas the right candidate should have a much higher count, assuming the characteristic has relatively high probability. So, the occurrence of right pairs is used to hopefully distinguish the correct subkey from wrong guesses.

The number of plaintext pairs needed before one can accurately identify the right subkey is typically a small multiple of $1/p$. The exact value depends upon the *signal to noise ratio*, which is the ratio of the number of right pairs to the average count. If the signal to noise ratio is very high, then the procedure will be complete after only a few right pairs. If it is low, then the number of right pairs required will be much higher.

In general, there are typically many characteristics having the same input and final differences. The specific path of how a characteristic gets from its input difference to its final difference is of lesser importance, so we use the broader concept of a *differential* which does not specify the intermediate rounds [20]. The probability of a differential is the sum of the probabilities of all characteristics having the same input and final differences as the differential.

For more information on differential cryptanalysis, see [2] for an excellent very detailed description or [20] for a short technical description.

2.1.2 Differential Cryptanalysis Attacks on RC5

The state of the art of attacks on RC5 is beautifully summarized in [18]. The most threatening attacks against RC5 are differential attacks. Below, we touch upon the details that will be the most relevant for our analysis of RC6.

All differential attacks published so far against RC5 have been based on the Kaliski-Yin approach from [17]. The attacks use exclusive-or as the difference operation. Consider the pairs $X_1 = (A_1, B_1)$ and $X_2 = (A_2, B_2)$ with difference $\Delta = (A', B')$. The first step in the RC5 half-round function is to exclusive-or B_1 onto A_1. Thus, the difference of $A_1 := A_1 \oplus B_1$ with $A_2 := A_2 \oplus B_2$ is $(A_1 \oplus B_1) \oplus (A_2 \oplus B_2) = A' \oplus B'$, which holds with probability 1.

The next operation is a left rotate of A_1 by the least significant $\lg w$ bits of B_1. With probability $1/w$ the least significant $\lg w$ bits of B_1 will be the same as the least significant bits of B_2. In this case, the difference between $A_1 := A_1 \lll B_1$ and $A_2 := A_2 \lll B_2$ is $A' := (A_1 \lll B_1) \oplus (A_2 \lll B_1) = (A_1 \oplus A_2) \lll B_1 = A' \lll B_1$. On the other hand, if the least significant $\lg w$ bits of B_1 are not the same as those from B_2, then the new difference will be $(A_1 \lll B_1) \oplus (A_2 \lll B_2)$. The difference in the rotate amounts causes different bits of A_1 to line up with different bits of A_2, making the outcome somewhat unpredictable. Let us look at an example of this phenomenom where A_1 and A_2 have a 1-bit difference ($w = 32$):

$$\begin{aligned} A_1 &= \texttt{10100011010111111101111011000001} \\ A_2 &= \texttt{10100011010111111111111011000001} \\ A' &= \texttt{00000000000000000010000000000000} \end{aligned}$$

If we left rotate both words by 6, we get:

$$\begin{aligned} A_1 \lll 6 &= \texttt{11010111111101111011000001101000} \\ A_2 \lll 6 &= \texttt{11010111111111111011000001101000} \\ (A_1 \lll 6) \oplus (A_2 \lll 6) &= \texttt{00000000000010000000000000000000} \end{aligned}$$

Clearly, the difference has been rotated left by 6. Now we will put a difference in the least significant bit of the rotates: we left rotate A_1 by 6 but A_2 by 7.

$$\begin{aligned} A_1 \lll 6 &= \texttt{11010111111101111011000001101000} \\ A_2 \lll 7 &= \texttt{10101111111111111011000001101001} \\ (A_1 \lll 6) \oplus (A_2 \lll 7) &= \texttt{01111000000010001101000010111001} \end{aligned}$$

So when there is a difference in the rotate amounts, there seems to be no simple way of classifying the output difference in terms of the input difference without taking into consideration the actual values of the inputs A_1 and A_2. Intuitively, one would expect the output differences to appear random. For this reason all existing differential attacks have been based upon the assumption that no differences occur in the rotation amounts, i.e. $B' \bmod w = 0$. We shall give some analytical justification for this in Section 2.3.

The final operation in the half-round function is adding on subkey $S[i]$. So, we are interested in the difference $(A_1 + S[i]) \oplus (A_2 + S[i])$. For now, let us only consider simple cases where the inputs differ in one bit. Specifically, we introduce the notation $A' = e_s = 2^s$ for some $0 \leq s < w$ which will be used throughout the chapter. The following lemma will be used many times:

Lemma 2.1 *Let $A_1 \oplus A_2 = e_s$. Then the probability that $(A_1 + S[i]) \oplus (A_2 + S[i]) = e_s$ when averaging over all w-bit word pairs $A_1, S[i]$ is $\frac{1}{2}$ when $s < w - 1$ and 1 when $s = w - 1$.*

Proof: The case of $s = w - 1$ is trivial, since there can be no carry propagating forward from bit $w - 1$. We proceed assuming $s < w - 1$.

We first note that the sum of the bits prior to s (i.e. lesser significant bits) is the same for both $(A_1 + S[i])$ and $(A_2 + S[i])$ and the sum of the bits after it will be the same if and only if the carry coming out from bit $\#s$ is the same. The carry out will be the same if and only if the carry in is equal to bit $\#s$ of $S[i]$. Thus, if the bit $\#s$ of $S[i]$ is 0 and the carry in to the bit $\#s$ is 0 or if the bit $\#s$ of $S[i]$ is 1 and the carry in to the bit $\#s$ is 1, then we are guaranteed to have $(A_1 + S[i]) \oplus (A_2 + S[i]) = e_s$.

We proceed by counting. We will use the notation $x \bmod n$ to mean the least non-negative representative of the congruence class. First suppose bit $\#s$ of S[i] is 1. Let j be $S[i] \bmod 2^s$ (i.e. the least significant s bits of $S[i]$). Then the carry in to bit s is 1 if and only if $A_1 \bmod 2^s = A_2 \bmod 2^s$ is any of the values $2^s - j, 2^s - j + 1, \ldots, 2^s - 1$. There are j such values. So the total number of values where bit $\#s$ of $S[i]$ is 1 and the carry in to bit $\#s$ is 1 is

$$\sum_{j=0}^{2^s-1} j = \frac{(2^s - 1)2^s}{2} \quad .$$

Similarly, the total number of values where bit $\#s$ of $S[i]$ is 0 and the carry in to bit $\#s$ is 0 is

$$\sum_{j=0}^{2^s-1} 2^s - j = 2^{2s} - \frac{(2^s - 1)2^s}{2} \quad .$$

Thus, in total there are $\frac{(2^s-1)2^s}{2} + 2^{2s} - \frac{(2^s-1)2^s}{2} = 2^{2s}$ combinations that guarantee an output difference of e_s. On the other hand, there are a total of 2^{2s+1} combinations for the least significant $s + 1$ bits of $S[i]$ and the least significant s bits of A_1 (= least significant bits of A_2). So, when $s < w - 1$, we have shown that the probability is $\frac{2^{2s}}{2^{2s+1}} = \frac{1}{2}$. □

The reader may become suspicious at this point, and rightfully so. In Lemma 2.1, we averaged over all subkeys to compute a probability, when in reality the subkey is a single, fixed value. The actual probability differs according to the values of the least significant $s + 1$ bits of $S[i]$ (it is $j/2^s$ when bit $\#s$ is 1 and $1 - j/2^s$ when bit $\#s$ is 0). So why are we averaging over all possibilities when the subkey is fixed? We alluded to the answer of this question in Section 2.1.1. When we string differences and probabilities together to form characteristics, we will be representing many probabilities by the average case. The real probability for a specific characteristic may be more or less than the average case, but we expect that when we sum the average case probabilities for numerous characteristics, we get a good *approximation* to the real differential probability. Such assumptions need to be tested experimentally, which they have been with RC5, and we shall later see a case of a simplified variant of RC6 where a seemingly reasonable assumption turned out to be false. Note that although this type of assumption/approximation is typical in differential cryptanalysis, it does leave open the possibility of *weak keys*. For the

time being, we shall ignore this issue, but we will return to the subject when we do our full analysis of RC6 in Section 2.4.5.

Let us now consider some half-round characteristics from [17, 18]. Let $\Delta = (0, e_s)$ where $s \geq \lg(w)$. After B_1 is exclusive-or'ed onto A_1, the difference is $\Delta = (e_s, e_s)$. The next step is a left rotate, where the rotates are the same since $s \geq \lg w$. With probability $1/w$, that rotate amount will be 0, hence keeping Δ the same. Finally, the subkey addition keeps Δ the same with probability at least $\frac{1}{2}$. Thus, the difference $X' = (0, e_s)$ with $s \geq \lg w$ becomes $Y' = (e_s, e_s)$ with probability $\geq \frac{1}{w} \cdot \frac{1}{2}$. This characteristic is denoted Ω^1.

Now take $X' = (e_s, e_s)$ again with $s \geq \lg(w)$. It is easily seen that $Y' = (e_s, 0)$ holds with probability 1. This characteristic is denoted Ω^2. Finally, consider $X' = (e_s, 0)$ with $s \geq \lg(w)$, and let t be any integer satisfying $\lg w \leq t < w$. Then X' will become $Y' = (0, e_t)$ with probability $\geq \frac{1}{w} \cdot \frac{1}{2}$. This characteristic is denoted Ω^3.

Define $\overline{\Omega}$ as the stringing together of Ω^1 with Ω^2 and then Ω^3. This is a characteristic that holds with probability $\geq \frac{1}{4} \cdot \frac{1}{w^2}$. We can string as many $\overline{\Omega}$'s together with each other as long as the output t from one is equal to the input s from the next. Such a characteristic is called an *iterative characteristic*.

More generally, we can consider differentials of this form. There are a total of $w - \lg w$ allowable values of the output t. So, iterating the $\overline{\Omega}$ characteristic $m-1$ times and allowing all legal values of t, we get a probability of at least $(\frac{w-\lg w}{(2w)^2})^{m-1}$. The research of [17] then shows how an additional 3 more half-rounds can be obtained for roughly the cost of 1. First, they start out with the difference $(0, e_{w-1})$ which will go through the first half-round with probability 1. Then, they choose the last two half-rounds to use larger Hamming weight differences in a way that assures that they have an efficient algorithm to derive the last subkey. The probability of their differential for $3m$ half-rounds is $\geq \frac{w-\lg w-1}{w} \cdot (\frac{w-\lg w}{(2w)^2})^{m-1}$. Similar characteristics exist for $3m+1$ and $3m+2$ half-rounds [17].

In general, the Kaliski-Yin approach uses iterative characteristics for about $r-2$ half-rounds when an r half-round characteristic is needed, and then finds a way to get 2 additional half-rounds almost for free. When we later get around to analyzing RC6 and its simplified variants, we shall make the assumption that an iterative characteristic can always be extended an additional 2 rounds for free. Note that these types of assumptions are safe to make when trying to establish security defenses since, if they turn out to be false, then it implies that the cipher is stronger than the analysis suggests.

See [17, 18] for efficient algorithms for deriving the subkey once enough plaintext pairs are sent in, and for more specific details on the attack. For the purpose of the research to be presented, these details do not need to be replicated here, since we intend to argue that high probability differentials do not exist for the full RC6. We remark that Kaliski and Yin have also implemented the attack on reduced round variants of RC5, and found the results to match the theoretical analysis reasonably well.

There have been two major improvements to the original attack presented by Kaliski and Yin. The first, due to Knudsen and Meier [19], shows how the attacks

can be improved by identifying plaintexts that cause no rotates in the first few half-rounds. The second, due to Biryukov and Kushilevitz [3], allows differences to have greater than Hamming weight 1 throughout the cipher, so long as the difference follows some "controlled" pattern. The Biryukov and Kushilevitz result is the most general and powerful attack known against RC5 at the time of writing. We shall talk a bit more about both attacks after we describe our best attacks against RC6.

2.2 RC6

RC6 was first presented in [34]. Like RC5, RC6 is fully specified by a word size w, a number of rounds r, and the number of bytes b in the key. The allowable values for the parameters are the same as for RC5 (see Section 2.1.2), and a specific choice of parameters for RC6 is denoted by RC6-$w/r/b$. For the AES candidate submission to NIST, the parameters $w = 32$, $r = 20$, and $b = 16, 24$, and 32 were the proposed values. The shorthand notation RC6 is used to mean $w = 32$ and $r = 20$.

The first step in RC6 is a one-time key expansion algorithm that takes the user entered $b-$byte key and expands it into a sequence of $2r+4$ subkeys of $w-$bits each, denoted $S[i]$ for $0 \leq i < 2r+3$. The key expansion algorithm is exactly the same as in RC5, and is given in Appendix 3.12. As before, the details are not so important for our analysis.

The RC6 encryption function is shown in Table 2.2. Note that it uses four w-bit words, thus encrypting a $4w$-bit plaintext into a $4w-$bit ciphertext. The notation $(A, B, C, D) = (B, C, D, A)$ means a parallel copying of B into A, C into B, etc.... See Figure 2.2 for a diagram of the round function. The following operations are used in encryption and/or decryption:

$X + Y$	Integer addition modulo 2^w.
$X - Y$	Integer subtraction modulo 2^w.
$X \oplus Y$	Bitwise exclusive-or.
$X \times Y$	Integer multiplication modulo 2^w.
$X \lll Y$	Left rotation of the $w-$bit word X by the amount given in the least significant $\lg(w)$ bits of Y.
$X \ggg Y$	Right rotation of the $w-$bit word X by the amount given in the least significant $\lg(w)$ bits of Y.

RC6 bears several similarities to RC5. The register A in a round of RC6 follows the same path as the register A in a half-round of RC5. However, the registers that affect it are different. In particular, the exclusive-or comes from $f(B) \lll (\lg w)$ where $f(X) = X(2X + 1)$, and the data-dependent rotate is determined by the amount of $(f(D) \lll (\lg w)) \bmod w$. We shall refer to the function f as the *quadratic function* and the left rotate by 5 as the *fixed rotate*. Both the quadratic function and the fixed rotate are used to make RC6 more resistant to differential cryptanalysis than RC5. In RC5, the data-dependent rotation amount is determined by only the 5 least significant bits of B: none of the other bits have any effect. In contrast, the combination of the quadratic function and the fixed rotate make the data-dependent

Encryption with RC6-$w/r/b$

Input: Plaintext stored in four $w-$bit input registers A, B, C, D.
Number of rounds r.
Subkeys $S[0], \ldots, S[2r+3]$, each $w-$bits.

Output: Ciphertext stored in A, B, C, D.

Procedure:
$B = B + S[0]$
$D = D + S[1]$
for $i = 1$ **to** r **do**
$t = (B \times (2B+1)) \lll \lg w$
$u = (D \times (2D+1)) \lll \lg w$
$A = ((A \oplus t) \lll u) + S[2i]$
$C = ((C \oplus u) \lll t) + S[2i+1]$
$(A, B, C, D) = (B, C, D, A)$
$A = A + S[2r+2]$
$C = C + S[2r+3]$

Table 2.2: RC6 Encryption

rotation amount a function of *all* bits of the input register. The quadratic function also plays an important role in the avalanche effect of the cipher. We also remark that RC6 uses four input registers instead of two since AES was required to have 128-bit block size, and the only way to do that with the RC5 design was to use 64-bit words which is inefficient on 32-bit processors. Hence, using four 32-bit registers seemed to be the better design.

2.3 Differential Properties of Data-Dependent Rotations

Before going into a detailed analysis of RC6, we first want to study the differential properties of data-dependent rotations. All published differential analyses on ciphers that use data-dependent rotations (RC5, RC6, and MARS) always have assumed that a difference never occurs in a rotation amount. This seems to be quite a natural assumption, since a difference in a rotation amount would seem to produce a random looking output difference.

In this section, we provide an analytical proof that justifies the above heuristic argument. Our main result is a complete characterization of all possible output differences that may occur when a difference is in the rotation amount. This allows us to compute precisely the probability of any characteristic for the data-dependent rotations. In particular, when the difference occurs in the rotation amounts, all

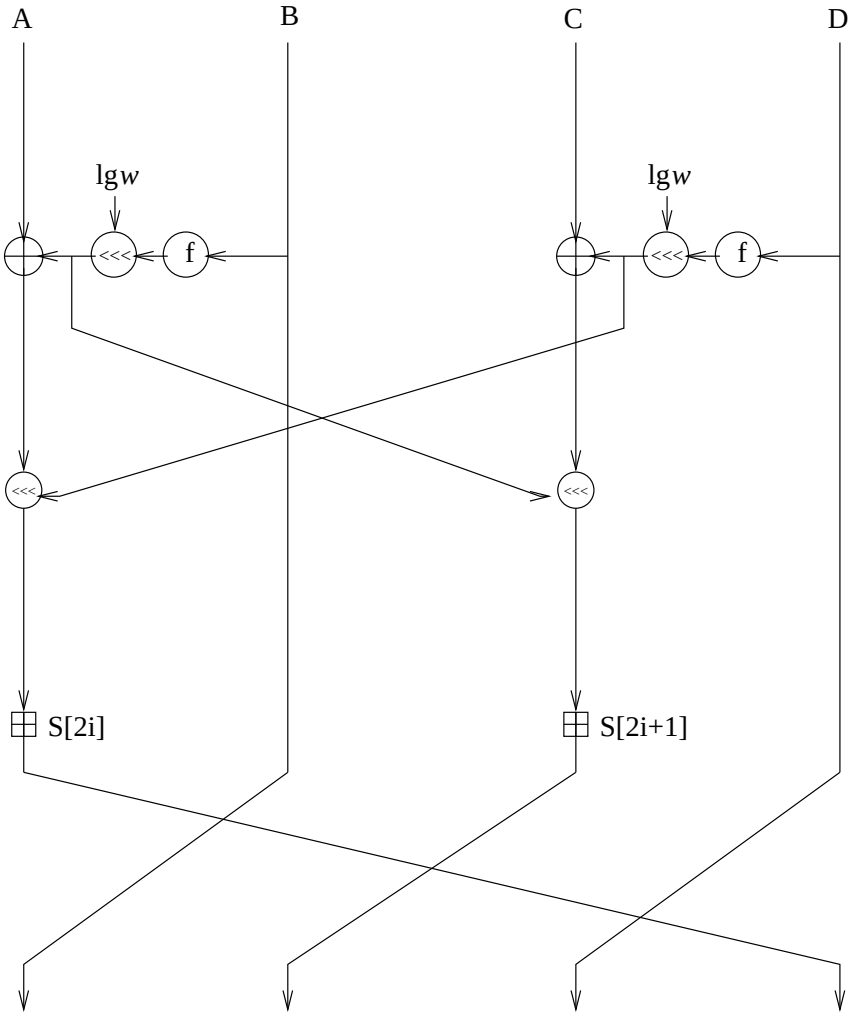

Figure 2.2: Diagram of a single round of RC6. Here, $f(X) = X(2X + 1)$ and $\boxplus$ is integer addition mod2^w.

possible characteristics for the data-dependent rotations hold with very small probability. Consequently, all differentials with small Hamming weight hold with very small probability. So these results quantify the effectiveness of data-dependent rotations in preventing differential attacks.

The results in this section were published in [10]. A preliminary version of the main theorem was given in [17].

2.3.1 Distribution of the Output Difference

Given a pair of inputs (X_1, R_1) and (X_2, R_2) where X_i is a word being rotated by the value R_i, we are interested in understanding the output difference in terms of the input differences. We call the input differences X' and R' and the output difference Y'. This is summarized in the following equations.

$$\begin{aligned} Y_1 &= X_1 \lll R_1, \\ Y_2 &= X_2 \lll R_2, \\ X' &= X_1 \oplus X_2, \\ R' &= R_1 \oplus R_2, \\ Y' &= Y_1 \oplus Y_2. \end{aligned}$$

We also introduce the variable

$$r' = (R_2 - R_1) \bmod w.$$

As we will show in the analysis, the variable r' is directly related to the probability of the characteristics for data-dependent rotations. Note that "no difference in the rotation amount" is equivalent to $r' = 0$ or $R' \bmod w = 0$.

For a fixed input difference X', consider the possible output differences Y'. Since Y' appears to depend on the two rotation amounts R_1 and R_2, this motivates us to define the function

$$\begin{aligned} f_{X',R_1,R_2}(X_1) &= (X_1 \lll R_1) \oplus ((X_1 \oplus X') \lll R_2) \\ &= (X_1 \lll R_1) \oplus (X_1 \lll R_2) \oplus (X' \lll R_2) \end{aligned}$$

which, for fixed X', R_1, and R_2, expresses the output difference in terms of the input X_1. We also define

$$\begin{aligned} I_{X',R_1,R_2} &= \{Y' : Y' = f_{X',R_1,R_2}(X_1) \text{ for some } X_1\}, \\ P_{X',R_1,R_2}(Y') &= \{X_1 : f_{X',R_1,R_2}(X_1) = Y'\}. \end{aligned}$$

That is, I_{X',R_1,R_2} (the image) is the set of output differences Y' when X_1 ranges over all possible values and $P_{X',R_1,R_2}(Y')$ (the pre-image) is the set of inputs X_1 which yield output difference Y'.

Theorem 2.2 *(1)* $|I_{X',R_1,R_2}| = 2^{w-\gcd(w,r')}$.
(2) For any $Y' \in I_{X',R_1,R_2}$*, the size of the set* $P_{X',R_1,R_2}(Y')$ *is* $2^{\gcd(w,r')}$.

Before proving the theorem, we first discuss some of its implications by contrasting the case where $r' = 0$ with the case where $r' \neq 0$:

1. $r' = 0$. The difference is not in the rotation amount.

   In this case, $\gcd(w, r') = w$ and $|I_{X',R_1,R_2}| = 1$, meaning that there is only *one* possible output difference Y'. All the characteristics used in existing differential attacks on RC5, RC6 and MARS belong to this category.

2. $r' \neq 0$. The difference is in the rotation amount.

   In this case, $gcd(w, r')$ is a power of 2 between 1 and $w/2$. Hence, $|I_{X',R_1,R_2}|$ ranges between $2^{\frac{w}{2}}$ and 2^{w-1}, and each possible output difference occurs *exactly* the same number of times. In other words, the output difference Y' is uniformly distributed in a set of size at least $2^{\frac{w}{2}}$ when the pair of inputs with a fixed difference ranges over all possible values.

From the above comparison, we can see that a difference in the rotation amount is spread out in the output difference in a drastic way.

Let us now move on to the proof of Theorem 2.2. We use some facts from group theory to simplify our understanding of the set of output differences I_{X',R_1,R_2}. The set of $w-$bit words form a group isomorphic to Z_2^w under the operation of exclusive-or. For a fixed integer r, the function $h(X) = X \lll r$ is a homomorphism: it has the property that $h(X \oplus Y) = h(X) \oplus h(Y)$. The function $p(X) = \text{parity}(X)$ is a homomorphism from Z_2^w to Z_2, and the kernel of p is the subgroup of even parity words, isomorphic to Z_2^{w-1}. Exclusive-oring any odd parity word to this subgroup yields the coset of odd parity words.

Proof of Theorem 2.2: We have $I_{X',R_1,R_2} = \{(X_1 \lll R_1) \oplus (X_1 \lll R_2) \oplus (X' \lll R_2)\}$. By replacing X_1 with the same value rotated right by R_1, we get a more convenient definition of the set:

$$I_{X',R_1,R_2} \;=\; \{X_1 \oplus (X_1 \lll r') \oplus (X' \lll R_2)\}.$$

Since $(X' \lll R_2)$ is a constant for fixed X' and R_2, the structure of I_{X',R_1,R_2} is determined by

$$g(X_1) = X_1 \oplus (X_1 \lll r'). \tag{2.1}$$

Let

$$S = \{g(X_1) : \; X_1 \text{ is a } w\text{-bit word}\}.$$

It is easy to verify that g is a homomorphism from the group of $w-$bit words (a group isomorphic to Z_2^w) to S, and S is a subgroup.

First, consider the special case where r' is odd. We claim that in this case S is isomorphic to Z_2^{w-1}. To prove this, we only need to show that the kernel of g has exactly two elements. The kernel consists of the X_1's satisfying

$$X_1 = X_1 \lll r'.$$

This property implies conditions on the bits of X_1: bit 0 must be the same as bit r', bit 1 must be the same as bit $r' + 1 \bmod w$, and so on. Since r' is relatively prime to w, we have that all bits must be the same, showing that the kernel is the two elements containing all 0's and all 1's.

Therefore, S is a subgroup isomorphic to Z_2^{w-1}. In particular, it is the subgroup of even parity words, and I_{X',R_1,R_2} is the coset of words having the same parity as X'. Hence $|I_{X',R_1,R_2}| = 2^{w-1} = 2^{w-\gcd(w,r')}$. The second part of the theorem follows from elementary group theory: the pre-image is the same size as the kernel.

For the general case, we write $r' = 2^e r$ where r is odd, so $2^{\gcd(w,r')} = 2^{2^e}$. A similar argument to the above shows that the kernel of g consists of elements of the form $a|a|\ldots|a$ where a is any of the 2^{2^e} values for a 2^e-bit vector (there are $w/2^e$ a's concatenated). Hence, the size of the kernel is 2^{2^e}, and the proof of the theorem follows immediately. □

Sean Murphy has suggested an alternate approach to proving the above theorem. His observation is that f_{X',R_1,R_2} is a linear transformation of a $w-$dimensional vector space given by

$$X_1 \mapsto R^{R_1}(I + R^{r'})X_1 + R^{R_2}X'$$

where R is the linear transformation of rotation by one place. Theorem 2.2 corresponds to the Rank-Nullity theorem applied to this transformation. The rank of the transformation is given by the rank of $I + R^{r'}$. Murphy claims this can be easily evaluated by a simple analysis of the eigenvalues of $R^{r'}$ [30].

2.3.2 Characteristics

Based on Theorem 2.2, we can easily compute the probability of any characteristic for the data-dependent rotations when the difference is in the rotation. For a binary vector X, we will use $w_H(X)$ to denote the *Hamming weight* of X.

Corollary 2.3 *For $i = 1, 2$, let $Y_i = X_i \lll R_i$. Let $r' = (R_2 - R_1) \bmod w$. Then each characteristic holds with either probability 0 or probability $2^{\gcd(w,r')-w}$.*

Note that for $w = 32$, the above theorem implies that the probability of any characteristic is at most $2^{-w/2} = 2^{-16}$ when the difference occurs in the rotation amount.

Corollary 2.4 *For $i = 1, 2$, let $Y_i = X_i \lll R_i$. Let $r' = (R_2 - R_1) \bmod w$. If $w_H(X') = 0$, then the probability that $w_H(Y') = 0$ is $2^{\gcd(w,r')-w}$.*

The following two corollaries follow from the proof of Theorem 2.2, since the parity of Y' must be the same as the parity of X'.

Corollary 2.5 *For $i = 1, 2$, let $Y_i = X_i \lll R_i$. Let $r' = (R_2 - R_1) \bmod w$. If $w_H(X') = 0$, then the probability that $w_H(Y') = 1$ is 0.*

Corollary 2.6 *For $i = 1, 2$, let $Y_i = X_i \lll R_i$. Let $r' = (R_2 - R_1) \bmod w$. If $w_H(X') = 1$, then the probability that $w_H(Y') = 0$ is 0.*

2.3.3 Differentials with Small Hamming Weights

From Theorem 2.3, we know that when the difference is in the rotation amount, all possible characteristics for data-dependent rotations hold with equal and very small probability. We now turn our attention towards differentials for the data-dependent rotations. This section will focus on differentials of Hamming weight one. The analysis can be extended to more general differentials.

Theorem 2.7 *For $i = 1, 2$, let $Y_i = X_i \lll R_i$. Let $r' = (R_2 - R_1) \bmod w$ be nonzero and write $r' = 2^e r$ where r is odd. For a given input difference X' such that $w_H(X') = 1$, the probability that $w_H(Y') = 1$ is $p = 2^{-(w - \lg(w) - 2^e + e)}$.*

Proof: The probability of the differential is

$$p = \frac{|\{Y' : Y' \in I_{X',R_1,R_2}| \text{ and } w_H(Y') = 1\}|}{|I_{X',R_1,R_2}|}. \tag{2.2}$$

Note that we are counting outputs here rather than inputs, but this is allowed since, as we showed earlier, each pre-image has the same size.

To evaluate the numerator, we rewrite Y' using the definition of $g(X_1)$ given in Equation 2.1.

$$Y' = g(X_1) \oplus (X' \lll R_2).$$

Since $w_H(X') = 1$, there are only two possibilities for $w_H(g(X_1))$ in order to have $w_H(Y') = 1$. That is, (1) $w_H(g(X_1)) = 0$, and (2) $w_H(g(X_1)) = 2$, and one of the two 1-bits in $g(X_1)$ lines up with the 1-bit in X'.

We claim that the set of values of $g(X_1)$ with Hamming weight 2 are those words in which the two 1-bits are a multiple of 2^e positions apart from each other. To prove this, we first show that $2^0 \oplus 2^e$ is in the set S of all outputs of $g(X_1)$. Since $g(2^0) = 2^0 \oplus 2^{r'}$ and $g(2^{r'}) = 2^{r'} \oplus 2^{2r'}$, using the homomorphism property of g we can construct a new Hamming weight 2 output by $g(2^0 \oplus 2^{r'}) = 2^0 \oplus 2^{2r'}$, where the exponents are taken modulo w. Continuing in this way, we see that $g(2^0 \oplus 2^{r'} \oplus \ldots \oplus 2^{(j-1)r'}) = 2^0 \oplus 2^e$, where $j = r^{-1} \bmod \frac{w}{2^e}$. Thus, we have $2^0 \oplus 2^e$, and by considering the left rotations of this word by i positions for $i = 0$ to $w - 2^e - 1$, we get a set of $w - 2^e$ linearly independent Hamming weight 2 elements in S. The linear combinations of these elements generate a subgroup of size 2^{w-2^e}, i.e. the entire set S. It is easy to see that any Hamming weight 2 word having the 1-bits separated by a multiple of 2^e positions from each other can be constructed from these basis words. $g(X_1)$ cannot contain a Hamming weight 2 word where the 1's are a distance 2^{e-1} apart (for example), since that would cause it to generate a subgroup of size $2^{w-2^{e-1}}$ which is larger than the subgroup under consideration.

So, the size of the set $\{g(X_1) : w_H(g(X_1)) = 2 \text{ and } w_H(Y') = 1\}$ is $\frac{w}{2^e} - 1$, since one of the 1-bits in $g(X_1)$ lines up with the 1-bit in X' and there are exactly

$\frac{w}{2^e} - 1$ positions for the other 1-bit. Adding on the one case where $g(X_1) = 0$, the numerator of Equation 2.2 becomes $\frac{w}{2^e}$ and hence the probability is $2^{-(w-\lg(w)-2^e+e)}$. □

Corollary 2.8 *Let the word size $w = 32$. If $w_H(X') = 1$ and there is a difference in the rotate amounts, then the probability that $w_H(Y') = 1$ is $\leq 2^{-15}$.*

Similar analysis shows that, for data-dependent rotations, all differentials with small Hamming weight hold with very small probability if there is a difference in the rotation amount. Therefore, it seems very unlikely that such differentials could be useful in attacking RC6.

2.4 Differential Cryptanalysis of RC6

We are now ready to begin the study of differential attacks on RC6. In order to understand how the new components, i.e. the quadratic function and the fixed rotate, contribute to the security, we shall start by analyzing some simplified variants of RC6. This analysis will be the building blocks for studying the full function.

The simplified variants are obtained by replacing the quadratic function with the identity function and/or removing the fixed rotate. The most basic variant, denoted RC6-I-NFR applies both of these changes. The "I" stands for *identity* and the "NFR" for *no fixed rotate*. Closer approximations to the real RC6 are obtained by only applying one of these changes: hence we have RC6-I and RC6-NFR. Graphically, we can view how well these variants approximate the security of RC6 as in Figure 2.3.

The purpose of studying the simplified variants is to get an idea on how to attack the full function. For this reason, we will not spend too much effort trying to optimize the attacks. There will be several trivial and not so trivial improvements, in particular ideas similar to [3]. Such ideas will be taken into consideration when we get to the full RC6.

The approach for attacking the simplified variants will be quite similar to how we described attacking RC5 in Section 2.1.2. The ideas follow the root work approach of Kaliski and Yin's security analysis of RC5 [17]. In contrast to RC5, when we consider the variants that use the quadratic function, we will find that subtraction is a better measure of difference than exclusive-or.

The analysis to be presented was published in [8].

2.4.1 Differential Cryptanalysis of RC6-I-NFR

RC6-I-NFR is the simplest variant of RC6 that we will look at. It eliminates both the quadratic function and the fixed rotate. Pseudo code is given in Table 2.3.

As we showed in Section 2.3, we want to avoid differences in the rotation amounts since such characteristics occur with very small probability. In order to do so, we must keep the avalanche effect under control. Let Δ be a generic difference in a

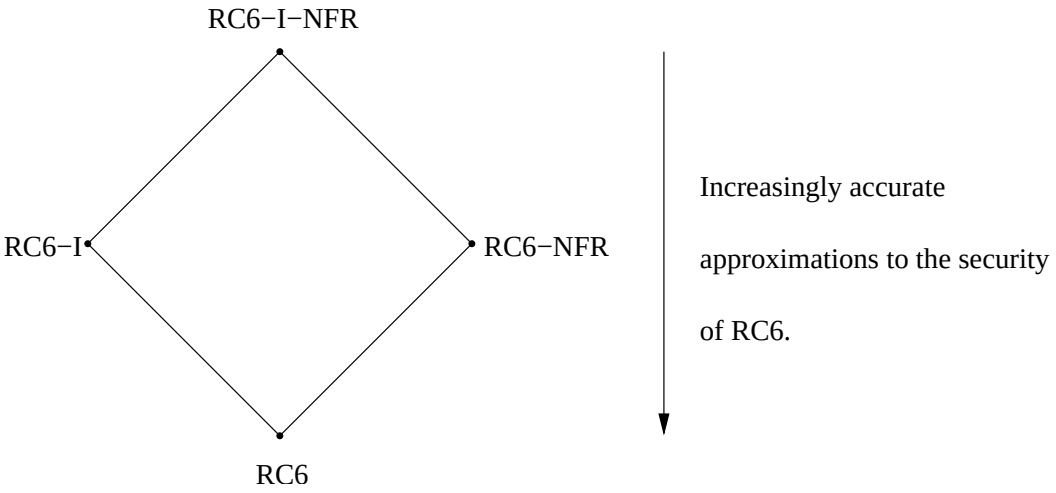

Figure 2.3: Simplified variants approximating the security of RC6.

Encryption with RC6-I-NFR $(w/r/b)$

Input: Plaintext stored in four $w-$bit input registers A, B, C, D.
Number of rounds r.
Subkeys $S[0], \ldots, S[2r+3]$, each $w-$bits.

Output: Ciphertext stored in A, B, C, D.

Procedure:
$B = B + S[0]$
$D = D + S[1]$
for $i = 1$ **to** r **do**
$t = B$
$u = D$
$A = ((A \oplus t) \lll u) + S[2i]$
$C = ((C \oplus u) \lll t) + S[2i+1]$
$(A, B, C, D) = (B, C, D, A)$
$A = A + S[2r+2]$
$C = C + S[2r+3]$

Table 2.3: RC6-I-NFR Encryption

(a)				(b)				(c)			
Δ	Δ	0	0	0	0	Δ	Δ	Δ	Δ	Δ	Δ
		$\downarrow$				$\downarrow$				$\downarrow$	
Δ	0	0	0	0	0	Δ	0	Δ	0	Δ	0
		$\downarrow$				$\downarrow$				$\downarrow$	
0	0	0	Δ	0	Δ	0	0	0	Δ	0	Δ
		$\downarrow$				$\downarrow$				$\downarrow$	
0	Δ	Δ	0	Δ	0	0	Δ	Δ	Δ	Δ	Δ
		$\downarrow$				$\downarrow$					
Δ	Δ	0	Δ	0	Δ	Δ	Δ				
		$\downarrow$				$\downarrow$					
Δ	Δ	Δ	0	Δ	0	Δ	Δ				
		$\downarrow$				$\downarrow$					
Δ	Δ	0	0	0	0	Δ	Δ				
$\alpha_\Delta^6 \rho_\Delta^6$				$\alpha_\Delta^6 \rho_\Delta^6$				$\alpha_\Delta^4 \rho_\Delta^4$			

Table 2.4: The three iterative characteristics for RC6-I-NFR, and their probabilities.

$w-$bit word, where $\Delta \equiv 0 \bmod w$. We consider possible characteristics that have the difference in each word as either 0 or Δ. Such characteristics are advantageous since they allow differences to cancel with each other.

Let ρ_Δ denote the probability that the difference Δ remains unchanged after a data-dependent rotation, and let α_Δ denote the probability that it remains unchanged after the subkey addition. Since there are 4 words which may or may not have a difference, and since we assume *some* difference exists, there are $2^4 - 1$ possible difference patterns. We can follow each such pattern from one round to the next assuming that when two differences meet, they will cancel. The result is one of three possible iterative characteristics . These characteristics and their corresponding probabilities are shown in Table 2.4. Observe that characteristic (b) is the same as characteristic (a) except with registers (A, B) are swapped with (C, D). Also, characteristic (c) has lower probability than characteristics (a) and (b) if we were to extend it to the same number of rounds. Therefore, characteristic (a) or (b) seems to be the most useful in designing a differential attack.

Assume $w = 32$, i.e. the word size used for RSA Data Security's submission as an AES candidate. By setting $\Delta = e_{31} = 2^{31}$, we have $\alpha_\Delta = 1$, and hence the characteristic is expected to hold with probabity $\rho_\Delta^6 = 2^{-30}$. This seems to be the best characteristic. Surprisingly, RC6-I-NFR has some hidden inter-dependencies which make this characteristic hold either with probability 0 or probability 2^{-20}, depending upon certain key bits. We will describe this phenomenom more in Section 2.4.6.

More generally, we consider characteristics where the main restrictions are that differences cannot occur in the rotation amounts and differences are lined-up so that they will cancel. Table 2.5 gives a general characteristic, where we assume

general characteristic			
e_s	e_s	0	0
	↓		
e_s	0	0	0
	↓		
0	0	0	e_t
	↓		
0	e_u	e_t	0
	↓		
e_u	e_u	0	e_v
	↓		
e_u	e_u	e_v	0
	↓		
e_u	e_u	0	0

Table 2.5: A general 6-round characteristic for RC6-I-NFR.

all of s, t, u, and v are between 5 and 31. For any given choice of s, t, u, and v, the characteristic has expected probability of about $(\frac{1}{2})^6 \times (\frac{1}{32})^6 = 2^{-36}$. Again, we emphasize that some of these characteristics will hold with higher probability (especially those that have the difference of e_{31}) and some with lower, but the average probability is expected to be close to 2^{-36}. For a specific choice of starting difference s, we can sum over all 27^3 choices for the intermediate differences, giving an expected differential probability of about $27^3 \times 2^{-36} \approx 2^{-22}$.

By iterating the above differential, we can attack an arbitrary number of rounds of RC6-I-NFR. Recall from Section 2.1.2 that if we want to attack $r-$rounds, we shall assume that we only need an iterative differential that lasts $r - 2$ rounds, since generally an extension of 2-rounds can be obtained for free (as explained in Section 2.1.2). Thus, to attack 20-rounds, we need differentials that last 18-rounds, which can be attained by chaining the above differential with itself 3 times. Hence, the probability of the differential for 18-rounds is about $(2^{-22})^3 = 2^{-66}$, which translates into needing on the order of 2^{66} chosen plaintexts to break 20-round RC6-I-NFR.

Table 2.6 summarizes the number of chosen plaintexts required for this attack for various numbers of rounds. The differential for 8-rounds has been verified experimentally. Note that there are obvious improvements to this attack, but we shall not consider such improvements until we get to the full RC6. Our goal here is just to outline the approach, which will be the basis for our more detailed analysis of RC6.

Differential Cryptanalysis of RC6-I-NFR					
Number of rounds	8	12	16	20	24
Number of chosen plaintexts	2^{22}	2^{32}	2^{45}	2^{66}	2^{76}

Table 2.6: Estimated number of chosen plaintexts required to attack RC6-I-NFR via the differential method described in this section.

2.4.2 Differential Cryptanalysis of RC6-I

The difference between RC6-I and RC6-I-NFR is the fixed rotate by $\lg w$ bits. Pseudo code for RC6-I is given in Table 2.7.

We can take the same approach for attacking it as we did for RC6-I-NFR: we use exclusive-or for difference, and concentrate on the cycle (a) from Table 2.4. However, we now have the added difficulty of lining up the difference bits. Table 2.8 gives the corresponding general characteristic for RC6-I. The exact restrictions are $0 \leq s, t, v \leq 26$ and $15 \leq u \leq 26$ so that differences do not occur in the rotation amounts. As in RC6-I-NFR, the general characteristic for 6-rounds holds with probability $(\frac{1}{2})^6 \times (\frac{1}{32})^6 = 2^{-36}$. Setting $u = s + 15$ gives an iterative characteristic with this probability.

Taking into consideration the effect of differentials, for a particular starting value s, there are 27 allowable values for each of t and v, and 12 for u. So, a 6-round differential for RC6-I holds with probability

$$2^{-36} \times 27^2 \times 12 \approx 2^{-23}.$$

Table 2.9 gives the expected number of chosen plaintexts required for attacking RC6-I for various choices of rounds, again under the assumption that we can get a differential to pass through an additional 2-rounds for free. It is evident that the extra fixed rotate does not add much security over RC6-I-NFR. However, we shall later see that the fixed rotate has a much more central security role when the quadratic function is put back in.

2.4.3 The Quadratic Function

In order to study RC6-NFR and RC6, we will need to understand some important differential properties of the quadratic function $f(X) = X(2X + 1)$. Unlike the simplified variants already analyzed, it is not clear that exclusive-or is the best measure of difference. In fact, given that the quadratic function uses both addition and multiplication, it seems more likely that subtraction is a better measure. So we start by analyzing subtraction, but we shall consider exclusive-or as well.

We first prove that the quadratic function is a permutation, which is a corollary of the following more general lemma.

Encryption with RC6-I $(w/r/b)$

Input: Plaintext stored in four $w-$bit input registers A, B, C, D.
Number of rounds r.
Subkeys $S[0], \ldots, S[2r+3]$, each $w-$bits.

Output: Ciphertext stored in A, B, C, D.

Procedure:
$B = B + S[0]$
$D = D + S[1]$
for $i = 1$ **to** r **do**
$t = B \lll \lg w$
$u = D \lll \lg w$
$A = ((A \oplus t) \lll u) + S[2i]$
$C = ((C \oplus u) \lll t) + S[2i+1]$
$(A, B, C, D) = (B, C, D, A)$
$A = A + S[2r+2]$
$C = C + S[2r+3]$

Table 2.7: RC6-I Encryption

general characteristic			
e_{s+5}	e_s	0	0
	$\downarrow$		
e_s	0	0	0
	$\downarrow$		
0	0	0	e_t
	$\downarrow$		
0	e_u	e_t	0
	$\downarrow$		
e_u	e_{u-5}	0	e_v
	$\downarrow$		
e_{u-5}	e_{u-10}	e_v	0
	$\downarrow$		
e_{u-10}	e_{u-15}	0	0

Table 2.8: A general 6-round characteristic for RC6-I.

Differential Cryptanalysis of RC6-I					
Number of rounds	8	12	16	20	24
Number of chosen plaintexts	2^{23}	2^{34}	2^{47}	2^{69}	2^{80}

Table 2.9: Estimated number of chosen plaintexts required to attack RC6-I via the differential method described in this section.

Lemma 2.9 *Let $\tilde{f}(X) = X(aX + b) \bmod 2^w$ where a is even and b is odd. Then $\tilde{f}$ is a one-to-one mapping from $\{0,1\}^w$ to $\{0,1\}^w$.*

Proof: Suppose $\tilde{f}(X_1) \equiv \tilde{f}(X_2) \bmod 2^w$. Then, $aX_1^2 + bX_1 - aX_2^2 - bX_2 \equiv 0 \bmod 2^w$, which implies

$$(X_1 - X_2)[a(X_1 + X_2) + b] \equiv 0 \bmod 2^w.$$

Since a is even and b is odd, the term $[a(X_1+X_2)+b]$ must be odd. But this implies $(X_1 - X_2) \equiv 0 \bmod 2^w$, so we have $X_1 \equiv X_2 \bmod 2^w$. Hence, the inputs must be the same. □

Let us now determine some basic differential properties of $f(X) = X(2X + 1) \bmod 2^w$ using subtraction as a measure of difference. We fix the following notation:

$$\begin{aligned} Y_1 &= f(X_1) \\ Y_2 &= f(X_2) \\ \delta_X &= X_2 - X_1 \qquad \text{(input difference)} \\ \delta_Y &= Y_2 - Y_1 \qquad \text{(output difference)} \end{aligned}$$

Substituting $X_2 = \delta_X + X_1$, we get

$$\delta_Y = (4X_1\delta_X + \delta_X + 2\delta_X^2) \bmod 2^w. \tag{2.3}$$

The following result is a trivial consequence of Equation 2.3, but is important enough to state as a lemma.

Lemma 2.10 *$\delta_Y = \delta_X$ if and only if $(4X_1\delta_X + 2\delta_X^2) \equiv 0 \bmod 2^w$.*

□

We shall hereafter restrict our analysis to the case $w = 32$, the word size in RSA Laboratories candidate submission for the AES. This restriction is only because certain results had to be verified experimentally. However, most of our analysis easily extends to other word sizes.

Using Integer Subtraction as a Measure of Difference

We begin the analysis of subtraction as measure of difference by studying differences of the form $\delta_Y = \delta_X$ for the quadratic function. We call such differences *static differences*. Static differences are particularly useful to begin our analysis since they easily chain together.

The next two lemmas characterize all static differences of the quadratic function.

Lemma 2.11 *If δ_X is odd then the characteristic $\delta_Y = \delta_X$ holds with probability zero.*

Proof: Applying Lemma 2.10 modulo 4, we have $\delta_Y = \delta_X$ if and only if $2\delta_X^2 \equiv 0 \bmod 4$. Since $\delta_X^2 \equiv 1 \bmod 4$, the left hand side is $2 \bmod 4$ and therefore the congruence cannot be satisfied. □

Lemma 2.12 *If $\delta_X = v2^i$ where v is odd and $1 \leq i \leq 30$, then the characteristic $\delta_X = \delta_Y$ holds with probability 2^{i-30}. If $\delta_X = 2^{31}$, then the characteristic $\delta_Y = \delta_X$ holds with probability one.*

Proof: By Lemma 2.10, we have $\delta_Y = \delta_X$ if and only if $4X_1v2^i + 2(v2^i) \equiv 0 \bmod 2^{32}$. This is equivalent to $X_1 \equiv -2^{i-1}v \bmod 2^{30-i}$ for $i \leq i \leq 30$. Note that if $i \geq 16$, the $-2^{i-1}v$ term is 0 modulo 2^{30-i}. In either case, we see that the relations holds with exactly probability 2^{i-30}, since the least significant $30-i$ bits of X_1 are completely determined and the remaining bits can be anything. The Lemma is trivially true when $\delta_X = 2^{31}$. □

Our initial analysis of RC6-NFR and RC6 shall use differences of Hamming weight 1. For such differences, we emphasize that the probability that $\delta_X = \delta_Y$ is negligibly small when the least significant difference bit i is small. This further justifies the use of the quadratic function as a tool to resist differential cryptanalysis: especially the way it is used in RC6.

Using Exclusive-Or as a Measure of Difference

We begin the analysis of exclusive-or as a measure of difference by proving a lemma analogous to Lemmas 2.11 and 2.12. The notion of difference with respect to exclusive-or is defined to be:

$$\begin{aligned} \delta_X^{\oplus} &= X_2 \oplus X_1, \text{ and} \\ \delta_Y^{\oplus} &= Y_2 \oplus Y_1. \end{aligned}$$

Lemma 2.13 *Let p_i denote the probability of the characteristic $\delta_Y^{\oplus} = \delta_X^{\oplus} = 2^i$. Then*

$$p_i = \begin{cases} 1 & \text{for } i = 31 \\ 2^{i-30} & \text{for } 15 \leq i \leq 20 \\ 0 & \text{for } 0 \leq i \leq 14 \end{cases}$$

$$\text{and } p_i \approx 2^{i-31} \text{ for } 21 \leq i \leq 30.$$

Proof: Assume $\delta_X^{\oplus} = 2^i$. Then either $X_2 = X_1 + 2^i$ with $X_1[i] = 0$ or $X_1 = X_2 + 2^i$ with $X_2[i] = 0$. Since these cases are symmetric with equal probabilities, we may assume $X_2 = X_1 + 2^i$ with $X_1[i] = 0$ without loss of generality. We divide the proof into cases according to i.

Case $16 \leq i \leq 30$**:** By Equation 2.3, we have

$$\begin{aligned} Y_2 - Y_1 &\equiv 2^{i+2}X_1 + 2^i + 2^{2i+1} \bmod 2^{32} \\ &\equiv 2^{i+2}X_1 + 2^i \bmod 2^{32} \text{ (since } i \geq 16) \end{aligned}$$

We have that $\delta_Y^{\oplus} = 2^i$ implies $Y_2 - Y_1 \equiv 2^i \bmod 2^{32}$ or $Y_1 - Y_2 \equiv 2^i \bmod 2^{32}$. It is easy to verify that the latter can never happen, so $\delta_Y^{\oplus} = 2^i$ if and only if both of these events hold:

$$\begin{aligned} &\text{Event A} \qquad 2^{i+2}X_1 \equiv 0 \bmod 2^{32}, \\ &\text{Event B} \qquad Y_1[i] = 0. \end{aligned}$$

Similar to the proof of Lemma 2.12, we see that $\text{prob}(A) = 2^{i-30}$. So, the characteristic holds with probability

$$p_i = \text{prob}(A) \times \text{prob}(B|A) = 2^{i-30} \times \text{prob}(B|A).$$

Event A implies the lower $30-i$ bits of X_1 are 0. Hence, the lower $2(30-i)+1 = 61-2i$ bits of $2X_1^2$ are 0, and the higher order bits are "approximately random." When $16 \leq i \leq 20$, this implies that bit i of $2X_1^2$ is 0, and since $Y_1 = 2X_1^2 + X_1$, we have $Y_1[i] = X_1[i]$. Since $X_1[i] = 0$, $\text{prob}(B|A) = 1$ for $16 \leq i \leq 20$ and therefore $p_i = 2^{i-30}$.

When $21 \leq i \leq 30$, bit i of $2X_1^2$ is "approximately random", and therefore $Y_1[i] = 0$ approximately $\frac{1}{2}$ of the time, so that $p_i \approx 2^{i-31}$. The "approximately random" is due to the fact that each bit of X^2 is not necessarily uniformly distributed, even when X is chosen uniformly at random. The exact probabilities, determined by exhaustive computation, are given in the following table.

i	2^{31-i}	$1/p_i$
21	1024	819.200000
22	512	455.111111
23	256	248.242424
24	128	126.025089
25	64	63.750731
26	32	31.938838
27	16	15.992672
28	8	7.998251
29	4	3.999823
30	2	1.999954

Case $i = 15$**:** This is similar to the previous case, but the details are slightly different. Setting $\delta_X = 2^{15}$ in Equation 2.3, we get

$$Y_2 - Y_1 \equiv 2^{17}X_1 + 2^{15} + 2^{31} \bmod 2^{32}.$$

bit position i	probability $\delta_Y = \delta_X = 2^i$	$\delta_Y^{\oplus} = \delta_X^{\oplus} = 2^i$
31	1	1
$21 \leq i \leq 30$	2^{i-30}	$\approx 2^{i-31}$
$15 \leq i \leq 20$	2^{i-30}	2^{i-30}
$1 \leq i \leq 14$	2^{i-30}	0
0	0	0

Table 2.10: Probabilities for single-bit, static differences of the quadratic function using both subtraction and exclusive-or as measure of difference.

So $\delta_Y^{\oplus} = 2^{15}$ if and only if both of these events hold:

$$\begin{aligned} &\text{Event A} \qquad 2^{17}X_1 \equiv 2^{31} \bmod 2^{32}, \\ &\text{Event B} \qquad Y_1[15] = 0. \end{aligned}$$

Following arguments similar to ones previously made, we have $\text{prob}(A) = 2^{-15}$. Therefore, the probability of the characteristic $\delta_Y^{\oplus} = \delta_X^{\oplus} = 2^{15}$ is

$$p_{15} = 2^{-15} \times \text{prob}(B|A).$$

Note that Event A implies the lower 14 bits of X_1 are 0 and that bit 15 is 1. Hence, the lower $(2 \times 14) + 1 = 29$ bits of $2X_1^2$ are 0. Since $Y_1 = 2X_1^2 + X_1$, we have $Y_1[15] = X_1[15] = 0$. Therefore, $p_{15} = 2^{-15} = 2^{i-30}$.

Case $1 \leq i \leq 14$**:** We have $\delta_Y^{\oplus} = 2^i$ if and only if

$$2^i \equiv 2^{i+2}X_1 + 2^i + 2^{2i+1} \bmod 2^{32},$$

which implies $X_1 \equiv -2^{i-1} \bmod 2^{30-i}$. This condition tells us that bits $i-1$ through $29 - i$ must be 1, which contradicts the assumption that bit i is 0. Hence, $p_i = 0$ for $1 \leq i \leq 14$.

Case $i = 0$ **and** $i = 31$**:** The case $i = 0$ follows from Lemma 2.11. For $i = 31$, we get $Y_2 - Y_1 = 2^{31}$ by substituting $i = 31$ in Equation 2.3. □

Comparing Exclusive-Or to Integer Subtraction

Using Lemmas 2.12 and 2.13, we can now compare integer subtraction to exclusive-or for static differences over the quadratic function. The probabilities are summarized in Table 2.10. In all cases, we have subtraction at least as good of a measure as exclusive-or. Furthermore, we see that exclusive-or is a particularly bad measure for $1 \leq i \leq 14$, since the probabilities are always 0.

This seems to suggest that subtraction is the better measure of difference. However, we should take one more factor into account: the exclusive-or that happens right after the quadratic function. This operation has no effect when we use exclusive-or as measure of difference, but it does for integer subtraction. Thus, we

need to examine the probabilities over both operations when integer subtraction is used.

Let $W = Z \oplus f(X)$. The analysis of this function using subtraction turns out to be more complicated than one might expect. In particular, one runs into the problem that the exclusive-or operation does not distribute over addition/subtraction. Thus, our analysis below involves computer enumeration, but in an intelligent way. After our results were published in [8], other researchers discovered efficient algorithms [23] [24] for closely related differential probabilities, however their algorithms cannot be directly applied to our problem. The research of [23] has the role of exclusive-or and subtraction the opposite of our problem. The research of [24] is almost what we are interested in, except their results do not apply due to properties of the quadratic function. More specifically, their result can only be directly applied when a function g is used such that $g(X_1 + \delta_X) - g(X_1)$ is a constant. The quadratic function does not work since Equation 2.3 is not a constant – it depends upon X_1.

For two sets of inputs X_1, Z_1 and X_2, Z_2, we define $W_1 = Z_1 \oplus f(X_1)$ and $W_2 = Z_2 \oplus f(X_2)$. The differences are $\delta_X = X_2 - X_1$, $\delta_Z = Z_2 - Z_1$, and $\delta_W = W_2 - W_1$.

Lemma 2.14 *Let p_i be the probability of the characteristic $(\delta_X, \delta_Z) \rightarrow \delta_W$ where $(\delta_X, \delta_Z) = (2^i, 0)$ and $\delta_W = 2^i$. Similarly, let q_i be the probability of the characteristic $(\delta_X, \delta_Z) = (2^i, 2^i)$ and $\delta_W = 0$. Then we have*

$$\begin{aligned} p_i = q_i = 2^{i-31} \quad & \textit{for } 15 \leq i \leq 31, \\ p_i = q_i \in [2^{i-35}, 2^{i-30}] \quad & \textit{for } \ 0 \leq i \leq 14. \end{aligned}$$

Proof: Consider p_i first. We determined the probabilities by computer enumeration which works as follows. For each value of i between 0 and 31, we count the number of times that $\delta_W = 2^i$ given that $(\delta_X, \delta_Z) = (2^i, 0)$. In total, there are 2^{64} possibilities for the X_1, Z_1 (which, combined with δ_X, δ_Z, determine the X_2, Z_2). A naïve search of this form would be infeasible, but the following observations make it doable on a single PC:

1. For integers a and b, we have $a - b \equiv 2^i \bmod 2^{32}$ only if $a \oplus b$ contains exactly one block of consecutive 1-bits. This block will always begin at bit i and end at some bit $k \geq i$, so the length of the block is $(k - i + 1)$ bits. Taking $a = Z \oplus f(X_2)$ and $b = Z \oplus f(X_1)$ (i.e. $Z = Z_1 = Z_2$) gives a necessary condition on $f(X_2) \oplus f(X_1)$ for δ_W to be 2^i.

2. Given $f(X_1)$ and $f(X_2)$ satisfying the first condition, the number of words for which $\delta_W = 2^i$ is

$$\begin{aligned} 2^{32-(k-i+1)} \quad & \text{if } k < 31 \text{ and} \\ 2^{32-(k-i)} \quad & \text{if } k = 31. \end{aligned}$$

   More precisely, the value of Z for the bits between indices i and k must be so that the corresponding bits in W_2 are equal to 2^k and the corresponding bits in W_1 are equal to $2^k - 2^i$ when $k < 31$. The other bits of Z are free

to be anything. When $k = 31$, there is one extra choice of Z that works: the choice where the bits between indices i and 31 in W_2 are all 0's, and the corresponding bits in W_1 all 1's.

The full search showed that p_i is exactly 2^{i-31} for $15 \leq i \leq 31$. For smaller values of i the results are given in the following table.

i	2^{31-i}	$1/p_i$	i	2^{31-i}	$1/p_i$
0	2^{31}	$2^{30.10}$	8	2^{23}	$2^{25.92}$
1	2^{30}	$2^{29.15}$	9	2^{22}	$2^{25.44}$
2	2^{29}	$2^{28.57}$	10	2^{21}	$2^{24.98}$
3	2^{28}	$2^{28.17}$	11	2^{20}	$2^{23.09}$
4	2^{27}	$2^{27.71}$	12	2^{19}	$2^{19.58}$
5	2^{26}	$2^{27.21}$	13	2^{18}	$2^{18.10}$
6	2^{25}	$2^{26.89}$	14	2^{17}	$2^{17.01}$
7	2^{24}	$2^{26.42}$			

Finally, we note that $p_i = q_i$ because there is a 1-to-1 correspondence between the solutions, simply by swapping the W values with the Z values. □

Observe that p_i decreases monotonically as i decreases, but not at a constant rate. For example, there is a big drop in the probabilities going from p_{12} to p_{11}.

Comparing Lemma 2.14 to Table 2.10 shows that $W = Z \oplus f(X)$ has integer subtraction at least as good of a difference measure as exclusive-or for all values of i except $15 \leq i \leq 20$. For this exceptional range, exclusive-or is twice as good as integer subtraction. However, once we take into consideration the addition of subkeys, the probability using exclusive-or will drop by a factor of 2 and the probability using integer subtraction will remain the same. So in the big picture, we see that integer subtraction is always at least as good as exclusive-or for difference measure in RC6-NFR and RC6. We also emphasize that half of the values of i have probability zero for exclusive-or. In summary, all analysis indicates that integer subtraction is the best measure.

Other Characteristics for the Quadratic Function

One can further consider characteristics that have more than one difference bit in the quadratic function, i.e. non-static characteristics. In fact, our Lemmas 2.11 and 2.12 were written general enough to allow for arbitrary differences. However, they do not tell the probability of these differences returning to a single bit difference. The following lemma fills in the gap:

Lemma 2.15 *Let m_i be the probability of the characteristic $\delta_X \rightarrow \delta_Y$ where $\delta_X = 2^i v$ for odd integer v and $\delta_Y = 2^i$. Then,*

$$m_i = \begin{cases} 2^{i-30} & \textit{for } 29 \geq i \geq 1 \textit{ and } v \equiv 1 \bmod 4, \\ 0 & \textit{for } i = 0 \textit{ and } v \equiv 1 \bmod 4, \\ 0 & \textit{for } 30 \geq i \geq 1 \textit{ and } v \equiv 3 \bmod 4, \\ 2^{i-30} & \textit{for } i = 0 \textit{ and } v \equiv 3 \bmod 4. \end{cases}$$

Proof: By Equation 2.3,

$$\delta_Y \equiv 2^{i+2} X_1 v + 2^i v + 2^{2i+1} v^2 \bmod 2^{32}.$$

This is 2^i if and only if

$$2^{i+2} X_1 v + 2^i (v-1) + 2^{2i+1} v^2 \equiv 0 \bmod 2^{32}.$$

We separate the cases according to v mod 4.

Case $v \equiv 1 \bmod 4$**:** If $i = 0$, then the term $2^{2i+1}v^2$ is 2 mod 4 while the other terms are 0 mod 4. Therefore, there is no solution. If $i \geq 1$, then all terms are divisible by 2^{i+2}. Dividing out the 2^{i+2}, we have:

$$X_1 v + \frac{(v-1)}{4} + 2^{i-1} v^2 \equiv 0 \bmod 2^{30-i}.$$

Basic modular arithmetic shows that there is a unique solution for X_1 modulo 2^{30-i}. This implies $m_i = 2^{i-30}$.

Case $v \equiv 3 \bmod 4$**:** If $i = 0$, then $v-1$ is 2 mod 4 and $2^{2i+1}v^2 = 2v^2$ is 2 mod 4, so the sum is 0 mod 4. Dividing out by 4, we have:

$$X_1 v + \frac{(v-1) + 2v^2}{4} \equiv 0 \bmod 2^{30}.$$

There is a unique solution modulo 2^{30}, so we get $m_i = 2^{-30}$. If $i \geq 1$, then divide out by 2^i to get:

$$4X_1 v + (v-1) + 2^{i+1} v^2 \equiv 0 \bmod 2^{30-i}.$$

The term $v-1$ is 2 mod 4 while the other terms are 0 mod 4, so there is no solution. □

Observe that this lemma tells us that a non-static difference gets transformed into a single-bit difference by the quadratic function with the same probability that a single-bit difference remains a single-bit difference. Although the computations have not been performed, it is assumed we have the same probabilities for an analogue of Lemma 2.14. By this, we mean the cases where there is a non-static input to the quadratic function and either a single-bit input to the exclusive-or, or a single-bit output from the exclusive-or.

2.4.4 Differential Cryptanalysis of RC6-NFR

In the preceding section, we developed the tools needed to study the resistance of both RC6-NFR and RC6 to differential cryptanalysis. We saw that subtraction is the better measure of difference, so we shall only consider subtraction hereafter. We begin with RC6-NFR, for which pseudo code is given in Table 2.11.

Similar to our analysis of RC6-I-NFR, we start with the characteristic from Table 2.5. The probability of this 6-round characteristic is

$$\rho^6 \times q_s \times p_t \times p_u \times q_u \times p_v \times q_u.$$

Encryption with RC6-NFR $(w/r/b)$

Input: Plaintext stored in four $w-$bit input registers A, B, C, D.
Number of rounds r.
Subkeys $S[0], \ldots, S[2r+3]$, each $w-$bits.

Output: Ciphertext stored in A, B, C, D.

Procedure:
$B = B + S[0]$
$D = D + S[1]$
for $i = 1$ **to** r **do**
$t = B \times (2B + 1)$
$u = D \times (2D + 1)$
$A = ((A \oplus t) \lll u) + S[2i]$
$C = ((C \oplus u) \lll t) + S[2i + 1]$
$(A, B, C, D) = (B, C, D, A)$
$A = A + S[2r + 2]$
$C = C + S[2r + 3]$

Table 2.11: RC6-NFR Encryption

Note that since we are using subtraction as difference, we are able to cross the integer subkey addition with probability one. The parmaters s, t, u, and v may take any value between 5 and 31. The probability is highest when they are all 31, since $p_{31} = q_{31} = 1$. Thus, we get a characteristic of probability $\rho^6 = 2^{-30}$. This may at first seem a bit disconcerting, since the probability is the same as the corresponding characteristic for RC6-I-NFR. However, the benefit becomes clear in the analysis of differentials.

To get an iterative differential with maximum probability, we fix $s = u = 31$ and allow t and v to take any value between 5 and 31.[2] The probability is then

$$\rho^6 \times \sum_{t=5}^{31} p_t \times \sum_{v=5}^{31} p_v \approx 2^{-28}.$$

Note that this is only 4 times better than the best characteristic probability. Compare that to RC6-NFR, where the differential was $\approx 2^8$ times better than the characteristic probability. The difference is that in RC6-I-NFR, each characteristic has approximately equal probability, whereas in RC6-NFR, only differences in the most

[2]In order to chain the differentials together, the output u must be equal to the input s. For smaller values of u and s, the probabilities drop significantly. Even if one does sum over all values of u, the probability comes out about the same as when we fix $u = 31$.

Differential Cryptanalysis of RC6-NFR					
Number of rounds	8	12	16	20	24
Number of chosen plaintexts	2^{28}	2^{47}	2^{61}	2^{84}	2^{103}

Table 2.12: Estimated number of chosen plaintexts required to attack RC6-NFR via the differential method described in this section.

significant bits have any effect on the overall summation. The quadratic function makes characteristics with differences in low order bits negligible in the big picture.

Table 2.12 summarizes the plaintext requirements to attack RC6-NFR with various numbers of rounds. We see that RC6-NFR provides moderate resistance to differential cryptanalysis, but not enough to meet the requirements of the AES.

2.4.5 Differential Cryptanalysis of RC6

Pseudo code for RC6 encryption is given in Table 2.2 and a diagram of the round function is in Figure 2.2.

We begin the differential cryptanalysis of RC6 by trying to adapt the attacks used against RC6-I to the full RC6. The left hand side of Table 2.13 shows a general characteristic which can be used to attack RC6. Similar to the computation in Section 2.4.4, its probability is

$$\rho^6 \times q_s \times p_t \times p_u \times q_{u-5} \times p_v \times q_{u-10}.$$

The allowable values for the parameters are $0 \leq s, t, v \leq 26$ and $15 \leq u \leq 26$. To make it iterative, we require $s + 5 = u - 10$. To maximise the probability, we choose the parameters as high as possible (see Lemma 2.14) subject to the above constraints. Thus, we take $s = 11$ and $t = u = v = 26$ (See right hand side of Table 2.13). The probability is $\approx 2^{-93}$.

For the iterative differential probability, we fix $s = 11$, $u = 26$ and allow t and v to vary over the 26 possibilities. The probability is

$$\rho^6 \times q_{11} \times \sum_{t=0}^{26} p_t \times p_{26} \times q_{21} \times \sum_{v=0}^{26} p_v \times q_{16} \approx 2^{-91}.$$

Similar to RC6-NFR, the iterative differential probability is only 4 times better than the best characteristic probability. We call this iterative differential I_6.

Other Static Iterative Differentials to Consider

In order to be sure that the differential I_6 is the static differential of maximum probability, we have also computed the probability of a cycle type (c) applied to RC6 (recall Table 2.4). Table 2.14 gives such a general iterative characteristic lasting

general characteristic					a specific choice			
e_{s+5}	e_s	0	0		e_{16}	e_{11}	0	0
	$\downarrow$					$\downarrow$		
e_s	0	0	0		e_{11}	0	0	0
	$\downarrow$					$\downarrow$		
0	0	0	e_t		0	0	0	e_{26}
	$\downarrow$					$\downarrow$		
0	e_u	e_t	0		0	e_{26}	e_{26}	0
	$\downarrow$					$\downarrow$		
e_u	e_{u-5}	0	e_v		e_{26}	e_{21}	0	e_{26}
	$\downarrow$					$\downarrow$		
e_{u-5}	e_{u-10}	e_v	0		e_{21}	e_{16}	e_{26}	0
	$\downarrow$					$\downarrow$		
e_{u-10}	e_{u-15}	0	0		e_{16}	e_{11}	0	0

Table 2.13: A general 6-round iterative characteristic for attacking RC6, and a specific choice of parameters giving maximum probability of 2^{-93}. The corresponding iterative differential known as I_6 has probability 2^{-91}.

6-rounds, which has probability

$$\rho^8 \times q_s^2 \times q_{t-5}^2 \times p_t^2 \times p_u^2.$$

Setting $s = 21$ and $t = u = 26$ gives a probability of 2^{-100}. In this differential, only the variable t can be free, which results in a probability of 2^{-99}. Hence, the differential is a factor of 2^8 times worse than I_6.

The Possibility of Multi-bit Differentials

It is tempting to try multi-bit (i.e. non-static) differentials in hope of finding a better iterative result than I_6. We have so far been unsuccessful in doing so. It is worth saying a few words about why it does not seem to work very well.

Similar to static differentials, there are bounds on where the difference bits can be in the multi-bit case. For example, if in the static case we have $0 \leq s \leq 26$, then the corresponding multi-bit bound would be differences where the least significant difference bit is at least 0 and the most significant is no more than 26. In the static case, we get the maximum probability by putting the difference bit in the most significant postion possible, because lower positions have much lower probabilities. But in the multi-bit case, often what decides the probability is where the least significant difference bit is: See Lemmas 2.12 and 2.15. So, to make the probability as high as possible in the multi-bit case, we want to put the *least* significant difference bit in the most significant position possible. This effectively suggests that we want to use static characteristics.

general characteristic					a specific choice			
e_{s+5}	e_s	e_{s+5}	e_s		e_{26}	e_{21}	e_{26}	e_{21}
	$\downarrow$					$\downarrow$		
e_s	0	e_s	0		e_{26}	0	e_{26}	0
	$\downarrow$					$\downarrow$		
0	e_t	0	e_t		0	e_{26}	0	e_{26}
	$\downarrow$					$\downarrow$		
e_t	e_{t-5}	e_t	e_{t-5}		e_{26}	e_{21}	e_{26}	e_{21}
	$\downarrow$					$\downarrow$		
e_{t-5}	0	e_{t-5}	0		e_{26}	0	e_{26}	0
	$\downarrow$					$\downarrow$		
0	e_u	0	e_u		0	e_{26}	0	e_{26}
	$\downarrow$					$\downarrow$		
e_u	e_{u-5}	e_u	e_{u-5}		e_{26}	e_{21}	e_{26}	e_{21}

Table 2.14: A general 6-round iterative characteristic for attacking RC6, and a specific choice of parameters giving maximum probability of 2^{-100}. The corresponding iterative differential has probability 2^{-99}, and therefore this differential is not as effective as I_6.

An alternate approach one may try is to not have differences cancel. This quickly results in differences in all four registers. Each time such a difference goes through the quadratic function, we cannot have a resulting difference in the most significant 5-bits, assuming we are to avoid differences in the data-dependent rotates. Since there are two quadratic functions per round, we immediately get a probability of about 2^{-10} of having allowable rotate values. Let us write a difference as $v2^i$, where v is odd (i.e. i represents the least significant difference bit). We may attempt to keep the differences small, for example, v is no more than 5-bits, or we may allow differences to be large. With the small difference approach, we get additional restrictions that need to be satisfied: a difference in one register must be rotated an amount so that it approximately lines up with a difference in another register, and each rotation value has probability 2^{-5}. This suggests a probability of about 2^{-20} per round, which is far below what we accomplish in the static case. Although sometimes the probability can be a little better, there are additional restrictions such as the quadratic function not causing more than a small difference. With the large difference approach, the problem seems to be that the difference paths are no longer distinguishable from random noise. Indeed, differential cryptanalysis is only effective if we can control the propagation of differences.

general characteristic					a specific choice			
e_{s+5}	e_s	0	0		e_{31}	e_{26}	0	0
		↓					↓	
e_s	0	0	0		e_{26}	0	0	0
		↓					↓	
0	0	0	e_t		0	0	0	e_{26}
		↓					↓	
0	e_u	e_t	0		0	e_{26}	e_{26}	0
		↓					↓	
e_u	e_{u-5}	0	e_v		e_{26}	e_{21}	0	e_{26}
		↓					↓	
e_{u-5}	e_w	e_v	0		e_{21}	e_{26}	e_{26}	0
		↓					↓	
e_w	e_x	0	$e_{u-5+y}+$ e_{w+5+y}		e_{26}	e_{31}	0	$e_{21}+$ e_{31}

Table 2.15: The left hand side is a general customized characteristic for attacking RC6. By choosing $s = 11$, we can get a differential E_6 that holds with probability 2^{-74} and can be appended to I_6. The right hand side contains a specific choice of parameters that make the characteristic hold with maximum probability. Although this specific choice cannot be appended to I_6, it is still useful for attacking 8-rounds of RC6. The corresponding differential holds with probability 2^{-56}.

Non-iterative Customized Differentials for RC6

Clearly we can attack an arbitrary number of rounds of RC6 by repeatingly using I_6. However, unlike the simplified variants of RC6, we now want to consider special optimizations that do not necessarily use iterative differentials, in order to attack RC6 as efficiently as possible. We call such non-iterative differentials *customized differentials*. We will specifically be seeking customized differentials that can be appended or prepended to I_6.

We begin with a 6-round customized characteristic that can be appended to I_6. Table 2.15 displays such a characteristic. As one can see, the first four rounds are the same as I_6, but the last two differ. Specifically, rather than enforcing that the e_{u-5} disappears between the fifth and sixth rounds, we have instead allowed a difference to be added onto it. The probability of this characteristic is

$$\rho^7 \times q_s \times p_t \times p_u \times q_{u-5} \times p_v \times p_w.$$

By taking $s = 11$ and $t = u = v = w = 26$, we get a characteristic that holds with probability

$$\approx 2^{-35} \times 2^{-23} \times 2^{-5} \times 2^{-5} \times 2^{-10} \times 2^{-5} \times 2^{-5} = 2^{-88}.$$

general characteristic			
e_{s+5}	e_s	0	0
	$\downarrow$		
e_s	0	0	0
	$\downarrow$		
0	0	0	e_t

Table 2.16: The characteristic E_2 which can be appended to I_6 by choosing $s = 11$. In this case the corresponding differential probability is 2^{-23}.

general characteristic			
e_{s+5}	e_s	0	0
	$\downarrow$		
e_s	0	0	0
	$\downarrow$		
0	0	0	e_t
	$\downarrow$		
0	e_u	e_t	0
	$\downarrow$		
e_u	e_w	0	e_v

Table 2.17: The characteristic E_4 which can be appened to I_6. The corresponding differential probability is 2^{-41}.

In the differential, we allow t, u, v, and w to take on any value between 0 and 26 which increases the probability by a factor of 2^4, and x and y to be any value between 0 and 31 which increases the probability by a factor of 2^{10} (i.e. all rotates are allowed). Thus, the differential probability is about $2^{-88} \times 2^4 \times 2^{10} = 2^{-74}$. We call this differential E_6.

Furthermore, if we wanted to attack only 8-rounds of RC6, then we can use a differential similar to E_6, but further improving it by starting with $s = 26$. This increases the probability by a factor of about 2^{18}, giving an 8-round differential attack on RC6 with complexity $\theta(2^{56})$. We shall refer to this differential as E_6'.

Depending upon the number of rounds we are attacking, we will also be interested in appending a 2-round or a 4-round differential to I_6. Table 2.16 shows a 2-round characteristic that holds with probability $\rho \times q_s$. For it to be appended to I_6, we must have $s = 11$. In the differential which we call E_2, t is allowed to take on any value, so the probability is $q_{11} = 2^{-23}$. Table 2.17 shows a 4-round characteristic that holds with probability $\rho^4 \times q_s \times p_t \times p_u$. Again, we require $s = 11$ to append it to I_6. In the differential which we call E_4, w and v can take on any value and t

Differential Cryptanalysis of RC6					
Number of rounds	8	12	16	20	24
Number of chosen plaintexts	2^{56}	2^{97}	2^{190}	2^{238}	2^{299}
Differential method	E'_6	$B_6 - E_4$	$B_6 - I_6 - E_2$	$B_6 - I_6 - E_6$	$B_6 - I_6 - I_6 - E_4$

Table 2.18: The estimated number of chosen plaintexts required to attack RC6, and the corresponding differential paths that give that attack. According to these estimates, RC6 is invulnerable to differential cryptanalysis as soon as the number of rounds is at least 16 because the number of required plaintexts is more than 2^{128}.

and u range between 0 and 26. This gives a probability of 2^{-41}.

Finally, we use a 6-round non-iterative differential which can be prepended to I_6. The differential is the same as I_6 (Table 2.15) except we start with $s = 26$, which improves the probability to 2^{-76}. We call this differential B_6.

By puzzling together these various differentials, we get estimates of the attack complexities for attacking various numbers of rounds of RC6, as show in Table 2.18. These are the best differential attacks so far published against RC6. Note that when the complexity (i.e. number of chosen plaintexts required) becomes more than 2^{128}, the attacks are infeasible since there are only 2^{128} plaintexts available. Therefore, it would require huge improvements to our attacks in order to cryptanalyze as few as 16-rounds of RC6.

On the Possibility of Weak Keys

When we analyzed RC5, we used the average probability of a static difference over subkey addition in our analysis, which is $\frac{1}{2}$ (see Lemma 2.1). However, we remarked that the real probability depends upon the least significant bits of the subkey, and can be anywhere between 0 and 1. For a particularly unlucky choice of subkeys, a characteristic or differential probability may in fact be substantially higher than our analysis suggests, potentially resulting in weak keys.

Such weak keys are unlikely to exist for RC6. If integer subtraction is fixed as the measure of difference, the subkey addition holds with probability 1, so our analysis is not affected. We also note that the best iterative differential we found, I_6 (Table 2.13), cannot hold for exclusive-or as measure of difference, since it requires the difference e_{11} to pass over the quadratic function. We proved in Lemma 2.13 that this cannot happen.

Statement on the Applicability of the Biryukov and Kushilevitz Attack

The Biryukov and Kushilevitz attack on RC5 [3] improves upon [17] by allowing more general but carefully controlled difference patterns. Rather than having differences always cancel, they allow differences to add onto each other, as long as the

number of difference bits does not get too large. This restriction is necessary so that expected output still occurs more likely than random noise.

Biryukov and Kushilevitz take advantage of the way differences propagate throughout RC5. Consider what happens when starting with a 1-bit difference and assuming that differences do not get in the rotate amounts, differences do not cancel with each other, and differences do not propagate from the subkey addition. The 1-bit difference becomes a 2-bit difference, then a 3-bit difference, then 5-bit, then 8-bit, and so on. The Hamming weight difference pattern is that of a Fibonacci sequence. Of course, this cannot continue forever.

Whenever a rotate amount is 0, a step back is taken in the Fibonacci sequence. For example, if a current difference has Hamming weight 5 (which comes from one register having Hamming weight 3 difference and the other having Hamming weight 2), then the result of the current half-round will be a Hamming weight 3 difference. Taking into consideration these backward steps, they define a *corrected Fibonacci sequence* with up to k corrections as a Fibonacci sequence having at most k backward steps. Thus, in their attack, any characteristic that follows the corrected Fibonacci sequence is the signal which allows them to uncover key bits after enough chosen plaintexts.

The same idea seemingly could apply to RC6. In order to do so, we would require that the quadratic function does not "disrupt" the difference in an unpredictable way. For instance, we would require that differences always remain the same after the quadratic function (Lemma 2.12) so that the corrected Fibonacci sequence remains intact. At first, this may seem to improve the attacks already presented in this chapter since there are more signals to look for. But on the other hand, remember that this quickly results in differences in all four registers which greatly reduces the probability of each characteristic because there are more ρ's and more applications of Lemma 2.12 that show up. It is our expectation that this negative aspect would make the Biryukov and Kushilevitz idea not very helpful in improving our attacks, with possible exception to when we only allow the differences to not cancel in the last few rounds. If this is indeed the case, then only small improvements could be made to the attacks we have presented. We leave the investigation to future researchers.

2.4.6 Addendum: Non-Random Behavior of RC6-I-NFR and RC6-NFR

In the analyses that we have done so far, we have made the preliminary assumptions that all probabilities are independent. It is prudent to verify that these probabilities do agree with experimental results. In doing so, we found that RC6-I-NFR and RC6-NFR had some hidden inter-dependencies from round to round. It turns out that the values of certain subkeys have a direct effect on whether or not certain characteristics can hold. A similar phenomenom was observed by Knudsen and Meier in [19]. In this section, we explain those inter-dependencies and argue that they do not seem to hold in the version that involve a fixed rotate. Experimental evidence supports this conclusion.

i	A_i	B_i	C_i	D_i
1	e_{31}	e_{31}	0	0
		$\downarrow$		
2	e_{31}	0	0	0
		$\downarrow$		
3	0	0	0	e_{31}
		$\downarrow$		
4	0	e_{31}	e_{31}	0
		$\downarrow$		
5	e_{31}	e_{31}	0	e_{31}
		$\downarrow$		
6	e_{31}	e_{31}	e_{31}	0
		$\downarrow$		
7	e_{31}	e_{31}	0	0

Table 2.19: A characteristic for RC6-I-NFR and RC6-NFR.

The results in this section were published in [9].

Refining the Analysis of RC6-I-NFR and RC6-NFR

Consider the characteristic for RC6-I-NFR and RC6-NFR given in Table 2.19. We are using the notation A_i (respectively B_i, C_i and D_i) to denote the values of registers A (respectively B, C, and D) at the beginning of round i. As an example, A_1, B_1, C_1, and D_1 contain the plaintext input after pre-whitening (i.e. after adding on $S[0]$ and $S[1]$) and for the six-round variants of the cipher, A_7, B_7, C_7 and D_7 contain the output prior to post-whitening.

In both variants, our analysis indicated that this characteristic should occur with probability 2^{-30} (Sections 2.4.1 and 2.4.4). Here, we show that it can only occur if certain subkey conditions are met. Further, once these subkey conditions hold, then the characteristic occurs with probability 2^{-20}. This is much higher than the initial estimate of 2^{-30} that was obtained by averaging over all subkeys.

In the analysis that follows we will concentrate on RC6-NFR. The same arguments and results can be applied to RC6-I-NFR by replacing $f(X) = X \times (2X + 1)$ with the identity function $f(X) = X$. We will use the fact that $X \bmod 2^i$ uniquely determines $(X \times (2X + 1)) \bmod 2^i$ (see Lemma 2.9).

Lemma 2.16 *If the characteristic given in Table 2.19 holds for RC6-NFR, then the following two conditions on the subkeys must hold:*

$$f(-S[9]) \equiv -S[7], \bmod 32$$
$$f(S[8]) \equiv -S[11] \bmod 32$$

Proof: First we observe that if the characteristic is to hold, then certain rotation amounts derived from the B and D registers must be zero. Note that we always

have that $B_i = A_{i+1}$ and that $D_i = C_{i+1}$. As a consequence, for the characteristic to hold we must have

$$\begin{array}{ll} D_2 \equiv C_3 \equiv 0 \bmod 32, & B_3 \equiv A_4 \equiv 0 \bmod 32, \\ B_4 \equiv A_5 \equiv 0 \bmod 32, & D_4 \equiv C_5 \equiv 0 \bmod 32, \\ B_5 \equiv A_6 \equiv 0 \bmod 32, & B_6 \equiv A_7 \equiv 0 \bmod 32. \end{array}$$

Using the fact that the rotation amounts are 0, we get the following two equations from rounds three and four and rounds four and five.

$$B_4 = (C_3 \oplus f(D_3)) + S[7], \tag{2.4}$$
$$B_5 = (C_4 \oplus f(D_4)) + S[9]. \tag{2.5}$$

Since $B_4 \equiv 0 \bmod 32$, $C_3 \equiv 0 \bmod 32$, $B_5 \equiv 0 \bmod 32$ and $D_4 \equiv 0 \bmod 32$, we have $S[7] \equiv -f(D_3) \bmod 32$ and $C_4 \equiv -S[9] \bmod 32$. Since $C_4 = D_3$, we obtain the first condition on subkeys $S[7] \equiv -f(-S[9]) \bmod 32$.

Similarly, looking at the computation from rounds four and five and rounds five and six, we get the following two equations.

$$D_5 = A_4 \oplus f(B_4) + S[8], \tag{2.6}$$
$$B_6 = C_5 \oplus f(D_5) + S[11]. \tag{2.7}$$

Since $A_4 \equiv 0 \bmod 32$, $B_4 \equiv 0 \bmod 32$, $B_6 \equiv 0 \bmod 32$ and $C_5 \equiv 0 \bmod 32$, we have $D_5 \equiv S[8] \bmod 32$ and $S[11] \equiv -f(D_5) \bmod 32$, and so $S[11] \equiv -f(S[8]) \bmod 32$.
□

The subkey dependencies in Lemma 2.16 were obtained using only four equations (those for B_4, B_5, D_5 and B_6). In total, one could write down 12 equations of the form $B_{i+1} = ((C_i \oplus f(D_i)) \lll f(B_i)) + S[2i+1]$ and $D_{i+1} = ((A_i \oplus f(B_i)) \lll f(D_i)) + S[2i]$ for this characteristic. Although there might be dependencies involving other equations, the four given above will be the focus of the rest of this section. Essentially, each equation involves four variables and the aim is to combine equations to obtain two expressions with a single variable. If the two expressions involve the same variable then we can obtain conditions on the subkeys involved.

It is worth noting that given such conditions on the subkeys involved, not only does the characteristic hold, but it does so with a higher probability than the expected value given in Section 2.4.4.

Lemma 2.17 *Assume that the characteristic given in Table 2.19 holds up to round five. Furthermore suppose that* $f(-S[9]) \equiv -S[7] \bmod 32$ *and* $f(S[8]) \equiv -S[11] \bmod 32$. *Then* $B_5 \equiv 0 \bmod 32$ *and* $B_6 \equiv 0 \bmod 32$.

Proof: From Lemma 2.16, we have that $S[7] \equiv -f(D_3) \bmod 32$. This is equivalent to $-S[7] \equiv f(C_4) \bmod 32$. Also, we have that $B_5 \equiv C_4 + S[9] \bmod 32$. So, if $-S[7] \equiv f(-S[9]) \bmod 32$ then $f(C_4) \equiv f(-S[9]) \bmod 32$ which implies that $C_4 \equiv -S[9] \bmod 32$ and so $B_5 \equiv 0 \bmod 32$. A similar argument can be used to show that $B_6 \equiv 0 \bmod 32$. □

Lemma 2.17 shows that when the subkey conditions hold, $B_5 \equiv 0 \bmod 32$ and $B_6 \equiv 0 \bmod 32$. In this case the probability of the characteristic will be $2^{-30} \times 2^5 \times 2^5 = 2^{-20}$, since two of the rotation amounts are always zero. Recall that the estimated probability for the characteristic when averaged over all keys is 2^{-30}. Here we have shown (Lemmas 2.16 and 2.17) that there is some irregularity in the distribution of the probability: For a fraction of 2^{-10} keys the probability is 2^{-20}, and for the rest of the keys the probability is much smaller than 2^{-30}. In fact, this set meets the definition of weak keys. In [19], Knudsen and Meier demonstrated that a similar irregularity of distribution that holds for RC5 can be exploited to improve differential attacks. We would expect the same to apply here. Similar subkey dependencies can be observed for some of the other characteristics for RC6-I-NFR and RC6-NFR. However in some cases the characteristic must be iterated more than once before dependencies exist.

Note that the behavior of the differential associated with some characteristic is typically of more importance in a differential attack. For RC6-I-NFR, while the characteristic displays the irregular behavior already described, the associated differential has been experimentally verified to hold with the expected probability. However the associated differential for RC6-NFR appears to have the same irregular behavior as the characteristic. Why is there this discrepancy? In Section 2.4.4 it is shown how the introduction of the quadratic function helps to reduce the additional effect of differentials. In short, for RC6-I-NFR there are many equally viable paths that match the beginning and end-points of the characteristic. If the characteristic fails to hold because of some choice of subkey values, other characteristics hold instead thereby maintaining the probability of the differential. However, with RC6-NFR we introduce the quadratic function and this typically reduces differentials to being dominated by the action of a single characteristic. Irregular behavior in the characteristic will therefore manifest itself as irregular behavior in the differential.

Differential Characteristics in RC6-I and RC6

Let us now consider the role of the fixed rotation that was omitted in RC6-I-NFR and RC6-NFR. We will find that this single operation removes the kind of subkey dependencies that occurred in these two variants.

We will focus on RC6-I in the analysis for simplicity, and the same arguments also apply to the full RC6. We will need to make some heuristic assumptions to make headway with our analysis. Nevertheless our experimental results confirm that the differential behavior of RC6-I closely matches the behavior described in 2.4.2.

The characteristic which seems to be the most useful for attacking RC6-I is shown again in Table 2.20. We first argue that there are no subkey dependencies of the form we described in Section 2.4.6 for this characteristic and we then broaden our discussion to include other, more general, characteristics.

At this stage we need some new notation: the exponent n will be used to denote when some quantity has been rotated to the left by n bit positions. For example, $D_2^5 \equiv 15 \bmod 32$ means that when D_2 is rotated five bits to the left, then the decimal

i	A_i	B_i	C_i	D_i
1	e_{16}	e_{11}	0	0
		$\downarrow$		
2	e_{11}	0	0	0
		$\downarrow$		
3	0	0	0	e_{26}
		$\downarrow$		
4	0	e_{26}	e_{26}	0
		$\downarrow$		
5	e_{26}	e_{21}	0	e_{26}
		$\downarrow$		
6	e_{21}	e_{16}	e_{26}	0
		$\downarrow$		
7	e_{16}	e_{11}	0	0

Table 2.20: A useful characteristic for RC6-I.

value of the least significant five bits is 15. Of course, this is the same as saying that the most significant five bits of D_2 take the value 15. For simplicity, we will assume that $(X+Y)^j = X^j + Y^j$ where j denotes a rotation amount. This is actually true if and only if there is no carry-out when adding the top j bits and no carry-out when adding the bottom $32-j$ bits. However, for the sake of our analysis, we make this assumption, since it should facilitate the construction of any potential subkey dependencies.

Following the arguments in Lemma 2.16, for the characteristic in Table 2.20 to hold the following rotation amounts must take the values indicated:

$$\begin{array}{ll} D_2^5 \equiv C_3^5 \equiv 15 \bmod 32, & B_3^5 \equiv A_4^5 \equiv 27 \bmod 32, \\ B_4^5 \equiv A_5^5 \equiv 27 \bmod 32, & D_4^5 \equiv C_5^5 \equiv 27 \bmod 32, \\ B_5^5 \equiv A_6^5 \equiv 17 \bmod 32, & B_6^5 \equiv A_7^5 \equiv 17 \bmod 32. \end{array}$$

We wish to write down four equations similar to Equations (2.4), (2.5), (2.6) and (2.7) which cause subkey dependencies in RC6-NFR. From round three to four, the difference e_{26} is copied from register D_3, is changed to e_{31} by the action of the fixed rotation, and then exclusive-or'ed into the C strand. For it to become the e_{26} that appears in B_4, the data dependent rotation B_3^5 must have the value 27. Hence, we must have $B_3^5 \equiv 27 \bmod 32$ and $B_4 = (C_3 \oplus D_3^5)^{27} + S[7] = C_3^{27} \oplus D_3 + S[7]$. In a similar way other equations can be derived:

$$B_4 = C_3^{27} \oplus D_3 + S[7], \tag{2.8}$$
$$B_5 = C_4^{27} \oplus D_4 + S[9], \tag{2.9}$$
$$D_5 = A_4^{27} \oplus B_4 + S[8], \tag{2.10}$$
$$B_6 = C_5^{17} \oplus D_5^{22} + S[11]. \tag{2.11}$$

i	A_i	B_i	C_i	D_i
1	e_{t+5}	e_t	0	0
		$\downarrow$		
2	e_t	0	0	0
		$\downarrow$		
3	0	0	0	e_s
		$\downarrow$		
4	0	e_u	e_s	0
		$\downarrow$		
5	e_u	e_{u-5}	0	e_v
		$\downarrow$		
6	e_{u-5}	e_{u-10}	e_v	0
		$\downarrow$		
7	e_{u-10}	e_{u-15}	0	0

Table 2.21: A generalized characteristic for RC6-I.

In Lemma 2.16 we observed a subkey dependency by combining the analogous equations to (2.8) and (2.9), and another dependency from combining the analogous equations to (2.10) and (2.11). In the case of RC6-I we can demonstrate that neither approach now works.

We first consider Equations (2.8) and (2.9). For Equation (2.9) we know that the values of $B_5^5 \bmod 32$, $D_4^5 \bmod 32$, and $S[9]^5 \bmod 32$ are fixed. This implies a condition on the least significant five bits of C_4. Since C_4 is the same as D_3, we have a condition on $D_3 \bmod 32$. We now have conditions on all the registers in Equation (2.8), namely, $B_4^5 \bmod 32$, $C_3^5 \bmod 32$, and $D_3 \bmod 32$. However, the bits from different words involved in this equation are from different positions. They do not lead to any constraints on $S[9]$, and there appear to be no subkey dependencies as a result.

Similar arguments also apply to Equations (2.10) and (2.11). One may also try to combine Equations (2.8) and (2.10), since they have the quantity B_4 in common, or Equations (2.9) and (2.11), since they have $C_5 = D_4$ in common. However, these combinations also fail to give any subkey dependencies.

We performed experiments on RC6-I to assess the probability of the characteristics given in Table 2.20. These results confirmed that the distribution of the characteristic probability was as expected, and there was no indication of any subkey dependencies for the characteristic.

More generally, we might consider characteristics of the form given in Table 2.21. The values which we need to fix if the characteristic is going to hold are

$$\begin{aligned} D_2^5 \equiv C_3^5 \equiv s - t \bmod 32, \qquad & B_3^5 \equiv A_4^5 \equiv u - 5 - s \bmod 32, \\ B_4^5 \equiv A_5^5 \equiv u - 5 - s \bmod 32, \qquad & D_4^5 \equiv C_5^5 \equiv v - u - 5 \bmod 32, \\ B_5^5 \equiv A_6^5 \equiv u - 15 - v \bmod 32, \qquad & B_6^5 \equiv A_7^5 \equiv u - 15 - v \bmod 32. \end{aligned}$$

Let $r_1 = u-5-s$, $r_2 = v-u-5$, and $r_3 = u-15-v$. Then the subkey dependencies we observed would be produced by the following equations:

$$\begin{aligned} B_4 &= C_3^{r_1} \oplus D_3^{5+r_1} + S[7], \\ B_5 &= C_4^{r_1} \oplus D_4^{5+r_1} + S[9], \\ D_5 &= A_4^{r_2} \oplus B_4^{5+r_2} + S[8], \\ B_6 &= C_5^{r_3} \oplus D_5^{5+r_3} + S[11]. \end{aligned}$$

Following similar arguments to those presented earlier, it can be verified that there is no choice for r_1, r_2, and r_3 that makes the characteristic depend upon the values of the subkeys. In particular, the most promising values to try are $r_1 = 0$; $r_1 = 27$; $r_3 = 0$ and $r_2 = 22$; and $r_3 = 0$, $r_2 = 27$, and $r_1 = 27$.

The fixed rotation is an important component of RC6. Not only does it help to hinder the construction of good differentials and linear approximations [8], but it helps to disturb the build-up of any inter-round dependencies. Here the fixed rotation ensures that equations can simultaneously hold without forcing any restriction on the values of the quantities involved.

2.5 Diffusion Properties of RC6

Recall, in Section 2.1.2, we discussed Kaliski and Yin's differential cryptanalysis of RC5. At the end of the section, we made the remark about the two improvements to their attack: the work of Knudsen and Meier [19] and the work of Biryukov and Kushilevitz [3]. Both of these attacks rely on the fact that RC5 has a relatively slow avalanche of change from one round to the next, assuming no difference in the data-dependent rotation amounts. To further justify (in addition to what was said in Section 2.4.5) the lack of applicability of these attacks to RC6, we shall analyze the avalanche properties of RC6 in this section. Since both the mentioned results used exclusive-or as measure of difference, we will do the same. Thus, we are interested in the distribution of the Hamming weights of output differences given an input difference of a certain form.

Even for a simple operation it can be difficult to fully characterize these probability distributions. We will study the problem by analyzing the *expected* Hamming weight of the output differences. This approach provides good insight into the roles of the different operations in RC6.

Our analysis shows that the quadratic function drastically increases the Hamming weight of the expected difference, especially when the Hamming weight of the input difference is small. This illustrates a nice effect whereby the use of the quadratic function complements that of the data-dependent rotation. As we have mentioned, the data-dependent rotation becomes an effective agent of change only when there is a difference in the rotation amount. With a small Hamming weight difference, it is less likely that non-zero difference bits appear in positions that affect a rotation amount. However, the quadratic function helps to drastically increase the avalanche of change so that the full benefit of the data-dependent rotations can be

gained as soon as possible.

2.5.1 Definitions and Assumptions

We will be interested in diffusion properties of RC6 with word size $w = 32$, but much of our analysis is general enough for arbitrary w. We only consider exclusive-or as measure of difference. Let $X' = X_1 \oplus X_2$, $Y' = Y_1 \oplus Y_2$, and $Z' = Z_1 \oplus Z_2$ and let x, y, z denote the Hamming weight of the differences X', Y', Z', respectively.

Consider the following two conditions that may be imposed on some difference that has Hamming weight x:

A: There is a single block of consecutive 1-bits of length x, and the block is distributed randomly at some position in the input difference. We do not assume words wrap-around.

B: There are $t > 1$ blocks of consecutive 1-bits of length $x_1, x_2, ..., x_t$ such that $x_1 + x_2 + \cdots + x_t = x$. In addition, each block is distributed randomly across the input difference.

Condition B is actually a good characterization for the differences in the intermediate rounds of RC6 and its variants. In each round (of RC6 or its variants) any difference in the A and C strands are rotated by a random amount due to the data-dependent rotations. Hence each block of 1-bits within the differences is distributed randomly. Condition A is a special case of Condition B. In the next two sections when we examine the diffusive properties of individual operations, we will first consider the special case Condition A and then generalize the results to Condition B.

2.5.2 Diffusive Properties of the Basic Operations

Here we analyze the basic operations of exclusive-or, addition, and rotation. The more complicated quadratic function will be considered in the next section.

Lemma 2.18 (exclusive-or) *For $i = 1, 2$ let $Z_i = X_i \oplus Y_i$. If X' and Y' satisfy Condition A, then the expected value of z is bounded by $E(z) \leq x + y - \frac{2xy}{w}$.*

Proof: The number of possible positions for each block of 1-bits is $w - x + 1$ and $w - y + 1$ respectively. We upperbound these values by w. Since the blocks of 1-bits are both distributed randomly, a 1-bit in X' has probability of $\geq \frac{1}{w}$ of overlapping with a 1-bit in Y'. So the expected number of bits that overlap is $\geq \frac{xy}{w}$. Since each overlap cancels two bits, the expected Hamming weight of the output is $\leq x + y - \frac{2xy}{w}$. □

Corollary 2.19 (exclusive-or) *For $i = 1, 2$ let $Z_i = X_i \oplus Y_i$. If X' and Y' satisfy Condition B then $E(z) \leq x + y - \frac{2xy}{w}$.*

Proof: The same proof from Lemma 2.18 applies here. □

Lemma 2.20 (addition) *For $i = 1, 2$ let $Z_i = X_i + S$, where S is the subkey. If X' satisfies Condition A then averaging over all possible X_1, X_2, and S one has that $E(z) = c + \frac{x+1}{2}$ where $c \in [0, 1]$ and depends on X'.*

Proof: First consider the special case where $x = w$, that is, X_1 and X_2 differ in all bits. We first prove that when averaging over all possible X_1, S,

$$prob(X_1 + S < 2^w \text{ and } X_2 + S \geq 2^w) = \frac{1}{4}. \tag{2.12}$$

Given any $X_1 \in \{0,1\}^w$, we define

$$d(X_1) = |\{S : S \in \{0,1\}^w, \text{s.t. } X_1 + S < 2^w \text{ and } X_2 + S \geq 2^w\}|.$$

If $X_1 \geq 2^{w-1}$, then there are no solutions ($d(X_1) = 0$), so we only need to consider the case $X_1 < 2^{w-1}$. Since $X_2 = 2^w - X_1 - 1$ (the 1's complement), we see that the valid solutions for S are $X_1 + 1 \leq S < 2^w - X_1$. Hence $d(X_1) = 2^w - 1 - 2X_1$, and

$$\begin{aligned} prob(X_1 + S < 2^w \text{ and } X_2 + S \geq 2^w) &= \frac{\sum_{X_1=0}^{2^{w-1}-1} d(X_1)}{2^w \times 2^w} \\ = \frac{\sum_{X_1=0}^{2^{w-1}-1} 2^w - 1 - 2X_1}{2^{2w}} &= \frac{2^{w-1}(2^w - 1) - (2^{w-1} - 1)(2^{w-1})}{2^{2w}} \\ &= \frac{1}{4}. \end{aligned}$$

Note that Equation 2.12 holds for any value of $w > 0$. So we can consider the least significant j bits of X_1, X_2, S. More precisely, for $1 \leq j \leq w$, define $X_1(j) = X_1 \bmod 2^j, X_2(j) = X_2 \bmod 2^j, S(j) = S \bmod 2^j$. Then,

$$prob(X_1(j) + S(j) < 2^j \text{ and } X_2(j) + S(j) \geq 2^j) = \frac{1}{4}. \tag{2.13}$$

By symmetry,

$$prob(X_1(j) + S(j) \geq 2^j \text{ and } X_2(j) + S(j) < 2^j) = \frac{1}{4}. \tag{2.14}$$

From Equations 2.13 and 2.14, we know that with probability $\frac{1}{2}$, exactly one of the two addition operations ($X_1 + S$ and $X_2 + S$) produces a carry into bit j. Z_1 and Z_2 will be the same in bit j if and only if this carry happens. Therefore, with probability $\frac{1}{2}$, the j^{th} bit ($j \geq 1$) of $Z' = Z_1 \oplus Z_2$ is 1. Since bit 0 of Z' is always 1, the expected Hamming weight of Z' is $\frac{w-1}{2} + 1 = \frac{w+1}{2} = c + \frac{w+1}{2}$ for $c = 0$. We have proved the Lemma for the special case where $x = w$.

For the general case where $1 \leq x \leq w$, we can apply the same type of argument. Let v be the index of the most significant 1 in X'. So X_1 and X_2 are the same

in bits $v+1$ through $w-1$. There are $\frac{x+1}{2}$ expected difference bits in Z' when considering up to bit index v. With probability $\frac{1}{2}$, a carry will propagate into the $v+1^{\text{st}}$ bit of exactly one of the words. This carry could propagate further, resulting in more difference bits. The expected number of difference bits coming from the $v+1^{\text{st}}$ to $w-1^{\text{st}}$ indices is $\sum_{i=1}^{w-v-1}(\frac{1}{2})^i$. Since this is a number c between 0 and 1, the expected Hamming weight of the output difference is $c+\frac{x+1}{2}$. □

Corollary 2.21 (addition) *For $i=1,2$ let $Z_i = X_i + S$, where S is the subkey. Suppose that X' satisfies Condition B and there are t blocks of 1's in X'. Then, averaging over all possible keys S, $E(z) \leq t + \frac{x+t}{2}$.*

Proof: Follows from Lemma 2.20. □

The fixed rotation $Z = X \lll \lg w$ always preserves the Hamming weight of the input difference in the output difference. For data-dependent rotations, it is straightforward to see that provided the input difference does not affect the rotation amount, then the Hamming weight of the difference is preserved. We can state this simple fact in the following lemma.

Lemma 2.22 (data-dependent rotation) *For $i=1,2$ let $Z_i = X_i \lll Y_i$. If $Y' \equiv 0 \bmod w$, then $z = x$.*

□

Recall that the analysis of the case where $Y' \not\equiv 0 \bmod w$ was treated in Section 2.3, where it was shown that except in very rare special cases, the Hamming weight is increased by a large amount. The probability of this occurring is given by the following lemma.

Lemma 2.23 *Let $y = w_H(Y')$ and let p be the probability that $Y' \not\equiv 0 \bmod w$. If Y' satisfies Condition A, then $p = \min\left(\frac{y+\lg w-1}{w}, 1\right)$.*

□

For the more general case when Y' satisfies Condition B, it is not so simple to derive a precise formula similar to the one given above. However, it is clearly the case that the higher the Hamming weight of Y', the larger the probability that some part of the non-zero input difference will have an effect on the rotation amount.

2.5.3 Diffusion Properties of the Quadratic Function

We begin this section with a lemma that gives some analytical justification for the use of the higher order bits of the quadratic function as a rotation amount in the full RC6. It tells us that changing a single bit of an input will likely change at least one of the high order bits. Thus, the quadratic function provides good avalanche properties.

Lemma 2.24 *Suppose input X_1 is chosen uniformly at random from $\{0,1\}^{32}$. Let $g_{i,j}$ denote the probability that flipping bit i of X_1 will flip bit j of $Y_1 = f(X_1)$. Then*

$$g_{i,j} = \begin{cases} 0 & \text{for } j < i, \\ 1 & \text{for } j = i, \\ 1 & \text{for } j = 1 \text{ and } i = 0, \text{ and} \end{cases}$$
$$g_{i,j} \in [1/4, 3/4] \text{ for } j > i \geq 1 \text{ or } j \geq 2 \text{ and } i = 0.$$

For the last case, $g_{i,j}$ is close to $3/4$ if $j = 2i + 2$, and for most of the other i, j pairs, $g_{i,j}$ is close to $1/2$.

Proof: Let $X_2 = X_1 \oplus 2^i$ and $Y_2 = f(X_2)$. Without loss of generality, we may assume $X_1[i] = 0$ so that $\delta_X = 2^i$. By Lemma 2.10, we have:

$$\delta_Y = Y_2 - Y_1 = 2^{i+2}X_1 + 2^i + 2^{2i+1} \bmod 2^{32} \ . \tag{2.15}$$

Case $j < i$: By Equation 2.15, it is clear that bits of δ_Y below index i are all zero. Therefore bits of $f(X_2)$ below index i are the same as those in $f(X_1)$.

Case $j = i$: By Equation 2.15, we see that bit i of δ_Y is one and all lower order bits are zero. So $Y_2[i]$ will necessarily be the opposite of $Y_1[i]$.

Case $j = 1$ and $i = 0$: By Equation 2.15, $\delta_Y = 4X_1 + 1 + 2$. Since $X_1[0] = 0$, $Y_1[0] = 0$ which implies that there is no carry from bit 0 into bit index 1 in the computation of $Y_2 = Y_1 + \delta_Y$. Thus, $Y_2[1] = Y_1[1] \oplus 1$.

Case $j > i \geq 1$ or $j \geq 2, i = 0$: An exhaustive computation of the probabilities reveals that $g_{i,j}$ always ranges between $\frac{1}{4}$ and $\frac{3}{4}$, but most of the probabilities are very close to $\frac{1}{2}$, especially when $i \geq 16$. □

The cases $j > i \geq 1$ and $j \geq 2, i = 0$ are of course unsatisfying due to not having a terse, mathematical proof. We therefore provide the following heuristic argument as further justification:

Heuristic 2.25 *Suppose input X_1 is chosen uniformly at random from $\{0,1\}^{32}$. Let $g_{i,j}$ denote the probability that flipping bit i of X_1 will flip bit j of $Y_1 = f(X_1)$. Then*

$$g_{i,j} \in [1/4, 3/4] \text{ for } j > i \geq 1 \text{ or } j \geq 2 \text{ and } i = 0.$$

Proof: By Equation 2.15, we see that

$$\delta_Y \approx 2^{i+2}X_1 + 2^{2i+1} \bmod 2^{32} \tag{2.16}$$
$$\approx 2^{i+2}X_1 \bmod 2^{32} \ . \tag{2.17}$$

We first consider the case where $j = i + 1$. By Equation 2.15, bit $j + 1$ of δ_Y is always zero and bit j is always one. When computing $Y_2 = Y_1 + \delta_Y$, there is a carry coming into bit $i + 1$ with probability $\approx \frac{1}{2}$ (whenever $Y_1[i] = 1$), implying that bit i of Y_2 will be different than bit i of Y_1 with probability $\approx \frac{1}{2}$.

Next consider $j = 2i + 2$. We analyze the carry effect when computing $Y_2 = Y_1 + \delta_Y$ where δ_Y is given by approximation 2.16. Since $X_1[i] = 0$, bit $2i + 2$ of $2^{i+2}X_1$ is zero. Thus, $Y_2[2i + 2] = Y_1[2i + 2]$ only if the incoming carry bit is zero. However, the incoming carry bit is one when $X_1[i - 1] = 1$ due to the 2^{2i+1} in approximation 2.16, and when $X_1[i - 1] = 0$, the incoming carry bit is one with probability $\approx \frac{1}{2}$. Thus, we get $g_{i,2i+2} \approx \frac{3}{4}$.

Finally, consider the case where $j \geq i + 2$ and $j \neq 2i + 2$. Using approximation 2.17, we see that $\delta_Y[j]$ is usually the same as $X_1[j - (i + 2)]$. The exceptional cases are when there is a carry coming in due to the 2^i and 2^{2i+1} terms in Equation 2.15. However, bits at indices i and $2i + 1$ are at least 2 positions away from the index j, so this carry effect is typically with probability $\leq \frac{1}{4}$. Thus, the probability is approximately $\frac{1}{2}$ that $Y_2[j]$ will be the same as $Y_1[j]$, where the approximation becomes more accurate the farther j is from indices i and $2i + 1$. □

Experimental evidence suggests a similar result to Lemma 2.24 holds true for the more general case where multiple input bits are flipped. If we let j be the index of the least significant 1-bit of X', we clearly will have no difference in Y' for bits 0 through $j - 1$, and a difference for bit j. For most of the other bits, we expect to have a difference with probability close to $\frac{1}{2}$. Since we are unable to prove this or complete an exhaustive search, we state it as an assumption:

Assumption 2.26 *For $i = 1, 2$ let $Y_i = f(X_i)$. Suppose X' satisfies Condition B and let j be the bit position of its least significant 1. Then most bits from positions $j + 1$ through $w - 1$ of Y' are 1 with probability close to $\frac{1}{2}$.*

We now get the following, perhaps surprising, result.

Lemma 2.27 (quadratic function) *For $i = 1, 2$ let $Y_i = f(X_i)$. Let $x = w_H(X')$ and $y = w_H(Y')$. If X' satisfies Condition B then $E(y) \approx 1 + \frac{x+w-2}{4}$.*

Proof: Let i be the index of the least significant 1 in X'. For a fixed i, the expected value of y is roughly $1 + (w - 1 - i)/2$ according to Assumption 2.26. If X' satisfies Condition A then i is uniformly distributed between 0 and $(w - x)$. Hence,

$$E(y) \approx \frac{1}{(w - x) + 1} \sum_{i=0}^{w-x} \left(1 + \frac{w - 1 - i}{2}\right) = 1 + \frac{x + w - 2}{4}.$$

□

Lemma 2.27 shows that even when the difference in some input to the quadratic function has Hamming weight 1, the average Hamming weight of the difference in the output is 8.75. This is a very important result. All the other basic operations in RC6, as well as those used in RC5, generally provide little or no additional change to the output difference when the Hamming weight of the input difference is very low. But Lemma 2.27 assures us that in this case, the expected change due to the quadratic function is quite large. Thus, the quadratic function provides excellent avalanche properties, and low Hamming weight characteristics seem difficult to attain.

We can illustrate this benefit in the following way. We experimentally measure the probability that the rotation amounts[3] at the end of a given number of rounds are unaffected by a single bit change in the first word of the input to the cipher. We consider rotation amounts in this exercise because current differential-style attacks on RC5 and RC6 require any difference propagating through the cipher to leave the rotation amounts unchanged. We use "-" to indicate that experimentally the probability is approximately (2^{-20}), which is indistinguishable from random noise.

Rounds	RC6-I-NFR	RC6-I	RC6-NFR	RC6
2	$2^{-0.54}$	$2^{-0.64}$	$2^{-1.32}$	$2^{-10.27}$
4	$2^{-2.15}$	$2^{-2.45}$	$2^{-6.27}$	-
6	$2^{-6.14}$	$2^{-7.04}$	$2^{-14.30}$	-
8	$2^{-12.76}$	$2^{-14.97}$	-	-
10	$2^{-19.07}$	-	-	-

For an increased number of rounds, the probability of unchanged rotation amounts gives a good illustration of the relative diffusive effect of RC6 and its weakened variants. It also illustrates the role of the quadratic function in the security of RC6.

2.5.4 Concluding Remarks

Basic differential-style attacks attempt to predict and control the change from one round to the next during encryption [17]. Improved attacks on RC5 [3, 19] do not attempt to predict the difference quite so closely. Instead, they rely on the relatively slow diffusive effect of RC5 to ensure that any change propagating through the cipher remains manageable and to some extent predictable. Even though single-bit starting differences might be used, differentials with an ending difference of Hamming weight 15, for example, can still be useful [3, 19].

The quadratic function was added to RC6 to address this particular shortcoming of RC5. In this section we have shown that the quadratic function tends to have a much higher diffusion property than the other operations, particularly when the input difference has small Hamming weight. This makes it quite difficult to find differentials following some controlled pattern with non-negligible probability. In general, we expect that the quadratic function will strongly hinder differential and other attacks that rely on a modest avalanche of change from one round to the next.

[3]By "rotation amounts" we mean the low five bits of the registers for RC6-I-NFR and RC6-NFR, the high five bits of the registers for RC6-I, and the high five bits of the output of $f(x)$ for RC6.

Chapter 3

Cryptanalysis of the SecurID Hash Function

The SecurID, developed by RSA Security, is a hardware token used for strengthening authentication when logging in to remote systems, since passwords by themselves tend to be easily guessable and subject to dictionary attacks. The SecurID adds an "extra factor" of authentication: one must not only prove themselves by getting their password correct, but also by demonstrating that they have the SecurID token assigned to them. The latter is done by entering the 6- or 8-digit code that is being displayed on the token at the time of login.

Each token has within it a 64-bit secret key and an internal clock. Every minute, or every half-minute in some tokens, the secret key and the current time are sent through a cryptographic hash function. The output of the hash function determines the next two authenticator codes, which are displayed on the LCD screen display. The secret key is also held within the "ACE/server", so that the same authenticator can independently be computed and verified at the remote end.

If ever a user loses their token, they must report it so that the current token can be deactivated and replaced with a new one. Thus, the user bears some responsibility in maintaining the security of the system. On the other hand, if the user were to temporarily leave his token in a place where it could be observed by others and then later recover it, then it should not be the case that the security of the device could be entirely breached, assuming the device is well-designed.

The scenario just described was considered in a recent publication by Biryukov, Lano, and Preneel [5], where they showed that the hash function that is alleged to be used by SecurID [40] (ASHF) has weak properties that could allow one to find the key much faster than exhaustive search. The attack they describe involves recording many outputs of the SecurID using a PC camera with OCR software, and then later searching the outputs for indication of a *vanishing differential* – two closely related input times that collide in the function, resulting in the same output hash. If one is discovered, the attacker then has a good chance of finding the internal secret key using a search algorithm that they estimated to be equivalent to 2^{48} hash function

operations. On a 2.4 GHz PC, 2^{48} hash operations take about 111 years[1]. It would require over 1300 of these PC's to find the key in a month.

In this chapter, we go through a deeper analysis of the [5] algorithm, giving further justification of their conjectured running time of 2^{48}. We then present three techniques to speed up the filtering step, which is the bottleneck of their attack. Our theoretical analysis and implementation experiments show that the time complexity can be reduced to about 2^{45} hash operations when using only a single vanishing differential. When the vanishing differential involves $\geq$ 4-bits, which happens about one third of the time, it appears that the running time should be faster.

We also investigate into the use of extra information that an attacker would ordinarily have, in order to speed up the attack further. This information consists of either multiple vanishing differentials, or knowledge that no other vanishing differentials occur in a nearby time period of the observed one. In either case, the running time can be reduced significantly. From our analysis we expect that after a vanishing differential is observed, the attacker would nearly always be able to perform the key search algorithm in 2^{40} hash operations or less. On a typical PC, this can be done in about 5 months, making the computing power requirements for the search attainable by almost any individual.

The success probability of all attacks (including [5]) depends upon how long the attacker must wait for a vanishing differential to occur – the longer the period is, the higher the chance that a token will have a vanishing differential. For example, simulations have shown that in any one-week period, about 1% of the SecurID cards will have a vanishing differential; in any one-year period, about 35% of the tokens will have a vanishing differential.

We should note, however, that the longer the device is out of a user's control, the more likely that the user will recognize it and have it deactivated. So, we consider two realistic scenarios in which the token could be compromised. In the first scenario, a user may be on vacation for one week and left his token behind in a place where others could observe it, in which case there is a 1% chance that a collision would happen. This probability is small, but definitely *non-negligible*, especially considering that a single large corporation may have many thousands of SecurID users. In the second scenario, the success is much more likely. Since the cost of SecurID tokens is very high, tokens are often reassigned to new users when a previous owner leaves a company [41]. This is a very bad idea, since the original user would have a high chance of being able to find the internal key, assuming he recorded many of the outputs while it was in his possession. See [5] for a diagram showing how the probability of a vanishing differential increases with time. In light of our new results, the token reassignment scenario becomes a serious risk.

[1] Requires some optimisations to the code from [40], such as re-ordering bytes to eliminate bswaps.

3.1 The SecurID Hash Function

We provide a high level description of the alleged SecurID hash function, following the same notation as in [5] wherever possible. More detailed descriptions can be found in [5, 40].

The function can be modeled as a keyed hash function $y = H(k, t)$, where k is a 64-bit secret key stored on the SecurID token, t is a 24-bit time obtained from the clock every time unit of 30 or 60 seconds, and y is two 6- or 8-digit codes. The two codes are output in consecutive time units. The function consists of the following steps:

- an expansion function that expands t into a 64-bit "plaintext",
- an initial key-dependent permutation,
- four key-dependent rounds, each of which has 64 subrounds,
- an exclusive-or of the output of each round onto the key (thus the four rounds each differ according to the changing of the key),
- a final key-dependent permutation (same algorithm as the initial one), and
- a key-dependent conversion from hexadecimal to decimal.

A diagram of the hash function is shown in Figure 3.1.

Throughout the chapter, we use the following notation to represent bits, nibbles, and bytes in a word: a 64-bit word b, consisting of bytes $B_0, ..., B_7$, nibbles $\mathtt{B}_0, ..., \mathtt{B}_{15}$, and bits $b_0 b_1 ... b_{63}$. The nibble $\mathtt{B}_0$ corresponds to the most significant nibble of byte 0 and the bit b_0 corresponds to the most significant bit. The other values are as one would expect.

The known weaknesses of the hash function only depend upon the time expansion, key-dependent permutation, and the key-dependent rounds. In the next three sections, we will describe them in more detail.

3.1.1 Time Expansion

The time t is a 24-bit number representing twice the number of minutes since January 1, 1986 GMT. So the least significant bit is always 0, and if the token outputs codes every minute, then the expansion function will clear the 2nd least significant bit as well. Let the result be represented by the bytes $T_0 T_1 T_2$ where T_0 is the most significant. The expansion is of the form $T_0 T_1 T_2 T_2 T_0 T_1 T_2 T_2$. Note that the least significant byte is replicated 4 times, and the other two bytes are replicated 2 times each.

3.1.2 Key-Dependent Permutation

The C code given by Wiener [40] (obtained by reverse-engineering the ACE/server code) of the key-dependent permutation is quite cryptic. It requires very careful

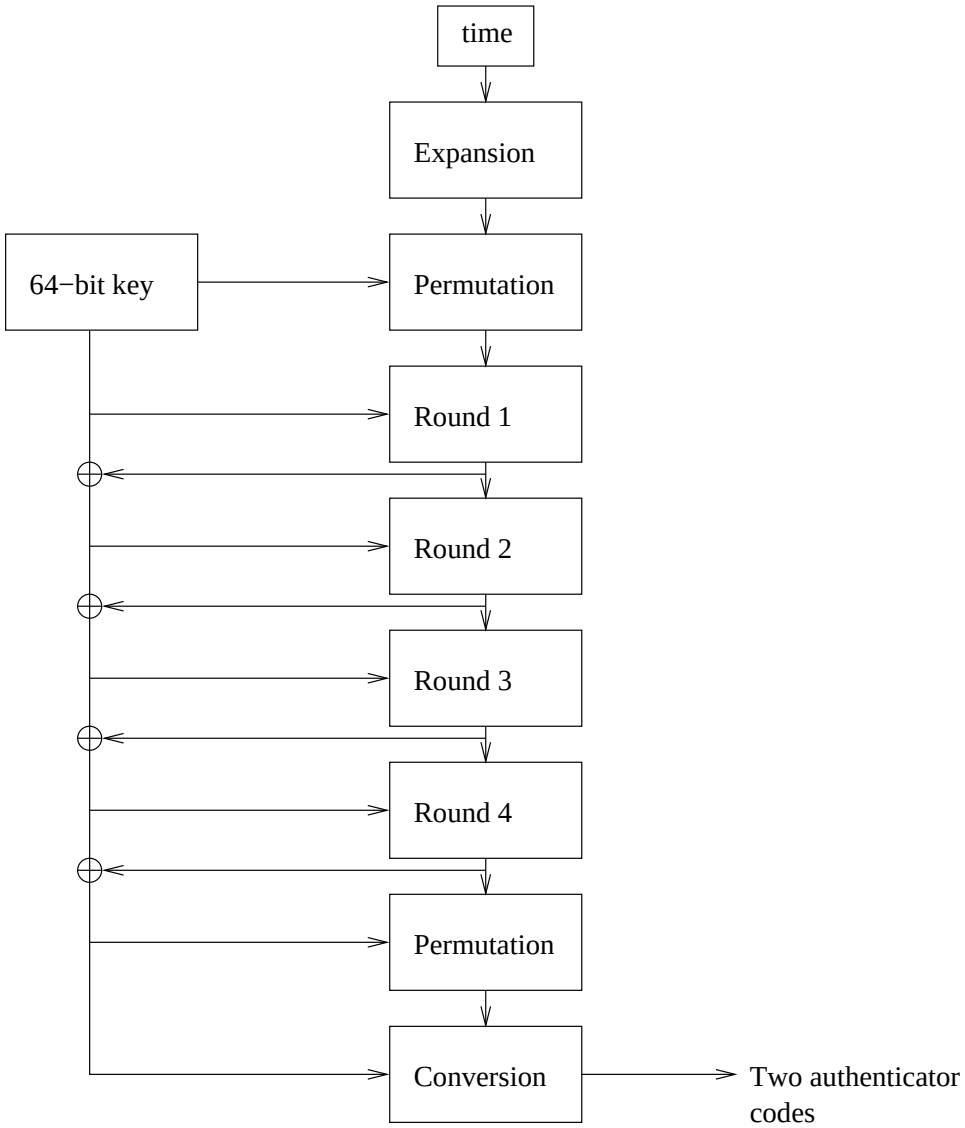

Figure 3.1: Diagram of the SecurID hash function.

study to see that it is really doing a permutation. In [7], Contini gave a different, more insightful description that produced an equivalent output. This description was one that was easy to understand if done with pencil and paper, but awkward to implement. Later, A.K. Lenstra came up with another description which is more natural for implemention purposes and uses no pointers. Here we include both the Contini and the Lenstra descriptions, since each has its merits.

Contini Description

The key-dependent permutation uses the key nibbles $\texttt{K}_0 \ldots \texttt{K}_{15}$ in order to select bits of the data for output into a permuted_data array. The data bits will be taken 4 at a time, copied to the permuted_data array from right to left (i.e. higher indexes are filled in first), and then removed from the original data array. Every time 4-bits are removed from the original data array, the size shrinks by 4. Indexes within that array are always modulo the number of bits remaining.

A pointer m is first initialized to the index $\texttt{K}_0$. The first 4-bits that are taken are those right before the index of m. For example, if $\texttt{K}_0$ is `0x2`, then bits 62, 63, 0, and 1 are taken. As these bits are removed from the array, the index m is adjusted accordingly so that it continues to point at the same bit it pointed to before the 4-bits were removed. The pointer m is then increased by a value of $\texttt{K}_1$, and the 4-bits prior to this are taken, as before. The process is repeated until all bits have been taken.

A small, 16-bit example may be helpful. Suppose we have the binary data `0110101110101001` and the hexadecimal key of `0x5ad3`. Processing the first key nibble, we move the pointer m up 5 places: `01101`$\boxed{\texttt{0}}$`1110101001`. Then, the 4-bits `1101` are moved to the permuted_data array, and the original data array becomes `0`$\boxed{\texttt{0}}$`1110101001`. Next, we move the pointer up 10 places (= `0xa`): `00111010100`$\boxed{\texttt{1}}$. Removing the 4-prior bits, we now have the permuted_data array containing `01001101` and the original array containing `0011101`$\boxed{\texttt{1}}$. After completing the last two key nibbles, the permuted_data array ends up as `1101001101001101`.

Lenstra Description

In the Lenstra description, we use a single 64-bit array. The array is divided into two parts: the left hand side which has the original data, and the right hand side which will contain the permuted_data. Initially, all 64-bits are considered to be on the left hand side. After each iteration, the left hand side shrinks by 4-bits and permuted_data grows by 4-bits.

In the first iteration, we rotate left the left hand side of the array by $\texttt{K}_0$ positions. The 4-bits that end up at the end (the far right hand side) then become the first nibble of the permuted_data array, while the other 60-bits remain on the left hand side. In the second iteration, the 60-bits on the left hand side are left rotated by $\texttt{K}_1$ positions. Again, the 4-bits at the end become the next nibble of permuted_data and the other 56-bits remain on the left hand side. The process continues in this way until all sixteen key nibbles are used.

We illustrate with the small, 16-bit example given above. The original binary data is `0110101110101001` and the key is `0x5ad3`. Left rotating the array by the first key nibble of 5, we get `011101010010|1101` where the '|' symbol divides the left hand side from the permuted_data. After processing the second key nibble, we have: `10011101|01001101`. In the third and fourth steps we get `1011|001101001101` and `1101001101001101` respectively.

3.1.3 Key-Dependent Rounds

Each of the four key-dependent rounds takes as inputs a 64-bit key k and a 64-bit value b^0, and outputs a 64-bit value b^{64}. The key k is then exclusive-ored with the output b^{64} to produce the new key to be used in the next round.

One round consists of 64 subrounds. For $i = 1, ..., 64$, subround i transforms b^{i-1} into b^i using a single key bit k_{i-1}. The subround involves a function f which does one of two different operations according to whether the key k_{i-1} is equal to b_0^{i-1}. See Figure 3.2. The exact details of these operations are not so important for our results, with the exception of two properties:

1. The function f involves only bytes B_0 and B_4 of b^i. These two bytes are modified, while the other six bytes remain the same.

2. The way f is used causes the hash function to have easy-to-find collisions after a small number of subrounds within the *first* round.

At the end of each subround, all the bits are rotated one bit position to the left. So, up to subround $N < 25$ of the first round, only $2N + 14$ data bits have been involved in the computation, as shown in Figure 3.3. This property is used in the Biryukov, Lano, and Preneel attack.

Subround Function

For completeness, we include a description of the actual f function. This explanation is only here for the curious reader: we shall not be concerned with the specifics later.

As mentioned above, the f computation depends upon whether key bit k_{i-1} is the same or not as data bit b_0^{i-1}. If it is the same, then f does:

$$\begin{aligned} B_0^i &= ((((B_0^{i-1} \ggg 1) - 1) \ggg 1) - 1) \oplus B_4^{i-1}, \\ B_4^i &= B_4^{i-1} \ . \end{aligned}$$

Otherwise, the computation is:

$$\begin{aligned} B_0^i &= B_4^{i-1}, \\ B_4^i &= 100 - B_0^{i-1} \ . \end{aligned}$$

Note that the "100" is given in decimal. See [5] for information on how collisions occur from f. We also have an abbreviated explanation in the Appendix.

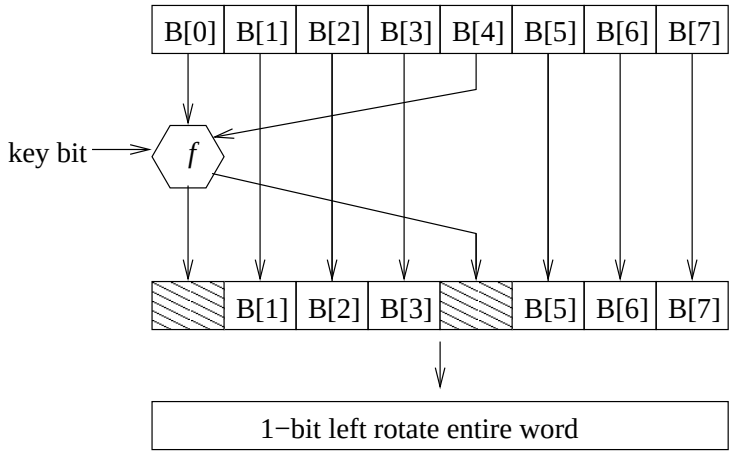

Figure 3.2: Diagram of the subround function. The function f does one of two operations, depending upon whether the key bit is equal to bit b_0. Only bytes B_0 and B_4 are modified in the subround function.

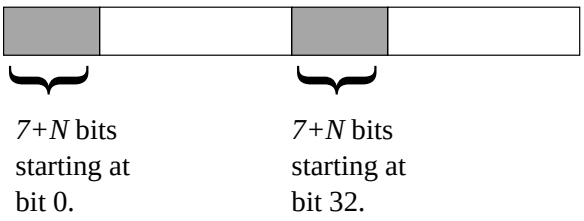

Figure 3.3: For subround $N < 25$ of the first round, only $2N + 14$ data bits are involved in the computation.

3.2 Vanishing Differentials

A vanishing differential is a collision that comes from two closely related inputs. In the case of SecurID, these collisions happen from two 24-bit times t and t' that differ only in a few bits (most often 1-bit) in the most significant two bytes, according to simulation experiments.

Since the output from the hash function is two 6- or 8-digit codes, a necessary condition for a collision is that a pair of two consecutive output codes must match another pair. But how do we distinguish a real collision from a fake one? In other words, since we are not given the full 64-bit output, it may be possible that a pair of consecutive output codes match another pair, but there was not a real collision.

Let us consider the probability of this happening over one year's output from a SecurID that outputs 6-digit codes every minute. For simplicity, we will restrict to vanishing differentials that occur from a 1-bit time difference. If this time difference is in the least significant byte, we consider the collision to be a fake since no such collision has ever been witnessed after numerous simulations.

In one year of data, about 2^{18} hashes are computed which can be grouped according to their least significant byte T_2. Since the least significant 2-bits of T_2 are 0, there are 2^6 such groups each containing 2^{12} elements. Among the 2^{12} elements of a group, there are $12 \times 2^{12}/2 \approx 2^{14.6}$ pairs that differ in exactly 1-bit. Since a pair of output codes is approximately 40-bits of information, the probability of a collision amongst a group is $\leq p := \binom{2^{14.6}}{2}/2^{40} \approx 2^{-11.8}$. Looking at all 2^6 groups, we see that the probability of a fake collision of this form is $\leq 1 - (1-p)^{64} \approx 0.017$. This is significantly lower than our observed probabilities of real collisions (see chapter introduction), so we see that almost always, matching consecutive pairs of outputs of the required form are real collisions.

3.3 A First Attack on the Hash Function

Before describing the Biryukov, Lano, and Preneel attack, we shall first describe a weakness that we found independently of [5], and publicized on the sci.crypt newsgroup [7]. The attack takes advantage of "redundancy" in the key-dependent permutation. Our attack was being developed at the same time as the attack of Biryukov, Lano, and Preneel, but their result turned out to be better than the idea presented here. Regardless, the two ideas are combined to some extent in [5].

Let t and t' be two input times that differ in a single bit. After the expansion and permutation, they have become two 64-bit words b and b' that differ in either two or four bits. For simplicity, assume 2-bit differences. We refer to the bits that differ between the two words as the *difference bits*. The difference bits will be rotated left from one subround to the next until at least one of them gets rotated into either byte B_0 or byte B_4. When this happens, usually the difference will propagate, causing two very different words. However, sometimes it will be the case that all difference bits are in B_0 and/or B_4, and the difference bits cancel by the f function. Thus,

we have a collision (vanishing differential). If this collision has happened in the first round, then it will result in identitical pairs of authenticators. Conversely, if the pairs of authenticators are the same, it is most likely that there was a collision in the first round [5].

We remark that any collision that happens after the first round does not necessarily produce identical authenticators. This is because if there was a difference at the end of the first round, then it gets mixed into the key (as shown in Figure 3.1). If the keys have a difference, then the round functions evolve differently.

The first observation we make is that almost always the internal collision will happen within 32 subrounds of the first round. If the collision happens after 32 subrounds, this means that each difference bit has passed through either B_0 or B_4 without disappearing. It will then be at least 24 more subrounds before the difference bits return to bytes B_0 and B_4 in order to have another chance to disappear. Note that this second chance for a collision is extremely unlikely to happen unless the differences follow a carefully controlled pattern the first time they enter B_0/B_4 (for example, difference bits ought not to propagate too much or to separate from each other by too much). While a technical analysis may be desirable, we were satisfied with simulations to confirm this heuristic. In 405 occurrences of vanishing differentials from randomly chosen keys, only 11 had the collision happen after the 32nd subround. So it appears that we can say that collisions happen within the first 32 subrounds 97% of the time.

We now illustrate how one can find the key faster than exhaustive search assuming a collision happened within 32 subrounds. The search involves testing keys to see if a vanishing differential occurs within 32 subrounds for a given pair of times t and t'. If it does, we do further testing to see if the key is the right one. Otherwise, we advance to the next candidate. Since we only have to try 32 subrounds, this is eight times faster than the full ASHF. On the other hand, we have to do it for two plaintexts, so each trial is four times faster than the full ASHF.

Note that in 32 subrounds, only the first 32 key bits come into play. We take advantage of this in our attack, which works as follows. Guess the first six bytes of the key. There are 2^{48} possibilities. For each guess, we perform the keyed permutation only for those first six bytes. After this step, we have six bytes in the permuted_data array and two bytes remaining. Although there are 2^{16} possible keys remaining, this is far more than we need to try since many of the resulting permutations of the remaining two bytes of data overlap. Thus, rather than guessing the last two key bytes, we try all possible permutations that could have come from some key. There are at most $16 \times 12 \times 8 \times 4 \approx 2^{12.58}$ permutations (see Section 3.1.2) remaining. If the collision does indeed happen, all possibilities for the remaining two key bytes are explored with the full ASHF to see if any is the right key. The probability of a collision is about 2^{-19} (see [5]), so the time for this second step is 2^{45} which is negligible compared to the first. Thus, the algorithm as stated gives at least a reduction factor of $2^{3.41}$ work in terms of total number of keys studied. Taking into consideration that each step is four times faster than ASHF, the speed-up is at least $2^{5.41}$ over exhaustive key search.

In actuality, we often can get away with less than the $2^{12.58}$ permutation trials since many of them overlap. For instance, if both difference bits have been placed in the six bytes already determined by the guessed part of the key (which happens heuristically with probability about $\binom{48}{2}/\binom{64}{2} \approx .56$), and if the Hamming weight of the remaining two bytes is zero, then there is only one possibility for the last two bytes. According to an exhaustive search, if both differential bits have already been placed, then the average number of possibilities for the last two bytes is about $2^{11.3}$. If we assume 56% of the time we can get away with this shortened search and the other 44% of the time we do the full search of $2^{12.58}$ work (which is not optimal), then the expected work for the last two bytes is $2^{12.0}$. Hence, the modified key search is $2^{6.0}$ times faster than full key search. The reader can independently verify that the most number of trials for the last two bytes when there is no difference is 4670, which comes from any bit rotation of the word `0x09bd`.

We should also emphasize that we can traverse the unique permutations trials efficiently. The most obvious way to do it is to precompute a table that has a list of all unique permutations for each possible value of the last two bytes. Another way which requires less storage was given in [7].

So, we see that we can recover the key in 2^{58} operations more than 97% of the time a collision has occurred. This is possible because the keyed permutation has redundancy built into it: many keys lead to the same permutation. In total, [5] observed that there are 12-bits of redundancy on average, and we are only taking advantage of 4 of them. So we should be able to further speed up the search by guessing less than six bytes in the initial part of the search, but it would come at the penalty of using more RAM.

Although we will discuss better attacks in the following sections, they will only be successful about 50% of the time. Thus, the attack here is still relevant since it almost always succeeds and is significantly faster than exhaustive search.

3.4 The Attack of Biryukov, Lano, and Preneel

The attack of Biryukov, Lano, and Preneel [5] can determine the full 64-bit secret key when given a single collision of the hash function. Is assumes that the two input times t and t' are known exactly, i.e. a known plaintext attack. Generally this will be true or very close to true. If the internal clock of the SecurID has drifted by x minutes, then the running time of their attack will be multiplied by a factor of x.

Suppose that input times t and t' are expanded and permuted to become 64-bit words b and b', and the two words collide in subround N of the first round. In their key recovery attack, the attacker first guesses the subround N, and then uses a *filtering algorithm* for each N to search the set of candidate keys that make such a vanishing differential possible. According to their simulations, one only needs to do up to $N = 12$ to have a 50% chance of finding the key.[2] For larger values of N,

[2]Our own simulations suggest that one needs to search up to $N = 16$. The discrepancy is due to differences in the way the attack is viewed, which we elaborate on in Section 3.10.1.

the cost of the precomputation stage is likely to become prohibitive. A summary of their description for $N = 1$ is given below. For simplicity, assume that a 2-bit vanishing differential is used, though this need not be the case.

A one-time cost precomputation table is needed before the filtering starts. The table contains entries the form

$$(k_0, B_0, B_4, B'_0, B'_4).$$

where k_0 represents a key bit, (B_0, B_4) represent data bytes of b after the initial keyed permutation, and (B'_0, B'_4) represent data bytes of b' after the permutation. The exact entries in the table are those where (B_0, B_4) differs from (B'_0, B'_4) in exactly 2-bits known as the "difference bits," and for which a vanishing differential occurs during the first subround. Since none of the other key bits or data bytes are involved in the first subround, whether a vanishing differential can happen or not for $N = 1$ is completely characterized by this table.

For each entry in the table, the filtering proceeds in two phases, each of which contains two steps.

- *First Phase.* (process the first half of the key bits)
  - *First Step.* Guess key bits $k_1, ..., k_{27}$. Together with k_0, 28 key bits are set, which determines 28-bits of b and b' after the initial key-dependent permutation. Since these bits overlap with the entries in the table in nibbles $\mathtt{B}_9$ and $\mathtt{B}'_9$, a key value that does not produce the correct nibbles for both b and b' is filtered out.
  - *Second Step.* Continue to guess key bits $k_{28}, ..., k_{31}$. Filtering is done using overlaps in nibbles $\mathtt{B}_8$ and $\mathtt{B}'_8$.
- *Second Phase.* (process the second half of the key bits)
  - *First Step.* Continue to guess key bits $k_{32}, ..., k_{59}$. Filtering is done using overlaps in nibbles $\mathtt{B}_1$ and $\mathtt{B}'_1$.
  - *Second Step.* Continue to guess key bits $k_{60}, ..., k_{63}$. Filtering is done using overlaps in nibbles $\mathtt{B}_0$ and $\mathtt{B}'_0$.

Finally, each candidate key that passes the filtering is tested by performing a full hash function to see if it is the correct key.

Note that Biryukov, Lano, and Preneel designed their attack to be practical: only one table entry needs to be in RAM at a time. An alternative algorithm would keep all table entries in RAM at the same time. In this case, rather than guessing key bits for each table entry, the algorithm would guess key bits only once and then do a table lookup to see if the result matches any entries. As we will see, this alternative algorithm is not so practical because the precomputed tables are too large to fit in RAM. They are not too large to fit on a modern hard-drive, however.

3.4.1 Assumptions

Many details about the Biryukov, Lano, and Preneel attack were omitted from their publication due to space limitations. In this section, we give assumptions about how their attack works for larger values of N, as well as some other subtle details. Under these reasonable assumptions, our analysis agrees very much with their experimental results.

For a general value of N, the precomputed table consists of entries of the following form:

- valid values for the key bits in indices $0, \ldots, N-1$,
- valid values for the plaintext pairs after the initial permutation in bit indices $32, 33, \ldots, 38+N$ which we label as (W_4, W_4') (we use the subscript 4 because the words begins at byte B_4), and
- valid values for the plaintext pairs after the initial permutation in bit indices $0, 1, \ldots, 6+N$ which we label as (W_0, W_0') (the word begins at byte B_0).

By "valid values" we mean that the combination of plaintext bits after the initial permutation and key bits will cause the difference to vanish in subround N. The words W_0, W_0', W_4, W_4' each consist of $7+N$ bits (again, see Figure 3.3) and the number of key bits is N.

In general, we assume there are two phases of filtering, but the number of steps per phase depends upon N. Filtering can be done based upon each nibble that overlaps with the words W_0, W_0', W_4, W_4'. Thus, we have a total of $\lceil \frac{7+N}{4} \rceil$ steps per phase. The first step in phase 2 is particularly important in the analysis. The exact number of key bits guessed in this step is $4 \times \lfloor \frac{29-N}{4} \rfloor$ (i.e. guessing just enough bits so that the resulting permuted data array begins to overlap with W_0 and W_0').

With exception to the last step, only part of the precomputed table is used in the filtering. For example, the last filtering step of the first phase uses only k_0, W_4, and W_4' but not W_0, W_0'. Because of this, there may be multiple precomputation table entries that are the same in the bits that are involved in the filter. It is a waste of time to deal with the repeated items separately. Instead, we assume the table is sorted and that the precomputed entries that are the same in the "active" filtering bits are grouped together as one until a later filtering step requires separating them. Since the separation task can be done very efficiently[3], we assume that the overhead time here is negligible. This optimization is actually quite a critical assumption in our analysis: without it, the Biryukov, Lano, and Preneel attack is much less efficient. The reader will recognize where this factors into our analysis when we specify certain *unique* parts of precomputed table entries.

[3]Simply by doing j comparisons where j is the number of overlaps. We will see in Section 3.7 that j is typically small, so this is much faster than doing j separate permutations and then j comparisons.

3.5 Analysis of the Biryukov, Lano, and Preneel Attack

Biryukov, Lano, and Preneel estimated the time complexity of their attack through simulation. They provided results for $N = 1$: step 1 of phase 1 reduced the number of possibilities to 2^{27}, step 2 of phase 1 further reduced the count to to 2^{25}, step 1 of phase 2 increased the count to 2^{45}, and step 2 of phase 2 resulted in 2^{41} true candidates. For larger values of N, they expect that the complexity of the attack would be lower due to stronger filtering.

Here we analyze their algorithm, giving some mathematical justification for the simulation results they observed and also showing that their conjecture of the filtering improving for larger N appears to be correct. In our analysis, we sometimes treat probabilities as if they are independent, which is not always true, but it is assumed that it provides a reasonable approximation. For instance, in the formula $P(A \cap B) = P(A|B) \times P(B)$, if the event B is expected to have little effect on the event A, then we may simply take $P(A \cap B) \approx P(A) \times P(B)$. Other similar approximations are used as well. In Section 3.8, we show that our estimates agree with real implementation experiments.

Before we begin, we mention a restriction on the precomputed table that was overlooked in [5]: the values of the two bits in b (or b') where the differences are located must be the same, due to the way the time expansion works. This reduces the number of precomputed table entries and results in a speed-up to the filtering. Although this is one of our three main filtering speed-ups, we apply it to the analysis of the original [5] algorithm in order to keep things as clean as possible.

3.5.1 Analysis of Final Number of Candidates

Analyzing the final step is equivalent to determining the true number of candidates that need to be tested with the full SecurID hash function. The expected number of true candidates can easily be determined since anything that matches an entry in the precomputed table will result in a vanishing differential. In other words, the entries in the table are not only a necessary set of cases for a vanishing differential to occur, but also sufficient.

For each entry in the precomputed table, we have:

- Only a portion of about $1/\binom{64}{2}$ of the 2^{64} keys will permute the two difference bits into the locations corresponding to what is in that table entry.
- With probability $\frac{1}{2}$, the value of the two difference bits will match those in the table (recall, the 2-bits in b must be the same, and the corresponding bits in b' are the complement).
- With probability $\frac{1}{2^{2N+12}}$, the remaining permuted data bits will match the table entry.
- With probability $\frac{1}{2^N}$ the guessed key bits will match the entry of the table.

Hence, the expected number of final candidates is:

$$\text{table size} \times 2^{64} \times \frac{1}{\binom{64}{2}} \times \frac{1}{2} \times \frac{1}{2^{2N+12}} \times \frac{1}{2^N} \ . \tag{3.1}$$

Some discussion is necessary about our analysis, in particular, it must be emphasized that these are approximations which measure probabilities as if bits are random and independent. For example, randomly chosen permutations would have each possible $\binom{64}{2}$ locations of the difference bits with equal probability, but it is not clear whether the SecurID permutation has the same property. Similarly, the $\frac{1}{2^{2N+12}}$ probability is an approximation that assumes we can match each bit with probability $\frac{1}{2}$. This is generally a reasonably close approximation to the true probability, but the exact probability will depend upon the particular inputs. Indeed, [5] noted that simulation results varied according to inputs. Thus, the reader should keep in mind that our approximations may not hold in extreme cases, however we expect them to be close most of the time.

3.5.2 Run Time Analysis of Phase 2, Step 1

Biryukov, Lano, and Preneel suggested that phase 2, step 1 was the most costly step in their attack. For now, we will assume this is true. Later, in section 3.7, the claim will be justified.

To analyze step 1 of phase 2, we must first determine the number of candidates passing phase 1. Define C_0 to be the number of unique table entries of the form $(k_0, \ldots, k_{N-1}, W_4, W_4')$ where $W_4 = W_4'$, define C_1 similarly except $W_4 \oplus W_4'$ having Hamming weight 1, and define C_2 similarly except $W_4 \oplus W_4'$ having Hamming weight 2.

Among the 2^{32} key bits considered in phase 1, a fraction of $\binom{57-N}{2}/\binom{64}{2}$ will put no difference in the tuple (W_4, W_4'). Of those, only a fraction of $\frac{C_0}{2^{7+N}}$ will match one of the C_0 unique entries in the table for W_4 (which is the same as W_4'). With probability $\frac{1}{2^N}$, the guessed key bits will match those in the table as well. Thus, the expected number of 32-bit keys resulting in no difference in (W_4, W_4') that pass phase 1 is:

$$2^{32} \times \frac{\binom{57-N}{2}}{\binom{64}{2}} \times \frac{C_0}{2^{7+N}} \times \frac{1}{2^N} \ = \ 2^{19-2N} \times \frac{3192 - 113N + N^2}{63} \times C_0 \ .$$

For 1-bit differences, we have a fraction of $\binom{57-N}{1}\binom{7+N}{1}/\binom{64}{2}$ keys that will put exactly one difference bit inside of (W_4, W_4') and one outside. For the one bit inside, the probability that it is in the right place to match a particular table entry is $1/(7+N)$. The probability that this difference bit has the right value is $\frac{1}{2}$ (i.e. the bit is either a 1 in W_4 or a 0). The probability that the other non-difference bits inside (W_4, W_4') match the table entry is 2^{-6-N}, and the probability that the key bits match the table entry is 2^{-N}. Hence, the expected number is:

$$2^{32} \times \frac{\binom{57-N}{1}}{\binom{64}{2}} \times \frac{1}{2} \times \frac{C_1}{2^{6+N}} \times \frac{1}{2^N} \ = \ 2^{20-2N} \times \frac{57-N}{63} \times C_1 \ .$$

N	Table size	C_0	C_1	C_2	T	Time for phase 2, step 1	Time for testing final candidates	Total time
1	12	5	2	0	$2^{25.0}$	$2^{47.0}$	$2^{40.6}$	$2^{47.0}$
2	152	11	64	44	$2^{24.3}$	$2^{43.2}$	$2^{41.3}$	$2^{43.5}$
3	1130	64	362	128	$2^{24.8}$	$2^{43.6}$	$2^{41.2}$	$2^{43.9}$
4	7292	453	1750	712	$2^{25.5}$	$2^{44.3}$	$2^{40.9}$	$2^{44.4}$
5	48212	2775	10614	3864	$2^{26.0}$	$2^{44.9}$	$2^{40.6}$	$2^{44.9}$
6	276788	15076	52716	19520	$2^{26.4}$	$2^{41.4}$	$2^{40.1}$	$2^{41.9}$

Table 3.1: Computing the running time estimates of algorithm [5] for $N = 1..6$.

For 2-bit differences, the equation is

$$2^{32} \times \frac{1}{\binom{64}{2}} \times \frac{1}{2} \times \frac{C_2}{2^{5+N}} \times \frac{1}{2^N} \;=\; 2^{21-2N} \times \frac{C_2}{63} \;.$$

The $\frac{1}{2}$ in this last equation accounts for whether the two difference bits in the first plaintext match the table entry (the bits must be the same). Thus, the expected number of candidates to pass phase 1 is

$$T = \frac{2^{19-2N}}{63} \times \left[(3192 - 113N + N^2)C_0 + (114 - 2N)C_1 + 4C_2\right] \;. \tag{3.2}$$

The first step in phase 2 involves guessing $4 \times \lfloor \frac{29-N}{4} \rfloor$ key bits. Under the assumption that the permutation is 5% of the time required to do the full SecurID hash [5], the running time is equivalent to

$$T \times 2^{4 \times \lfloor \frac{29-N}{4} \rfloor} \times \frac{4 \times \lfloor \frac{29-N}{4} \rfloor}{64} \times 2 \times 0.05 \times s \tag{3.3}$$

full hash operations, where s is the speed-up factor that can be obtained by taking advantage of the redundancy in the key with respect to the permutation. The value of s is $\frac{96}{256}$ for $N = 1$, $\frac{12}{16}$ for $N = 2..5$, and 1 for all other values.

3.5.3 Combined Analysis

Assuming that other filtering steps are negligible (to be discussed in Section 3.7), the running time of algorithm [5] for a particular value of N is expected to be approximately the sum of Equations 3.3 and 3.1. For $N = 1..6$, these running times are given in Table 3.1. Again, we reiterate that the table sizes are different from [5] because of an extra condition due to the time expansion, which also gives a small improvement in the running time. The analysis for $N = 1$ is very close to the simulated results from [5].

Even though the number of candidates T after the first phase is approximately the same as N goes from 1 to 2 and also from 5 to 6, the running times of the

phase 2, step 1 drop significantly. This is because one less nibble of the key is being guessed, and an extra filtering step is being added. In general, we see the pattern that larger values of N are contributing less and less to the sum of the running times, which agrees with the conjecture from [5]. The total running time for $N = 1$ to 6 is $2^{47.7}$ and larger values of N would appear to add minimally to this total.

3.5.4 Remark on Table Size Growth

It has been observed experimentally that 2-bit vanishing differentials happen with probability 2^{-19}, and that the distribution of which subround that collision happens in is not far from uniform for $N \in [1..32]$. Thus, we would expect about $2^{64} \times 2^{-19} \times 2^{-5} = 2^{40}$ final candidates for each such value of N. This agrees with Table 3.1.

We can use this observation to predict table growth for other values of N. The probability of a collision in subround N is approximately given by Equation 3.1 divided by 2^{64}, and we expect this to be about $2^{-19} \times 2^{-5} = 2^{-24}$. Therefore, we get the following estimation for the table size:

$$\text{table size} \;\approx\; 2^{-24} \times \binom{64}{2} \times 2^{3N+13} \;. \tag{3.4}$$

The estimates given by Equation 3.4 are reasonably close to the values given in Table 3.1. They are within a factor of 2, except for $N = 2$ and $N = 3$, where the error is slightly larger. For N up to 6, the estimate from Equation 3.4 is smaller than the actual value. However, Equation 3.4 grows by a factor of 8 as N increments whereas the actual table sizes seem to be growing less. Coupling this observation with the fact that the estimate is very close for $N = 6$, it seems reasonable to assume that Equation 3.4 may be slightly pessimistic for large values of N.

We have not found a way of estimating C_0, C_1, and C_2.

3.6 Faster Filtering

Table 3.1 illustrates that the trick to speeding up the key recovery attack in [5] is faster filtering. We have found three ways in which their third filtering can be sped up:

1. Only include entries in the precomputed table that actually can be derived from the time expansion. In particular, the values of the two bits in b (or b') where the differences are located must be the same.

2. In the original filter, a separate permutation is computed for each trial key. This is inefficient, since most of the permuted bits from one particular permutation will overlap with those from many other permutations. Thus, we can amortize the cost of the permutation computations.

3. We can detect ahead of time when a large portion of keys will result in "bad"

permutations in steps 1 of both phase 1 and phase 2, and the filtering process can skip past chunks of these bad permutations.

The first technique was already applied to the analyses in the previous section. Without this improvement, the running time would have been about 50% worse.

The second technique is aimed at reducing the numerator of the factor $\frac{4\times\lfloor\frac{29-N}{4}\rfloor}{64} = \frac{\lfloor\frac{29-N}{4}\rfloor}{16}$ in Equation 3.3. To do this, we view the key as a 64-bit counter, where k_0 is the most significant bit and k_{63} is the least. In phase 2, step 1 of the filter, the bits $k_0, \ldots, k_{31}$ are fixed and so are some of the least significant bits (the exact number depends upon N), so we can exclude these for now. The keys are tried in order via a recursive procedure that handles one key nibble at a time. At the j^{th} recursive branch, each of the possibilities for nibble $\mathtt{K}_{7+j}$ is tried. The part of the permutation for that nibble is computed, and then the $j+1^{\text{st}}$ recursive branch is taken. The level of recursion stops when key nibble $\mathtt{K}_{7+\lfloor\frac{29-N}{4}\rfloor}$ is reached. Thus, the $\lfloor\frac{29-N}{4}\rfloor$ from Equation 3.3 gets replaced with the average cost per permutation trial, which is $\sum_{i=0}^{\lfloor\frac{29-N}{4}\rfloor-1} 2^{-4i} \approx 1.07$. Observe that when $N = 1$, this results in a factor of $\frac{7}{1.07} \approx 6.5$ speed-up. This trick alone knocks more than 2 bits off the running time.

The third speed-up is dependent upon the second. It will apply in both phases of the filtering. During the process of trying a permutation, there will be large chunks of bad trial keys that can be identified immediately and skipped. In particular, whenever a difference bit is placed outside of words (W_0, W_0') and (W_4, W_4'), the key can be skipped because the difference is not in a valid position. Moreover, any other key with the same most significant bits (up to the key nibble that placed the difference bit) will also result in invalid values, implying that the entire recursive branch can be skipped. Heuristically, one would expect that the number of keys that get tested for filtering in phase 2, step 1 to be about a fraction of about $\binom{14+2N}{2}/\binom{64}{2}$ of the number for the attack in [5]. However, this oversimplifies the analysis. We briefly summarize a more proper analysis in the next paragraph.

Let U be the number of key possibilities through step 1 of phase 2. That is, $U = 2^{60}$ for $N = 1$, $U = 2^{56}$ for $N = 2..5$, and $U = 2^{52}$ for $N = 6$. Then the number of candidate keys up to this point that have both differences in W_0, W_0' is about

$$x = U \times \frac{\binom{7+N}{2}}{\binom{64}{2}} \times \frac{C_0}{2^{7+N}} \times \frac{1}{2^N} \ .$$

The number of candidates with a 1-bit difference in W_0, W_0' (the other difference bit is in W_4, W_4') is about

$$y = U \times \frac{\binom{7+N}{1}}{\binom{64}{2}} \times \frac{C_1}{2^{6+N}} \times \frac{1}{2} \times \frac{1}{2^N} \ .$$

and the number of candidates with no differences in W_0, W_0' (both differences are in W_4, W_4') is about

$$z = U \times \frac{1}{\binom{64}{2}} \times \frac{C_2}{2^{5+N}} \times \frac{1}{2} \times \frac{1}{2^N} \ .$$

N	Time for phase 2, step 1	Time for testing final candidates	Total time
1	$2^{38.7}$	$2^{40.6}$	$2^{40.9}$
2	$2^{36.4}$	$2^{41.3}$	$2^{41.3}$
3	$2^{37.1}$	$2^{41.2}$	$2^{41.3}$
4	$2^{37.9}$	$2^{40.9}$	$2^{41.1}$
5	$2^{38.6}$	$2^{40.6}$	$2^{40.9}$
6	$2^{35.7}$	$2^{40.1}$	$2^{40.2}$

Table 3.2: Running times using our improved filter, for $N = 1..6$.

Hence the expected run time of phase 2, step 1 is

$$(x + y + z) \times \frac{4 \times 1.07}{64} \times 2 \times 0.05 \times s \ .$$

The three filtering speed-ups give the run times in Table 3.2. In all cases, phase 2, step 1 has become faster than the time for testing the final candidates. The running time for $N = 1..6$ is $2^{43.6}$, so we conjecture that the run time for N up to 12 is no more than $12/6 \times 2^{43.6} \approx 2^{44.6}$. This assumes that the speed of other filtering steps is negligible, which we analyze in section 3.7.

Although it appears that we cannot do much better using only a single 2-bit vanishing differential, we can improve the situation if we use other information that an attacker would have. In later sections we will show that we can improve the time greatly if we take advantage of multiple vanishing differentials, or if we take advantage of knowledge that no other vanishing differentials occur within a small time period of the observed one. We will also show that $\geq$ 4-bit vanishing differentials appear to be to the advantage of the attacker.

3.7 The Speed of Other Steps

We have so far assumed that phase 2, step 1 is the most time consuming step of the Biryukov, Lano, and Preneel filter. Biryukov, Lano, and Preneel suggested the same thing in their publication [5]. In fact, it is not completely clear that this is true. Here we give some justification to this belief. Unfortunately, we must emphasize the word *belief*, since a full analysis remains an open problem.

Due to our filtering speed-ups, the running time of each step is roughly a constant times the number of candidates considered in that step. Thus, we want to argue that the number of candidates is maximum in step 1 of phase 2. In fact, sometimes step 2 of phase 2 will be comparable to step 1 due to the fact that step 1 does not always filter based upon a full nibble, but step 2 should add only a fractional amount to the run time. We expect that one of these two steps is the dominant filtering cost, and that filtering cost is always significantly less than the testing of

	beginning of step 1	beginning of step 2	beginning of step 3	end of step 3
c_0	15076	29792	52034	75038
c_1	52716	57100	98358	142918
c_2	19520	21984	41176	58832

Table 3.3: The blow-up factor for $N = 6$.

final candidates.

It would be ideal if we could argue that the number of candidates generally gets whittled down in every step. Working against this claim is a "blow-up" factor due to the assumption that entries which are the same in the active filtering bits can be grouped together and treated as one (see Section 3.4.1). Eventually, these grouped elements must be separated, resulting in an increase in candidates.

Here is an example of the blow-up for $N = 1$. The precomputed table entries are derived in the Appendix. The two entries of the form $(k_0, B_0, B_4, B'_0, B'_4)$ (`0x0, 0x06, 0x9e, 0xc6, 0x9e`) and (`0x0, 0xc6, 0x9e, 0x06, 0x9e`) will be grouped together until the last filtering step, since they only differ in nibbles $(\mathtt{B}_0, \mathtt{B'}_0)$. Any candidate matching this entry up until the last step will be separated into two candidates before processing that last step.

Since we have no analytical means for predicting the blow-up factor, we are cornered into using empirical evidence to understand the effect. Let c_0, c_1, and c_2 be the analagous variables to C_0, C_1, and C_2 except taking into considerating the blow-up factor at various steps in phase 2. Thus, at the first step in phase 2, we have $c_0 = C_0$, $c_1 = C_1$, and $c_2 = C_2$, but in later steps the values of c_0, c_1, and c_2 become larger until eventually $c_0 + c_1 + c_2 =$ table size. We have computed these values for $N = [1..6]$ and found that blow-up factor per step is almost always within a factor of 2. The values for $N = 6$ are given in Table 3.3.

Let us first consider phase 1. The number of starting candidates for step 1 of phase 1 is equal to the number of key bits guessed times the number of precomputed table entries that are unique in the key bits and rightmost nibble (or partial nibble) of (W_4, W'_4). Since the table size increases the fastest, this is largest for highest values of N. The Biryukov, Lano, and Preneel attack was specified only up to $N = 12$ because of precomputation and storage requirements, so the most threatening case is $N = 12$. For $N = 12$, we guess 16 key-bits. According to Equation 3.4, the full table size is about 2^{39}. For simplicity of analysis, we will begin by overestimating the cost of phase 1 by analyzing the case where all table entries are used – in other words, we do the less efficient algorithm that has no blow-up factor. In this case, the number of initial candidates is $2^{16} \times 2^{39} = 2^{45}$. Note that this is much smaller than the number of candidates that currently dominates the filtering time (when $N = 1$, step 1 of phase 2 has an estimated 2^{53} candidates). We can conclude that step 1 of phase 1 is certainly much faster than the most costly steps of phase 2, and therefore it can be considered negligible.

It is easy to see that other steps in phase 1 will have less candidates than the 2^{45} over-estimate for step 1 with $N = 12$. For each surviving candidate, we guess another key nibble. Thus, we have 2^4 new candidates for each previous candidate. On the other hand, the probability of the candidate surviving is less than 2^{-4}. This is because the new data nibbles that are computed from the permutation must match that of the precomputed table entry for that candidate. There are *two* data nibbles that must match: one for W_4 and one for W'_4. Due to the difference bits, the two nibbles may not be the same. For example, if the precomputed table entry has a difference in the active nibble, then most candidates will be dropped because they will likely not have the difference bit – and if they do, it needs to be in the right place. Also, if the table has no difference in the active nibble, then the current candidate could be filtered out if it incorrectly places a difference there. So, the probability of a candidate surviving is *definitely* less than 2^{-4}, implying that other steps in phase 1 are also negligible.

Unfortunately, the over-estimate just provided is not sufficient for showing that step 1 of phase 2 is negligible for values of $N > 6$. We need to take into consideration the savings from combining similar table entries as well as the expected reduction in the number of candidates through filtering to get an accurate estimate of the number of candidates starting phase 2 (i.e. the variable T). This remains an open problem. But if Table 3.1 is any indication, we would expect T to be on the order of 2^{26} for nearby[4] values of N – or perhaps less due to our third filtering speed-up. Under this assumption, the first step of phase 2 will generally become less costly as N increases, consistent with our observations in Section 3.5.3. So, the number of candidates going through step 1 is expected to be about $2^{26+4\times\lfloor\frac{29-N}{4}\rfloor}$. Note that for larger values of N, the number of candidates decreases. This is countered by an increased blow-up factor per step.

By plugging in some numbers, it appears that the number of candidates is small enough so that the blow-up factor will not have much of an impact. If this turns out to be false, the following extra filtering trick can be added to eliminate a large portion of false candidates:

- For each surviving candidate, check that the Hamming weight of the remaining (not yet permuted) data bits matches that of the corresponding table entries. Reject the candidate if it does not.

If there are x nibbles remaining, this trick will eliminate at least a fraction of $\binom{4x}{2x}/2^{4x}$ of the false candidates. For example, a fraction of $2^{-1.87}$ of the false candidates are eliminated when $x = 2$ and a fraction of $2^{-2.15}$ are eliminated when $x = 3$.

It is also worth noting that the redundancy of the key with respect to the permutation also mitigates the cost of other steps. Taking everything into consideration, it seems quite likely that other filtering steps will have little impact on the overall run time.

[4]It certainly cannot remain at 2^{26} forever. Nevertheless, the growth appears to be very slow.

3.8 Software Implementation

The attack of Biryukov, Lano, and Preneel was specially designed to keep RAM usage low – only one of the precomputed table entries needs to be in program memory at a time. We tested our ideas only for $N = 1$ and 2-bit differences, and since the table size is small, we took the freedom of implementing a slight variant of their attack which kept the whole precomputed table in memory at once.

We programmed all filtering steps of both phases and the three main filtering speed-ups. In addition, we programmed an extra "table lookup" speed-up that would improve the running time by a factor of 8 for $N = 1$. The extra speed-up is only applicable for small values of N due to the memory requirements. Thus, the running time is expected to be 8 times faster than the $2^{38.7}$ listed in Table 3.2. On our 2.4 GHz PC, this translates to about 8 days of effort.

Our code did the search in numerical order, when the key is viewed as a counter as described in Section 3.6. The only thing we did not do was testing the final candidates using the real function. Instead, we just stopped when we arrived at the target key. So our implementation was designed to test and time the filtering only, in order to confirm that filtering is significantly faster than testing of the final candidates.

We have not done the full key search. However, we have done a search that starts out knowing the correct first nibble of the key. The key we were searching for is `356b48b3ae15c271` which yields a vanishing differential when times t =`0x1c3ba8` and t' =`0x1c3aa8` are sent in. We were able to find the key in 13.8 hours. If we pessimistically assume that the full search will take at most 2^4 times longer, the full running time would be 9.2 days, which is on target of expectations.

To understand why we believe this is pessimistic, observe that the first difference is at bit index 15. So, whenever the first two key nibbles add up to a value between 16 and 19, the entire recursive branch corresponding to the second key nibble is skipped, according to our second filtering speed-up. Since our search fixed the first key nibble at 3 and the second nibble went through values from 0 to 5, none of these big skips have happened yet. Thus, the second filtering speed-up will become more effective during a full search.

3.9 Vanishing Differentials with $\geq$ 4-bit Differences

According to our simulations, about 25% of the first collisions (first occurrence of a vanishing differential for a given key) are actually from a 4-bit difference, and about 7% from larger differences. We would expect that our filtering algorithm still performs quite well in these circumstances. Here, we give a first analysis assuming 4-bit differences.

When $N = 1$, we would expect our third filtering speed-up to skip all except a fraction of $\binom{16}{4}/\binom{64}{4} \approx 2^{-8.4}$ of the incorrect keys between phase 1 and the first step of phase 2. Without going through the analysis, it seems reasonable to assume that

N	Table size	Run time
1	90	$2^{37.2}$
2	1234	$2^{38.0}$
3	5904	$2^{37.3}$
4	32458	$2^{36.7}$

Table 3.4: Expected cost of the final step using a 4-bit differential for $N = 1..4$, where difference bits in b are all 1's or all 0's.

the testing of final candidates is still the bottleneck. The formula for the number of final candidate keys for 4-bit differences is derived similarly to that of Equation 3.1, except we have to distinguish between the type of differences. If the difference bits in b are all the same, then the formula is:

$$\text{table size} \times 2^{64} \times \frac{1}{\binom{64}{4}} \times \frac{1}{2} \times \frac{1}{2^{2N+10}} \times \frac{1}{2^N} \ .$$

Table 3.9 shows the expected running times in this case, for $N = 1..4$. When two difference bits in b are 1 and the other two are 0, then the formula is:

$$\text{table size} \times 2^{64} \times \frac{1}{\binom{64}{2}} \times \frac{1}{\binom{62}{2}} \times \frac{1}{2^{2N+10}} \times \frac{1}{2^N} \ .$$

Table 3.9 gives the running times for $N = 1..4$ for this case.

Joe Lano pointed out to us [21] that the assumptions about independence of permuted difference bits may make some of these running time estimates inaccurate. Neither he nor we have seen any cases where the 4-difference bits have come from a 1-bit difference in the time (see Section 3.2). Instead, they all seem to be coming from 2-bit differences in the time, where the differences are in bytes T_0 and T_1 (see Section 3.1.1). Consequently, the difference bits occasionally tend to be grouped together, and in the cases that they are, that grouping is often preserved after the key-dependent permutation because of the way it works (i.e. taking four consecutive bits at a time). So in summary, Tables 3.9 and 3.9 may not represent the actual run time for all cases where a 4-bit difference occurs, but it should be somewhat accurate for most cases. We conjecture that finding the key for $\geq$ 4-bit vanishing differentials is generally much faster than for 2-bit vanishing differentials. The downside is that the memory and time for the precomputation is larger.

3.10 Multiple Vanishing Differentials

There are two scenarios for multiple vanishing differentials: when they have the same difference and when they have different differences. The former is more likely to occur, but in either case we can speed up the attack.

N	Table size	Run time
1	344	$2^{37.6}$
2	3364	$2^{37.9}$
3	22176	$2^{37.6}$
4	119112	$2^{37.0}$

Table 3.5: Expected cost of the final step using a 4-bit differential for $N = 1..4$, where difference bits in b consist of two 1's and two 0's.

3.10.1 Multiple Vanishing Differentials with the Same Difference

According to computer simulations, about 45% of the keys that had a collision over a two month period will actually have at least 2 collisions. There is a simple explanation for this, and a way to use the observation to speed up the key search even more.

Consider a vanishing differential which comes from times $t = T_0T_1T_2$ and $t' = T'_0T'_1T'_2$. As we saw earlier, the only bits that determine whether the vanishing differential will occur at a particular subround are those that get permuted into words W_0, W'_0, W_4, and W'_4. Suppose we flip one of the bits in T_2 and T'_2 (the same bit in each). This bit will be replicated four times in the time expansion. If, after the permutation, none of those bits end up in W_0, W'_0, W_4, or W'_4, then we will witness another vanishing differential. The new vanishing differential will follow the same difference path and disappear in the same subround. Thus, new information is learned that can be used to speed up the key search, which we explain below. In the case that another vanishing differential does *not* occur, information is also learned which can improve the search, which is detailed in Section 3.11.

Following the above thought process, it is evident that:

- Flipping time bits in T_1, T'_1 or T_0, T'_0 will only replicate the flipped bit twice in the expansion. Since there are only two bits that are not allowed to be in W_0, W'_0, W_4, and W'_4, the collision is more likely to occur. On the other hand, the time between the collisions is increased, since these are more significant time bits.

- Multiple vanishing differentials are more likely to occur when the first collision happened in a small number of subrounds. This is because the words W_0, W'_0, W_4, and W'_4 are smaller, giving more places where the flipped bits can land without interfering with the collision.[5]

[5]This is the reason for the apparent discrepancy between our research claiming that one needs to precompute up to $N = 16$ in order to have a $\geq 50\%$ chance of finding the key and [5] claiming 12. In our view, the attacker has a single token and will perform a key search once a single vanishing differential has occurred. In their view, the attacker has several tokens for a fixed period of time, and the attacker selects a vanishing differential randomly among all vanishing differentials that have occurred [21]. Since their view includes multiple vanishing differentials, the expected number of subrounds is less.

N	Number of final cands using single collision	Number of final cands with $z = 2$	Number of final cands with $z = 4$	Number of final cands with $z = 8$
1	$2^{40.6}$	$2^{39.8}$	$2^{38.9}$	$2^{37.0}$
2	$2^{41.3}$	$2^{40.3}$	$2^{39.3}$	$2^{37.2}$
3	$2^{41.2}$	$2^{40.1}$	$2^{39.0}$	$2^{36.6}$
4	$2^{40.9}$	$2^{39.7}$	$2^{38.4}$	$2^{35.7}$
5	$2^{40.6}$	$2^{39.2}$	$2^{37.8}$	$2^{34.8}$
6	$2^{40.1}$	$2^{38.6}$	$2^{37.0}$	$2^{33.6}$

Table 3.6: Number of final candidates assuming the attacker became aware of z-bits that do not get permuted into words W_0, W_0', W_4, or W_4'.

- The converse of these observations is that when multiple vanishing differentials occur, it is most often the case that the collisions all happened in the same subround and followed the same difference path. Moreover, the collisions usually happen within a few subrounds.

By simply inspecting the time data that caused the multiple vanishing differentials, one can determine with high accuracy whether this situation has happened. The signs of it are: 1) Same input difference for all vanishing differentials, 2) All input times differ in only a few bits, and 3) It is the same bits that differ in all cases. An example is given below.

The attacker learns $z \geq 2$ bits which cannot be permuted into words W_0, W_0', W_4, or W_4'. This new knowledge can be combined with our third filtering speed-up to skip past more bad keys. The expected number of final key candidates to be tested becomes a fraction of $\binom{50-2N}{z}/\binom{64}{z}$ of the values given in Table 3.2. See Table 3.6 for a summary of these figures when $z = 2$, $z = 4$, and $z = 8$. The times can be further reduced using information about where certain related plaintexts did not cause a vanishing differential: see Section 3.11.

Example of Multiple Vanishing Differentials

Table 3.7 is an example where sixteen vanishing differentials happened within 1.3 days. All had the same difference path, which collided at $N = 2$. One can see that only the four least significant bits of time byte T_1 differ. Since each of these bits are replicated twice in the time expansion, the expected running time of the last steps is given by $z = 8$ in Table 3.6. Taking into consideration $N = 2$, the total time is expected to be on the order of 2^{38} operations.

3.10.2 Multiple Vanishing Differentials with Different Differences

Given two vanishing differentials with different differences, the number of candidate keys can be reduced significantly by constructing more effective filters in each step. Denote the two pairs of vanishing differentials V_1 and V_2, and their N values N_1

First plaintext	Second plaintext
1e 80 8c 8c 1e 80 8c 8c	1e 90 8c 8c 1e 90 8c 8c
1e 81 8c 8c 1e 81 8c 8c	1e 91 8c 8c 1e 91 8c 8c
1e 82 8c 8c 1e 82 8c 8c	1e 92 8c 8c 1e 92 8c 8c
1e 83 8c 8c 1e 83 8c 8c	1e 93 8c 8c 1e 93 8c 8c
1e 84 8c 8c 1e 84 8c 8c	1e 94 8c 8c 1e 94 8c 8c
1e 85 8c 8c 1e 85 8c 8c	1e 95 8c 8c 1e 95 8c 8c
1e 86 8c 8c 1e 86 8c 8c	1e 96 8c 8c 1e 96 8c 8c
1e 87 8c 8c 1e 87 8c 8c	1e 97 8c 8c 1e 97 8c 8c
1e 88 8c 8c 1e 88 8c 8c	1e 98 8c 8c 1e 98 8c 8c
1e 89 8c 8c 1e 89 8c 8c	1e 99 8c 8c 1e 99 8c 8c
1e 8a 8c 8c 1e 8a 8c 8c	1e 9a 8c 8c 1e 9a 8c 8c
1e 8b 8c 8c 1e 8b 8c 8c	1e 9b 8c 8c 1e 9b 8c 8c
1e 8c 8c 8c 1e 8c 8c 8c	1e 9c 8c 8c 1e 9c 8c 8c
1e 8d 8c 8c 1e 8d 8c 8c	1e 9d 8c 8c 1e 9d 8c 8c
1e 8e 8c 8c 1e 8e 8c 8c	1e 9e 8c 8c 1e 9e 8c 8c
1e 8f 8c 8c 1e 8f 8c 8c	1e 9f 8c 8c 1e 9f 8c 8c

Table 3.7: Example of 16 vanishing differentials that happened within 1.3 days, using key `b5 a9 f4 8c 16 23 a6 1a`.

and N_2.

We first make a guess of (N_1, N_2). The number of guesses will be quadratic in the number of subrounds tested up to. The following is a simplified sketch for the new filtering algorithm.

- *First Phase.* Take V_1 and guess the first 32-bits of the key. For each 32-bit key that produces a valid (W_4, W_4'), test it against V_2 to see if it also produces a valid (W_4, W_4').
- *Second Phase.* For 32-bit keys that pass phase 1, do the same thing to guess the second 32 bits of the key.

The main idea here is to do double filtering within each stage so that the number of candidate keys is further reduced in comparison to when only a single vanishing differential is used.

When $N_1 = N_2 = 1$, the probability that a 32-bit key passes phase 1 (see Table 3.1) is $2^{25.0}/2^{32} = 2^{-7.0}$ (assuming using the original filter of [5] – it is even more reduced using our improved filter), and the probability that a 64-bit key passes both phases is $2^{40.6}/2^{64} = 2^{-23.4}$. If the two vanishing differentials are indeed *independent*, we would expect the number of keys to pass the first phase to be

$$2^{32} \times 2^{-7.0} \times 2^{-7.0} = 2^{18}$$

and the number of keys to pass both phases to be

$$2^{64} \times 2^{-23.4} \times 2^{-23.4} = 2^{17.2}.$$

Experimentation should be done to determine whether these figures are attainable in practice, but even if they are not, a big speed up is still expected. The situation should be better in the cases where differences with Hamming weights ≥ 4 are involved.

We should mention the caveat that the chances of success using the above technique are lower, since we need *both* difference pairs to disappear within sixteen subrounds. On the other hand, the cost of trying this algorithm for two difference pairs is expected to be substantially lower than trying the previous algorithms for only one. Therefore, the double filtering should add negligible overhead to the search in the cases that it fails, and would greatly speed up the search when it is successful.

3.11 Using Non-Vanishing Differentials with a Vanishing Differential

In Section 3.10.1, we argued that even if only a single vanishing differential occurs over some time period, the search can still be sped up if one takes advantage of knowing where related differentials do not vanish. Here, we give the details.

Assume a vanishing differential occurred at times t and t', but no vanishing differential occurred among the time pairs $(t \oplus 2^i, t' \oplus 2^i)$ for $i = 2, \ldots, j$. We start with $i \geq 2$ because in the most typical case, where authenticators are displayed every minute, the least two significant bits of the time are 0 (see Section 3.1.1). For the values $2 \leq i \leq 7$, the difference is replicated 4 times in the time expansion, and for $i \geq 8$, it is replicated twice.

For each value of i, we learn a set of 2 or 4 bits for which at least one in each set must be permuted into the words W_0, W_0', W_4, or W_4'. Let us label these sets as $U_2, \ldots, U_j$. For simplicity, we will take $j = 13$, which corresponds to no other vanishing differential within a window of 2.8 days before or after the observed one. So, we are interested in the probability of at least one bit in each of these sets getting permuted into words W_0, W_0', W_4, or W_4'.

We say a set U_i is *represented* with $c_i \geq 1$ bits if exactly c_i bits from U_i get permuted into W_0, W_0', W_4, or W_4'. The number of ways $2N+14$ bits can be selected to end up in W_0, W_0', W_4, or W_4' is $\binom{64}{2N+14}$. The number of ways that exactly c_i bits are represented in the selection for $2 \leq i \leq 13$ is

$$\prod_{i=2}^{7} \binom{4}{c_i} \times \prod_{i=8}^{13} \binom{2}{c_i} \times \binom{28}{2N + 14 - \sum_{i=2}^{13} c_i}.$$

The first product tells the number of ways of selecting c_i bits from each set that has 4 bits, the second product is the same except for among sets with 2 bits, and the third product is the number of ways of selecting the remaining bits from the 28 bits that are not among any of the U_i. Thus, our desired probability is:

$$\sum_{\text{all valid } (c_2, \ldots, c_{13})} \frac{\prod_{i=2}^{7} \binom{4}{c_i} \times \prod_{i=8}^{13} \binom{2}{c_i} \times \binom{28}{2N+14-\sum_{i=2}^{13} c_i}}{\binom{64}{2N+14}} \tag{3.5}$$

N	Fraction of keys having property	Time for testing final candidates
1	$2^{-14.3}$	$2^{26.3}$
2	$2^{-11.7}$	$2^{29.6}$
3	$2^{-9.7}$	$2^{31.5}$
4	$2^{-8.1}$	$2^{32.8}$
5	$2^{-6.7}$	$2^{33.9}$
6	$2^{-5.7}$	$2^{34.4}$

Table 3.8: Assuming no more vanishing differentials occur within 2.8 days before or after of a given vanishing differential, the final testing of candidates can be improved by the amounts given in this table.

where *valid* $(c_2, \ldots, c_{13})$ means that each value is at least 1, but the sum of all values is no more than $2N + 14$.

We have computed these probabilities using the Magma [42] computer algebra package. The probabilities, and corresponding running time for the testing of final candidates are given in Table 3.8. Monte Carlo experiments have been done to double-check the accuracy of these results. The fact that the probabilities are so small for low values of N is consistent with the argument in Section 3.10.1 that when a collision happens early, other collisions are likely to follow soon after.

One should not assume that the times for the testing the final candidates given in Table 3.8 are the dominant cost in applying this strategy. Unlike the filtering speed-ups given in Sections 3.6 and 3.10.1, the use of non-vanishing differentials seem to require more overhead in checking the conditions. So although we do not have an exact running time, we confidently surmise that the use of non-vanishing differentials will reduce the time down below 2^{40} hash operations.

3.12 Concluding Remarks

The design of the alleged SecurID hash function appears to have several problems. The most serious appears to be collisions that happen far too frequently and very early within the computation. The involvement of only a small fraction of bits in the subrounds exacerbates the problem. Moreover, the redundancy of the key with respect to the initial permutation adds an extra avenue of attack. Altogether, ASHF is substantially weaker than one would expect from a modern day hash function.

Our research has shown that the key recovery attack in [5] can be sped up by more than a factor of 8, giving an improved attack with time complexity about 2^{45} hash operations. In practice, the attacker can actually obtain more information than just a single collision. We have shown evidence that with this extra information, the time complexity can be further reduced to about 2^{40} hash operations, making the attack doable by anyone with a modern PC.

Bibliography

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

[2] E. Biham and A. Shamir. Differential cryptanalysis of DES-like cryptosystems. Journal of Cryptology, volume 4, no. 1. pages 3–72, 1991.

[3] A. Biryukov and E. Kushilevitz. Improved cryptanalysis of RC5. In K. Nyberg, editor, *Advances in Cryptology — Eurocrypt '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 85–9 9, 1998. Springer Verlag.

[4] A. Biryukov and E. Kushilevitz. From differential cryptanalysis to ciphertext-only attacks. In H. Krawczyk, editor, *Advances in Cryptology — Crypto '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 72–88. Springer Verlag.

[5] A. Biryukov, J. Lano, B. Preneel. Cryptanalysis of the alleged SecurID hash function. In M. Matsui and R.J. Zuccerato, editors, Proceedings of SAC 2003, volume 3006 of *Lecture Notes in Computer Science*, pages 130–144. Springer Verlag. A longer version of this paper is available online from `http://eprint.iacr.org/2003/162`.

[6] C. Burwick, D. Coppersmith, E. D'Avignon, R. Gennaro, S. Halevi, C. Jutla, S. Matyas Jr., L. O'Connor, M. Peyravian, D. Safford, and N. Zuric. MARS - a candidate cipher for AES. IBM Corporation, June 10, 1998.

[7] S. Contini, The effect of a single vanishing differential in ASHF. sci.crypt post, 6 Sep, 2003. `http://groups.google.com.au/groups?selm=6f35025c.0309061607.26d110a6%40posting.google.com`.

[8] S. Contini, R.L. Rivest, M.J.B. Robshaw and Y.L. Yin. The security of the RC6 block cipher. v1.0, August 20, 1998. Available at `http://www.rsa.com/rsalabs/aes/`.

[9] S. Contini, R.L. Rivest, M.J.B. Robshaw and Y.L. Yin. Improved analysis of some simplified variants of RC6. In *Fast Software Encryption*, volume 1636 of *Lecture Notes in Computer Science*, pages 1–15, 1999. Springer Verlag.

[10] S. Contini and Y.L. Yin. On differential properties of data-dependent rotations and their use in MARS and RC6. Second AES Conference, 2000.

[11] S. Contini and Y.L. Yin. Fast software-based attacks on SecurID. In B. Roy and W. Meier, editors, *Fast Software Encryption*, volume 3017 of *Lecture Notes in Computer Science*, pages 454–471, 2004. Springer Verlag.

[12] D. Coppersmith. The Data Encryption Standard and its strength against attacks. In *IBM Journal of Research and Development*, 38(3), pages 243–250. Available at `http://www.research.ibm.com/journal/rd/383/coppersmith.pdf`

[13] W. Diffie and M. E. Hellman. New directions in cryptography. In *IEEE Transactions on Information Theory*, IT-22, pages 644–654, November 1976.

[14] W. Diffie and M. E. Hellman. Exhaustive cryptanalysis of the NBS Data Encryption Standard. In *Computer*, volume 10, no. 6, pages 74–84, June 1977.

[15] Electronic Frontier Foundation. *Cracking DES, Secrets of Encryption Research, Wiretap Politics & Chip Design*. O'Reilly & Associates, Sebastopol, 1998.

[16] M.H. Heys. Linearly weak keys of RC5. *IEE Electronic Letters*, Vol. 33, pages 836–838, 1997.

[17] B.S. Kaliski and Y.L. Yin. On differential and linear cryptanalysis of the RC5 encryption algorithm. In D. Coppersmith, editor, *Advances in Cryptology — Crypto '95*, volume 963 of *Lecture Notes in Computer Science*, pages 171–184, 1995. Springer Verlag.

[18] B.S. Kaliski and Y.L. Yin. On the security of the RC5 encryption algorithm. RSA Laboratories Technical Report TR-602. Available at `http://www.rsa.com/rsalabs/aes/`.

[19] L.R. Knudsen and W. Meier. Improved differential attacks on RC5. In N. Koblitz, editor, *Advances in Cryptology — Crypto '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 216–228, 1996. Springer Verlag.

[20] X. Lai, J. L. Massey, and S. Murphy. Markov ciphers and differential cryptanalysis. In D. W. Davies, editior, *Advances in Cryptology — Eurocrypt '91*, volume 547 of *Lecture Notes in Computer Science*, pages 17–38, 1991. Springer Verlag.

[21] J. Lano, private communication, 28 Oct, 2003.

[22] A.K. Lenstra, private communication, 29 Sep, 2004.

[23] H. Lipmaa and S. Moriai. Efficient algorithms for computing differential properties of addition. In M. Matsui, editor, *Fast Software Encryption*, volume 2355 of *Lecture Notes in Computer Science*, pages 336–350, 2002. Springer Verlag.

[24] H. Lipmaa, J. Wallén, and P. Dumas. On the additive differential probability of exclusive-or. In B. Roy and W. Meier, editors, *Fast Software Encryption*, volume 3017 of *Lecture Notes in Computer Science*, pages 317–331, 2004. Springer-Verlag.

[25] M. Matsui. Linear cryptanalysis method for DES cipher. In T. Helleseth, editor *Advances in Cryptology — Eurocrypt '93*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397, 1994. Springer Verlag.

[26] V. McLellan. SecurID crypto. sci.crypt post, 10 July, 2000.

[27] V. McLellan. Newsgroup post, 10 August, 2004. `http://www.webservertalk.com/archive268-2004-8-310475.html`.

[28] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography* CRC Press, 1996.

[29] S. Moriai, K. Aoki, and K. Ohta. Key-dependency of linear probability of RC5. March 1996.

[30] S. Murphy. Comments on *Case Studies in Symmetric Key Cryptography*. June 2005.

[31] National Bureau of Standards. *Data Encryption Standard* US Department of Commerce, FIPS pub 46, January 1977.

[32] National Institute of Standards and Technology Announcing development of a federal information processing standard for advanced encryption standard. Available at `http://csrc.nist.gov/CryptoToolkit/aes/pre-round1/aes_9701.txt`.

[33] R.L. Rivest. The RC5 encryption algorithm. In B. Preneel, editor, *Fast Software Encryption*, volume 1008 of *Lecture Notes in Computer Science*, pages 86–96, 1995. Springer Verlag.

[34] R.L. Rivest, M.J.B. Robshaw, R. Sidney and Y.L. Yin. The RC6 block cipher. v1.1, August 20, 1998. Available at `http://www.rsa.com/rsalabs/aes/`.

[35] R.L. Rivest, M.J.B. Robshaw, and Y.L. Yin. The case for RC6 as the AES. Available at `http://csrc.nist.gov/CryptoToolkit/aes/round2/comments/`.

[36] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. volume 21 of *Communications of the ACM*, pages 120–126, 1978.

[37] SecurID product description. Available at `http://www.rsasecurity.com/node.asp?id=1156` .

[38] A.A. Selcuk. New results in linear cryptanalysis of RC5. In S. Vaudenay, editor, *Fast Software Encryption*, volume 1372 of *Lecture Notes in Computer Science*, pages 1–16, 1998. Springer Verlag.

[39] C.E. Shannon. Communication theory of secret systems. volume 28 of *Bell Systems Technical Journal*, pages 656–715, 1949.

[40] I.C. Wiener. Sample SecurID token emulator with token secret import. post to BugTraq, 21 Dec, 2000. `http://archives.neohapsis.com/archives/bugtraq/2000-12/0428.html`.

[41] *Tips on reassigning SecurID cards and requesting new securID cards*, AMS Newsletter, March 2002, Issue No. 117. Available at `http://www.utoronto.ca/ams/news/117/html/117-5.htm`.

[42] The Magma Computer Algebra Package. Information available at `http://magma.maths.usyd.edu.au/magma/`.

Key Expansion Algorithm for RC5 and RC6

RC5 and RC6 both have the same key expansion algorithm. The algorithm uses two constants

$$\begin{aligned} P_w &= \text{Odd}((e-2)2^w) \\ Q_w &= \text{Odd}((\phi-1)2^w) \end{aligned}$$

where e is the base for natural logarithms and $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio. The function $\text{Odd}(x)$ rounds x to the nearest odd integer. Pseudo code for the key expansion is given in Table 9.

Key expansion for RC5 and RC6 ($w/r/b$)

Input: $b-$byte key stored in array K.
Number of rounds subkeys required, t
($t = 2r + 2$ for RC5 and $t = 2r + 4$ for RC6).

Output: $w-$bit subkeys stored in $S[0, \ldots, t-1]$.

Procedure:

$u = w/8$
$c = \lceil b/u \rceil$

/* load user key into $c-$word array L */
for $i = 0$ **to** $c - 1$ **do**
    $L[i] = 0$
for $i = b - 1$ **downto** 0 **do**
    $L[i/u] = (L[i/u] \lll 8) + K[i]$

/* initialize $t-$word array S */
$S[0] = P_w$
for $i = 1$ **to** $t - 1$ **do**
    $S[i] = S[i-1] + Q_w$

/* mix S with L */
$i = j = 0$
$A = B = 0$
do $3 * \max(t, c)$ **times**
    $A = S[i] = (S[i] + A + B) \lll 3$
    $B = L[j] = (L[j] + A + B) \lll (A + B)$
    $i = (i + 1) \bmod t$
    $j = (j + 1) \bmod c$

Table 9: Key expansion for RC5 and RC6.

Analyzing Precomputation Tables in SecurID

Using computer experiments, we were able to exhaustively search for valid entries in the precomputed table up to $N = 6$ for 2-bit vanishing differentials (Table 3.1) and up to $N = 4$ for 4-bit differentials (Tables 3.9 and 3.9 in Section 3.9). It was predicted in [5] that the size of the table gets larger by a factor of 8 as N grows and it may take up to 2^{44} steps and 500GB memory to precompute the table for $N = 12$.

Here we make an attempt to derive the entries in the table analytically when $N = 1$. If we could extend the method to $N > 1$, we may be able to enumerate the entries analytically without expensive precomputation and storage.

Recall from Section 3.1.3 the details of the subround function. To find a condition for a collision, we will require constraints for the values in the subround $i-1$. So for simplicity, we will omit the superscript $i-1$ from now on. Using the definition of f, one can derive the exact conditions for a collision which appeared as Equation (6) in [5]:

$$B'_4 = ((((B_0 \ggg 1) - 1) \ggg 1) - 1) \oplus B_4, \quad (6)$$

$$B'_0 = 100 - B_4 \ . \quad (7)$$

We first note that B_0 and B'_0 have to be different in the first bit since a collision can only happen if each plaintext undergoes a different transformation. Therefore, there is at least one bit difference in (B_0, B'_0). The other bit difference can be placed either in the remaining 7 bits of (B_0, B'_0) or any of the 8 bits in (B_4, B'_4).

Rewriting Equation 6, we have

$$B_0 = (((B_4 \oplus B'_4) + 1) \lll 1) + 1) \lll 1.$$

Since there is at most one bit difference in (B_4, B'_4), it can only take on 9 possible values: 0 (for no bit difference) or 2^i (for one bit difference in bit i). Below, for each possible value of (B_4, B'_4), we enumerate the possible values of (B_0, B'_0). During the enumeration, we also take into consideration the additional requirement that the two bits in b where the differences occur must be the same (See Section 3.5).

- If $B_4 \oplus B'_4 = 0$, then $B_0 =$`0x06`. Since there is no bit difference in (B_4, B'_4),

we know that B_0 and B'_0 differ in two bits – one of them must be the first bit, and the other can be any of the remaining 7 bits.

$B_4 \oplus B'_4$	B_0	B'_0	k_0
0x00	0x06	0x87, 84, 82, 8e, 96, a6, c6	0

The exact value of $B_4 = B'_4$ is then determined by Equation 7. The additional requirement rules out two possible values of B'_0 (0x84, 0x82), leaving 5 possible combinations.

- If $B_4 \oplus B'_4 = 2^i$, then there is only one bit difference in (B_0, B'_0), which is the first bit. In this case, there is only one choice for B'_0 for each B_0.

$B_4 \oplus B'_4$	B_0	B'_0	k_0
0x01	0x0a	0x8a	0
0x02	0x0e	0x8e	0
0x04	0x16	0x96	0
0x08	0x26	0xa6	0
0x10	0x46	0xc6	0
0x20	0x86	0x06	1
0x40	0x07	0x87	0
0x80	0x08	0x88	0

The additional requirement rules out every combination above except the first one (B_0 =0x0a and B'_0 =0x8a).

Combining the above two cases, we have $5 + 1 = 6$ pairs of (B_0, B'_0), each of which giving a valid tuple $(k_0, B_0, B_4, B'_0, B'_4)$, where k_0 is the first bit of B_0.

Finally, note that if (k_0, a, b, c, d) is a valid tuple, than (k_0, c, d, a, b) is also a valid tuple. For example, if (0, 0x06, 0xdd, 0x87, 0xdd) is valid, then (0, 0x87, 0xdd, 0x06, 0xdd) is also valid. Therefore, the table consists of a total of $2 \times 6 = 12$ entries. These entries match the results from our simulation.

So far, we have been unable to extend this analysis for higher values of N.

Magma Code for Computing Non-Vanishing Differentiali Probabilities

Below is the Magma code that was used to compute the probabilities from Section 3.11 above.

```
/* this code assumes  j >= 7 */
j := 13;
cardinality_sets := 24 + (j-7)*2;
for N in [1..6] do
    w0w4size := 2*N + 14;
    c := [1: i in [2..j]];
    sum := 0.0;
    done := false;
    while not done do
        bits_taken := &+c;
        if bits_taken le w0w4size then
            prod1 := &*[ Binomial( 4, c[i] ) : i in [1..6] ];
            if j gt 7 then
                prod2 := &*[ Binomial( 2, c[i] ) : i in [7..j-1] ];
            else
                prod2 := 1;
            end if;
            prod3 := Binomial( 64 - cardinality_sets, w0w4size - bits_taken );
            sum +:= prod1*prod2*prod3;
        end if;
        index := 1;
        c[index] +:= 1;
        while ((index le 6 and c[index] eq 5) or
          (index ge 7 and c[index] eq 3)) do
            /* there is a 'carry' in the counter */
            c[index] := 1;
            index +:= 1;
```

```
            if index ge j then
                done := true;
                break;
            end if;
            c[index] +:= 1;
        end while;
    end while;

    prob := sum/Binomial(64, w0w4size );
    print "Probability is 2^",Log(prob)/Log(2), "for N = ",N;
end for;
```

Index

Acknowledgements

I would first like to thank my coauthors, Ron Rivest, Matt Robshaw, and Yiqun Lisa Yin, for allowing me to include work that I did jointly with them.

I owe a special thanks to Igor Shparlinski, for allowing me to work on this thesis while being employed under him.

I am very grateful to my PhD core committee of Andries Brouwer, Arjen Lenstra, Sean Murphy, Henk van Tilborg, and Yiqun Lisa Yin. Not only did they provide extremely valuable comments that improved the overall quality of this work, but they did so in a very timely manner. In fact, although I was confident about my original submission, I had some ideas in the back of my mind of a few subtle issues that may deserve a bit more attention. To my surprise, the committee also recognized these issues, as well as a few others. Thanks to their feedback, I feel much more assured about the final version of this work.

Of the people mentioned, two people deserve special recognition: Arjen Lenstra and Yiqun Lisa Yin. My career as a cryptography researcher never would have taken off had I not had the opportunity to work with Arjen Lenstra in the summers of 1994 and 1995. Since then, he has provided much support, and in fact, he has made this PhD possible. I am similarly indebted to Yiqun Lisa Yin. Everything from this thesis has been joint work with her. I must emphasize that Yiqun was the lead on differential cryptanalysis of RC6: my involvement was proving lemmas and theorems that were used to evaluate the differential and characteristic probabilities, as well as some other contributions at a higher level. Thus, this thesis contains many of her ideas and contributions.

Anita Klooster's invaluable assistance with filling out forms and, most importantly, the printing process of this thesis, is gratefully acknowledged.

Finally, I dedicate this thesis to my parents, for pointing me the right direction in life, putting up with me (especially my anti-Republican political viewpoints), and providing years of support.

Samenvatting

Deze thesis gaat over twee symmetrische cryptografische functies die allebei van groot praktisch belang zijn – het RC6 blokcijfer en de SecurID hashfunctie. RC6 was een van de uiteindelijke 5 kandidaten voor de Geavanceerde Encryptie Standaard (AES). Het werd ontwikkeld door het bedrijf RSA Security, en zal waarschijnlijk wereldwijd worden gebruikt in talloze producten. SecurID is een apparaatje waarmee eenmalige passwords worden gegenereerd voor authentificatie doeleinden. Het wordt door RSA Security geproduceerd en gebruikt door meer dan 13 miljoen mensen in meer dan 80 landen.

Dit proefschrift concentreert zich op de veiligheidsanalyse van RC6 en het gebrek aan veiligheid van SecurID. Het RC6 blokcijfer werd ontworpen door cryptografische deskundigen en ondersteund door een uitgebreide openbare veiligheidsanalyse. Een deel van die analyse wordt gepresenteerd in het tweede hoofdstuk van dit proefschrift. Er wordt aangetoond dat RC6 voldoende weerstand biedt tegen differentiële cryptanalyse, wat een belangrijk onderdeel is van de volledige veiligheidsanalyse van RC6. In tegenstelling tot de manier waarop RC6 werd ontwikkeld, werd de SecurID hashfunctie in het geheim ontworpen, en de resulterende methode werd nooit openbaar gemaakt. Het apparaat werd evenwel met succes 'reverse engineered', waarna een nauwkeurige beschrijving van de methode op het Internet werd gepubliceerd. Zwakheden in het ontwerp werden toen spoedig aangetoond. Het derde hoofdstuk van dit proefschrift beschrijft nieuwe resultaten die de veiligheid van SecurID op een praktisch realiseerbare manier aantasten.

Curriculum Vitae

Scott Contini completed his Bachelor degree in Computer Science and Mathematics at Purdue University, a Master's degree in Computer Science at the University of Wisconsin-Milwaukee, and a Master's degree in Mathematics at the University of Georgia. Since then, he has worked at RSA Data Security, the University of Sydney, Motorola, and now the Macquarie University. His research emphasis is on applied mathematics to solve real world engineering problems, especially in cryptology. He also enjoys research in number theory, discrete mathematics, and algorithms.