REBOUND ATTACKS ON GRϕSTL

Student Name: KOMAL KOCHAR

IIIT-D-MTech-CS-IS-11-001
Dec 6, 2011

Indraprastha Institute of Information Technology
3rd Floor, Library Building
NSIT Campus, Dwarka, Sector 3
New Delhi - 110078

Thesis Committee
Somitra Sanadhya (Chair)
Shishir Nagaraja
Gaurav Gupta

Submitted in partial fulfillment of the requirements
for the Degree of M.Tech. in Computer Science,
with specialization in Information Security

Abstract

Cryptographic hash Functions are widely used for a wide range of applications such as authentication of information, digital signatures and protection of pass-phrases. In the last few years, the cryptanalysis of hash functions has gained much importance within the cryptographic community. In 2004 a series of attacks by Wang et al. [19, 20] have exposed security vulnerabilities in the design of the most widely deployed SHA-1 hash function. As a result, the US National Institute for Standards and Technology (NIST) recommended the replacement of SHA-1 by the SHA-2 hash function family and in 2008, they announced a call for the design of a new SHA-3 hashing algorithm.

On October 31, 2008, the "SHA-3 competition", organised by the National Institute of Standards and Technology (NIST), was launched [17]. 64 algorithms were submitted, out of which, 51 were accepted for the first round of the competition. On July 24, 2009, 14 candidates were chosen by NIST to advance to the second round of the competition. One of the candidates accepted for the second round is called Grϕstl [11], developed by Praveen Gauravaram, Lars R. Knudsen and Krystian Matusiewicz. Grϕstl further advanced to the final round along with BLAKE [2], JH, Keccak [3], Skein [10] and became one of the top 5 proposals for SHA-3.

The report breifly specifies the Grϕstl family of cryptographic hash algorithms, one of the top 5 finalists of the SHA-3 hash function competition and a well known attack named Rebound Attack on Grϕstl. The rebound attack is a freedom degrees utilization technique that was first proposed by Mendel et al. in [15] as an analysis of round-reduced Grϕstl and Whirlpool [18]. The main idea of the rebound attack is to use the available degrees of freedom in a collision attack to effeciently bypass the low probability parts of a truncated differential trail. The rebound attack consists of an inbound phase with a match-in-the-middle part to exploit the available degrees of freedom, followed by a subsequent probabilistic outbound phase. Report discusses available rebound attacks on reduced rounds of Grϕstl-256.

The report first describes a simple method to utilize the available freedom degrees. The original idea of rebound is then applied to reduced rounds of Grϕstl- 256. Report describes attack on 4 rounds of Grϕstl-256. It further explains same rebound technique applied on 5 and 6 rounds Grϕstl-256. The new technique Super Sbox Cryptanalysis [12] introduced by Thomas Peyrin and Henri Gilbert is explained in the report alongwith its application on 7 rounds of Grϕstl-256.

Acknowledgments

I wish to express my sincere gratitude to Prof. Pankaj Jalote, Director at Indraprastha Institute of Information Technology, Delhi and Security and whole Security and Privacy Group of IIIT-Delhi for providing me an opportunity to do my project work on REBOUND ATTACKS ON GRϕSTL. This project bears on imprint of many peoples. I sincerely thank to my project guide Dr. Somitra Sandhya, Assistant Professor at Indraprastha Institute of Information Technology, Delhi for guidance and encouragement in carrying out this project work. I would also like to thank all people had many interesting research discussions. Especially, thanks to Doonhoon Chang and Shuang Wu and Thomas Peyrin for answering conceptual queries on mail. I would also like to thank all my friends for their welcome distractions in my spare time. This important distance allowed me to focus on research without losing the fun of it. Last but not least I wish to avail myself of this opportunity, express a sense of gratitude and love to my friends and my beloved parents for their manual support, understanding, and simply for everything.

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

This report starts with an introduction to cryptographic hash functions. It further introduces the important terms associated with hash functions, such as iterated hash functions, Merkle Damgard Strengthening [4] used for iterated hash functions, block cipher based and permutation based compression functions. Chapter further explains the security requirements of hash functions that is collision resistance, preimage resistance, second preimage resistance. It further gives a brief overview of Grϕstl [11] and a well known rebound technique used to attack the compression function used in hash functions. Rebound technique can be applied to both block cipher based compression function and permutation based compression function. Original idea of rebound is to avail the degrees of freedom to do a collision [13] or a semi-free-start collision attack. Rebound technique is one of the differential cryptanalytic technique, using truncated differential trial. As Grϕstl uses AES like permutation for its compression function, rebound technique applied on Grϕstl uses the differential properties of operations of AES permutaion. Rebound technique along with an introduction to the differential properties used for attack are explained next.

1.2 Hash Functions

Hash functions are a building block for numerous cryptographic applications. Cryptographic hash functions are used in a variety of applications like digital signatures, authentication schemes and message integrity. Hash function are defined as one way functions, that com-

preses a message of arbitrary length to a result of fixed length. Informally, a hash function H is a function that takes an arbitrarily long message as input and outputs a fixed-length hash value of size n bits. There are two types of hash functions keyed and unkeyed. The keyed hash functions are often called as Message authentication code or MAC. In case of unkeyed hash fnctions, the message digest must be securely kept so that it cannot be altered whereas in a keyed hash function, you have to transfer the key along with the message. The classical security requirements for one way hash functions are collision resistance and (second)- preimage resistance. Namely, it should be impossible for an adversary to find a collision (two distinct messages that lead to the same hash value) in less than $2^{n/2}$ hash computations, or a (second)-preimage (a message hashing to a given challenge) in less than 2^n hash computations.

1.2.1 Iterated hash function

In the literature, most of the hash functions are iterated hash functions, i.e., they are based on an easily computable function h(·,·) from two binary sequences of respective lengths m and l to a binary sequence of length m. The message M is split into blocks X_i of l bits or X = (X_1,X_2,.....,X_n). If the length of R is not a multiple of l, X is padded using an unambiguous padding rule. The value H_n of length m is obtained by computing iteratively

H_i = h(H_{i-1},X_i) i = 1, 2, ... ,t

where H_0 is a specified initial value denoted by IV . Iterated hash function along with output transformation is shown in Fig. 1.1. The function h is called the compression function or the hash round function. The hash result is then obtained by applying an output transformation(basically an identity function) g to H_n, or Hash(IV,M) = H = g(H_t).

1.2.2 MD strengthening

Iterated hash functions are vulnerable to extension attacks(extending the message with another block to generate its hash). Merkle and Damgard proposed an operation to be applied on iterated structure to make hash function immune to extension attack. MD strengthening

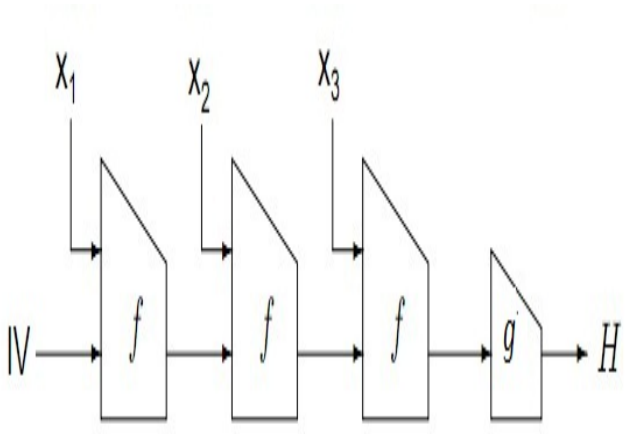

Figure 1.1: Iterated hash function

makes the hash function theoretically secure. The major objective of the Merkle-Damgard iteration is collision security preservation- showing that the Merkle-Damgard hash function is collision secure when the underlying compression function is assumed to be also collision secure. The preservation is achieved by applying the Merkle-Damgard-strengthening [4]: a suffix-free message padding function used in conjunction with a fixed initialization value IV. In Merkle-Daamgard strengthening an extra message block or padding is done to make it collision secure. In MD strengthening, few 0 bits are added to make last block x_t of message of length l. An extra block x_{t+1} containing length of original message in bits is added. An extra block addition puts restriction on the length of the message to be padded, but the restriction is not significant in real life.

1.2.3 Compression Function

Compression function used in hash function are of two types- Block cipher based compression function and Permutation based compression function [1, 14]. During introductory years of hash functions, hash functions were mostly block cipher based. As block ciphers have long been the central primitive in symmetric key cryptography, and there exists some measure of confidence in block cipher designs, thus block cipher based design is widely used for hash functions. Two major block cipher based design schemes are explained in following subsections.

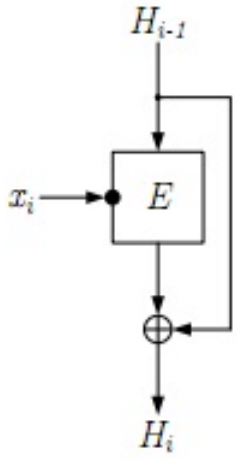

Figure 1.2: Davies Meyer Scheme- block based compression function

1.2.4 Davies-Meyer compression function

The Davies-Meyer compression function makes a simple use of the underlying block cipher E (Fig. 1.2). The input blocks x_i serve as the key to E. Thus, the block size of x_i must match the excepted key size of the specic block cipher. The previous hash-value H_i serves as the plaintext to be handled with appropriate bit-length. The output H_i is then concatenated with the previous output H_{i1} with aid of the exclusive-or operator. The final output H_t is dened by the iterated formula

$$H_i = E_{x_i}(H_{i1}) \oplus H_{i1}$$

for $1 \leq \mathrm{i} \leq \mathrm{t}$, $H_0 = \mathrm{IV}$.

1.2.5 Matyas-Meyer-Oseas compression function

The Matyas-Meyer-Oseas compression function is most widely identical to Davies-Meyer with the input x and the previous hash-value H_i interchanged (Fig. 1.3). Here, the input blocks x_i serve as plaintext to be handled and the previous hash-value H_i serve as the key to block cipher E. Due to the possible different bit-lengths k and n a function g precedes the key input of E. It maps the n-bit previous hash-value to a suitable k-bit key. The nal output H_t is dened by the iterated formula

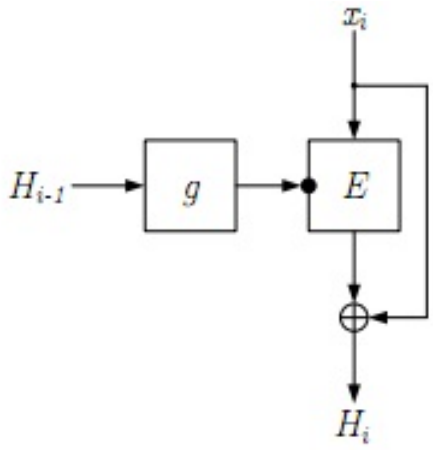

Figure 1.3: Matyas-Meyer-Oseas Scheme- block based compression function

Figure 1.4: JH- permutation based compression function

$$H_i = E_{G_{h_{I-1}}}(x_i) \oplus x_i$$

for $1 \leq \mathrm{i} \leq \mathrm{t}$, $H_0 = \mathrm{IV}$.

1.2.6 Permutation based compression functions

Traditionally, hash functions were constructed on existing block ciphers. But with advancement the cryptography, existing confidence in security of block ciphers decreased, which leads to introduction of permutation based hash functions. JH,one of the top 5 finalists uses a single permutaion as compression function(Fig. 1.4) and Grϕstl hash function is based on two permutations(Fig. 1.5)

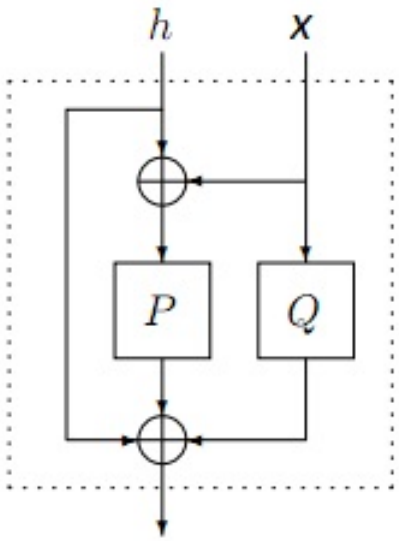

Figure 1.5: Grϕstl- permutation based compression function

1.3 Grϕstl

Informally, Grϕstl can be defined as AES based hash function with permutaion based design. Grϕstl [11] is a wide-pipe Merkle-Damgrd [4] hash algorithm with an output transformation. In Grϕstl, the iterated state is wider than the final hash output, which classifies it as a type of a wide-pipe design [5]. Grϕstl processes its inputs by first calling the compression function f iteratively, then applying a final output transformation to the state and finally truncating the result to the desired output length. The compression function f is built out of two permutations P and Q and the output transformation is designed on top of the permutation P. The compression function of Grϕstl uses two fixed 2n-bit permutations together, and produces a 2n-bit compression function. Main goal of the commpression function is to achieve collision and preimage resistance as an ideal n-bit-wide compression function. Intermediate chaining values in Grϕstl-256 are 512 bits wide; for Grϕstl-512, they are 1024 bits wide. The output transformation processes the final chaining state, and discards half the bits of the result, yielding an n-bit hash output.

The underlying fixed permutations of Grϕstl are based closely on the structure of AES, reusing the S-box, but expanding the size of the block to 512 bits (for Grϕstl-256) or 1024 bits (for Grϕstl-512).

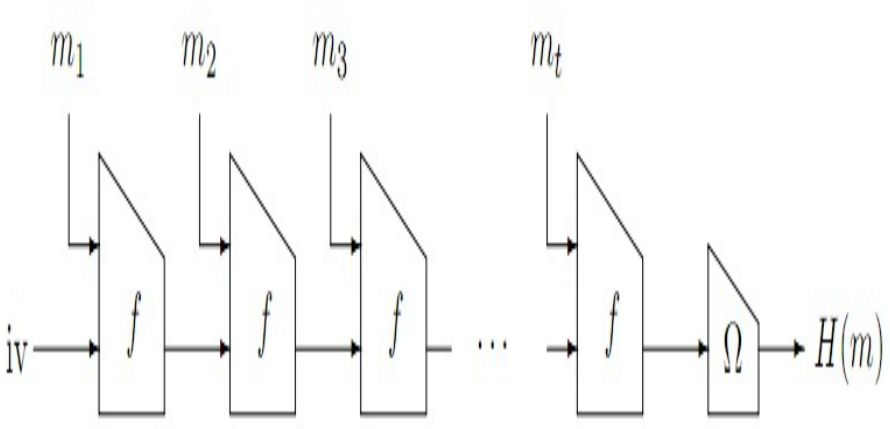

Figure 1.6: The Grϕstl hash function

1.4 Specification of Grϕstl

Grϕstl is a collection of hash functions, capable of returning message digests of any number of bytes from 1 to 64, i.e., from 8 to 512 bits in 8-bit steps. The variant returning n bits is called Grϕstl-n. As stated earlier, Grϕstl is an iterated hash function with a compression function built from two distinct permutations (see Fig. 1.7). Grϕstl is a byte-oriented SP-network which borrows components from the AES.

Major components of Grϕstl are described as follows-

1.4.1 The Grϕstl hash function

The Grϕstl hash functions iterate the compression function f as follows. The message M is padded and split into l-bit message blocks m_1, h_2, . . ., m_t, and each message block is processed sequentially. An initial l-bit value h_0 = iv is defined, and subsequently the message blocks mi are processed as [see Fig. 1.6]

$h_i \rightarrow$ f(h_{i-1},mi) for i = 1, . . . , t.

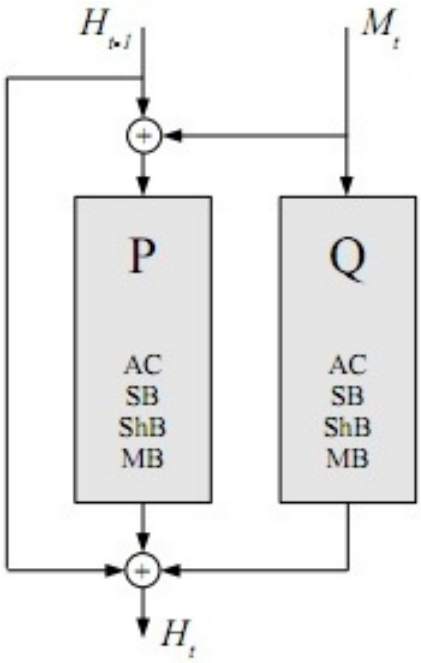

Figure 1.7: The Grϕstl compression function f

1.4.2 The Grϕstl compression function

The compression function f is based on two underlying l-bit permutations P and Q. It is defined as follows:

$$f(h,m) = P(h \bigoplus m) \bigoplus Q(m) \bigoplus h$$

The two permutations P and Q are constructed using the wide trail design strategy and borrow components from the AES. The S-box used is identical to the one used in the block cipher AES and the diffusion layers are constructed in a similar manner to those of the AES. As a consequence there is a very strong confusion and diffusion in Grϕstl.

The design of P and Q was inspired by the Rijndael block cipher algorithm. Thus, design consist of a number of rounds R, which consists of a number of round transformations. In Grϕstl, a total of four round transformations are defined for each permutation [see Fig. 1.7]. These are

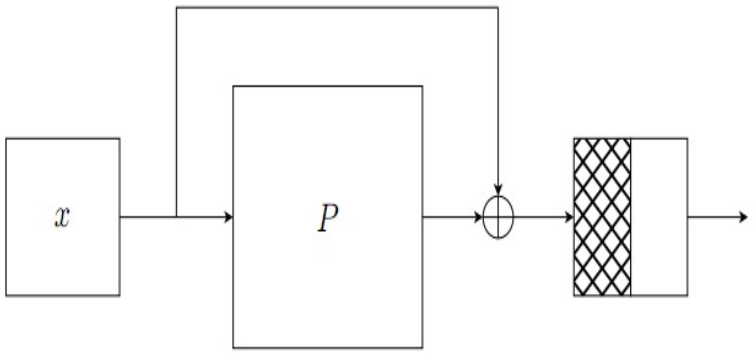

Figure 1.8: The Grϕstl output transformation

1. AddRoundConstant (AC) adds different one-byte round constants to the 8×8 states of P and Q.

2. The non-linear layer SubBytes (SB) applies the AES S-Box to each byte of the state independently.

3. The cyclical permutation ShiftBytes (SH) rotates the bytes of row j left by j positions.

4. The linear diffusion layer MixBytes (MB) multiplies the state by a constant matrix.

In each round, the state is updated by round transformation R_i as follows:

$$R_i = \text{MB} \circ \text{SH} \circ \text{SB} \circ \text{AC}$$

1.4.3 The Grϕstl output function

Let truncn(x) be the operation that discards all but the trailing n bits of x. Then the output transformation ψ is defined as [see Fig. 1.8]

$$\psi(x) = trunc_n(\text{P(x)} \bigoplus \text{x}).$$

1.5 Securirty Requirements of Hash Functions

One major security criteria for hash functions is that they should be collision-resistant, which means that given two distinct messages, M1 and M2, the probability that the corresponding hashes, H(M1) and H(M2), being equal is negligible. Due to the birthday paradox, there is a generic attack that find collisions after about $2^{n/2}$ evaluations of the hash function, where n is the size in bits of the hash values. The attack works by randomly choosing messages and computing their hash values until a collision occurs. Typically, with iterated hash functions, the size of messages blocks is often larger than the size of the hash values themselves, and this attack usually works on the compression function itself. Other important security goals for hash functions are preimage resistance and second-preimage resistance. An attack against preimage resistance is an attack that, given some target value y, finds a message M such that H(M) = y. An attack against second preimage resistance, given a message M, finds another message M1 such that H(M1) = H(M). Rebound attack explained in next sections tends to find collisions in the Grϕstl hash function.

1.6 Rebound Attack

Rebound [16] is a widely used tool in the cryptanalysis of hash functions. This was invented during the cryptanalysis of Whirlpool [18]. It basically works on the AES- based designs. Thus, rebound can be applied to wide range of hash functions such as Echo, Grϕstl, Lane, Skein, Twister. The idea of the Rebound attack is to divide an attack into two phases, an inbound and outbound phase. The inbound phase is an efficient meet-in-the-middle phase, which exploits the available degrees of freedom in the middle of a (truncated) differential path to guarantee that the expensive part of a differential path holds. In the probabilistic outbound phase the solutions of the inbound phase are computed backwards and forwards to obtain an attack on the hash or compression function.

Rebound applies to both block cipher and permutation based designs. Broadly speaking, in the Rebound attack, the internal cipher of a hash or compression function is considered as three sub-ciphers. Let E be a block cipher, then

$$\mathrm{E} = E_{fw} \circ E_{in} \circ E_{bw}.$$

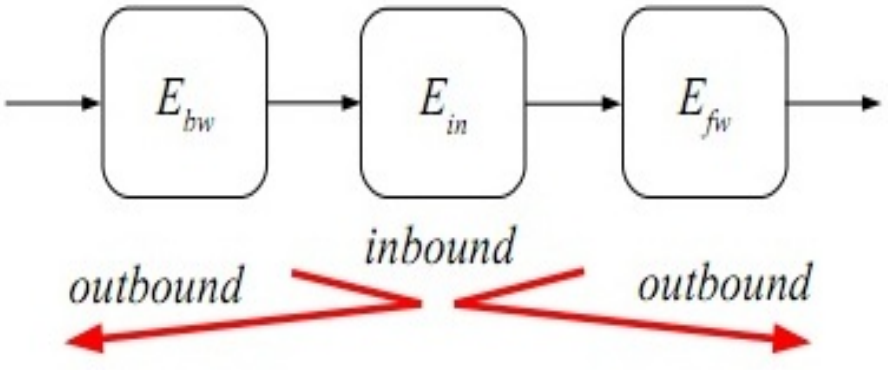

Figure 1.9: Schematic view of Rebound Attack

Alternatively, for a permutation based construction, we decompose a permutation P into three sub-permutations

$$\mathrm{P} = P_{fw} \circ P_{in} \circ P_{bw}.$$

The rebound attack can be described by two phases (see Fig. 1.9):

1.6.1 Inbound phase

It is a meet-in-the-middle phase in E_{in} (or P_{in}), which is aided by the degrees of freedom that are available to a hash function cryptanalyst. The combination of meet-in-the-middle technique and exploitation of degrees of freedom leading to very efficient matches is termed as match-in- the-middle approach. This phase usually starts with several chosen input/output differences of E_{in}(or P_{in}) that are propogated through linear layers forward and backward. Then, one can carry out meet-in-the-middle (MITM) match for differences and generate all possible value pairs validating the matches. Inbound Phase is also referred as controlled rounds as adversary can control the propogation of differences.

1.6.2 Outbound phase

In this second phase, truncated differentials in both forward and backward direction are used through E_{fw} and E_{bw} (or P_{fw} and P_{bw}) to obtain desired collisions or near-collisions. If the truncated differentials have a low probability in E_{fw} and E_{bw}, we can repeat the inbound phase to obtained more starting points for the outbound phase. Outbound Phase is also referred as uncontrolled rounds as in this phase propogation of differences can't be controlled by adversary.

In most cases, the inbound phase can be done fast due to the MITM nature and generates solution pairs with very low average complexity. Hence, attackers usually choose the position of E_{in}(or P_{in}) in the differential path so that it covers a low probability portion of the trail in order to increase the success probability of the outbound phase.

1.7 Prelimineries of the Rebound Attack

In the following, we want to briefly summarize some well known facts that will be frequently used in the subsequent sections.

1.7.1 Truncated differentials

Knudsen [9] proposed truncated differentials as a tool in block cipher cryptanalysis. In a standard differential attack, the full difference between two inputs/outputs is considered whereas in the case of truncated differentials, the differences is only partially determined, i.e. for every byte, we only check if there is a difference or not. A byte having a non-zero difference is called active.

1.7.2 Differential Properties of SubBytes

In the attacks, few differential properties of the AES S- box are used. Most of these properties can simply be veried by computing the differential distribution tables (DDT) of the

S-box (or inverse S-box).

1. For a given input (or output) difference of the AES S-box, the number of possible output (or input) differences is restricted to 127.

2. For a given input and output difference, the number of possible input values is limited to either 2 or 4 values.

3. For a given input and output difference, there are only 2 or 4 solutions per S-box.

1.7.3 Differential Properties of MixBytes

In the case of MixBytes, we use the property of an n $\times$ n MDS matrix that, given any bytes of input and output, the other bytes can be uniquely determined. Since MixBytes is linear, this also holds for differences. In the following attacks, we use differential paths with a minimum number of active S-boxes. Hence, also the number of differences in the MixBytes transformation is minimal and every active MixBytes operation contains zero differences in exactly 7 (3 for MixBytes) input/output bytes. It follows, that choosing a single byte difference uniquely determines all other 8 (4 for MixBytes) differences.

Hence, for a fixed position of active bytes, we get 255 possibilities for the difference propagation of MixBytes [8]. These cases can be precomputed and stored in tables.

1.7.4 AddColumns and ShiftRows

AddColumns and ShiftRows are linear operations. Active Bytes after applying AddColumns operations remains same as that before. Also in ShiftRows operation, number of active bytes remains same in both input and output, only active bytes shifts to new bytes after applying ShiftBytes operation.

Chapter 2

Rebound Attack on Grϕstl-256

2.1 Overview

In the last chapter we gave an introduction to hash functions and terminology associated with hash functions such as permutation based compression functions, Mekle Damgard strengthening for iterated hash functions and widepipe design. Next, we discussed a well known hash function Grϕstl based on above mentioned characteristics. Analysis of Grϕstl introduced rebound technique explained in section 1.7.

In this chapter, we will look how simple rebound technique can be used to attack Grϕstl. Document will discuss attacks on reduced rounds of Grϕstl-256 only. Chapter starts with an introduction to differential trials and describes a way to construct 4 round differential trial for Grϕstl-256. It further discussed the way to attack 4 rounds of Grϕstl using the truncated differential trial. Same rebound technique can be applied to 5 and 6 rounds of Grϕstl-256. Same technique is used to do a collision attack on 5 rounds of Grϕstl-256 and a semi-free-start collision attack on 6 rounds of Grϕstl-256(in paper [16]).

2.2 Differential Paths

As explained before, inbound phase of the rebound attack exploits the available degrees of freedom in the middle of a (truncated) differential path to guarantee that the expensive part of a differential path holds and in the probabilistic outbound phase the solutions of the

inbound phase are computed backwards and forwards to obtain an attack on the hash or compression function. Thus, before applying rebound attack on Grϕstl, we will first look how differences propogate in the differential path. As truncated differential path is used for rebound attack(see section 1.8.1), we just need to check for active bytes(bytes having a non zero difference). Differential path considered for analysis is

$$1 \rightarrow 8 \rightarrow 64 \rightarrow 8 \rightarrow 1$$

2.2.1 Notation

Since only MixBytes and SubBytes have differential properties(Section 1.8.2 and 1.8.3), these are used to denote the state. Rest two operations of the permutation, ShiftRows and MixBytes do not have differential properties, they only have diffusion property. The SubBytes layer of round i is denoted by SB_i , its input state by SB_i^{in} and its output state by SB_i^{out}. Similarly, the MixBytes layer of round i is denoted by MB_i , its input state by MB_i^{in} and its output state by MB_i^{out}. The first and second column show differences at the input and output of the S-boxes (SB_i^{in} and SB_i^{out}), and column three and four show differences at the input and output of the MixBytes transformations (MB_i^{in} and MB_i^{out}).

2.2.2 4 round Truncated Differential Trial

In this section, we will see how differences propogate across different operations of permutation and also paths which follow the differential trial($1 \rightarrow 8 \rightarrow 64 \rightarrow 8 \rightarrow 1$). For this trial, we will start with MixBytes operation of round 2 MB_2. Complete trial is shown in Fig. 2.1.

1. For first column of MB_2, 7 input byte differences are required to be made zero, if we choose a single difference, all other differences of column 1 of MB_2^{out} can be uniquely determined. Thus we have 255 choices for fixing first byte of column1 of MB_2^{out}, or we can choose first byte difference of column 1 of SB_3^{in}, as AddColumns operation is a linear operation. Also differences of column 1 of MB_2^{in} will propogate linearly to SB_2^{out} as inverse ShiftBytes is a linear operation.

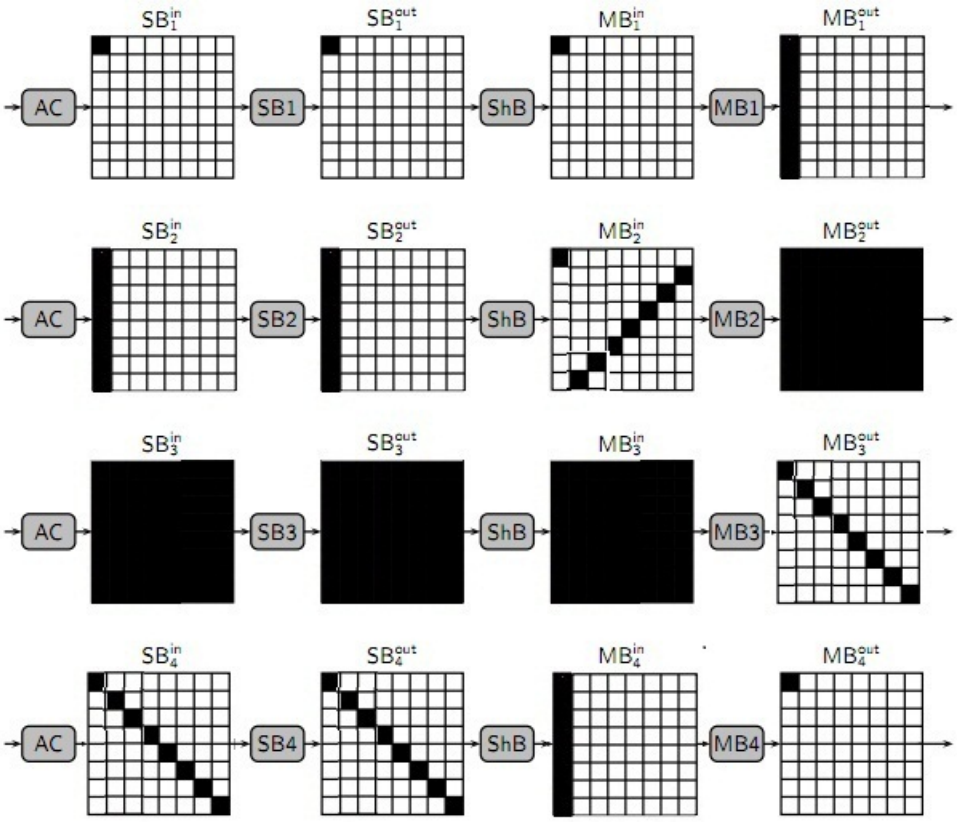

Figure 2.1: Truncated Differential paths

2. Next, we move on with column 1 of MB_3. Same as above, for first column of MB_3, 7 input byte differences are required to be made zero, if we choose a single difference, all other differences of column 1 of MB_3^{in} can be uniquely determined. Differences of column 1 of MB_3^{in} will propogate linearly to SB_3^{out} as inverse ShiftBytes is a linear operation. As first byte of SB_3^{out} has already been fixed from above, we have 127 choices for fixing first byte of column1 of MB_3^{out}, or we can choose first byte difference of column 1 of SB_4^{in}, as InverseAddColumns operation is a linear operation.

3. Next, we start with second column of MB_2, 7 input byte differences are required to be made zero, if we choose a single difference, all other differences of column 2 of MB_2^{out} can be uniquely determined. Also differences of column 2 of MB_2^{in} will propogate linearly to SB_2^{out} as inverse ShiftBytes is a linear operation. Due to differential behaviour of sbox(explained in section 1.8.2) will have 127 choices for fixing first byte of column 2 of MB_2^{out}, or we can choose first byte difference of column 2 of SB_3^{in}, as AddColumns operation is a linear operation.

4. Next, we move on with column 2 of MB_3. Same as step 2, for second column of MB_3, 7 input byte differences are required to be made zero, if we choose a single difference, all other differences of column 2 of MB_3^{in} can be uniquely determined. Differences of column 2 of MB_3^{in} will propogate linearly to SB_3^{out} as inverse ShiftBytes is a linear operation. As first byte of SB_3^{out} has already been fixed from above, we have 64 choices for fixing first byte of column2 of MB_3^{out}, or we can choose first byte difference of column 2 of SB_4^{in}, as InverseAddColumns operation is a linear operation.

5. Similarly, above 2 steps are repeated for each column. The approximate number of possible S-box dierences are halved for each additional MixBytes column. From above steps, approximate number of choices are around 2^{64}.

In these we have found differences for SB_2^{in}, SB_2^{out}, SB_3^{in}, SB_3^{out}. It is the middle differential path which is exploited by attacker as an inbound phase for attacking Grϕstl.

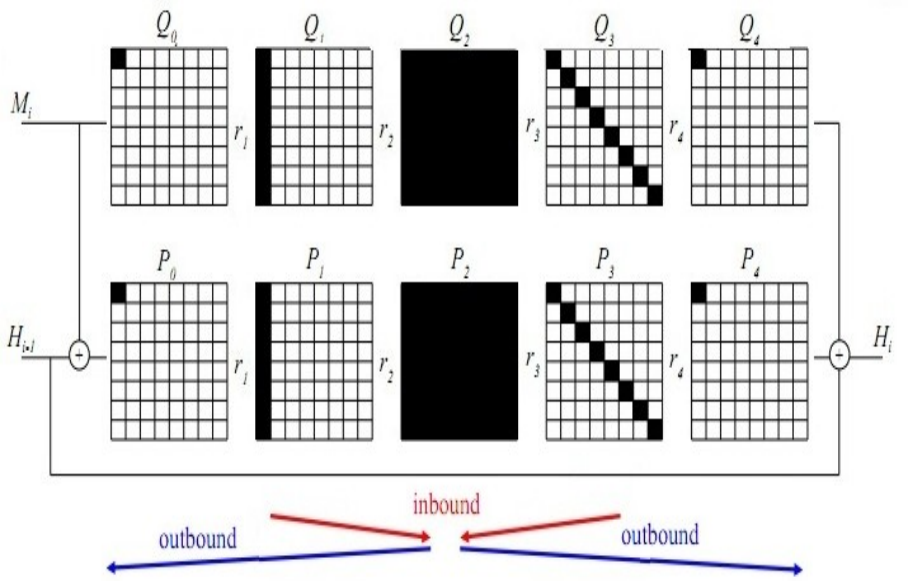

Figure 2.2: Rebound Attack on 4 rounds Grϕstl-256

2.3 Rebound Attack on 4 rounds Grϕstl-256

As stated above, main idea of Rebound Attack is to start from the middle of the truncated differential path and then propogate outwards. Thus, middle part of the differential trial is solved for the differences and values by exploiting available degrees of freedom(Inbound phase), and then differences and values are propgated outwards probabilistically(Outbound phase). This attack is based on the differential path explained in the previous section. As we have seen the truncated differential path that is used to attack 4 rounds of Grϕstl has a full active state in the middle and low active bytes on ends. The detailed path is given in Fig. 2.2 and the sequence of active bytes between each round r_i is as follows:

$$1 \rightarrow^{r_1} 8 \rightarrow^{r_2} 64 \rightarrow^{r_3} 8 \rightarrow^{r_4} 1$$

Next, we explain how the trial is being constructed. The main idea is to search for differences according to the 4-round middle part ($1 \rightarrow 8 \rightarrow 64 \rightarrow 8 \rightarrow 1$) of the path. Phases of the attack are explained in subsequent subsections.

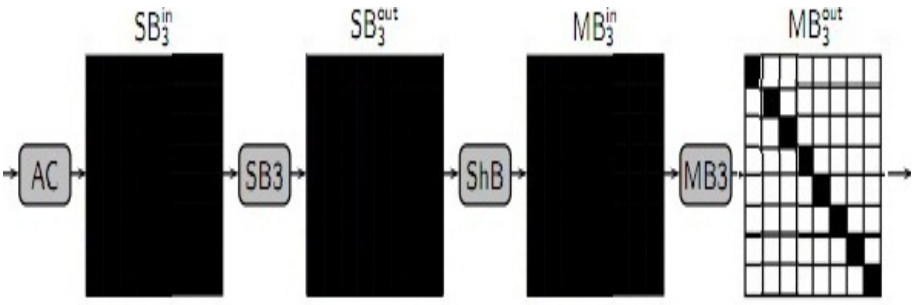

Figure 2.3: Phase 1 of the Attack on 4 rounds Grϕstl-256

2.3.1 Inbound Phase

The rebound attack is started with the inbound phase in round r_2 and r_3. It follows the truncated differential path explained above and deterministically propagate to the full active SubBytes layer in the middle. This phase starts with 8 input/output differences of MB_2^{out} and SB_4^{in} and propogates through linear layers and match at SB_3. Inbound Phase can be further subdivided into three phases.

1. In Phase 1, we randomly select 8 byte differences for SB_4^{in} and propgate backwards to MB_3^{out}. The output differences will remain same as inverseAddColmns is a linear operation. Propogating backwards and applying inverseMixBytes operation, all the 64 bytes would get active and we will obtain 64 byte differences for MB_3^{in}. After applying inverseShiftBytes, differnces will propogate linearly to SB_3^{in}. Next is an inverseSBox operation, which is not linear, so we will compute all valid byte pairs for SB_3^{in} using SBox differential tables..ref.. Detailed trial is shown in Fig. 2.3

2. In phase 2, we will choose byte differences of SB_3^{in}. these byte differences will propogate linearly to MB_3^{in}(inverseAddColumns is a linear operation). These difference must be chosen such that each of the such that after applying inverseMixBytes, we obtain only 8 active bytes at MB_2^{in}. Thus, for each column of SB_3^{in} (or MB_2^{out} inverse MixBytes table is used to choose each byte difference of SB_3^{in}, such that we obtain single active byte at MB_2^{in}(see Fig. 2.4). For each of the 255 differences each column of inverse MixBytes table, we can choose each byte of SB_3^{in} from 127 possible differences, thus probabilty of success is 127/255, that is 1/2 approx.Thus for each column, that is 8 bytes, probability would be 2^{-8}. Since we had 255 input differences for each column of MB_2, probability of finding solution for a column is

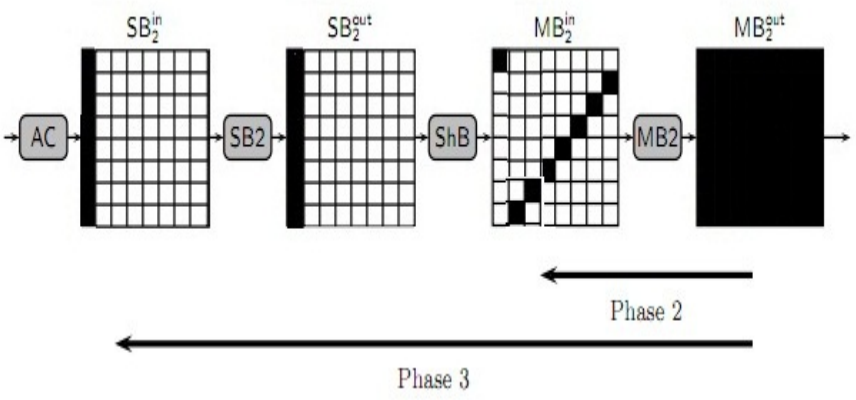

Figure 2.4: Phase 2 and 3 of the Attack on 4 rounds Grϕstl-256

$1(1(127/255)^8)^{255} \simeq 0.\ 62$. For 8 columns, it would be $(0.62)^8 \simeq 2^{-5.5}$. Thus phase 1 needs to be repeated $2^{5.5}$ times to obtain solution following the trial.

3. Phase 3 - From above phase we have obtained differences for SB_2^{in}. These differences will propogate backwards linearly to SB_2^{out}(inverseShiftBytes being linear operation). Moving backwards, 8 byte differences will be obtained by inverseSubBytes operation. Now it looks like differences in SB_2^{in} and SB_4^{out} can't be chosen anymore. But, Thomas Peyrin has made an observation in paper [12] that differences of active bytes located at row j of SB_2^{in} depend only on byte pairs of column j of SB_4^{in}. Thus one solution for each column of SB_3^{in} provides 2^8 valid candidates.

2.3.2 Outbound Phase

The uncontrolled phase is probabilistic. In the outbound phase, we probabilistically propagate the pairs of the inbound phase outwards, to match the differences at the input and output of the permutations. In the backward direction (from MB_1^{in} to SB_1^{in}), the probability to follow the path is almost one. In the forward direction (from SB_4^{in} to MB_4^{out}) the probability for the propagation from 8 to 1 active byte through the MixBytes transformation in round r_4 is $2^{-8\times7} = 2^{-56}$. Hence, we can construct one pair conforming to the truncated differential path for each of P and Q with a complexity of 2^{56}.

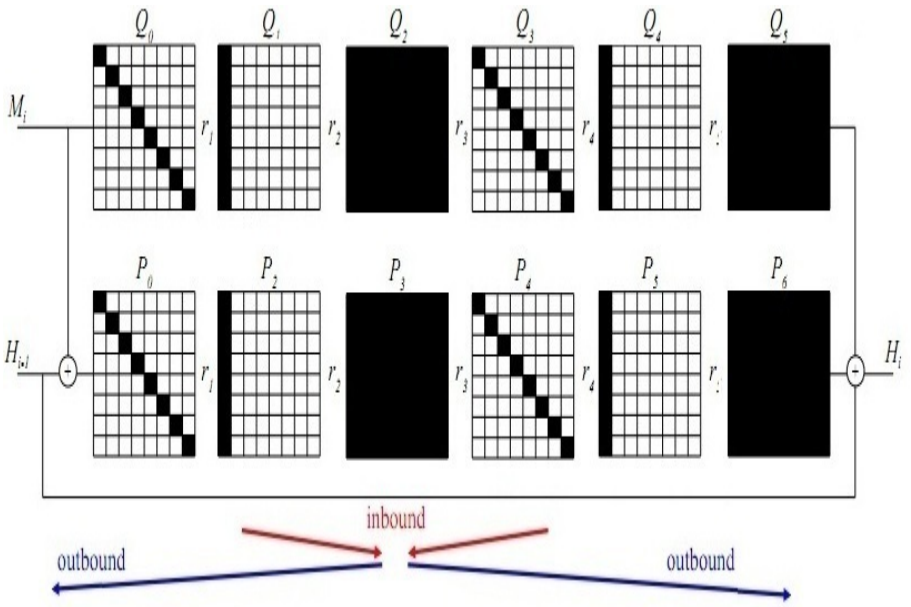

Figure 2.5: Rebound Attack on 5 rounds Grϕstl-256

2.4 Rebound Attack on 5 rounds Grϕstl-256

The Rebound technique dissussed in section 1.7 or applied in above section can also be used to attack 5 rounds of Grϕstl-256. The collission attack on 5 rounds of the Grϕstl-256 compression function, explained by Kota Ideguchi and Bart Preneel in paper [13], uses same technique. The truncated differential path used to attack has high number of active bytes in the middle and in the end. The truncated differential path is (see Fig.2.5)

$$8 \rightarrow^{r_1} 8 \rightarrow^{r_2} 64 \rightarrow^{r_3} 8 \rightarrow^{r_4} 8 \rightarrow^{r_5} 64$$

Same truncated differntial path is used for both the permutations P and Q. Both the phases-Inbound phase and Outbound phase of rebound attack on 5 rounds of Grϕstl is explained as follows. Complete truncated differential path showing all active bytes differences after each operation of permutation is shown in Fig. 2.6.

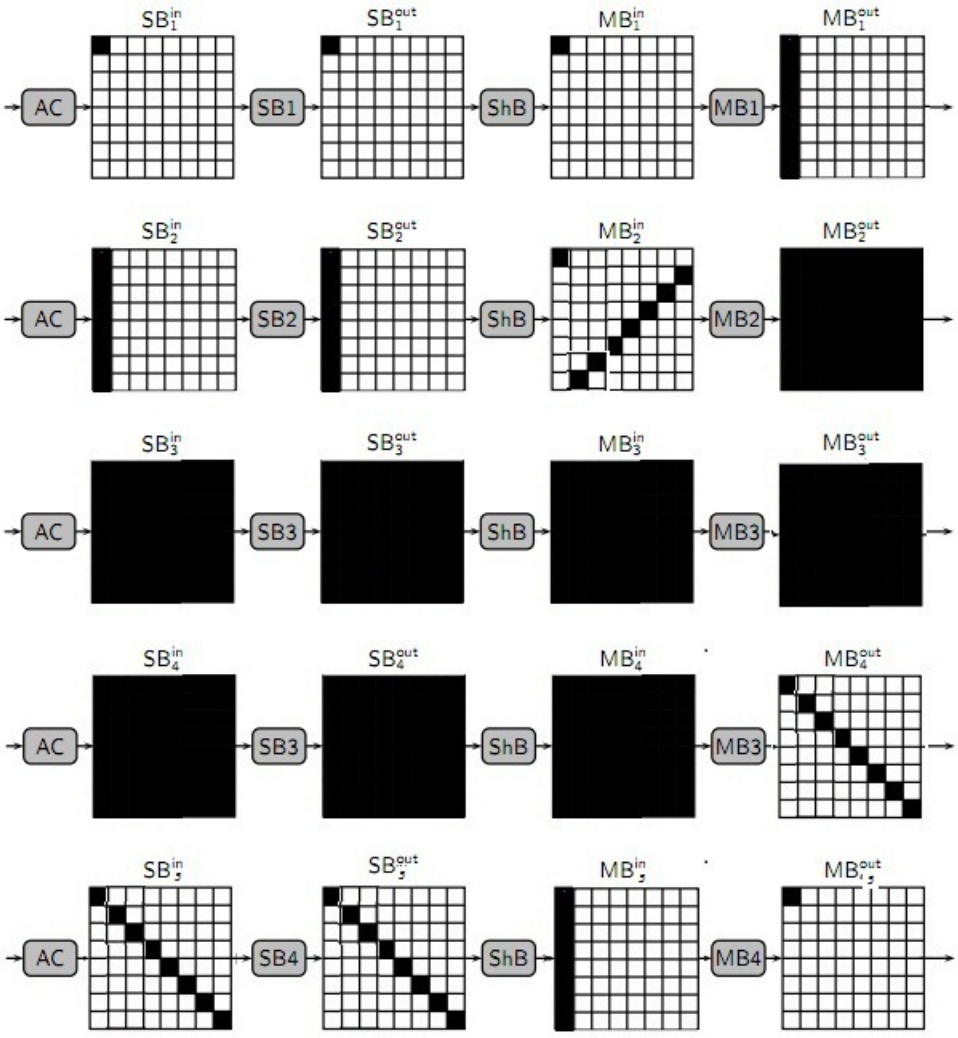

Figure 2.6: Truncated Differential trial for 5 rounds Grϕstl-256

2.4.1 Inbound Phase

The rebound attack starts with the inbound phase in round r_2 and r_4. It starts from the input of 4th round SubBytes(SB_4^{in}) and input of 1st round MixBytes(MB_1^{out}) and follows the truncated differential path explained above and deterministically propagate to the full active SubBytes layer in the middle. This phase starts with 8 input/output differences of MB_1^{out} and SB_4^{in} and propogates through linear layers and match at SB_3. The average time complexity to generate an internal state pair which follows the path of the controlled phase is one.

2.4.2 Outbound Phase

The remaining steps of the path are probabilistic. For the forward direction (from the SB_4 to MB_5^{out}), the probability to follow the path is almost one. For the backward direction (from MB_1^{out} toSB_1^{in}), it takes 2^{16} computations.

2.5 Rebound Attack on 6 rounds Grϕstl-256

The same rebound technique which is used for attacking 4 rounds and 5 rounds of Grϕstl-256 can also be applied to attack 6 rounds. The truncated differential path used for this attack has high number of active bytes in the middle and a low number of active bytes at the input and output of each permutation P and Q. This is done to control degrees of freedom in the inbound phase and the rest rounds of the trial are probabilistic. This rebound attack on 6 rounds of Grϕstl can also be used to construct semi-free-start collission for Grϕstl-256(explained by Florian Mendel in paper [16]). This attack follows the following sequence:

$$8 \rightarrow^{r_1} 1 \rightarrow^{r_2} 8 \rightarrow^{r_3} 64 \rightarrow^{r_4} 8 \rightarrow^{r_5} 8 \rightarrow^{r_6} 64$$

The truncated differential path followed for the attack is shown in Fig. 2.7. Phases of the Rebound Attack are explained in the following subsections.

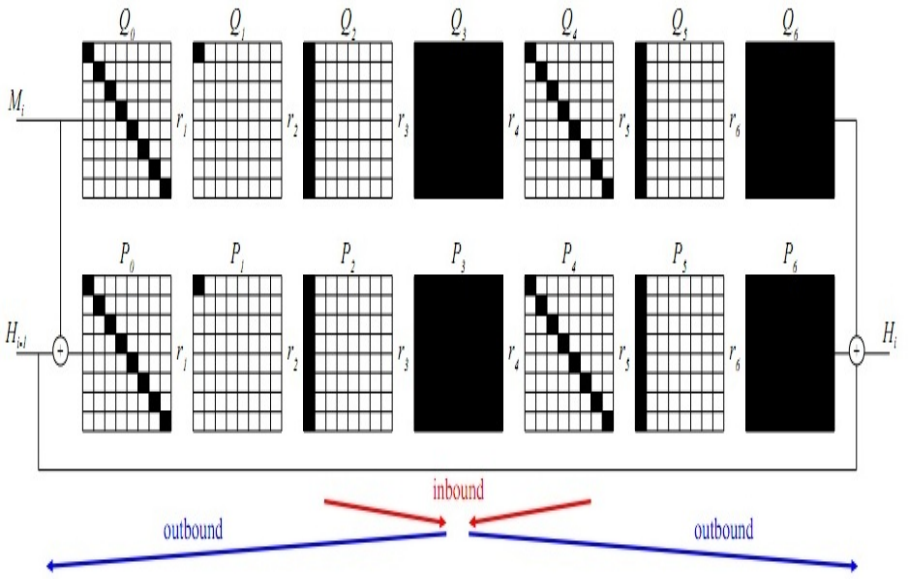

Figure 2.7: Rebound Attack on 6 rounds Grϕstl-256

2.5.1 Inbound phase

The rebound attack is started with the inbound phase in round r_3 and r_4. It follows the truncated differential path shown above and deterministically propagate to the full active SubBytes layer in the middle. This phase starts with 8 input/output differences of MB_4^{out} and SB_3^{out} and propogates through linear layers and match at SB_3(see Fig. 2.8). Inbound Phase can be explained as follows:

1. Initially we randomly select 8 byte differences for SB_5^{in} and propgate backwards to MB_4^{out}. The output differnces will remain same as inverseAddColmns is a linear operation. Propogating backwards and applying inverseMixBytes operation, all the 64 bytes would get active and we will obtain 64 byte differences for MB_4^{in}. After applying inverseShiftBytes, differences will propogate linearly to SB_4^{out}. Next is an inverseSBox operation, which is not linear, so we will compute all valid byte pairs for SB_4^{in} using SBox differential tables.

2. Next, we will choose byte differences of SB_3^{out}.and propogate in forward direction to have 64 byte differnces at MB_3^{out} and these byte differences will propogate linearly to SB_3^{in}(inverseAddColumns is a linear operation).

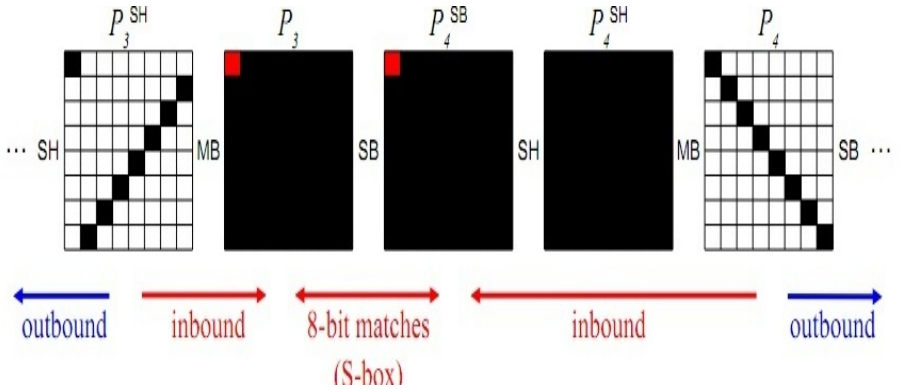

Figure 2.8: Inbound phase of Rebound Attack on 6 rounds Grϕstl-256

3. Differences obtained in step 1 and 2 at SB_3 needs to be matched to check for the existence of SBox differential.

Form differential properties of SubBytes(see section 1.8.2), we know that for a single S-box, the probability that a random S-box differential exists is 1/2. Moreover, for a given input and output difference, the number of possible input values is limited to either 2 or 4 values. Thus for each column, that is 8 bytes, probability would be 2^{-8}. Thus, For each column, we will try 2^8 non-zero differences of the according byte in (SB_3) and thus, expect one valid differential for all 8 S-boxes of that column. With two independent solutions for each S-box, we get at least 2^8 pairs for one column. Hence, the average complexity to find a valid pair is 1. We repeat this for all 8 active bytes of (SB_3) and get about 2^{64} solutions for the inbound phase.

2.5.2 Outbound Phase

In the outbound phase, the airs obtained using inbound phase are propogated probabilistically outwards so as to match the differences at the input and output of the permutations. For the backward direction, that is for the propagation from 8 to 1 active bytes through the MixBytes transformation, probability is 2^{56}. Hence, we can construct one pair conforming to the truncated differential path for each of P and Q with a complexity of 2^{56}.

Chapter 3

Extending Rebound Attack

3.1 Overview

In the last chapter we discussed simple rebound technique applied on 4, 5 and 6 rounds of Grϕstl-256. In this chapter, we will discuss how a rebound technique can be extended such that it can be used to attack more rounds of Grϕstl-256. Rebound technique can be extended using Super Sbox technique. This is known as Super Sbox cryptanalysis, introduced by Thomas Peyrin and Henri Gilbert [12]. Super SBox technique was introduced for AES like permutations, and thus improved the cryptanalysis of AES, Grϕstl, Echo.

Chapter starts with an intrduction to super Sbox view. Super Sbox view was introduced by Deamen and Rijmen [6, 8]. SuperSbox view is explained for AES like permutations. The central idea is to view two consecutive rounds of AES as application of SuperSBox. It further explains Super Sbox cryptanalysis and rebound attack using super sbox on 7 rounds of Grϕstl-256.

3.2 Super S Box View

Super SBox concept introduced by Deamen and Rijmen [7] is used for two consecutive rounds of AES. It represents two rounds of AES permutations using one big SBox layer. Precisely speaking, it exchanges the order of SubBytes and ShiftBytes and represnts two rounds of AES permutation

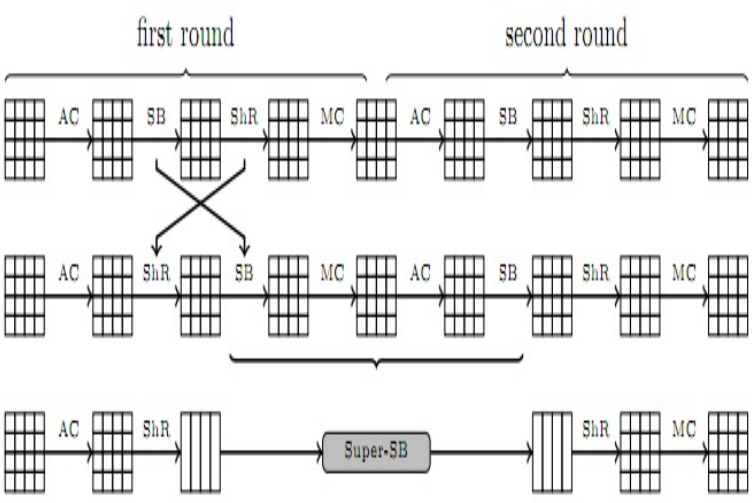

Figure 3.1: Super Sbox view for 2 rounds AES Permutation

$$MC \circ ShR \circ SB \circ AC \circ MC \circ ShR \circ SB \circ AC(S)$$

as

$$MC \circ ShR \circ SB \circ AC \circ MC \circ SB \circ ShR \circ AC(S).$$

The middle part $SB \circ AC \circ MC \circ SB$ is known as Super Sbox(see Fig. 3.1).

Thus two rounds of AES permutations can be represented as-

$$MC \circ ShR \circ \text{Super-SB} \circ ShR \circ AC(S).$$

3.3 Super Sbox view of Grϕstl-256

As we already mentioned Super Sbox view was introduced for AES like permutations, thus Sbox view mentioned above can be applied to Grϕstl. A SuperBox of Grϕstl is quite similar to

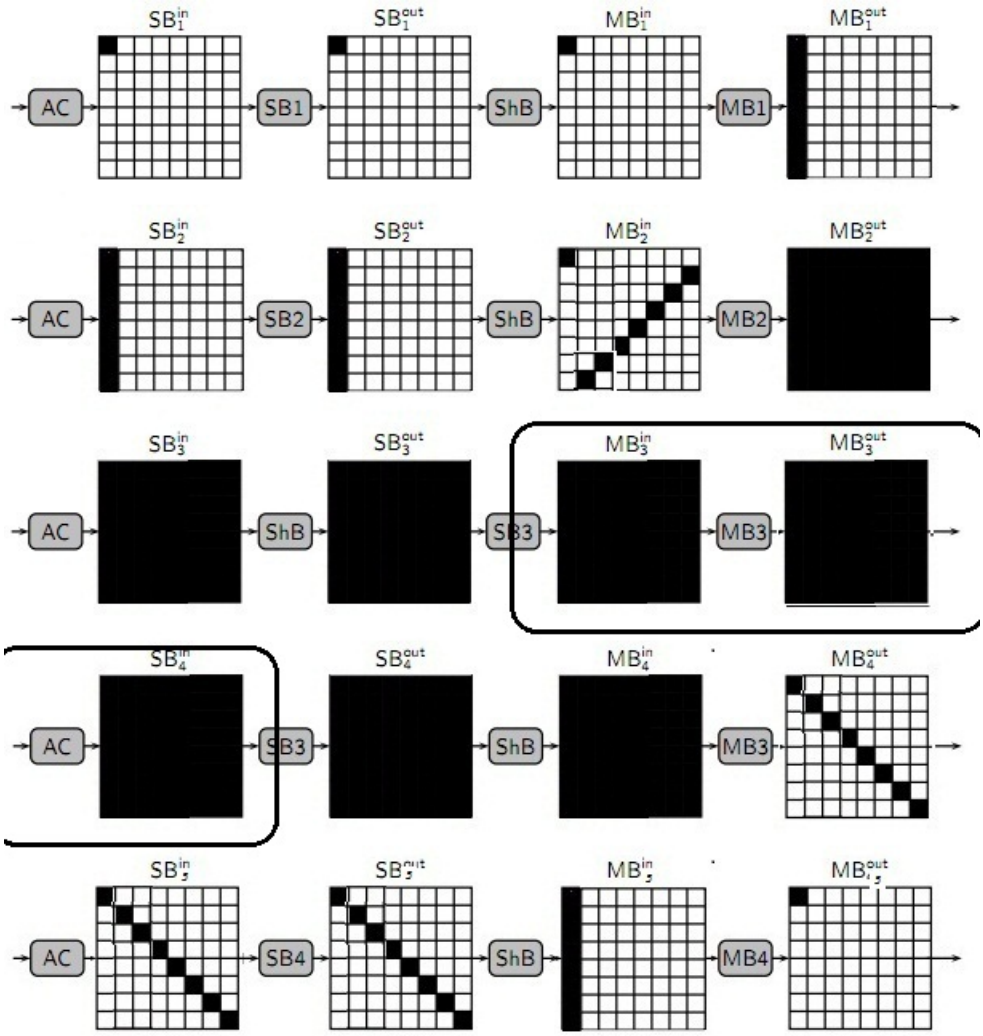

Figure 3.2: Super Sbox view of Grϕstl-256

the SuperBox of the AES. Super Sbox for 5 rounds of Grϕstl is shown in Fig 3.2. For Grϕstl, the SuperBox consists of 8 parallel S-boxes, followed by one MixBytes transformation and another 8 parallel S-boxes: SB $\circ$ MB $\circ$ SB. As stated earlier, the SubBytes and ShiftBytes transformations can be interchanged. Hence, a Super Box behaves like a non-linear 64-bit S-box.

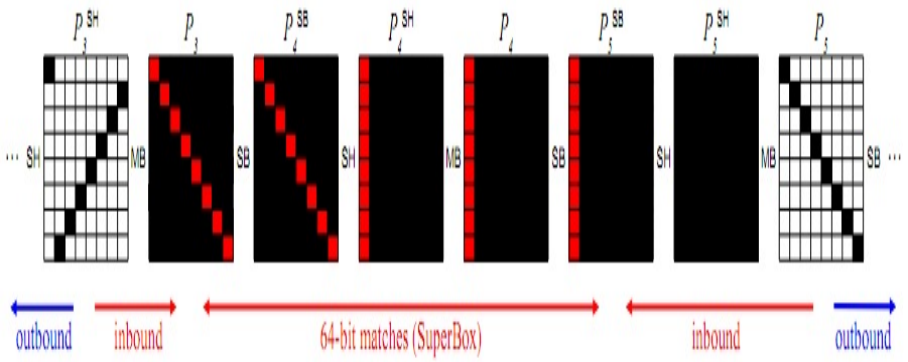

Figure 3.3: Inbound Phase with Super Sbox view for Grϕstl-256

3.4 Extending Inbound Phase

Super Sbox view of Grϕstl-256, mentioned above, can be used to extend the rebound technique explained in previous chapter. This is known as Super Sbox cryptanalysis. This idea has already been applied in the improved attack on the Whirlpool hash function. Super Sbox concept just extends the inbound phase of the rebound attack as outbound phase is probabilistic, and thus can't be improved. By considering SuperBoxes, inbound phase can easily be extended by one full active state.

The rebound attack is started with the inbound phase in round r_3 and r_5. Following the truncated differential path, it deterministically propagate to the full active SubBytes layer in the middle. This phase starts with 8 input/output differences of MB_5^{out} and SB_3^{out} and propogates through linear layers. Inbound Phase can be explained as follows:

1. Initially we will randomly select 8 byte differences of SB_3 and propogates in forward direction through MIxBytes to get 64 byte differences for MB_3 and store the resulting differences.

2. Next we randomly select 8 byte differences for MB_5 and propgate backwards through MixBytes and ShiftBytes to SB_5. Propogating backwards and applying inverseMixBytes operation, all the 64 bytes would get active and we will obtain 64 byte differences for MB_5^{in}. After applying inverseShiftBytes, differences will propogate linearly to SB_5^{out}. Next is an inverseSBox operation, which is not linear, so we will compute all valid byte pairs for SB_5^{in} using SBox differential tables.

3. Connect the output differences of the 8 parallel SuperBoxes (i.e. SB_5) with the corresponding input differences of the SuperBoxes (state SB_3). For each column(SuperBox) at state SB_5 , take all 2^{64} possible values and propgates backward to state MB_3 and store the differnces and values in initial list. Now, for each SuperBox we will obtain 2^{64} input differences for each SuperBox in state P_3 and store the resulting differences and values in second list. Next, match the differences of initial list and second list. Since both lists have 2^{64} entries and we have a condition on 64 bits, we get $2^{64} \times 2^{64} \times 2^{-64} = 2^{64}$ solutions and update the initial list. Repeat this for every column of state SB_5.

4. Complete inbound phase will obtain 2^{64} solutions with a complexity of 2^{64} in time and memory.

3.5 Attack on 7 rounds Grϕstl-256

Same as attack on 6 rounds Grϕstl-256, attack on 7 rounds Grϕ-256 uses a truncated differential path with a high number of active bytes in the middle and a low number of active bytes at the input and output of each permutation. For the attack on 7 rounds, 2 full active states are placed in the middle of each permutation. The improved inbound phase using the SuperBox allows to extend the 6-round semi-free-start collission attack on Grϕstl-256 by one round. The truncated differential path is given in Fig. 8. The sequence of active bytes in each round for both, P and Q are given as follows:

$8 \rightarrow^{r_1} 1 \rightarrow^{r_2} 8 \rightarrow^{r_3} 64 \rightarrow^{r_4} 64 \rightarrow^{r_5} 8 \rightarrow^{r_6} 8 \rightarrow^{r_6} 64$

Inbound Phase of the attack is explained in section 3.4. The solutions of the inbound phase are propagated outwards probabilisticaly, same as in the attack on 6 rounds.

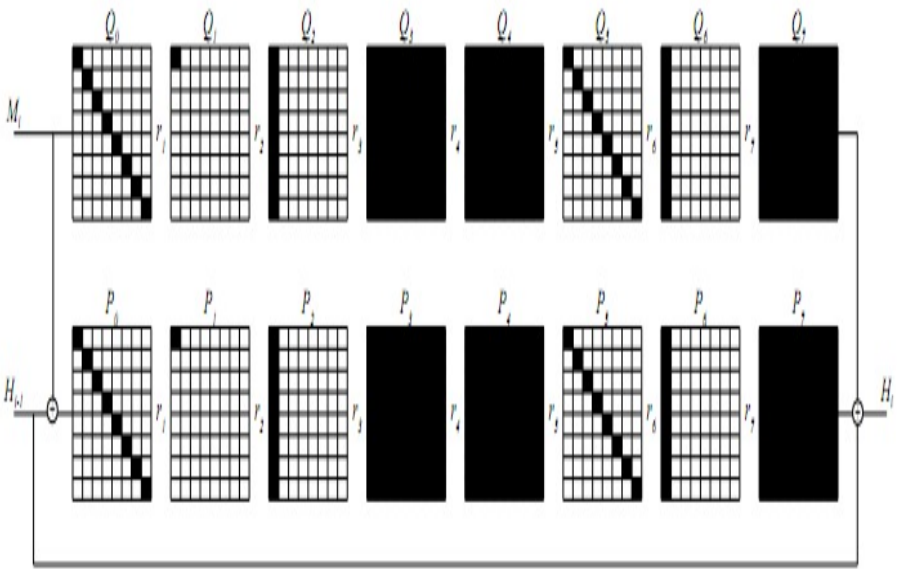

Figure 3.4: Rebound Attack on 7 rouunds Grϕstl-256

Chapter 4

Conclusion and Future Work

In this report, we have focussed on the cryptanalysis of Grϕstl-256. As Grϕstl uses an AES like permutaion, thus standard differential attack or linear attack is out of scope. Cryptanalysis of Grϕstl introduced Rebound attacks. The rebound attack consists of inbound and outbound phases. Using the techniques described in this thesis, we can find right pairs for truncated differential path. In this document we have presented a variety of results of rebound attacks on SHA-3 candidate Grϕstl. As Grϕstl uses two strong permutations with wide trial design, thus the available degrees of freedom is limited. Thus, only single inbound and outbound phases are possible in the rebound attack on Grϕstl. We have considered only Grϕstl-256 in the document. Rebound attacks on reduced rounds of Grϕstl-256 has been discussed. Simple rebound technique is used to attack 4, 5 and 6 rounds Grϕstl-256. Super Sbox technique improved the cryptanalysis of Grϕstl, so as to apply rebound technique on more number of rounds. Attack on 7 rounds of Grϕstl-256 is discussed by using simple concept of Super boxes initially given for AES permutations. Report has focussed rebound attacks on Grϕstl. Similar techniques can also be applied to hash functions using AES permutation as compression function. Future work might be applying similar techniques on hash functions based on block based compression functions.

Bibliography

[1] Ross Anderson. The classification of hash functions, 1993.

[2] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C.-W. Phan. Sha-3 proposal blake. Submission to NIST (Round 3), 2010.

[3] Guido Bertoni, Joan Daemen, Michal Peeters, and Gilles Van Assche. Keccak specifications, 2009.

[4] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle–Damgård revisited: How to construct a hash function. pages 430–??, 2005.

[5] Joan Daemen and Vincent Rijmen. The wide trail design strategy. In *Proceedings of the 8th IMA International Conference on Cryptography and Coding*, pages 222–238, London, UK, 2001. Springer-Verlag.

[6] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES — the Advanced Encryption Standard*. Springer-Verlag, 2002.

[7] Joan Daemen and Vincent Rijmen. Two-round aes differentials, 2006.

[8] Joan Daemen and Vincent Rijmen. Understanding two-round differentials in aes. In *SCN'06*, pages 78–94, 2006.

[9] Joan Daemen and Vincent Rijmen. Plateau characteristics. *IET information security*, 1:11 – 18, 2007.

[10] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker. The skein hask function family, 2009.

[11] Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Thomsen Søren S. Grøstl - a sha-3 candidate. In

Helena Handschuh, Stefan Lucks, Bart Preneel, and Phillip Rogaway, editors, *Symmetric Cryptography*, number 09031 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2009. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[12] Henri Gilbert and Thomas Peyrin. Super-sbox cryptanalysis: Improved attacks for aes-like permutations. *IACR Cryptology ePrint Archive*, 2009:531, 2009.

[13] Kota Ideguchi, Elmar Tischhauser, and Bart Preneel. Improved collision attacks on the reduced-round grøstl hash function. In Mike Burmester, Gene Tsudik, Spyros S. Magliveras, and Ivana Ilic, editors, *ISC*, volume 6531 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2010.

[14] Lars Knudsen and Bart Preneel. Hash functions based on block ciphers and quaternary codes. In *Advances in Cryptology ASIACRYPT*, pages 77–90. Springer Verlag, 1996.

[15] Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. The rebound attack: Cryptanalysis of reduced whirlpool and grøstl. In Orr Dunkelman, editor, *FSE*, volume 5665 of *Lecture Notes in Computer Science*, pages 260–276. Springer, 2009.

[16] Florian Mendel, Christian Rechberger, Martin Schlffer, and Sren S. Thomsen. Rebound attacks on the reduced grstl hash function. In *CT-RSA'10*, pages 350–365, 2010.

[17] Bart Preneel. Information security and cryptology. chapter The State of Hash Functions and the NIST SHA-3 Competition, pages 1–11. Springer-Verlag, Berlin, Heidelberg, 2009.

[18] Vincent Rijmen and Paulo S. L. M. Barreto. The WHIRLPOOL hash function. World-Wide Web document, 2001.

[19] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full sha-1. In *International Crytology Conference*, pages 17–36, 2005.

[20] Xiaoyun Wang and Hongbo Yu. How to break md5 and other hash functions. In *EUROCRYPT*, pages 19–35, 2005.