Cryptanalysis of Hash Functions

In particular the SHA-3 contenders Shabal and Blake

Nieke Aerts

August 2011

Cryptanalysis of Hash Functions

In particular the SHA-3 contenders Shabal and Blake

Nieke Aerts

Under supervision of

Josef Pieprzyk & Benne de Weger

Macquarie University & Eindhoven University of Technology

August 22, 2011

Abstract

Since NIST announced the SHA–3 competition in 2007, many new attacks to hash functions have been born. We tried to understand and apply these new attacks to the second round candidate Shabal and the final round candidate BLAKE.
In Chapter 2 we set out the definitions used in this thesis and we describe different attacks. In Chapter 3 we set forth the previous analysis of Shabal and our own ideas. In the same way we discuss BLAKE in Chapter 4. In the last Chapter we discuss the comparison of these two functions.

Contents

Chapter 1

Introduction

1.1 NIST competition

NIST began the standardization of hash algorithms in 1993 when they published the SHA-0 algorithm. Soon after, the algorithm was replaced by SHA-1 due to security issues of SHA-0. In 2001 the Merkle-Damgård based SHA-2 hash family was added to the standard hash algorithms by NIST. In 2005 SHA-1 was theoretically broken which called for the need to use the stronger algorithms of the SHA-2 family.
In November 2007 NIST wrote out the request for candidate new hash algorithm families referred to as SHA-3. Up until now, no attack on SHA-2 is known, but a collision in the SHA-2 family would have catastrophic effects for digital signatures. Therefore the new hash family should be able to immediately replace SHA-2 if necessary. The requirements for submissions for the SHA-3 contest were published in [2].
NIST received sixty–four entries in 2008, of which fifty–one advanced to the first round. In 2009 fourteen candidates were selected for the second round. Meanwhile cryptanalysts all over the world were analyzing the candidates.
In December 2010 the five finalists were selected, of which one will be selected as winner in the spring of 2012.

1.1.1 Security Requirements of SHA-3

The security requirements were published in [2, Part 4.A]. Here I will state the most important ones considering this thesis[1].

1. It should be possible and secure to use the hash family for a wide variety of cryptographic applications, including digital signatures, key derivation, hash-based message authentication codes and deterministic random bit generators.
2. Support HMAC, Pseudo Random Functions (PRF) and Randomized Hashing
   - the PRF must resist distinguishing attacks that require much fewer than $2^{n/2}$ queries and significantly less computation than a preimage attack.
   - the construction for Randomized Hashing must be resistent to the following attack: the attacker chooses m_1, the hashing algorithm processes this message with a to the attacker unknown randomization value r_1, now the attacker tries to find a second

[1] As the research for this thesis consists of Cryptanalysis, we will only state the requirements considering this.

message m_2 and a randomization value r_2 such that m_2 with r_2 is mapped to the same hash as m_1 with r_1. The construction should have at least n bits of security.

3. Collision resistance of approximately $n/2$ bits
4. Preimage resistance of approximately n bits
5. Second-preimage resistance of approximately $n - k$ bits for any message shorter than 2^k bits[2]
6. Resistance to length extension attacks
7. Any m–bit hash function specified by taking a fixed subset of the candidate's function output bits is expected to meet the above requirements with m replacing n

1.2 Attacks on Hash functions

Hash functions are used in many different applications, so an attack in one application is not necessarily an attack on every application. For a hash function to be broken one should be able to find a preimage, a second-preimage or a collision in feasible time. A hash function is computationally broken if one of those can be found with effort less than $2^{\#output\ bits}$, but none has been found yet. The security of a hash function is questioned if there is a distinguishing attack, such that the attacker can distinguish the hash output from the output of a random oracle.

1.3 Cryptanalysis in the competition

The authors of contending functions and many more cryptologists are currently analyzing the hash fucntions of the SHA–3 contest. Several functions in the first round were broken. All of the second round contenders were thoroughly analyzed. This has helped the committee of NIST to decide which functions progress to the next round.

1.4 Thesis Outline

Chapter 2 is a preliminary chapter, it contains a small introduction to hash functions, many notations are borrowed from Menezes, van Oorschot and Vanstone [4], and a description of analysis and some attacks to hash functions.
In Appendix A we summarize the notations used in this thesis.
The third and fourth chapter are organized identically, they start with a description of the function, secondly an overview of recent work is given, followed by some analysis of my own and ending with a conclusion. Chapter 3 considers the second round candidate of the NIST competition Shabal and Chapter 4 considers the final round candidate BLAKE.
Chapter 5 compares the two functions Shabal and BLAKE.

[2] A method to find second preimages in less than 2^n time is given by Kelsey and Sneier [3].

Chapter 2

Preliminaries

2.1 Hash Function/Hash Algorithm

A hash algorithm converts an arbitrary length message into a fixed length output. If x is mapped to y by the hash function, we will call y the hash or the digest of x.
Definition 1 (Hash function)**.** *$\mathcal{H}$ is called a hash function if it maps messages of arbitrary length to images of finite length l:*

$$\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^l$$

and given an input x, $\mathcal{H}(x)$ is easy to compute.

2.1.1 Cryptographic hash function versus standard hash function

This report only considers cryptographic hash functions. There is a difference between cryptographic and other hash functions, but the sets are not disjoint. By cryptographic hash functions, you should think of one-way functions, it should be impossible to find the inverse in feasible time. Standard hash functions are mostly used to define a lookup system, for example the way we use a dictionary. A word starting with an "A" can be found in the beginning of the dictionary, a word starting with a "W" almost at the end, our brain maps the letter "A" to "somewhere in the first part of the dictionary". This is certainly not one-way, as the hash will reveal (a part of) the message. Realizing that you are looking in the first part of the dictionary, you can tell you were looking for something starting with an "A".
Obviously, using a (partly-)invertible function while trying to hide the message is of no use. But using a cryptographic hash function to define a lookup function will not lead to a big problem, it might not be as fast since cryptographic hash functions tend to be more complex than standard ones.
From here, hash function or hash algorithm will always refer to the cryptographic variant.

2.1.2 Basic Properties

A hash algorithm should have the following three properties:

- **Preimage resistance** Given an output y, it is computationally infeasible to find an input x such that y is the hash of x.
- **Second–preimage resistance** Given an input x and its hash y, it is computationally infeasible to find a second input x' such that y is the hash of x'.

- **Collision resistance** It is computationally infeasible to find two inputs, x and x' such that that are mapped into the same output.

All three contain the phrase *computationally infeasible*, this statement will be explained in Section 2.3.1.

2.1.3 Applications

Hash algorithms can be used to validate the **identity** of the author or the **integrity** of a message. A hash function can be used in a **digital signature scheme** to validate the authenticity of a message. A valid digital signature gives a recipient reason to believe that the message was created by a known sender, and that it was not altered in transit. As a hash algorithm should be second–preimage resistant, given the hash of a message, it is impossible to find a second message consistent with this hash. So given a message[1] and its hash one can check whether the message has been altered after the hash was generated.
In **commitment schemes** hash algorithms are often used to prove knowledge of the message without revealing the message. In example, Alice wants to show Bob that she knows the answer to a question without revealing the answer, she sends the hash of the answer to Bob. Later Alice can show the answer and Bob can check if the hash Alice gave earlier really is the hash of the answer.
The hash of a document can be used to **identify** the document.
A hash algorithm can be used to simulate a random bit stream, using it as a so–called **pseudo-random** bit generator.

2.2 Iterative Hash functions

A hash algorithm is used to hash a message of arbitrary length. But most mathematical functions are defined on a space in which all elements have the same "size". So for a hash algorithm to be able to process arbitrary length messages we often use a transformation to transform the message into blocks of specified length. Then a particular function (on a fixed size domain) can be applied to all the blocks consecutively.
An iterative hash function consists of the following steps:

- Initialization: Padding, State–initialization.
- Compression: Processing the message blocks.
- Finishing: Truncation, Final Function.

Now some compression functions are similar to others in terms of the operations they contain, the following classes are quite common[2]:

- ARX: The compression function **only** contains modular Addition, Rotation and eXclusive OR.
- S–box: The compression function contains a substitution box.
- Wide Pipe: The internal state in the compression function is larger than the digest.

[1] This can either be the message, the message and the name of the author or only the name of the author.
[2] Some notations are borrowed from [5]

- Narrow Pipe: The internal state in the compression function is smaller than the digest.

2.2.1 Padding

To transform a message into a sequence of blocks of fixed length n, a message can be padded and then chopped into n–bit blocks. There are different padding methods. The most trivial one is to add a minimal number of 0-bits at the end of the message, until the message has length a multiple of n. Obviously, this is not a bijection as messages x and $x||0$ are both mapped to the same message as long as the length of x is not a multiple of n.
To obtain a bijection one could first append a 1-bit to the message, followed by as many 0-bits as needed.
Another way of padding is to reserve a fixed number of bits, k, at the end where the length of the original message is stated. So a message is padded with a 1, a certain number of 0's followed by the bit-representation of the length in k bits. This method leads to prefix-free messages, but not necessarily prefix-free message blocks.
Definition 2 (Prefix–free)**.** *A set is said to be prefix–free if for all elements x there does not exist an element $y \neq x$ such that y concatenated with some non–empty c gives $x = y||c$.*

2.2.2 Merkle-Damgård

Merkle and Damgård both published about the same such transformation at almost the same time [6, 7], this explains why nowadays this construction is called the Merkle-Damgård construction (or MD–construction).
We let X be a set of elements and we want $\mathcal{H}$ to map a concatenation of an arbitrary number of elements of X to a concatenation of m elements of X. We let X^* the set of arbitrary length concatenations and X^k the set of length k concatenations.
Let $F : X^n \to X^m$ be a function that maps elements of X^n to elements of X^m, for $n > m$. Now we define $\mathcal{H}$ using F.
Let $M \in X^*$ be the input of $\mathcal{H}$. We want the length of M to be a multiple of $n - m$ so we pad M such that $length(M) = t(n - m)$, where t is a strictly positive integer. Let $M = M_0||\ldots||M_{t-1}$.
The Merkle-Damgård construction consists of t rounds of F. In the first round an *initialization value*[3] is used, $IV \in X^m$. After each round an intermediate value is computed $\mathcal{H}_i$, which we will call the *Chaining Variable*[3]. Now let $\mathcal{H}_0 = IV$. Then:

$$\mathcal{H}_i = F[M_{i-1}||\mathcal{H}_{i-1}], i = 1, \ldots, t$$

Now $\mathcal{H}_t$ is the output of $\mathcal{H}$.
Obviously $M_{i-1}||\mathcal{H}_{i-1}$ is of length n and so $F[M_{i-1}||\mathcal{H}_{i-1}]$ is well defined. Thus we can conclude that $\mathcal{H}$ is defined to map an arbitrary length input to a fixed length output and $\mathcal{H}$ is a hash function.

Vulnerability of MD

The Merkle-Damgård construction is vulnerable to *length extension attacks*, if the hash of an unknown message y is known, then the hash of $Pad(y)||c$ can be found for any c, where Pad is the padding function.
Let us look into an MD construction $\mathcal{H}$ with compression function $\mathcal{F}$. The padding function

[3] We will use either "value" or "variable".

consists of adding an appropriate sequence of zero's.
Suppose the hash of a certain message y of length precisely one block is known, $h = \mathcal{H}(y) = \mathcal{F}(y)$. Then for any extension of this message $x = y||c$ the hash can be produced without querying x to the construction, in the following way: query $c||h$ to $\mathcal{F}$ to produce

$$h^* = \mathcal{F}(c||h) = \mathcal{F}(c||\mathcal{H}(y)) = \mathcal{F}(c||\mathcal{F}(y||IV)) = \mathcal{H}(y||c) = \mathcal{H}(x)$$

where IV is the initialization value.

How to protect an MD construction to length extension attacks

Coron et al. describe several methods to protect an MD construction against length extension attacks [8].

Prefix–free Recall that a code is said to be prefix-free if for all codewords x there does not exist a codeword $y \neq x$ such that y concatenated with some c gives $x = y||c$. Now to make the full sequence of message blocks to be prefix–free, there are many mappings, of which we will describe two, given by Coron et al. [8].
After the message is padded and divided in blocks, we prepend one block containing the message length. Let n be the bitlength of a message block. Now for two full sequences of message blocks $x \neq y$ to have $x = y||c$ we must have that the representation of the original message length is the same. For this to happen either the length of x equals the length l of y, which gives a contradiction, or the length of x must be $l + 2^n$, which will be represented as l too. Most hash functions admit at most a 2^{64}–bit input, and the message blocks are at least 64 bits, so the latter case will not cause any problem.
A second method prepends a zero to every but the last message block. As the compression function requires n–bit input blocks, the message is padded to be a multiple of $n - 1$, then divided in blocks of length $n - 1$, then a zero is prepended to all but the last block, the last block is prepended with a one. Since only the last block can start with a one in this method, this must be a prefix–free set.
Coron et al. prove that the combination of any prefix–code in an MD construction yields an indifferentiable system. Indifferentiability will be discussed later in this chapter. For now, a prefix–free code will surely make the length extension attack impossible and that is what we were after, so we consider this system secure.

The Chop solution Another way to protect against the message extension attack is to chop off part of the last chaining variable before returning it, i.e. if the chaining value consists of n bits, only output a subset of those bits of size $n - s$. This way the attacker does not know the full chaining variable, and the message extending technique shown before, can not be used. If the set of dropped bits is too small, the attacker can guess the missing bits of the chaining variable and use the message extension technique to find the hash of the larger message x with high probability. For example, if $s = 1$, there is only one output bit missing from the chaining variable, it can either be 1 or 0 and the attacker guesses the hash of x correctly with probability $1/2$.
The drawback of this method is that its security is strongly related to s, the number of bits that are chopped. It is proportional to $N^2 2^{-s}$, where N is the number of queries, so for the function to be secure s has to be relatively high, which means that short-output hash functions such as SHA-1 and MD5 cannot be fixed using this method. However, functions such as SHA-512 can

naturally be fixed (say, by setting s = 256).
The chop solution is also referred to as *truncation*.

NMAC An NMAC construction applies a final transformation to the last chaining value before it is returned. This transformation should be independent of the compression function. The length extension attack is no longer possible as the intermediate chaining value will not collide with the hash output of the smaller message y.

HMAC Instead of an independent final transformation as in NMAC, HMAC uses the compression function again but as input we use the last chaining variable and the initial value. The chaining variable of length m takes the place of the message block of length k and is either padded with zeros if $m < k$, chopped if $m > k$ or left unchanged if $m = k$. Again, the length extension attack is no longer possible as the intermediate chaining value will not collide with the hash output of the smaller message y.

2.3 Cryptanalysis

Recently a lot of new attacks have been introduced to the world of cryptography. Especially since the NIST SHA-3 competition started in 2007, many new ways of attacking a hash function have been discovered. In the following paragraphs some attacks are clarified, ordered by date of first use. But first we will explain the criteria of security of a hash function.

2.3.1 Security Criteria of Hash functions

The security of a hash function is often measured in how many queries are needed to find

(1) a preimage,

(2) a second–preimage and

(3) a collision.

We say that a problem is *computationally infeasible* if even the fastest computer can not solve the problem within a normal amount of time, e.g. a human lifetime. If (1), (2) and (3) are computationally infeasible, the hash function is considered secure.

Breaking

A hash function is *theoretically broken* if the number of queries for any of the three is less than the best brute force attack. A hash function is *broken* if a collision is known or if there exists an algorithm that given an output finds a preimage (or a second–preimage) in feasible time. Let n be the bit–size of the digest, then the number of queries needed for a brute force attack are

- 2^n for a preimage,
- 2^n for a second–preimage and
- $2^{n/2}$ for a collision.

Brute Force Attacks

Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^n$ be a hash algorithm that maps an arbitrary length message to a hash of length n.
Given a hash y, a brute force attack to find a preimage, selects messages randomly, queries the messages, until the function returns y. Using this, one expects to find the preimage after at most 2^n queries.
The birthday–attack is a brute force attack to find a collision, there are many different algorithms for this attack, for example with Yuval's birthday attack using Floyd'd cycle finding algorithm [4]. The birthday–attack uses that with high probability, there exists a pair of colliding messages in a random selection of $2^{n/2}$ messages.

Semi–breaking

As cryptanalysis advanced, newer hash functions needed to resist more attacks and their designs are often more complex to protect the function against to these attacks. Because of this, one started to relax the requirements, and defined three types of relaxations, so–called *near*, *pseudo* and *free–start*.
For a **pseudo**–collision one can choose the chaining variable independently for each message. That is, instead of using the initial values or a valid[4] chaining value, one can choose two (not necessarily equal) chaining values. These chaining values together with two (not necessarily) different messages that are mapped to the same image using the chaining values, are called pseudo–collision. So if for chaining variables h^*, h^{**} and messages m, m'

$$\mathcal{H}_{h^*}(m) = \mathcal{H}_{h^{**}}(m')$$

holds, it is a pseudo–collision.
Similarly a pseudo–preimage is a message which given some chaining value, is mapped to the image.
A **free–start**–collision is a little stronger than a pseudo–collision. Here only one chaining value can be chosen, and two different messages. Now if the messages are mapped to the same image using the chosen chaining variable, we have found a free–start collision. Formally, for the chaining variable[5] h^* and the messages m, m' we have a free–start collision if

$$\mathcal{H}_{h^*}(m) = \mathcal{H}_{h^*}(m').$$

When two different messages are mapped to almost the same image starting with the same initial values, we call it a **near**–collision. We say that the images are almost the same if the Hamming distance (HD) is very small. Let the Hamming distance of a near–collision be at most $n - l$, then we say that we have a l–bit near–collision. So we should have

$$HD[\mathcal{H}_{IV}(m), \mathcal{H}_{IV}(m')] < n - l$$

where n is the number of bits of the digest. And likewise, if a message is mapped into a value close to value for which we seek a preimage, we call it a near–preimage.
One can also combine two of the relaxations, e.g. a near–pseudo–collision is a has two different chaining values and two messages which are mapped to almost the same image.

[4] We call a chaining variable valid if a sequence of message blocks that leads to this chaining value is known.
[5] The chaining variable does not have to be a valid chaining variable, that is, the attacker does not need to know how to achieve this.

Security Concerns

Even if a hash function has not been broken in any way described above, there might be concerns about its security. Ideally the digest will give no information on the input, even a little information can be crucial for the security of a hash function. Hash digests are therefore often compared to random outputs, when a hash function behaves like a random oracle then the digests will not reveal any information. We will discuss the comparison of a hash function with a random oracle in the following section.

2.3.2 (In)Differentiability

Two systems $\mathcal{E}$ and $\mathcal{F}$ are said to be differentiable if there exists a distinguisher $\mathcal{D}$ which is able to tell whether it is communicating with $\mathcal{E}$ or $\mathcal{F}$ in feasible time. For example, if there exists a set of inputs S such that for $f \in S$ we have:

$$\Pr[\mathcal{E}(f) = d] > \epsilon + \Pr[\mathcal{F}(f) = d]$$

then querying a small number of elements of S will suffice to decide with which system you are communicating. Here a small number is approximately ϵ times the number of elements in S. If there exists no such distinguisher then two systems are said to be indifferentiable. (In)differentiability is formally defined by Maurer et al. [9].

Indifferentiability is based on the comparison of two systems, the implementation of the cryptographic construction and a random oracle. A random oracle selects outputs of n bits randomly. If there exists a distinguisher $\mathcal{D}$ which is able to decide correctly (with high probability) with which of the two systems it is communicating, we call the system *differentiable from a random oracle*. The two systems are named $\mathcal{C}^{\mathcal{P}}$ for the implementation of a cryptographic construction $\mathcal{C}$ based on the random function $\mathcal{P}$ and $\mathcal{S}^{\mathcal{O}}$ for the simulator based on a random oracle $\mathcal{O}$.
A system is indifferentiable if the probability that the distinguisher chooses the correct system, is negligible. The advantage of the distinguisher is defined as

$$\mathrm{ADV}(\mathcal{D}, \mathcal{C}) = |\Pr\left[\mathcal{D}^{\mathcal{Q}} = 1 | \mathcal{Q} = (\mathcal{C}^{\mathcal{P}}, \mathcal{P})\right] - \Pr\left[\mathcal{D}^{\mathcal{Q}} = 1 | \mathcal{Q} = (\mathcal{S}^{\mathcal{O}}, \mathcal{O})\right]|$$

where $\mathcal{Q}$ represents the system (undefined which one).
If for a hash algorithm, there exists a distinguisher with a large advantage, the algorithm is not yet broken, but there are concerns about the security. In the near future it could be possible that more is known on how to proceed if there is a non-negligible bias in the output of a hash algorithm and therefore such a system might be broken in the near future.

2.3.3 Linear Cryptanalysis

Linear cryptanalysis was first used on the cipher FEAL by Matsui and Yamagishi [10] and is still mainly used on ciphers to recover key bits. First the system is linearized and if there exists some (approximate) expression for a plaintext, ciphertext pair P, C, and the key K, we could use this to recover some key bits. We call this a known–plaintext attack. Let i_s, j_s and k_s represent bit positions, if

$$P[i_1] \oplus \ldots \oplus P[i_n] \oplus C[j_i] \oplus \ldots \oplus [j_m] = K[k_1] \oplus \ldots \oplus K[k_l]$$

will hold with probability $p \neq 1/2$, some key bits can be recovered with an algorithm, e.g. the following algorithm [11]:

Algorithm: Key Recovery
1. Let T be the minimum number of plaintexts such that $P[i_1] \oplus \ldots \oplus P[i_n] \oplus C[j_i] \oplus \ldots \oplus [j_m] = 0$
2. If $T > N/2$, where N is the number of plaintexts known,
then guess $K[k_1] \oplus \ldots \oplus K[k_l] = 0$ (when $p > 1/2$) and 1 (when $p < 1/2$)
else guess $K[k_1] \oplus \ldots \oplus K[k_l] = 1$ (when $p > 1/2$) and 0 (when $p < 1/2$)

The number of plaintexts, T, can be determined by finding the success probability of the equation and multiply this with the total number of plaintexts.

Application to Hash Functions

Unless the bias is extremely high, linear cryptanalysis will not lead to an attack for a hash function. So linear cryptanalysis will not provide us a way to find a collision, pre–image or second–pre–image, but it might give us insight on the security.
Hash functions usually contain a non-linear operation, this could be a substitution box (S-box)[6], modular addition, modular multiplication etcetera. Linearizing this operation is often expensive, i.e. the probability of success will be very small.

Linearizing modular addition We approximate modular addition with the bitwise exclusive or (XOR) operator $\oplus$. For the first bit we will always have equality, but for the other bits we only have equality if there is no carry from the previous bits. We have for the modular sum s of a and b:

$$\begin{aligned} s_0 &= a_0 \oplus b_0 \\ c_1 &= a_0 \cdot b_0 \\ s_i &= a_i \oplus b_i + c_i \qquad ,i > 0 \\ c_{i+1} &= a_i \cdot b_i \oplus a_i \cdot c_i \oplus b_i \cdot c_i \quad ,i > 0 \end{aligned}$$

where c represents the carry.
Now for bit–position $i > 0$ we have $(a + b)_i = (a \oplus b)_i$ if and only if the carry $c_i = 0$. We let n be the bitlength of the words, so we have addition modulo 2^n. We estimate the probability as follows:

$$\begin{aligned} \Pr[a + b = a \oplus b] &= \Pr[\forall i > 0 : c_i = 0] \\ &= \Pr[\forall i \geq 0 : a_i \cdot b_i = 0] \\ &= \Pr[\forall i \geq 0 : a_i = 0 \vee b_i = 0] \\ &= \left(\tfrac{3}{4}\right)^{n-1}. \end{aligned}$$

So if we replace all modular additions by XOR's in a hash function operating on n–bit words and originally containing r modular additions we find probability

$$\left(\frac{3}{4}\right)^{r(n-1)}.$$

that the output is the same as in the original hash function.

[6] S-box is a permutation of the (small) subsequences of bits, e.g. for a 128–bit sequence an S-box could replace every four bits with the four–bits–image under the permutation.

Linearizing modular multiplication with a constant First we will look at linearizing multiplication by three, and then we will look at the more general case. Multiplication with by 3 can be written as a combination of shifts and additions:

$$3 \cdot a = a + a \ll_1 .$$

So again we have to linearize the modular addition, so we approximate $3 \cdot a$ with $a \oplus a \ll_1$ and for the probability of equality we find:

$$\begin{aligned} \Pr[3 \cdot a = a \oplus a \ll_1] &= \Pr[a + a \ll_1 = a \oplus a \ll_1] \\ &= \left(\tfrac{3}{4}\right)^{n-2}. \end{aligned}$$

There is one factor less than for addition, since for $a \ll_1$ we have $(a \ll_1)_0 = 0$.
For the general case we have to use the binary representation of the constant d, let this be $d_{n-1} \ldots d_0$. Then we find

$$d \cdot a = \sum_{i=0}^{n-1} d_i \cdot a \ll_i .$$

Let $\{j_0, j_1, \ldots, j_t\} = \{i : d_i = 1\}$. Then we approximate the modular addition with exclusive or, and find the probability of equality to be

$$\sum_{k=1}^{t} \left(\frac{3}{4}\right)^{n-1-j_k}.$$

Linearizing AND ($\wedge$) First we will look at the AND operation on two bits, as it is a bitwise function, this can be easily extended to a word. We estimate the probability as follows:

$$\Pr[x_i \wedge y_i = \overline{x_i \oplus y_i}] = \Pr[x_i \cdot y_i = \overline{x_i \oplus y_i}] = \frac{3}{4}.$$

As for only one bit–combination ($x_i = 1, y_i = 1$) we have $x_i \wedge y_i = 1$ and for any linear operation on two bits we find two combinations, there is no better linear approximation. Since the bits are independent, we can easily extend this to words:

$$\Pr[x \wedge y = \overline{x \oplus y}] = \left(\frac{3}{4}\right)^n$$

Nowadays most hash function contain many elements to make the function not linear, e.g. the permutation of Shabal has 48 multiplications by 3, 48 multiplications by 5, 48 AND's and 36 additions for each message block. It operates on 32–bit words, so $n = 32$. The probability that a message block input in the linearized version gives the same output as in the original permutation is approximately 2^{-2276}. For an ideal random permutation on 896 bits this would be 2^{-896}, so the linearization does not help us in this case. In Section 3.10 linear cryptanalysis on Shabal is discussed.

2.3.4 Differential Cryptanalysis

Differential cryptanalysis has been discovered by two parties independently, the first publication is attributed to Biham and Shamir, they used XOR differences to attack DES (Data Encryption

Standard) [12]. A few years later Coppersmith stated that IBM had known about the differential attack since the development of DES and had built it to be strong against this particular attack [13].
Differential analysis studies the behavior of a pair of inputs with a known difference through the hash function. There are different types which could be studied, for example the XOR difference ($\oplus$), the rotational difference ($\lll$), the shift difference ($\ll$) or the modular addition ($+$).

For a hash function a differential input pair $(x, x \boxdot \Delta)$, where $\boxdot$ represents the type of difference, could give a related output pair $(\mathcal{F}(x), f_\Delta(\mathcal{F}(x)))$. If such a relation holds, then one can certainly distinguish the hash function from a random oracle.
For most hash functions, such a relation will never hold with probability one. So there are two ways of relaxing this statement. First the relation does not have to hold for the whole output, if there are certain bits for which a differential relation holds, the function may already be distinguishable from a random oracle. Secondly, the probability with which the relation holds can be estimated, and if this is high enough the function can also be distinguished.
In the attack the difference between the two variables is known, but this does not define the variables, i.e. one could have a difference in the first byte, so at least one of the eight bits in the byte of $\Delta x = x \boxdot x \boxdot \Delta$ is flipped with respect to x, but the precise difference is unknown. This is why the relation will not always hold with probability 1.

Differential properties

Differential cryptanalysis has evolved over the years and there is still much to discover. Different type of differences have different advantages and disadvantages. Many hash functions are designed to have many different operations, such that there is no difference proceeding through the whole function with probability 1. Over the years more different ways of differential attacks became known, we will highlight a few of those.
Daum proved a few theorems on XOR and modular addition differences [14]. The proof of each theorem is given in Appendix D.
Daum does not only look for equality, but also some low–weight non–zero difference, e.g. 2^{n-r}.
We will state some of Daum's results now:

$$\begin{aligned}
\Pr[(x+y) \lll_r = x \lll_r + y \lll_r] &= \tfrac{1}{4}\left(1 + 2^{-n-r} + 2^{-n} + 2^{-r}\right) \\
\Pr[(x+y) \lll_r = x \lll_r + y \lll_r + 2^{n-r}] &= \tfrac{1}{4}\left(1 + 2^{-n-r} + 2^{-n} + 2^{-r}\right) \\
\Pr[(x+y) \lll_r = x \lll_r + y \lll_r + 1] &= \tfrac{1}{4}\left(1 + 2^{-n-r} + 2^{-n} + 2^{-r}\right) \\
\Pr[(x+y) \lll_r = x \lll_r + y \lll_r + 2^{n-r} + 1] &= \tfrac{1}{4}\left(1 + 2^{-n-r} + 2^{-n} + 2^{-r}\right).
\end{aligned}$$

Lipmaa and Moriai studied XOR differentials through modular addition and multiplication [15]. Indesteege and Preneel used this study for XOR differences of words with small Hamming weight to attack EnRUPT[7] [16]. Isobe and Shirai use this for a differential attack on Shabal [17].
Let

$$i_j = 0\underbrace{1\ldots1}_{n-j-1}\underbrace{0\ldots0}_{j}$$

[7] EnRUPT is a first–round–candidate of the SHA–3 competition

and wt is the weight function, i.e. the Hamming distance of the word and the all zero word. Let $\wedge$ represent the AND function, then the following probabilities are given:

$$\begin{aligned} \Pr[(x+y) \oplus ((x \oplus \alpha) + (y \oplus \beta)) = \alpha \oplus \beta] &= 2^{-wt((\alpha \vee \beta) \wedge (2^{32}-1))} \\ \Pr[3x \oplus 3(x \oplus \alpha) = \alpha \oplus \alpha \ll_1] &= 2^{-wt((\alpha \vee (\alpha \ll_1)) \wedge i_1)} \\ \Pr[3x \oplus 5(x \oplus \alpha) = \alpha \oplus \alpha \ll_2] &= 2^{-wt((\alpha \vee (\alpha \ll_2)) \wedge i_2)} \end{aligned} .$$

In Appendix D we also prove some (more) statements for shifted pairs, rotational pairs, and pairs with a XOR difference.

Combined differential analysis

As most hash functions are based on modular addition as well as exclusive or, the combined differential has a higher success probability.
For example if we have two XOR differences, say $\Delta x = 0001, \Delta y = 1111$, which go trough addition, we find many possibilities for he new XOR difference:

x	x'	y	y'	$\Delta(x+y)$
0001	0000	1111	0000	0000
0001	0000	0000	1111	1110
0011	0010	1111	0000	0000
0011	0010	0000	1111	0010
0101	0100	1111	0000	0000
				...

And if we would have modular addition differences, say $\delta x = x - x' = 1, \delta y = 15$, which go through exclusive or, we again find many possibilities for the new difference:

x	x'	y	y'	$\delta(x+y)$
1	0	15	0	14
2	1	15	0	12
3	2	15	0	14
4	3	15	0	8
				...

Studying the combination of a XOR difference and a modular addition difference, leads to a more precise variable. Suppose we know the start differences $\delta x = 1, \delta y = 15$, then $\delta(x+y) = 0$

δx	δy	x	x'	y	y'	$\Delta(x+y)$	$\delta(x+y)$	$\delta x + \delta y$
1	15	0001	0000	1111	0000	0000	0	0
1	15	0011	0010	1111	0000	0000	4	0
1	15	0101	0100	1111	0000	0000	8	0
1	15	0111	0110	1111	0000	0000	12	0
1	15	1001	1000	1111	0000	0000	0	0
1	15	1011	1010	1111	0000	0000	4	0
1	15	1101	1100	1111	0000	0000	8	0
1	15	1111	1110	1111	0000	0000	12	0

Now we know that the variables must be either

$$x = 0001, x' = 0000, y = 1111, y' = 0000 \textbf{ or } x = 1001, x' = 1000, y = 1111, y' = 0000$$

Wang used this method to break MD5 [18].
The advantage of having a smaller solution space is that more information is known about what a difference does in certain operations of the hash function, for example, for which bitpositions there is a carry when modular addition is applied.

2.3.5 Boomerang Attack

The boomerang attack was first described by David Wagner, he used this to attack several ciphers, COCONUT98, FEAL and Khufu [19]. The attack relies on the existence of high probability differential paths for half the encryption operation.
For example, assume that the encryption operation E consists of two quite similar rounds, E_0 and E_1, thus $E(m) = E_1(E_0(m))$. Now if for both of the rounds there exists a high probability differential, but it is difficult to match the two differentials, the boomerang attack can be very useful.
In the boomerang attack, we need three differential message pairs: P, P', P, Q and P', Q'. Let $E(P) = C, E(P') = C', E(Q) = D$ and $E(Q') = D'$. Now for these message pairs and their images, we want the following equations to hold:

$$\begin{array}{rclcrcl} P \oplus P' & = & \Delta & \wedge & E_0(P) \oplus E_0(P') & = & \Delta^* \\ C \oplus D & = & \nabla & \wedge & E_1^{-1}(C) \oplus E_1^{-1}(D) & = & \nabla^* \\ C' \oplus D' & = & \nabla & \wedge & E_1^{-1}(C') \oplus E_1^{-1}(D') & = & \nabla^*. \end{array}$$

That is, we want to constrain the differences of the messages P, P', their images, and the differences of the outputs C, D and C', D' and their preimages. The goal of the attack is that now $Q \oplus Q' = \Delta$ and $E_0(Q) \oplus E_0(Q') = \Delta^*$.
The boomerang attack is an out-in-out attack, it starts with the messages and the digests, meets in the middle and goes back to the messages.

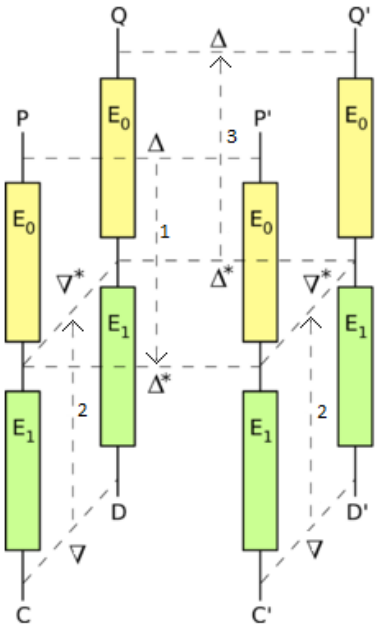

Figure 2.1: The boomerang attack

Construction of the attack We start with generating messages P and $P' = P \oplus \Delta$. These are queried to find the digests C and C'. Then we set the other two digests $D = C \oplus \nabla$ and $D' = C' \oplus \nabla$. Finally D and D' are decrypted to find Q and Q'.
If all characteristics are satisfied, you will end with two messages Q and Q' such that $Q' = Q \oplus \Delta$.
When the equations above hold, we have:

$$\begin{aligned} E_0(Q) \oplus E_0(Q') &= E_0(P) \oplus E_0(P') \oplus E_0(P) \oplus E_0(Q) \oplus E_0(P') \oplus E_0(Q') \\ &= E_0(P) \oplus E_0(P') \oplus E_1^{-1}(C) \oplus E_1^{-1}(D) \oplus E_1^{-1}(C') \oplus E_1^{-1}(D') \\ &= \Delta^* \oplus \nabla^* \oplus \nabla^* = \Delta^*. \end{aligned}$$

Application to Hash Functions

The boomerang attack is immediately applicable to hash functions. The attack can be applied to the inner primitive and is very useful if there exists a high probability differential path for the first half of the inner primitive, but when the path is extended to the full primitive the probability of success decreases rapidly.

2.3.6 Rebound Attack

The rebound attack is an attack with two phases, an inbound phase and an outbound phase. The element to attack is divided in three parts. For example, if the hash algorithm consists of 5 rounds, it can be divided in 2, 1 and 2 rounds. If the first two and the last two rounds are linear functions, and the middle round is not, then the differentials for the middle round will decrease the probability for the differential of the full function. Therefore it would be convenient to solve the middle round first and then expand this forward and backward to the input respectively the output.
In the rebound attack the middle part can contain one or more rounds. The differential for the middle part is solved first. We choose the input and the output of the middle part and match this via a meet-in-the-middle attack. Then the two outer parts are solved from the inside out. This attack is especially useful when the element contains substitution boxes (S-BOX).

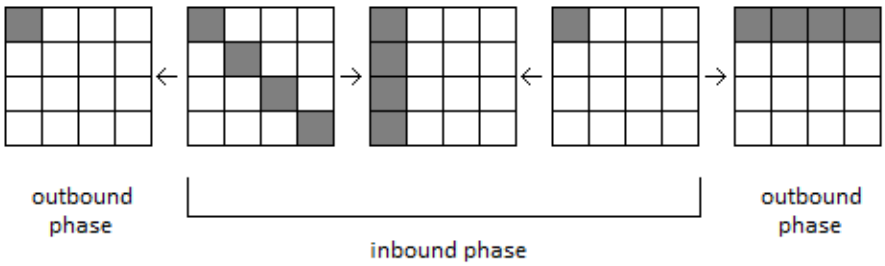

Figure 2.2: The rebound attack

In Figure 2.2 we consider a function with an inner state of 16 words. The state is changed four times and the grey blocks represent the words which contain a difference. We start by choosing the second and the fourth state, these are matched in the middle (state 3) and if that is successful, we try to expand the attach by moving outwards from state 4 and 2.

2.3.7 AIDA/Cube Attack

The cube attack was first described by Dinur and Shamir [20]. Although Vielhaber stated that it is the same as his earlier published attack "AIDA" [21, 22], the general view is that the Cube Attack is surely an improvement of AIDA. The goal of the cube attack is to describe secret variables by a low degree polynomial of public variables. Each boolean function[8] $p(x_1, \ldots, x_n)$ admits an Algebraic Normal Form (ANF), i.e.

$$p(x_1, \ldots, x_n) = a_0 + a_1 x_1 + \ldots + a_n x_n + a_{1,2} x_1 x_2 + \ldots + a_{1,\ldots,n} x_1 \cdot \ldots \cdot x_n.$$

Let I be a subset of indices. We write t_I for the product of all x_i for which $i \in I$ and let l_I be the number of elements in I. Now we can factor t_I out of some terms of the ANF polynomial:

$$p(x_1, \ldots, x_n) = t_I p_{S(I)} + q(x_1, \ldots, x_n).$$

Here $p_{S(I)}$ is called the *superpoly* of I in p. The superpoly is as large as possible, i.e. every term of q misses at least one variable of t_I and the superpoly is the sum of the terms of t_I. The elements in t_I are called the cube variables. Now t_I defines a l_I–dimensional cube C_I and if all elements of t_I are set (to either 0 or 1) we have a vertex of the cube. Each vertex v of the cube defines a polynomial p_v, as the binary presentation of the vertex give the coefficients of the polynomial.

Lemma 1. *For any polynomial p and a subset of variables $\{x_i : i \in I\}$ we have*

$$\sum_{v \in C_I} p_v \equiv p_{S(I)} \mod 2.$$

First we note that for every term of $q(x_1, \ldots, x_n)$ is added an even number of times in $\sum_{v \in C_I} p_v$, since for $x_j \in t_I$ and $x_j \notin q(x_1, \ldots, x_n)$ it is added for $x_j = 0$ and for $x_j = 1$. So these terms will cancel modulo 2.
The term $t_I p_{S(I)}$ only exists when v is the all 1 vector (otherwise $t_I = 0$).
Now it is easy to see that

Proof.

$$\begin{array}{rcl} \sum_{v \in C_I} p_v \mod 2 & = & \sum_{v \in C_I} t_I \cdot p_{S(I)} + q(x_1, \ldots, x_n) \mod 2 \\ & = & \sum_{v \in C_I} t_I \cdot p_{S(I)} \mod 2 \\ & = & 1 \cdot p_{S(I)} \mod 2. \end{array}$$

□

We call t_I *maxterm* if $p_{S(I)}$ is of degree 1, so $p_{S(I)}$ is a linear polynomial but not a constant. Now the superpoly of a maxterm can be computed via blackbox queries to the algorithm.

- The free term is the sum of the images of all the vertices of the cube (all other bits are zero).
- The coefficient of $x_j \notin t_I$ is the sum of the images of all the vertices of the cube, where bit x_j is one and the remaining bits are zero.

Now each output bit can be described by a polynomial in terms of the input and key bits.
This is especially applicable to stream and block ciphers, since they use key bits (secret) and have a bit output (public). For a cipher (part of) the key can be exposed, for a hash function,

[8] A boolean function is a function that maps $\{0,1\}^n$ to $\{0,1\}^m$ for some n and m.

some output bits will be known without a query to the function and thus is it possible to distinguish the function from an ideal random function.
Biham and Chen first described neutral bits in the context of cryptographic functions [23]. They used neutral bits to find collisions.

Definition 3 (Neutral bits)**.** *Let $\mathcal{R}$ be a relation that holds for couples of messages, for example $m \oplus m' = \delta_{\mathcal{R}}$, and let $\mathcal{M}_{\mathcal{R}}$ be the set of couples that satisfy this relation. Let m be a message, then $m[i]$ is m with bit i complemented. And for a set $s = \{i, j, \ldots\}$, all bits at the bit–positions in s are complemented in $m[s] = m[i, j, \ldots]$.*
For $(m, m') \in \mathcal{M}_{\mathcal{R}}$, a bit i is said to be neutral if also $(m[i], m'[i]) \in \mathcal{M}_{\mathcal{R}}$.
A pair of bits i, j is said to be neutral if for $(m, m') \in \mathcal{M}_{\mathcal{R}}$ also the messages arising from complementing any subset of the bits, are in $\mathcal{M}_{\mathcal{R}}$:

$$(m[i], m'[i]), (m[i, j], m'[i, j]), (m[j], m'[j]) \in \mathcal{M}_{\mathcal{R}}.$$

Similarly, a set of bits S is neutral if for any subset $s \subset S$:

$$(m, m') \in \mathcal{M}_{\mathcal{R}} \Rightarrow (m[s], m'[s]) \in \mathcal{M}_{\mathcal{R}}.$$

And, a set of bits S is k–neutral if any subset $s \subset S$ of size k is a neutral set.

The precise usage of neutral bits may differ, but the general idea is that the relation is invariant under alternating the bits.
To find a set that is maxterm, Dinur and Shamir describe a random start search [20].

Random start search

Use the black box polynomial as a polynomial in the secret variables with as input the public variables. When searching for an image, use input respectively output, when searching for a preimage, use output respectively input.
First pick a random index set of chosen size. Then compute the sum over the cube given by the index set (the other public positions are static, could be random or all zero[9]). Now if this gives a (non-constant) linear function in the secret variables, this set might lead to an attack. If the function is non-linear, the index set might be too small and we should restart after adding one more public variable to the cube. If the function is constant, the index set might be too big, so restart after dropping a public variable.
If after a restart one is send backwards, there might not be a solution containing this set, so restart with a new randomly chosen set.
There is some uncertainty in the precious statement because there are situations in which one can not find a helpful index set.

For the attack we need a sufficient number of index sets which give a linear polynomial in the secret variables. Then by solving the set of equations one could find linear equations for some secret bits.

[9] Dinur and Shamir state that these can be random, but use all zero in their application [20]. Zhu, Yu and Wang state that using random values will lead to many failures [24]

Cube tester

A Cube tester is a distinguisher between the system with the hash algorithm and a random function, based on cube search. For example there could be output bits of the hash function, which are neutral if compared over specified input bits (the cube variables), e.g. the sum over the cube with the neutral bit 0 is equal to the sum over the cube with the neutral bit 1. For example, if we had

$$f(x_1, \ldots, x_n) = x_2 \oplus \ldots \oplus x_n$$

we could let the set of cube variables be the empty set and x_1 is neutral under this set:

$$f(0, x_2, \ldots, x_n) = f(1, x_2, \ldots, x_n).$$

The bigger the set of cube variables, the larger the complexity of the attack, as we need to sum over all cube variables.

Chapter 3

Shabal

Fourteen hash functions advanced to the second round of the NIST SHA–3 competition [2], Shabal is one of them. Shabal has been submitted by Jean–François Misarsky et. al. The name of the algorithm was chosen as a tribute to Sébastien Chabal, a French rugby player known for his aggressive playing as well as his beard and long hair which got him the nickname of "Caveman" [25].
The authors claim Shabal to be indifferentiable from a random oracle, even when the inner primitive is biased.
In December 2010 NIST announced the final round candidates, Shabal was not one of them. In February NIST released the document describing the reasons for their decisions [26]. The security claimed by the authors was not threatened at the time of the decision, but the non–randomness of the inner primitive raised concerns. The renewed proof of indifferentiability which takes the bias of the inner primitive into account, did not convince NIST that Shabal would remain secure and is the main reason that Shabal did not advance to the final round.
Nevertheless I enjoyed working on Shabal even after knowing that they where out of the competition. In this chapter I will first describe Shabal, discuss the recent analysis and the indifferentiability proofs. Then I will explain how I tried to attack Shabal.

3.1 The mode of Operation

Shabal is a hash function based on a keyed–permutation, this permutation is an NLFSR–construction[1].
The compression functions of Shabal operates on 512–bit message blocks and a counter that states which block is inserted this round. The message is padded with a 1–bit followed by as many 0–bits such that the length of the message is an integral multiple of 512–bits. The internal state of Shabal consists of three arrays A, B and C of length 384–, 512– and 512–bits respectively. The message blocks are inserted in the message rounds, and when all message blocks are inserted three final rounds follow. Each round consists of applying the compression function. The final rounds are similar to the message rounds, but instead of a new message block, the last block is inserted and the counter w is fixed to the total number of message blocks.
The length of the digest is l words, the digest is given by the last l 32–bit words of C. In this chapter we will use word to refer to a 32–bit sequence unless stated otherwise.

[1] NLFSR means Non–Linear–Feedback–Shift–Register

3.1.1 The Compression function $\mathcal{R}$

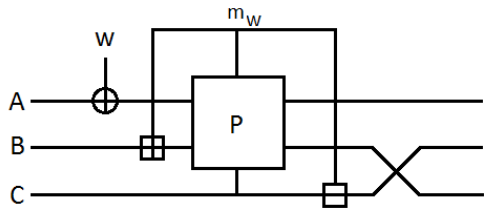

Figure 3.1: Compression function of Shabal

- A is XOR–ed with the message block counter w
- the message block m_w is added modulo 2^{32} to B (wordwise)
- a permutation is applied to A and B with keys C and m_w
- m_w is subtracted modulo 2^{32} from C (wordwise)
- B and C switch places

The Permutation

The permutation consists of three steps.
In the first step only B is modified, in the second step A and B are synchronically modified and in the last step only A is modified.
Here $+$ is addition modulo 2^{32}, $\oplus$ is the XOR operator, $\mathcal{U}$ is multiplication by 3 modulo 2^{32}, $\mathcal{V}$ is multiplication by 5 modulo 2^{32}, $\wedge$ is the bitwise AND operator, $\lll_n$ is rotation to the left by n bits and $\overline{a}$ is the bitwise negation of a.
The three steps of the permutation are:

1. The words of B are left–rotated over 17 bits, i.e. $B = B \lll_{17}$.
2. The functions in this step are applied wordwise. The words of B are updated thrice, thus each word of A is updated four times. We have:

for $i = 0$ to 2 do
  for $j = 0$ to 15 do
    $A[j + 16i \mod 12] \leftarrow \mathcal{U}(A[j + 16i \mod 12] \oplus \mathcal{V}(A[j - 1 + 16i \mod 12] \lll_{15})$
      $\oplus C[8 - j \mod 16])$
      $\oplus B[j + 13 \mod 16]$
      $\oplus (B[j + 9 \mod 16] \wedge \overline{B[j + 6 \mod 16]})$
      $\oplus M[j]$
    $B[j] \leftarrow (B[j] \lll_1) \oplus A[j + 16i \mod 12]$
  end for
end for.

3. In the last step each word of A is updated three times,
   for $j = 0, \ldots, 35$ do $A[j \mod 12] \leftarrow A[j \mod 12] + C[j + 3 \mod 16]$

The initial Values

Shabal–256

A: 52F84552 E54B7999 2D8EE3EC B9645191 E0078B86 BB7C44C9 D2B5C1CA B0D2EB8C
14CE5A45 22AF50DC EFFDBC6B EB21B74A

B: B555C6EE 3E710596 A72A652F 9301515F DA28C1FA 696FD868 9CB6BF72 0AFE4002
A6E03615 5138C1D4 BE216306 B38B8890 3EA8B96B 3299ACE4 30924DD4 55CB34A5

C: B405F031 C4233EBA B3733979 C0DD9D55 C51C28AE A327B8E1 56C56167 ED614433
88B59D60 60E2CEBA 758B4B8B 83E82A7F BC968828 E6E00BF7 BA839E55 9B491C60

Shabal–512

A: 20728DFD 46C0BD53 E782B699 55304632 71B4EF90 0EA9E82C DBB930F1 FAD06B8B
BE0CAE40 8BD14410 76D2ADAC 28ACAB7F

B: C1099CB7 07B385F3 E7442C26 CC8AD640 EB6F56C7 1EA81AA9 73B9D314 1DE85D08
48910A5A 893B22DB C5A0DF44 BBC4324E 72D2F240 75941D99 6D8BDE82 A1A7502B

C: D9BF68D1 58BAD750 56028CB2 8134F359 B5D469D8 941A8CC2 418B2A6E 04052780
7F07D787 5194358F 3C60D665 BE97D79A 950C3434 AED9A06D 2537DC8D 7CDB5969

3.2 Recent Analysis

3.2.1 On the permutation only

Aumasson noted the existence of a distinguisher based on a cube–tester which has a complexity of 2^{300} against 2^{448} ideally [27]. This implies that the permutation is not pseudo–random, but this has no effect on the security of Shabal. This attack is evaluated in Section 3.5.
Knudsen et al. describe how to find state conserving values (fixed points) A, B, C and M and key–collisions for the permutation [28]. They choose A, B and C in both cases, and since Shabal uses initial values which are different this is not applicable to full Shabal, this is evaluated in Section 3.8.
Aumasson et al. described a related–key distinguisher [29], the distinguisher queries for $P_{M,C}(A, B)$ and $P_{M',C'}(A, B)$, so only values for M and C are chosen. Again this is not applicable to full Shabal, since then C and C' can not be chosen. They also found pseudo–collisions and pseudo–second–preimages for a reduced version of the permutation, they reduce the number of rounds in step 3 from 36 to 24. They prove that this method can not be applied to the full permutation.
The submitters of Shabal responded to the non–pseudo–randomness of the permutation by stating that Shabal does not need it to be pseudo–random [30]. They have rewritten the original security proofs (based on the indifferentiability of the permutation of a random oracle) to security proofs based on the fact that for the permutation there exists a certain, known input–output relation. The proof is described in Section 3.4.
A rotational distinguisher for the permutation function was described by Van Assche [31]. This again shows that the permutation can be distinguished from an ideal permutation, but it can not be applied to the full Shabal since the initial values of Shabal are not rotational, the addition of the block counter is not rotational and the final rounds of Shabal will decrease the rotational probability. This distinguisher does not require less queries than the previous distinguishers.
In addition Van Assche showed that a reduced version of Shabal's mode of operation can be distinguished from a random oracle. The reduced version uses different IV's, no final rounds and no block counter. This implies that the security of Shabal relies on the non–symmetric IV's, the counter and the final rounds. This attack is evaluated in Section 3.12.

Novotney introduces a distinguisher based on differential analysis which has complexity 2^{23} [32]. This can not be applied to full Shabal because of the initial values and the final rounds.

3.2.2 On the compression function $\mathcal{R}$

Aumasson shared an observation on the compression function [33], which is a pseudo–near–collision. For chosen A, B, C, m and M' we have

$$\mathcal{R}(A, B, C, m) = (A', C - M, M)$$

and

$$\mathcal{R}(A, B, C, m') = (A', C - M', M').$$

Here $C - M$ and $C - M'$ differ only on 1 bit in each word (so 16 differences). Here only the message blocks contain a difference, but again, it can not be used for full Shabal since A, B and C are chosen.

3.2.3 On full Shabal

The first results (except for the submission document) on full Shabal were published by Isobe and Shirai [17].
They first describe a near–collision attack on the compression function backwardly, and then use this to describe a pseudo–collision attack. They start with (A, B, C) and (A', B', C') which differ in 45 bits, apply the compression function to those internal states and M and M' respectively to end with a collision with probability 2^{-184}.
Secondly they describe an attack on two reduced versions of Shabal, first where step 2 of the permutation is only repeated twice and then on the reduction where A has length 256–bits and step 2 of the permutation is repeated only 1.5 times. For these attacks they use the Guess–and–Determine technique.
The Guess–and–Determine technique (GD) consists of three steps, first, obtain equations for relations between the message block and the internal state. The guess part of the message block. In step three we use this to determine the rest of the message block using the equations of step one.
For the first variant Isobe and Shirai find complexity 2^{497} (with 2^{400} memory) and for the second variant they find complexity 2^{497} (with 2^{272} memory). Again this has no consequences for the security of Shabal. This attack is evaluated in Section 3.11.

3.3 Indifferentiability

In the submission document of Shabal [25] it is claimed that the mode of operation of Shabal is indifferentiable from a random oracle up to $2^{(l_a+l_m)/2}$ queries, where l_a is the length in bits of A and l_m is the length in bits of B and C. The proof is based on the ideal cipher model, which was showed not to be correct. The second proof [30] uses a biased inner primitive for which two values play a special role, the forward bias τ and the backward bias λ of the relation that holds (with high probability) for the inner primitive.

First we will discuss the original proof, followed by the proof including the biased permutation. Then we will address a third proof [34].

3.3.1 The original proof of Indifferentiability

The first proof was given in the submission document of Shabal, this proof does not take the inverse of the permutation into account, but this is improved in the second proof.
The authors of Shabal claim that for a distinguisher with respect to the mode of operation of Shabal after at most N calls

$$\mathrm{ADV}(\mathcal{D}, \mathcal{C}) \leq N(2N-1) \cdot 2^{-l_a - l_m}$$

holds. Recall that l_a, l_m represents the bit length of A respectively B.

Definitions

M the message, element of $\{0,1\}^*$.

m one block of the message, element of $\{0,1\}^{512}$.

(A, B, C, m) element of $\{(\{0,1\}^{384}, \{0,1\}^{512}, \{0,1\}^{512}, \{0,1\}^{512})\}$.

Insert applies all the operations except for the permutation to a tuple:
$\mathrm{Insert}[m_i, i](A, B, C, m) = (A \oplus i, C - m + m_i, B, m_i)$.

x a tuple (A, B, C, m) *before* Insert is applied.

y a tuple (A, B, C, m) *after* Insert is applied, i.e. $\mathrm{Insert}[m](x) = y$.

$\mathcal{X}$ is the set of all x.

$\mathcal{Y}$ is the set of all y.

x_0 the initial value (A, B, C, m), where m is the all zero sequence.

h element of $\{0,1\}^{512}$.

$\mathcal{P}$ is the permutation (input (A, B, C, m); output (A', B')).

$\mathcal{C}^{\mathcal{P}}$ represents the mode of operation (initialization, message rounds and final rounds; input M; output h).

$\mathcal{H}$ is a random oracle, which outputs a 512–bit value.

$\mathcal{S}$ is a simulator for $\mathcal{P}$ (input (A, B, C, m); output (A', B')).

$\mathcal{I}$ is a simulator of $\mathcal{C}^{\mathcal{P}}$ (input: m; output h).

$\mathcal{G}$ is the graph that the simulator keeps up to date during the game, the graph contains vertices x and y, edges starting in y ending in x representing a permutation step.

PATH is a path in $\mathcal{G}$, given by the vertices
$y_1 = \mathrm{Insert}[m_1](x_0), x_1, y_2 = \mathrm{Insert}[m_2](x_1), \ldots, y_k = \mathrm{Insert}[m_k](x_{k-1}), x_k$ for some message block $M = m_1 || m_2 || \ldots || m_k$.

We will also view the operations of the compression function in a different order (see Figure 3.2), in intermediate states this makes no difference from the original order. For the first and last step

it does differ, but if we modify the start and end procedure of the original version as follows, the two views collide. We use

- modified start values (B and C are swapped)
- zero for the first message block (m in Figure 3.2)
- the output B instead of C.

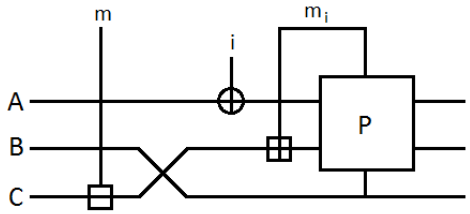

Figure 3.2: Different view on the compression function of Shabal

The proof is game based, between every two games the advantage of the distinguisher is given by the difference of the probability that the distinguisher outputs a one in the two games. The first game is the original game and in the last game the system is a random oracle. W_i represents the event of a 1 output in game i. The advantage of the distinguisher can then be bounded from above by:

$$|\Pr[W_0] - \Pr[W_9]| \leq \sum_{i=1}^{9} |\Pr[W_{i-1}] - \Pr[W_i]|.$$

The idea is to make a simulator $\mathcal{S}$ for the (assumed to be random) permutation $\mathcal{P}$. The simulator keeps track of all the previous queries in a graph. In the graph there is a difference between queries directly from the distinguisher $\mathcal{G}_{\mathcal{D}}$ and queries from the cryptographic construction $\mathcal{G}_{\mathcal{C}}$.

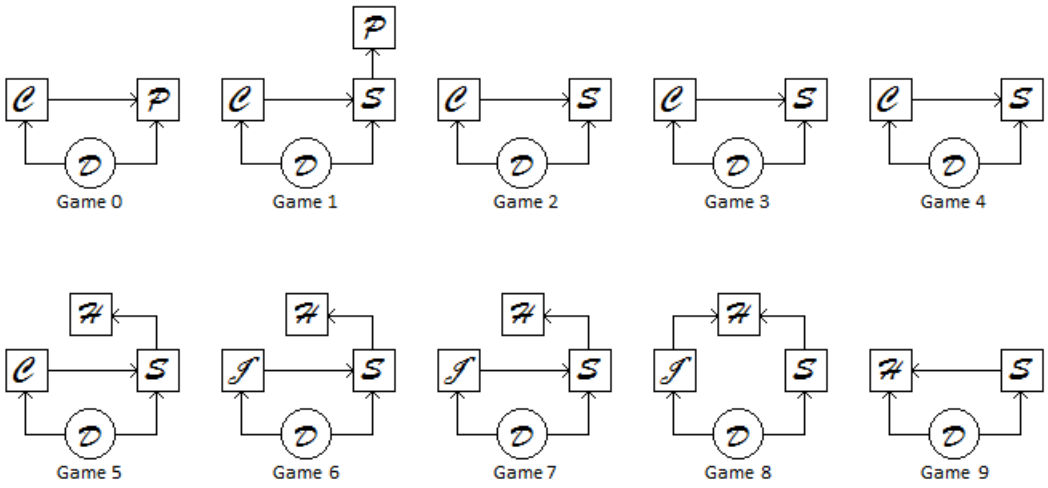

Figure 3.3: Graphical representation of the games

Game 0 This is the original game, $\mathcal{D}$ interacts with $\mathcal{P}$ and $\mathcal{C}^{\mathcal{P}}$ which represents a random keyed permutation respectively the compression function. The compression function executes for every message block the following two subroutines:

$$(A, B, C, m) = \text{Insert}[m_i, i](A, B, C, m)$$

$$(A, B) = \mathcal{P}(A, B, C, m)$$

where

$$\text{Insert}[m_i, i](A, B, C, m) = (A \oplus i, C - m + m_i, B, m_i). \tag{3.1}$$

And afterwards the final rounds (which are defined to apply the two subroutines using the last message block and the number of message blocks as counter i).

Game 1 $\mathcal{P}$ is replaced by the simulator $\mathcal{S}$, which forwards calls to $\mathcal{P}$ and returns the responses. $\mathcal{S}$ constructs the graph $\mathcal{G} = \mathcal{G}_{\mathcal{C}} \cup \mathcal{G}_{\mathcal{D}}$ throughout the game. $\mathcal{D}$ can see no difference between the two games, thus $\Pr[W_1] = \Pr[W_0]$. The simulator is given by:

```
Initialisation of S in Game 1
  No input, no output
      1.  set G = ∅
Simulation of P
  Input:  y = (A, B, C, m), origin O
  Output: (A', B')
      1.  add node y to G_O
      2.  call P to get (A', B') = P(A, B, C, m)
      3.  add node x = (A', B', C, m) and edge y → x to G_O
      4.  return (A', B') to O
```

Game 2 In this game $\mathcal{S}$ no longer queries $\mathcal{P}$, but selects a response randomly. The authors state that under the assumption that $\mathcal{P}$ is an ideal random permutation, this does not change the distribution. So $\Pr[W_1] = \Pr[W_2]$.

In my opinion, this is not true. As $\mathcal{P}$ is a permutation and thus injective, we find

$$\Pr[\mathcal{P}_{m,c}(A, B) = \mathcal{P}_{m',c'}(A', B') | (A, B) \neq (A', B')] = 0$$

and for a random selection this is 2^{-896}, which is small, but not zero. And for the q^{th} query this gives probability at most $(q-1)2^{-896}$. So

$$|\Pr[W_2] - \Pr[W_1]| \leq (q-1)2^{-896}.$$

A solution would be to randomly choose another value every time a colliding value is selected, then $\Pr[W_1] = \Pr[W_2]$ would hold. This would (slightly) increase the time between a query and a response for $\mathcal{S}$, so a side channel attack might be possible. Another option is to remove this game, and go from game 1 to game 3, nothing in the evaluation of $|\Pr[W_3] - \Pr[W_1]|$ will be different of the analysis between game 2 and 3 as we will see later.

Game 3 We define two events for which the simulator will abort, these events protect the simulator from giving outputs that the permutation would never give. The simulator has to

check whether the value it selects gives a collision in the future, it aborts if one of the following events is true for unequal $x \neq \tilde{x}$:

Coll0 x and $\tilde{x}$ admit a PATH of k consecutive evaluations of the subroutines of the compression function in the graph $\mathcal{G} = \mathcal{G}_\mathcal{C} \cup \mathcal{G}_\mathcal{D}$ and there exists a y such that $y = \text{Insert}[m_k, k](x) = \text{Insert}[\tilde{m}_k, k](\tilde{x})$. In words, after digesting two sequences of $k+1$ message blocks the same internal state is reached, this is an internal collision. If after this (at least once) only equal message blocks are digested to both evaluations, the hash outputs will collide,

Coll1 x and $\tilde{x}$ both admit a PATH of the subroutines of the compression function and all but one of the final rounds, in the graph $\mathcal{G} = \mathcal{G}_\mathcal{C} \cup \mathcal{G}_\mathcal{D}$ and there exists a y such that applying the last final round to x and $\tilde{x}$ would result in y. In words, after digesting two sequences of message blocks and the final rounds the same state is reached, so the hash outputs are colliding.

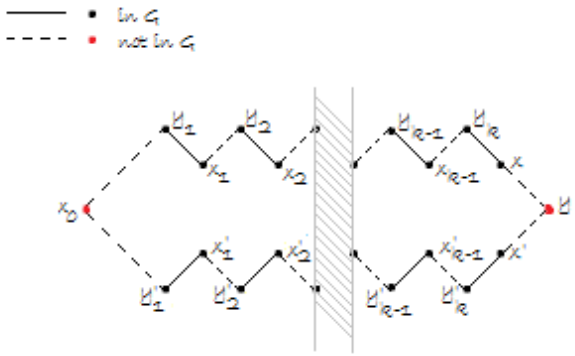

Figure 3.4: A graphical representation of colliding paths in the graph.

Note that in the latter case we must have that the last message blocks are equal, as the state y contains the last message block too. The first thought one might have is that (A, B, C, m) and (A', B, C', m') would give a collision but the abort events return false. But this happens with the same probability in game 3 as in the original game under the assumption that $\mathcal{P}$ is ideal, so we can ignore this case while estimating the difference in the probabilities between the two games. The new simulator is given by:

Initialisation of $\mathcal{S}$ in Game 3
No input, no output
1. set $\mathcal{G} = \emptyset$
Simulation of $\mathcal{P}$
Input: $y = (A, B, C, m)$, origin $\mathcal{O}$
Output: (A', B')
1. add node y to $\mathcal{G}_\mathcal{O}$
2. if there exists $y \to x \in \mathcal{G}_\mathcal{O}$
(a) return (A', B') where $x = (A', B', C, m)$
(b) add node x to $\mathcal{G}_\mathcal{O}$
3. randomly select $A' \leftarrow \{0,1\}^{l_a}$ and $B' \leftarrow \{0,1\}^{l_m}$
4. add node $x = (A', B', C, m)$ and edge $y \to x$ to $\mathcal{G}_\mathcal{O}$
5. if $\exists \tilde{x} \in \mathcal{G}_\mathcal{O}$ such that ABORT_1 then abort
6. return (A', B') to $\mathcal{O}$.

The simulator $\mathcal{S}$ now aborts if it selects a response x for which there exists an $\tilde{x}$ such that one of the events **Coll0** or **Coll1** happens, we name this ABORT_1. In this game, each state in the graph $\mathcal{G}$ admits at most one PATH, since a second path would have caused the simulator to abort.
When the events described above are not true, the probability that the distinguisher outputs a one in game 3 is equal to the probability that the distinguisher outputs a one in game 2, so we can use the **difference lemma**.

Lemma 2. *Difference Lemma.*
Let U, V, E be three events such that

$$\Pr[U \wedge \neg E] = \Pr[V \wedge \neg E],$$

then

$$|\Pr[U] - \Pr[V]| \leq \Pr[E].$$

Proof.

$$\begin{aligned} |\Pr[U] - \Pr[V]| &= |\Pr[U \wedge E] + \Pr[U \wedge \neg E] - \Pr[V \wedge E] - \Pr[V \wedge \neg E]| \\ &= |\Pr[U \wedge E] - \Pr[V \wedge E]| \leq \Pr[E] \end{aligned}$$

□

So we have

$$|\Pr[W_3] - \Pr[W_2]| \leq \Pr[\text{ABORT}_1].$$

An upper bound for $\Pr[\text{ABORT}_1]$ is an upper bound for the advantage of the distinguisher in game 3 compared to game 2.
Let us estimate $\Pr[\text{ABORT}_1]$ using:

$$\begin{aligned} \Pr[\text{ABORT}_1] &= \Pr[\textbf{Coll0} \vee \textbf{Coll1}] \\ &\leq \Pr[\textbf{Coll0}] + \Pr[\textbf{Coll1}]. \end{aligned}$$

When the q^{th} query is requested to $\mathcal{S}$, there are at most $q-1$ states $\tilde{x}$ in graph $\mathcal{G}$. As every (A, B) part of the states has been chosen randomly we can bound $\Pr[\textbf{Coll0}]$ at the q^{th} query as follows:

$$\begin{aligned} \Pr[\textbf{Coll0}] &\leq (q-1)\Pr[\exists m' : \text{Insert}[m', k](A, B, C, m) = \text{Insert}[m', k](\tilde{A}, \tilde{B}, \tilde{C}, \tilde{m})] \\ &= (q-1)\Pr[A = \tilde{A} \wedge B = \tilde{B} \wedge C - m = \tilde{C} - \tilde{m}] \\ &= (q-1)2^{-l_a}2^{-l_m}\Pr[C - m = \tilde{C} - \tilde{m}] \\ &\leq (q-1)2^{-l_a}2^{-l_m}. \end{aligned}$$

And the bound for $\Pr[\textbf{Coll1}]$ is given by

$$\begin{aligned} \Pr[\textbf{Coll1}] &\leq (q-1)\Pr[\text{Insert}[m_i, i](A, B, C, m) = \text{Insert}[m_j, j](\tilde{A}, \tilde{B}, \tilde{C}, \tilde{m})] \\ &= (q-1)\Pr[A = \tilde{A} \wedge B = \tilde{B} \wedge C = \tilde{C} \wedge m_i = m_j] \\ &= (q-1)2^{-l_a}2^{-l_m}\Pr[C = \tilde{C} \wedge m_i = m_j] \\ &\leq (q-1)2^{-l_a}2^{-l_m}. \end{aligned}$$

Therefore $\Pr[\text{ABORT}_1(q)] \leq 2(q-1)2^{-l_a}2^{-l_m}$. As we are allowing only N calls, we have:

$$\Pr[\text{ABORT}_1] \leq \sum_{q=1}^{N} \Pr[\text{ABORT}_1(q)].$$

And so

$$|\Pr[W_3] - \Pr[W_2]| \leq \Pr[\text{ABORT}_1] \leq N(N-1)2^{-l_a - l_m}.$$

In the case that game 2 does not exist, so we want to estimate $|\Pr[W_3] - \Pr[W_1]|$*, we find that again*

$$|\Pr[W_3 \wedge \neg\text{ABORT}_1] = \Pr[W_1 \wedge \neg\text{ABORT}_1]|,$$

and thus

$$|\Pr[W_3] - \Pr[W_1]| \leq \Pr[\text{ABORT}_1].$$

So in order to minimize the upper bound for the advantage of the distinguisher it is better to remove game 2 and go from game 1 directly to game 3

Game 4 Now we define two events which make sure that an input y either has a PATH on the moment it is added to the graph or if it does not have a PATH when it is added, it will never get one. We do this to catch the situation that an attacker correctly guesses an intermediate state of a path, since in this case, the attacker could find the hash of a message without querying the whole message (see Figure 3.5).

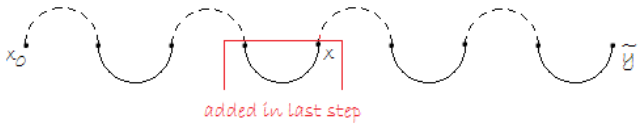

Figure 3.5: When adding x to the graph which would give $\tilde{y}$ a PATH the simulator aborts.

The simulator now checks if the image x it selected, is a dependency for some $\tilde{y}$ already in the graph, it aborts if one of the following events is true for this x and $\tilde{y} \in \mathcal{G}$:

Dep0 There is a path of k compression steps from x to $\tilde{y}$ in $\mathcal{G}$; i.e. $\tilde{y}$ has been queried before, and now we would introduce a PATH for $\tilde{y}$.

Dep1 There is a path of k compression steps and one or more final rounds from x to $\tilde{y}$; i.e. $\tilde{y}$ has been queried before, and now we would introduce a PATH for $\tilde{y}$.

The simulator $\mathcal{S}$ now aborts if it selects a response x for which there exists an $\tilde{y}$ such that one of the events **Dep0** or **Dep1** happens, we name this ABORT_2. The new simulator is given by:

Initialisation of $\mathcal{S}$ in Game 4
No input, no output
1. set $\mathcal{G} = \emptyset$
Simulation of $\mathcal{P}$
Input: $y = (A, B, C, m)$, origin $\mathcal{O}$
Output: (A', B')
1. add node y to $\mathcal{G}_{\mathcal{O}}$
2. if there exists $y \to x \in \mathcal{G}_{\mathcal{O}}$
(a) return (A', B') where $x = (A', B', C, m)$
(b) add node x to $\mathcal{G}_{\mathcal{O}}$
3. randomly select $A' \leftarrow \{0,1\}^{l_a}$ and $b' \leftarrow \{0,1\}^{l_m}$
4. add node $x = (A', B', C, m)$ and edge $y \to x$ to $\mathcal{G}_{\mathcal{O}}$
5. if $\exists \tilde{x} \in \mathcal{G}_{\mathcal{O}}$ such that ABORT_1 then abort
6. if $\exists \tilde{y} \in \mathcal{G}_{\mathcal{O}}$ such that ABORT_2 then abort
7. return (A', B') to $\mathcal{O}$.

Where $\mathcal{O}$ is either $\mathcal{D}$ or $\mathcal{C}$.
We use the difference lemma again to see:

$$|\Pr[W_4] - \Pr[W_3]| \leq \Pr[\text{ABORT}_2].$$

Let us estimate $\Pr[\text{ABORT}_2]$ using:

$$\begin{aligned} \Pr[\text{ABORT}_2] &= \Pr[\mathbf{Dep0} \vee \mathbf{Dep1}] \\ &\leq \Pr[\mathbf{Dep0}] + \Pr[\mathbf{Dep1}]. \end{aligned}$$

When the q^{th} query is requested to $\mathcal{S}$, there are at most $q-1$ states $\tilde{x}$ in graph $\mathcal{G}$. As every (A, B) part of the states has been chosen randomly we can bound $\Pr[\mathbf{Dep0}]$ at the q^{th} query as follows:

$$\begin{aligned} \Pr[\mathbf{Dep0}] &\leq (q-1)\Pr[\exists m' : \text{Insert}[m', k+1](x) = \tilde{y}] \\ &= (q-1)\Pr[m' = \tilde{m} \wedge A \oplus (k+1) = \tilde{A} \wedge B = \tilde{C} \wedge C - m + m' = \tilde{B}] \\ &= (q-1)2^{-l_a}2^{-l_m}\Pr[C - m + \tilde{m} = \tilde{B}] \\ &\leq (q-1)2^{-l_a}2^{-l_m}. \end{aligned}$$

Remember that we are now only looking at the final rounds (so the message block that we insert must be the same as the last message block) thus the bound for $\Pr[\mathbf{Dep1}]$ is given by

$$\begin{aligned} \Pr[\mathbf{Dep1}] &\leq (q-1)\Pr[\text{Insert}[m, k](x) = \tilde{y}] \\ &= (q-1)\Pr[m = \tilde{m} \wedge A \oplus k = \tilde{A} \wedge B = \tilde{C} \wedge C = \tilde{B}] \\ &= (q-1)2^{-l_a}2^{-l_m}\Pr[m = \tilde{m} \wedge C = \tilde{B}] \\ &\leq (q-1)2^{-l_a}2^{-l_m}. \end{aligned}$$

Therefore $\Pr[\text{ABORT}_2(q)] \leq 2(q-1)2^{-l_a}2^{-l_m}$. As there are at most N calls allowed, we have:

$$\Pr[\text{ABORT}_2] \leq \sum_{q=1}^{N} \Pr[\text{ABORT}_2(q)],$$

and thus

$$|\Pr[W_4] - \Pr[W_3]| \leq \Pr[\text{ABORT}_2 \leq N(N-1)2^{-l_a-l_m}].$$

Game 5 In game 5 the random oracle $\mathcal{H}$ is added to the game. $\mathcal{H}$ will be used to give the hash of a complete message. Thus $\mathcal{S}$ wants to know whether the current request x, is the last step of hashing a message, that is if it is the third final round. To obtain this information, $\mathcal{S}$ checks whether x has a PATH in the graph $\mathcal{G}$ which contains two final rounds before ending in x. If so, instead of selecting the B part randomly itself, $\mathcal{S}$ now queries the complete message M to $\mathcal{H}$. We call the function which finds the PATH and returns the complete message if required, **unpad**. This is to make sure that, when we replace $\mathcal{C}$ by $\mathcal{H}$, the hash obtained via the compression function, collides with the permutation of the last block if requested to the permutation directly. The abort rules stay intact, as both $\mathcal{H}$ and $\mathcal{S}$ return a random value for B nothing changes in respect to the previous game and thus $\Pr[W_5] = \Pr[W_4]$.

Game 6 In game 6 $\mathcal{C}$ is replaced by $\mathcal{I}$. When a query M is send to $\mathcal{I}$, $\mathcal{I}$ first queries $\mathcal{S}$ the same way $\mathcal{C}$ would do and then completely ignores the output, queries $\mathcal{H}$ and outputs $\mathcal{H}(M)$. We assume the permutation to be ideal, so the attacker will not see the difference between an output of $\mathcal{H}$ and an output of $\mathcal{P}$. And when the simulator $\mathcal{S}$ finds that this is the query for the last round, it will use the function unpad to get M and query M to $\mathcal{H}$. Now as $\mathcal{H}$ is queried M for the second time, it will output the same value.
The distinguisher will not notice the change and thus $\Pr[W_6] = \Pr[W_5]$.

Game 7 In this game we want to estimate the probability that a path exists in $\mathcal{G}_\mathcal{C}$ and not in $\mathcal{G}_\mathcal{D}$. Later $\mathcal{I}$ will no longer query $\mathcal{S}$, then $\mathcal{S}$ does not know the queries previously made to $\mathcal{H}$. For example, assume an attacker is able the find the penultimate inner state without querying the permutation and queries this to $\mathcal{S}$, then $\mathcal{S}$ does not know it is running a last final round of the hashing of a message, and will most probably not return the same value as $\mathcal{H}$ will give for the whole message.
We use the following event to describe the situation

Guess Given the graphs $\mathcal{G}_\mathcal{C}, \mathcal{G}_\mathcal{D}$ and a node $y \in \mathcal{G}_\mathcal{C} \cup \mathcal{G}_\mathcal{D}$, the event is true if and only if y admits a path in $\mathcal{G}_\mathcal{C} \cup \mathcal{G}_\mathcal{D}$ and not this path in $\mathcal{G}_\mathcal{D}$.

The simulator $\mathcal{S}$ now aborts if it is queried y for which the event **Guess** is true, we name this event ABORT$_3$. The new simulator is given by:

```
Initialisation of 𝒮 in Game 7
  No input, no output
      1.  set 𝒢 = ∅
Simulation of 𝒫
  Input:  y = (A, B, C, m), origin 𝒪
  Output: (A', B')
      1.  add node y to 𝒢_𝒪
      2.  if there exists y → x ∈ 𝒢_𝒪
          (a) return (A', B') where x = (A', B', C, m)
          (b) add node x to 𝒢_𝒪
      3.  if y has a path in graph 𝒢_𝒪
          (a) compute M = unpadμ
          (b) call ℋ to get h = ℋ(M)
          (c) set B' = h
      4.  else
          (a) randomly select b' ← {0,1}^{l_m}
      5.  randomly select A' ← {0,1}^{l_a}
      6.  add node x = (A', B', C, m) and edge y → x to 𝒢_𝒪
      7.  if ∃x̃ ∈ 𝒢_𝒪 such that ABORT_1 then abort
      8.  if ∃ỹ ∈ 𝒢_𝒪 such that ABORT_2 then abort
      9.  return (A', B') to 𝒪.
```

We use the difference lemma again to see:

$$|\Pr[W_7] - \Pr[W_6]| \leq \Pr[\text{ABORT}_3].$$

The authors state that $\Pr[\text{ABORT}_3(y)] \leq 2^{-l_a - l_m}$ due to the following methodology.

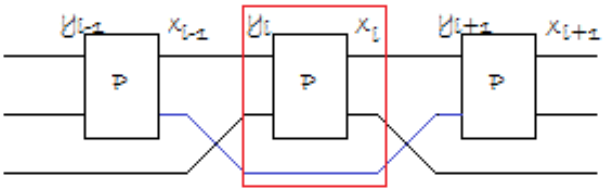

Figure 3.6: The two possibilities for event **Guess** to be true.

Assume **Guess** happens, then there must have been a query M to $\mathcal{I}$, such that there exists a PATH to y, for which the B–part is $\mathcal{H}(M)$ in $\mathcal{G}_\mathcal{C}$. Now assume this path is not in $\mathcal{G}_\mathcal{D}$, then there must be an edge of the path, say $y_i \to x_i$ in $\mathcal{G}_\mathcal{C}$ such that either x_i or y_i is not a vertex of $\mathcal{G}_\mathcal{D}$.
If $y_i \in \mathcal{G}_\mathcal{D}$ then there must be some x_i^* such that there exists an edge $y_i \to x_i^*$ in $\mathcal{G}_\mathcal{D}$. Now we must have that $x_i^* = x_i$ since for each input a unique output is selected by $\mathcal{S}$, so we have a contradiction.
Now let $y_i \notin \mathcal{G}_\mathcal{D}$. Now suppose $y_{i-1} \in \mathcal{G}_\mathcal{D}$, then the attacker knows the B value of y_{i-1}, and thus also the B value of y_{i+1} as it knows the message blocks and $B_{i+1} = (B_{i-1} - m_i) + m_{i+1}$ mod 2^{32} as shown in Figure 3.7. Given that the distinguisher correctly guesses the values for A and C of the state y_{i+1}, it can then query y_{i+1} which would result in the event **Guess**. The

probability of guessing this correctly is $2^{-l_a-l_m}$.

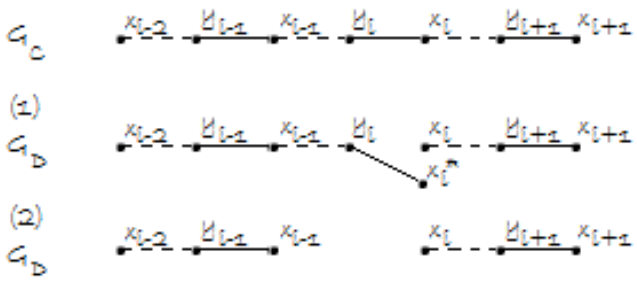

Figure 3.7: The attacker is able to find the B part of the second following state.

And thus $\Pr[\text{ABORT}_3(y)] \leq 2^{-l_a-l_m}$.
Assuming that there have been at most N queries, we find

$$\Pr[\text{ABORT}_3] \leq N2^{-l_a-l_m}.$$

Game 8 In this game $\mathcal{I}$ no longer queries $\mathcal{S}$, but just queries $\mathcal{H}$ and returns the response. The graph $\mathcal{G}$ of $\mathcal{S}$ now only consists of $\mathcal{G}_{\mathcal{D}}$. The test for ABORT$_3$ can be removed.
The distinguisher has no advantage due to this change and thus $\Pr[W_8] = \Pr[W_7]$.

Game 9 We now remove the interface $\mathcal{I}$, so the distinguisher communicates with $\mathcal{H}$ directly. Again this does not change the advantage of the distinguisher, $\Pr[W_9] = \Pr[W_8]$.
Game 9 is the final game, the distinguisher now interacts with the system $(\mathcal{H}, \mathcal{S}^{\mathcal{H}})$.

Summing up Summing up we find:

$$\begin{aligned} |\Pr[W_0] - \Pr[W_9]| &= \textstyle\sum_{i=1}^{9} |\Pr[W_i] - \Pr[W_{i-1}]| \\ &= 2N(N-1)2^{-l_a-l_m} + N2^{-l_a-l_m} \\ &= N(2N-1)2^{-l_a-l_m}. \end{aligned}$$

Complexity of the Simulator $\mathcal{S}$

The proof of indifferentiability does not take the complexity of the simulator into account. Via a side channel attack, an attacker may be able to differentiate between the original system and the simulating system, as the simulator may take longer to respond than the permutation. The simulator in the final game is given by:

```
Initialisation of S in the Final Game
  No input, no output
      1.  set G = ∅
Simulation of P
  Input:  y = (A, B, C, m)
  Output: (A', B')
      1.  add node y to G_D
      2.  if there exists y → x ∈ G_D
          (a) return (A', B') where x = (A', B', C, m)
      3.  if y has a path in graph G_D
          (a) compute M = unpad μ
          (b) call H to get h = H(M)
          (c) set B' = h
      4.  else
          (a) randomly select b' ← {0,1}^{l_m}
      5.  randomly select A' ← {0,1}^{l_a}
      6.  add node x = (A', B', C, m) and edge y → x to G_D
      7.  if ∃x̃ ∈ G_D such that ABORT_1 then abort
      8.  if ∃ỹ ∈ G_D such that ABORT_2 then abort
      9.  return (A', B').
```

The complexity depends mainly on the size of the graph. Let N be the number of previous queries. In step 2, in the worst case scenario we have to compare the new value with all N old values. The cost of step 3, the length of the path, can be neglected as this is most of the time much smaller than N. The complexity of step 7 and step 8 can be combined, if the simulator checks for every message block m and for every existing node $x' \in \mathcal{G}$ if $\text{Insert}[m](x) = \text{Insert}[m](x')$, it can decide whether to abort. The worst case complexity of step 7 and 8 is $n \cdot |M|$ where the latter represents the size of the message space.
The other parts are negligible compared to the above, so the overall worst case complexity of the simulator is of order

$$n + n \cdot |M| = \mathcal{O}(n \cdot |M|).$$

If the message space is large, already for the second query the time needed to respond for the simulator may differ significantly from the time the permutation needs.

3.4 Indifferentiability with a biased permutation

After the non–randomness of the permutation of Shabal was exposed, the authors of Shabal responded with a proof of indifferentiability of Shabal with a biased permutation [30]. The proof is based on the original proof and uses the same games, but the simulator for $\mathcal{S}$ now uses the relation that is known for the permutation. There is one more difference, the inverse of the permutation is also considered, this was a flaw in the first proof which is now corrected.

Forward Bias (τ) Let $\mathcal{R}$ be an input–output relation for the permutation. Given A, B, B', C and m, let $\text{PERM}_{\mathcal{R},m,A,B,C,B'}$ be the set of permutations P for which $\mathcal{R}$ holds and $P_{m,C}(A, B) \to (A', B')$ for some A'. Assume that there exists an algorithm which, given A, B, C, B' and m selects randomly a permutation P from $\text{PERM}_{\mathcal{R},m,A,B,C,B'}$. Then the forward bias τ is defined to be the

smallest real such that given A, A', B, C, m we have

$$\Pr\left[B' \in \{0,1\}^{512} : P_{m,C}(A,B) = (A', B')\right] \leq 2^{-384+\tau}.$$

For example, if the relation $\mathcal{R}$ is such that all tuples (A, B, C, m) are mapped to (A^*, B') for some fixed A^* and B' completely random. Then we choose A' to be A^* to find a bias of 384:

$$\Pr\left[B' \in \{0,1\}^{5}12 : P_{m,C}(A,B) = (A^*, B')\right] = 1 \leq 2^{-384+384}.$$

Backward Bias (λ) Now let $\text{PERM}_{\mathcal{R}}$ the set of all permutations that satisfy the relation $\mathcal{R}$. Assume that there exists an algorithm which, given A', B', C, m, selects randomly a permutation P from $\text{PERM}_{\mathcal{R}}$ and returns A, B such that $P_{m,C}(A,B) = (A', B')$. Then the backward bias λ is defined to be the smallest real such that

$$\Pr\left[P_{m,C}(A,B) = (A', B')\right] \leq 2^{-384-512+\lambda}.$$

If we take the same relation as above, then if we select B', C and m randomly, and let (A^*, B') be the image, we would find a backward bias $\lambda = 384$:

$$\Pr\left[P_{m,C}(A,B) = (A^*, B')\right] = 2^{-512} \leq 2^{-384-512+\lambda}.$$

In this example we know that A^* will be the image, so we only need B' to be the other part, which would happen with probability 2^{-512} since the B part of the permutations is random.

Indifferentiability Proof with a biased inner primitive (I) The submitters of Shabal use the original proof, taken away the final rounds and the counter [30]. So instead of Equation 3.1 we now use

$$\text{Insert}[m_i](A, B, C, m) = (m_i, A, C - m + m_i, B). \tag{3.2}$$

We assume that the permutation satisfies a relation $\mathcal{R}$, thus has forward bias, τ, and there exists an implementation J which generates a' given A, B, B', C, m such that for some permutation P for which $\mathcal{R}$ holds, $(A', B') = P_{m,C}(A, B, C, m)$.
Before the simulator selected a value for A' randomly, it now uses the implementation J to select A'. So the proof is rewritten by slightly changing the simulator and therefore the abort probabilities have to be estimated again.
The first change is for the simulator in Game 3. Recall that the simulator used to select A' randomly if the requested value is not yet in its graph, and test for a collision afterwards. The simulator is given by:

```
Initialisation of S in the Final Game
  No input, no output
      1.  set G = ∅
Simulation of P
  Input:  y = (A, B, C, m)
  Output: (A', B')
      1.  add node y to G_D
      2.  if there exists y → x ∈ G_D
          (a) return (A', B') where x = (A', B', C, m)
      3.  if y has a path in graph G_D
          (a) compute M = unpad μ
          (b) call H to get h = H(M)
          (c) set B' = h
      4.  else
          (a) randomly select b' ← {0,1}^{l_m}
      5.  get A' ← {0,1}^{l_a} from the implementation J
      6.  add node x = (A', B', C, m) and edge y → x to G_D
      7.  if ∃x̃ ∈ G_D such that ABORT_1 then abort
      8.  if ∃ỹ ∈ G_D such that ABORT_2 then abort
      9.  return (A', B')
```

We want to estimate $\Pr[\text{ABORT}_1]$ for the new simulator. As we no longer take the final rounds into account, the simulator aborts only when event **Coll0** evaluates true.
When the q^{th} query is requested to $\mathcal{S}$, there are at most $q-1$ states $\tilde{x}$ in graph $\mathcal{G}$. Using that the permutation is biased, we can bound $\Pr[\textbf{Coll0}\}$ at the q^{th} query as follows (recall that we are no longer taking the counter into account either):

$$\begin{aligned}
\Pr[\textbf{Coll0}] &\leq (q-1)\Pr[\exists m' : \text{Insert}[m'](A,B,C,m) = \text{Insert}[m'](\tilde{A},\tilde{B},\tilde{C},\tilde{m})] \\
&= (q-1)\Pr[A=\tilde{A} \wedge B=\tilde{B} \wedge C-m=\tilde{C}-\tilde{m}] \\
&\leq (q-1)\Pr[A=\tilde{A}|B=\tilde{B}]\Pr[B=\tilde{B}] \\
&\leq (q-1)2^{-l_a+\tau}2^{-l_m}.
\end{aligned}$$

Also $\Pr[\text{ABORT}_2]$ changes. As we no longer take the final rounds into account, the simulator aborts only when event **Dep0** evaluates true.
When the q^{th} query is requested to $\mathcal{S}$, there are at most $q-1$ states $\tilde{x}$ in graph $\mathcal{G}$. Using that the permutation is biased, we can bound $\Pr[\textbf{Dep0}]$ at the q^{th} query as follows:

$$\begin{aligned}
\Pr[\textbf{Dep0}] &\leq (q-1)\Pr[\exists m' : \text{Insert}[m'](x) = \tilde{y}] \\
&= (q-1)\Pr[m'=\tilde{m} \wedge A=\tilde{A} \wedge B=\tilde{C} \wedge C-m+m'=\tilde{B}] \\
&\leq (q-1)\Pr[A=\tilde{A}|B=\tilde{C}]\Pr[B=\tilde{C}] \\
&\leq (q-1)2^{-l_a+\tau}2^{-l_m}.
\end{aligned}$$

The authors use the value of ABORT_3 in their final estimation, we assume that they mean that this is not changed compared to the previous proof. Recall that ABORT_3 happens if there exists a path in $\mathcal{G}_\mathcal{C}$ and not in $\mathcal{G}_\mathcal{D}$. The final rounds nor the counter are considered as a special case and removing them will not give the attacker any advantage, so we conclude again

$$\Pr[\text{ABORT}_3] \leq N2^{-l_a-l_m}.$$

In this paper the simulator for the inverse of the permutation, $\mathcal{P}^{-1}$, is introduced. Here we assume that there is a relation which provides a backward bias λ and there exists an implemen-

tation K such that given (A', B', C', m), selects a permutation P for which the relation holds, and returns (A, B) for which $P_{m,C}(A, B) = (A', B')$. The simulator is given by:

Initialisation of $\mathcal{S}$ in the Final Game
  No input, no output
    1. set $\mathcal{G} = \emptyset$

Simulation of $\mathcal{P}$
  Input: $y = (A, B, C, m)$
  Output: (A', B')
    1. add node y to $\mathcal{G}_{\mathcal{D}}$
    2. if there exists $y \to x \in \mathcal{G}_{\mathcal{D}}$
       (a) return (A', B') where $x = (A', B', C, m)$
    3. if y has a path in graph $\mathcal{G}_{\mathcal{D}}$
       (a) compute $M = \text{unpad}\mu$
       (b) call $\mathcal{H}$ to get $h = \mathcal{H}(M)$
       (c) set $B' = h$
    4. else
       (a) randomly select $b' \leftarrow \{0,1\}^{l_m}$
    5. get $A' \leftarrow \{0,1\}^{l_a}$ from the implementation J
    6. add node $x = (A', B', C, m)$ and edge $y \to x$ to $\mathcal{G}_{\mathcal{D}}$
    7. if $\exists \tilde{x} \in \mathcal{G}_{\mathcal{D}}$ such that ABORT$_1$ then abort
    8. if $\exists \tilde{y} \in \mathcal{G}_{\mathcal{D}}$ such that ABORT$_2$ then abort
    9. return (A', B')

Simulation of $\mathcal{P}^{-1}$
  Input: $x = (A', B', C, m)$
  Output: (A, B)
    1. add node x to $\mathcal{G}_{\mathcal{D}}$
    2. if there exists $y \to x \in \mathcal{G}_{\mathcal{D}}$
       (a) return (A, B) where $y = (A, B, C, m)$
    3. else, get (A, B) from the implementation K
    4. add node $x = (A, B, C, m)$ and edge $y \to x$ to $\mathcal{G}_{\mathcal{D}}$
    5. if $\exists \tilde{x} \neq x \in \mathcal{G}_{\mathcal{D}}$ such that $y \to \tilde{x} \in \mathcal{G}_{\mathcal{D}}$ then abort (ABORT$_4$)
    6. if $\exists \tilde{x} \in \mathcal{G}_{\mathcal{D}}$ such that $\text{Insert}[m](\tilde{x}) = y$ for some $m \in \{0,1\}^{l_m}$ then abort (ABORT$_5$)
    7. return (A, B)

There are two abort events defined in the simulator for $\mathcal{P}^{-1}$. The first event, ABORT$_5$, prevents the simulator from choosing an output y that is already the preimage of another input $\tilde{x}$, i.e. an output that has two preimages, and this would imply that $\mathcal{P}$ maps y to two different states. The second event, ABORT$_5$, prevents the simulator for $\mathcal{P}^{-1}$ from outputting an intermediate value y for a path from $\tilde{x}$ to x, i.e. from outputting y for which there exists a message block m such that $x = \mathcal{P}(\text{Insert}[m](\tilde{x}))$.
We estimate the probabilities for the two events similarly to the other abort events.
For event ABORT$_4$ to be true for x and y, we need that there exists an $\tilde{x}$ such that both of them will be mapped to y by the permutation simulator. As the simulator for $\mathcal{P}$ will never map to elements to the same point, the only possibility is $x = \tilde{x}$, so the implementation K selects a y that is already in the graph. Recall that q is the number of this query. We now have

$$\begin{aligned}
\Pr[\text{ABORT}_4(q)] &\leq (q-1)\Pr[y = \tilde{y}] \\
&= (q-1)\Pr[A = \tilde{A} \wedge B = \tilde{B} \wedge C = \tilde{C} \wedge m = \tilde{m}] \\
&\leq (q-1)\Pr[(A, B) = (\tilde{A}, \tilde{B})] \\
&\leq (q-1)2^{-l_a+\tau}2^{-l_m}.
\end{aligned}$$

Event ABORT_5 to be true for x and y, we need that there exists there exists a message block m such that $x = \mathcal{P}(\text{Insert}[m](\tilde{x}))$. We find

$$\begin{aligned}\Pr[\text{ABORT}_5(q)] &\leq (q-1)\Pr[\exists m^* : \text{Insert}[m^*](\tilde{x}) = y] \\ &= (q-1)\Pr[\exists m : \tilde{A} = A \wedge \tilde{C} - \tilde{m} + m^* \wedge \tilde{B} = C \wedge m^* = m] \\ &\leq (q-1)\Pr[(A,B) = (\tilde{A}, \tilde{C} - \tilde{m} + m^*)] \\ &\leq (q-1)2^{-l_a - l_m + \lambda}.\end{aligned}$$

Summing up The advantage of the attacker at the q^{th} query is the sum of the above:

$$\begin{aligned}Adv_q(\mathcal{D}, \mathcal{C}) &\leq 3(q-1)2^{-l_a-l_m+\tau} + (q-1)2^{-l_a-l_m+\lambda} + 2^{-l_a-l_m} \\ &= (q-1)2^{-l_a-l_m}(3 \cdot 2^{\tau} + 2^{\lambda}) + 2^{-l_a-l_m}.\end{aligned}$$

And for at most N queries this gives:

$$\begin{aligned}Adv(\mathcal{D}, \mathcal{C}) &\leq \sum_{q=1}^{N} \left\{ 3(q-1)2^{-l_a-l_m+\tau} + (q-1)2^{-l_a-l_m+\lambda} + 2^{-l_a-l_m} \right\} \\ &= \tfrac{3}{2}N(N-1)2^{-l_a-l_m+\tau} + \tfrac{1}{2}N(N-1)2^{-l_a-l_m+\lambda} + N2^{-l_a-l_m} \\ &= \tfrac{1}{2}N(N-1)2^{-l_a-l_m}(3 \cdot 2^{\tau} + 2^{\lambda}) + N2^{-l_a-l_m}.\end{aligned}$$

The submitters of Shabal stated that if one could find a relation with backward bias $\lambda > 384$ and thus larger than the forward bias, this distinguishing relation can be used to break Shabal [30]. The example given before does not satisfy this. A simple permutation that does satisfy $\lambda > 384$ is the identity function, the backward bias $\lambda = 896$ and the forward bias would be zero in this case. This is trivial since the set of permutations P which satisfy the relation $P_{m,C}(A,B) = (A,B)$ contains only the identity function, so when computing the backward bias this permutation is chosen with probability 1.

The forward bias is defined to be a bias for sampling A', so the forward bias is a real between 0 and the length of A. But as B is larger than A, it seems logical to also define a forward bias for B'. Let us call this the B–forward bias β. Now β is defined between 0 and the length of B, 512. The new proof of Shabal's security, based on the biased permutation, states that as long as the forward bias is smaller than 384 the mode of operation of Shabal remains ideal [30]. But let us now redo that proof with a B–forward bias (Note that we use the notations of the most recent publication [30], not the notations of earlier versions).

$$\begin{aligned}p_1(\tilde{x}, y) &= \Pr\left[A' = \tilde{A} \wedge B' = \tilde{B}\right] \\ &= \Pr\left[A' = \tilde{A}\right]\Pr\left[B' = \tilde{B} | A' = \tilde{A}\right] \\ &= 2^{-l_a-l_m+\beta}.\end{aligned}$$

The same way we can do this for p_2 and p_4, so in the end we will find

$$\text{Adv}(\mathcal{D}, \mathcal{R}) \leq \frac{3N(N-1)}{2}2^{-l_a-l_m+\beta} + \frac{N(N-1)}{2}2^{-l_a-l_m+\beta} + N2^{-l_a-l_m}.$$

Now if $\beta > \tau > 0$ then $N^2 2^{-l_a-l_m+\beta}$ dominates the adversary's advantage. If $\beta = l_m$ we find the limit of $\mathcal{O}(2^{l_a/2})$ adversarial observations, less than ideal.

The reason that a B–forward bias is not taken into account, is that a bias in B is leading to a much bigger problem, as B contains the hash output. For example a bias in B will be useful to

find preimages, as the values that not satisfy the bias will easily be excluded and therefore the search space becomes smaller.

Indifferentiability Proof with a biased inner primitive (II) A second paper describes biased inner primitives in indifferentiability proofs in a more general way [34]. The new proof for Shabal gives a tighter upper bound for the advantage of the attacker, due to the different handling of the forward bias. As previously explained, non uniform behavior in B will be of more concern than non uniform behavior in A.
The new methodology is defined when the inner primitive is a compression function and when it is a permutation, but we will only take the situation with a permutation into account.
Recall that with x we mean a state before message insertion and y is a state after message insertion.

First we will introduce some new notations and then we will combine this with the previously described forward bias to find a tighter bound on the indifferentiability of Shabal. First we will describe the simulator, the advantage of the attacker can then again be computed using the probabilities that the simulator aborts.

A biased permutation can be exposed by selecting an input for which some relation holds for the output. If such an input–output pair can be identified before making any query to the permutation, the input of the pair is called an *atypical input*. Other related pairs can only be identified after some queries. In this case, we have a history of queries $\mathcal{L}$ and a related input–output pair, and we call the input of the pair a *relative* of the set of inputs $\mathcal{L}_{in}$ of $\mathcal{L}$. The simulator of the permutation will abort when it is queried an atypical input or a relative of a previously requested input.
To formally define atypical inputs and relatives we use the statistical distance. For two statistical objects, in our case, two distributions D_1 and D_2, defined over a common set of events $\mathcal{E}$, the statistical distance is

$$||D_1 - D_2|| = \frac{1}{2} \sum_{e \in \mathcal{E}} \left| \Pr_{X \leftarrow D_1}[X = e] - \Pr_{X \leftarrow D_2}[X = e] \right|.$$

Given a history $\mathcal{L}$, as a set of input–output pairs, let $\textsc{perm}(\mathcal{L})$ be the set of all permutations such that for all $p \in \textsc{perm}(\mathcal{L})$ the following holds:

$$\forall (y, x) \in \mathcal{L} : p(y) = x.$$

Thus, $\textsc{perm}(\mathcal{L})$ is the set of all functions that satisfy the history $\mathcal{L}$. Now for a given input y', we call the set of admissible outputs $\mathcal{A}_{\mathcal{L}}(y')$. That is, given a history $\mathcal{L}$ and an input y', all possible outputs x' such that there exists a $p \in \textsc{perm}(\mathcal{L})$ such that $p(y') = x'$.

Definition 4 (ϵ–relatives of a set of inputs)**.** *Let $\mathcal{A}_{\mathcal{L}}(y)$ be the set of admissible outputs, $\epsilon \in [0, 1]$ a real and $\mathcal{L}$ a history of queries. We assume that there exists an algorithm which samples all admissible outputs, given history $\mathcal{L}$ and an input y, and let $D_{\mathcal{L},y}$ be its distribution.*
The ϵ–relatives of $\mathcal{L}_{in}$ are then given by

$$\textsc{rel}(\mathcal{L}_{in}, \epsilon) = \{y \notin \mathcal{L}_{in} : ||D_{\mathcal{L},y} - \textsc{unif}|| > \epsilon\}.$$

where unif *represents the uniform distribution.*

As explained above, the atypical inputs are inputs for which the output is constrained, even without any knowledge of other input–output pairs. This is formalized in the following definition.

Definition 5 (ϵ–Atypical inputs)**.** *Let $\epsilon \in [0,1]$ a given real, then the set of ϵ–atypical inputs is*

$$\textsc{at}(\epsilon) = \textsc{rel}(\emptyset, \epsilon).$$

The simulator is again making a graph of all previous queries, and will for every query check whether a state leads to an atypical input after inserting some message. If so, the simulator will abort. Again we assume that there exists an algorithm to test this.

A second problem arises when the outputs of two relatives of some other value, must be made consistent with the random oracle. As the outputs will be very correlated, by the definition of relatives, this is impossible. Thus in this situation the simulator will also abort. Formally, the relatives of an input are defined as follows.

Definition 6 (ϵ–relatives of an input)**.** *Let $\mathcal{A}_{\mathcal{L}}(y)$ be the set of admissible outputs, $\epsilon \in [0,1]$ a real and $\mathcal{L}$ a history of queries.*
Let y' be an input which is not yet queried, so $y' \notin \mathcal{L}_{in}$. The set of ϵ–relatives of y', $\mathcal{R}(y, \mathcal{L}_{in}, \epsilon)$, is the smallest set R such that $y' \in r$ and

$$\textsc{rel}(\mathcal{L}_{in} \cup R, \epsilon) \subset \textsc{at}(\epsilon).$$

Now we are ready to define the games, which are not entirely the same as in the original proof, so we shall briefly address all games.

Original Game In the original game the distinguisher interacts with $\mathcal{C}^{\mathcal{P}}$ and by definition we have

$$\Pr[W_0] = \Pr[\mathcal{D}^S | S = (\mathcal{C}^{\mathcal{P}}, \mathcal{P})].$$

Game 1 & 2 In Game 1, $\mathcal{P}$ is replaced by the simulator $\mathcal{S}$ which queries $\mathcal{P}$. There is no difference between this and the original game, so there is no advantage for the attacker. In Game 2 $\mathcal{S}$ no longer queries $\mathcal{P}$ but uses the implementations J and K to get the response. The definition of the implementations make sure that there are no inconsistencies, so there is no advantage for the attacker in this game compared to the previous game:

$$\Pr[W_2] = \Pr[W_1] = \Pr[W_0].$$

Note that we have equality here, since the simulator chooses a permutation for which the history is satisfied, this permutation will be bijective and thus will never output a value that is already assigned to another input.

Game 3-7 In these games we are preparing for the replacement of the permutation by a random oracle. The simulator has to abort in a case where the permutation would give an output inconsistent with the random oracle, i.e. if the distribution is too far from the uniform distribution.

When the simulator is queried an input y, it simultaneously computes the image of y and of all

its ϵ–relatives. We indicate four events for which the new simulator aborts, while evaluating a state $\tilde{y}$. These events represent the situations in which the permutation differs from a random oracle.

(i) If $\tilde{y}$ is mapped to a state that already exists in the graph, the simulator aborts. If both the image and the vertex already exist in the graph and have a path starting in the initial value, then there could be a collision.

(ii) The number of relatives that the simulator checks must be upper bounded by some R_{MAX} and the simulator will abort if it identifies more than R_{MAX} relatives. Now we modify the simulator such that it is consistent with a random oracle. Therefore we assume that if a state has more relatives, the attacker is not able to identify them offline, i.e. it needs to query the simulator to identify more relatives.

(iii) The simulator also aborts if there exist message blocks m, m' and a state x such that for the new output $\tilde{x}$ we have

$$\text{Insert}[m'](\tilde{x}) \in \text{REL}(\text{Insert}[m](x), \mathcal{L}_{in}, \epsilon)$$

and both x and $\tilde{x}$ have a path from the initial value in the graph. In this case, we would have two relatives of a certain state in the graph, which both have a path.

(iv) If $\tilde{y}$ is mapped to a state that can be transformed to an atypical input by inserting a particular message block, the simulator also aborts.

The simulator now uses the implementations $\mathcal{J}$ and $\mathcal{K}$ where it used to select A and B randomly. The abort rules that we use are (i), (ii), (iii) and (iv).

Initialisation of $\mathcal{S}$
  No input, no output
    1. choose $\epsilon \in [0,1]$
    2. set $\mathcal{G} = (\{x_0\}, \emptyset)$

Simulation of $\mathcal{P}$
  Input: $y = (A, B, C, m)$
  Output: (A', B')
    1. if there exists $y \to x \in \mathcal{G}$
       (a) return (A', B') where $x = (A', B', C, m)$
    2. if $|\mathcal{R}(y, \epsilon)| > R_{\texttt{MAX}}$ then abort (ii).
    3. if $\exists y^* \in |\mathcal{R}(y, \epsilon)$ that has a pathμ in graph $\mathcal{G}$
       (a) compute $M = \text{unpad}(\mu)$
       (b) get $B' \leftarrow \{0,1\}^{l_a}$ from the implementation K
       (c) get $A' \leftarrow \{0,1\}^{l_a}$ from the implementation J
       (d) set $x = (A', B', c^*, m^*)$
       (e) if $x \in \mathcal{G}$ then abort (i).
       (f) if $\exists x' \in \mathcal{G}$ which has a path and such that x is a relative of x' then abort (iii).
       (i) if $\exists m$ such that $\text{Insert}[m](x) \in \text{AT}(\epsilon)$ then abort (iv).
       (j) add nodes $x = (A', B', C, m), y$ and edge $y \to x$ to $\mathcal{G}$
    5. for all $\tilde{y} \in \mathcal{R}(y, \epsilon) \backslash \{x\}$
       (a) get $\tilde{B}' \leftarrow \{0,1\}^{l_m}$ from the implementation K
       (b) get $\tilde{A}' \leftarrow \{0,1\}^{l_a}$ from the implementation J
       (c) set $\tilde{x} = (\tilde{A}', \tilde{B}', \tilde{C}, \tilde{m})$
       (d) add node $\tilde{x} = (A', B', C, m), \tilde{y}$ and edge $y \to x$ to $\mathcal{G}$
    6. return (A', B')

The event (i) is related to the events previously defined ABORT_1 and ABORT_2, the other events are based on the concept of relatives. We use the difference lemma to see that the advantage of the distinguisher between game 7 and game 1 is smaller than or equal to the probability that the simulator aborts in game 7:

$$|\Pr[W_7] - \Pr[W_1]| \leq \Pr[\text{ABORT}_{(\text{I})}] + \Pr[\text{ABORT}_{(\text{II})}] + \Pr[\text{ABORT}_{(\text{III})}] + \Pr[\text{ABORT}_{(\text{IV})}].$$

The bounds for these probabilities are given in the third paper of the authors of Shabal [34]. *We have not been able to reconstruct these bounds and the full version of the paper has not been issued yet. Therefore we will try to explain the part we do understand.*

$$\Pr[\text{ABORT}_{(\text{I})}] \quad \leq \quad 2^{-(l_a - \tau(\epsilon))} \left(2^{-l_m} + 4\epsilon\right) R_{max} N^2$$

The term $2^{-l_a - l_m + \tau(\epsilon)}$ represents the probability that the A and the B part are equal to a state that is already in the graph. There are at most NR_{max} states in the graph, for each query (total is N) we add at most R_{max} neighbors of the input and their outputs, plus the input and the output itself. Assuming that the last two are included in R_{max} we can see that for the input we check NR_{max} states for equality and for the output we check N. This results in $N^2 R_{max}$. Understandably we do not compare the outputs with the ϵ–relatives, since ϵ–relatives are only defined (and useful) for the inputs. But there are already NR_{max} outputs in the graph and these should be compared with the new output too in our opinion.
The term $\epsilon \cdot 2^{-l_a + \tau(\epsilon)}$ represents the probability that a state with the A part equal to a state in the graph, is an ϵ–relative of this state. Since an ϵ–relative of a state has a B–part which is known with some probability through the state.

Now let us look at $\text{ABORT}_{(II)}$:

$$\Pr[\text{ABORT}_{(II)}] \quad \leq \quad \max_{m,y} \left(\Pr[|\mathcal{R}(m, A, B, C, \epsilon)| > R_{max}]\right) N.$$

Note that y represents the output of the permutation.
The probability of $\text{ABORT}_{(II)}$ is bounded by the maximum probability that there is a set of relatives which size exceeds R_{max}.
In our opinion this probability should be maximized over all inputs and all message blocks, so we would get:

$$\Pr[\text{ABORT}_{(II)}] \quad \leq \quad \max_{m,x} \left(\Pr[|\mathcal{R}(m, A, B, C, \epsilon)| > R_{max}]\right) N.$$

To bound the probability of $\text{ABORT}_{(III)}$ we first define $\mathcal{A}$.
$\mathcal{A}$ is the size of the maximal set of all the possible preimages of a state which is an ϵ–relative of some state in the graph (of the simulator). Formally,

$$\mathcal{A} = \max_{y \in \mathcal{X}, c \in \{0,1\}^{l_m}} |\{(a', b') : \text{Insert}^{-1}[\tilde{m}](\tilde{x}) = (a', b', c), (\tilde{(m)}, \tilde{(x)}) \in \mathcal{R}(m, \text{Insert}[m](y))\}|,$$

now the probability is given by

$$\Pr[\text{ABORT}_{(III)}] \quad \leq \quad \mathcal{A} 2^{-(l_a - \tau(\epsilon))} (2^{-l_m} + 4\epsilon) \tfrac{N(N+1)}{2}.$$

The simulator aborts when it has to add a state for which there exists a message block under which it is mapped to a set of relatives for which we can find another state which is mapped to this set under a certain message block.

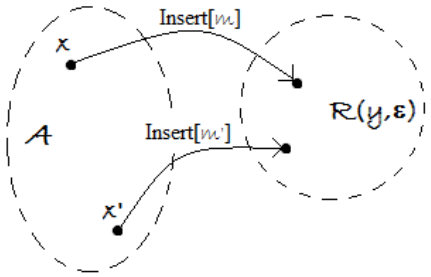

Figure 3.8: Graphical presentation of $\text{ABORT}_{(III)}$

We have to check at most $\mathcal{A}$ preimages, since this is the size of the biggest set of preimages for a certain set of relatives. Secondly we could either have equality or an ϵ–relative, we assume these probabilities give the factor $2^{-(l_a - \tau(\epsilon))}(2^{-l_m} + 4\epsilon)$.
The last factor is $\frac{N(N+1)}{2} = \sum_{i=1}^{N} i$, so for the second query we only check one set of relatives, for the third query we check three and for the fourth we check six.
The reasoning behind this is not clear to us and we would expect to see something related to the number of message blocks that the simulator needs to check.

To bound the probability of $\text{ABORT}_{(IV)}$ we first define $\mathcal{B}$.
$\mathcal{B}$ is the size of the maximal set of preimages, such that the image is an atypical value. Formally,

$$\mathcal{B} = \max_{c} |\{(a, b) : \text{Insert}^{-1}[\tilde{m}](\tilde{A}, \tilde{B}, \tilde{C}) = (a, b, c), (\tilde{m}, \tilde{A}, \tilde{B}, \tilde{C}) \in \text{AT}(\epsilon)\}.$$

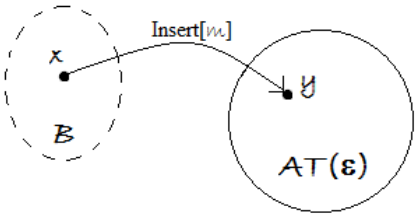

Figure 3.9: Graphical presentation of $\text{ABORT}_{(\text{IV})}$

And the probability is then given by

$$\Pr[\text{ABORT}_{(\text{IV})}] \quad \leq \quad \mathcal{B}2^{-(l_a-\tau(\epsilon))}(2^{-l_m}+4\epsilon)N.$$

We have to check at most $\mathcal{B}$ preimages, since this is the size of the biggest set of preimages for which the image is a atypical input. Secondly we could either have equality or an ϵ–relative, we again assume these probabilities give the factor $2^{-(l_a-\tau(\epsilon))}(2^{-l_m}+4\epsilon)$.
The last factor is N, that is the number of previous queries, why this factor is added is not clear, as the set of atypical inputs does not depend on previous queries.

Game 8 Game 8 is similar to Game 5 in the original proof, here we add the random oracle $\mathcal{H}$. The simulator will query $\mathcal{H}$ for the B part of the output if the input has a relative which has a path in the graph. Given that none of the previous abort events occur, the distance between the distribution of the sampling algorithm and the distribution of $\mathcal{H}$ will be smaller than ϵ. After N queries the difference between the uniform distribution of $\mathcal{H}$ and the sampling algorithm is at most $N\epsilon$:

$$|\Pr[W_8] - \Pr[W_7]| \leq N\epsilon.$$

Game 9 & 10 In the final game, the simulator does not have access to the hash requests issued by the distinguisher, therefore it does not know whether the input is already queried to $\mathcal{H}$. We define $\text{ABORT}_{(\text{V})}$ such that the simulator would abort if $\mathcal{H}$ has already assigned a value to the input.
Bresson et. al. describe this as "when $\mathcal{S}$ notices that it has to use the part of its memory that was build to answer hash requests".
And they state:

$$|\Pr[W_{10}] - \Pr[W_8]| \leq \text{ABORT}_{(\text{V})}.$$

And:

$$\text{ABORT}_{(\text{V})} = N^2 2^{-(l_a-\tau(\epsilon))}\left(2^{-l_h}(1-\epsilon)^{-1} + 2^{-l_h}p_C(1+\exp) + p_C\epsilon\right) + N2^{-(l_a-\tau(\epsilon)-1)}l_h p_C$$

where n_f represents the number of final rounds, l_h is the length of the hash output and

$$p_C = \begin{cases} \max_{m,c,c'} \Pr_{(a,b)\leftarrow\{0,1\}^{l_a+l+m}}[\text{Insert}_C[m](a,b,c) = c'] & \text{if } n_f \geq 1 \\ 1 & \text{if } n_f = 0. \end{cases}$$

Game 11 & 12 Now we actually remove the access of the simulator to the hash requests to $\mathcal{H}$, but since we already defined $\text{ABORT}_{(v)}$ there is no change:

$$\Pr[W_{12}] = \Pr[W_{10}].$$

Initialisation of $\mathcal{S}$
  No input, no output
  1. choose $\epsilon \in [0,1]$
  2. set $\mathcal{G} = (\{x_0\}, \emptyset)$

Simulation of $\mathcal{P}$
  Input: $y = (A, B, C, m)$
  Output: (A', B')
  1. if there exists $y \to x \in \mathcal{G}$
     (a) return (A', B') where $x = (A', B', C, m)$
  2. if $|\mathcal{R}(y, \epsilon)| > R_{\text{MAX}}$ then abort (ii).
  3. if $\exists y^* \in |\mathcal{R}(y, \epsilon)$ that has a pathμ in graph $\mathcal{G}$
     (a) compute $M = \text{unpad}(\mu)$
     (b) call $\mathcal{H}$ to get $h = \mathcal{H}(M)$
     (c) set $B' = h$
     (d) get $A' \leftarrow \{0,1\}^{l_a}$ from the implementation J
     (e) set $x = (A', B', c^*, m^*)$
     (f) if $x \in \mathcal{G}$ then abort (i).
     (h) if $\exists x' \in \mathcal{G}$ which has a path and such that x is a relative of x' then abort (iii).
     (i) if $\exists m$ such that $\text{Insert}[m](x) \in \text{AT}(\epsilon)$ then abort (iv).
     (j) add nodes $x = (A', B', C, m), y$ and edge $y \to x$ to $\mathcal{G}$
  5. for all $\tilde{y} \in \mathcal{R}(y, \epsilon) \backslash \{x\}$
     (a) get $\tilde{B}' \leftarrow \{0,1\}^{l_m}$ from the implementation K
     (b) get $\tilde{A}' \leftarrow \{0,1\}^{l_a}$ from the implementation J
     (c) set $\tilde{x} = (\tilde{A}', \tilde{B}', \tilde{C}, \tilde{m})$
     (d) add node $\tilde{x} = (A', B', C, m), \tilde{y}$ and edge $y \to x$ to $\mathcal{G}$
  6. return (A', B')

Computing the security bound for Shabal We take $\epsilon = 0$, since the sample algorithm, based on the distinguishers defined for Shabal, is set to either a fixed value or the uniform distribution. The forward bias is represented by $\tau(\epsilon)$ and we find $\tau(0) = 96$ due to the non–dependencies for the inverse permutation given by Naya–Plasencia (only published in French) [35]. We will shortly address Naya–Placensia's method.
When computing the inverse permutation of a state there are three blocks of B (these are 11,14,15) which depend on at most four message blocks (these are 0,4,8,12). As a result one can find up to three input words before computing the inverse. The number of possible preimages for a given state will be only 2^{96} in some cases. Therefore, the forward bias is 96.
We define the maximum number of relatives to be 2^{18}.
We use that $\mathcal{A}$ is the maximum number of states that are colliding on the C part that are the preimage (under insert) of a certain group of relatives to compute $\mathcal{A} \leq 2^{136.25}$.
We can compute the precise value of $\mathcal{B}$, this is the number of atypical inputs which lead to values with an equal C part, which is 2^{128}.
The authors use that the Insert function of Shabal can be written as

$$\text{Insert}_C[m](A, B, C) = (A_2 - m_1, B - m_2)$$

where A_2 denotes the last l bits of A, $l = |m|$ minus the number of output bits. In the definition of Shabal we find that $l \leq 512$ and thus

$$\text{Insert}_C[m](A, B, C) = B - m.$$

Recall that K is the implementation that selects an output value for B' given some input. Now

$$\begin{aligned} p_C &= \max_{u \in \mathcal{X}, \mathcal{L}, m, C, C'} \Pr_{(a,b) \leftarrow \{0,1\}^{l_a + l + m}}[\text{Insert}_C[m](a, b, c) = c'] \\ &= \max_{m,c} \Pr_{b \leftarrow K(m,u,\mathcal{L})}[B - m = C] \\ &\leq 2^{-l_m - \tau} = 2^{-416} \end{aligned}$$

Using the above we find

$$\begin{aligned} \Pr[\text{ABORT}_{(\text{I})}] &\leq 2^{-800} R_{max} N^2 \leq 2^{-782} N^2 \\ \Pr[\text{ABORT}_{(\text{II})}] &\leq 2^{-654} N \\ \Pr[\text{ABORT}_{(\text{III})}] &\leq 2^{-801} \mathcal{A} N(N+1) \leq 2^{-664} N(N+1) \\ \Pr[\text{ABORT}_{(\text{IV})}] &\leq 2^{-800} \mathcal{B} N \leq 2^{-672} N \\ \Pr[\text{ABORT}_{(\text{V})}] &\leq 2^{-800}(1 + 2^{-416}(1 + \exp))N^2 + l_h 2^{-704} N \leq (2^{-800} + 2^{-1214})N^2 + l_h 2^{-704} N. \end{aligned}$$

For $\text{ABORT}_{(\text{V})}$ we have used that $1 + \exp \leq 2^2$ and $l_h \leq 512$.
The authors conclude that for $l_h \leq 512$ Shabal is indifferentiable from a random oracle up to 2^{332} requests, due to the term $2^{-664}N^2$.

3.4.1 Conclusion on the Indifferentiability proofs

The authors claim that the last proof [34] does not contain mistakes. The full version of this article is not yet issued. In the small version not all steps are defined, and for this reason we are not convinced by this proof.
The first proof was based on an assumption which was proven wrong. The second proof did not have tight bounds, as the others claimed, but it is also based, for example, on the assumption that there is no forward bias on the B part.
The third proof has not yet been fully explained.

3.5 Neutral Bits

Recall that the general idea of neutral bits is that the relation is invariant under alternating the bits.
Aumasson described a cube–distinguisher attack [27]. He noted that summing over 7 cube–variables and comparing for 6 super–poly variables, one finds 3 unchanged bits in the output of $B[5]$ after two internal rounds of one permutation round. The attack is described as follows:

- invert the finalization loop of $\mathcal{P}_{M,C}$
- guess $M[5] \ldots M[13]$
- invert the last 11 loops of the third round, observe $B[5]$

This would give the attack a complexity of 2^{301}, there are $2^{32 \cdot 9} = 2^{288}$ possible message words, and 2^{13} for summing over all cube and all super–poly variables.
But to observe $B[5]$ after the second round of the permutation, one does not need to guess

any message word nor invert the complete 11 loops of the third round. Let $B^3[5]$ be the final word and $B^2[5]$ the word after the second round of the permutation, then their relation can be described as follows:

$$B^3[5] = B^2[5] \lll_1 \oplus\overline{A^4[1]}$$

where $A^4[1]$ is the first word of A after the third round, which can be found after inverting the finalization loop.
So the complexity of this attack is 2^{13}, summing over 7 cube–variables (2^7) times comparing over 6 super–poly variables (2^6).

Important Note

Unfortunately our tests have not led to the same results. Aumasson noted that it should be some bits of the fourth word $B^2[4]$, and not the fifth. But this result could not be reproduced by us.

How to translate the attack of Aumasson into a bias

The attack of Aumasson can be described (simplified) as follows:

$$\Pr\left[\left(\sum_{cube} B^2[4,k] \middle| M[i,j] = 0\right) = \left(\sum_{cube} B^2[4,k] \middle| M[i,j] = 1\right)\right] = 1.$$

We could transform this into a forward bias relation, suppose we have n bits of B which we observe. First pick a super–poly variable, sum over the cube–variables with the super–poly variable active, subtract the sum over the cube–variables minus one setting with the super–poly variable not active. Now one knows the n specified bits of B'. So given A, B, C, m and B':

$$\Pr\left[A' \in \{0,1\}^{384} : \mathcal{P}_{M,C}(A,B) = (A',B')\right] = 2^{-512+n}.$$

But the proof of indifferentiability for Shabal [30] does not contain a bias on B, nor a bias over a sum.
Also, this seems not to be the optimal way to use the observation of the neutral bits. Assume that we have a set of observation bits for which we have a set of neutral super–poly variables over a cube, after one full permutation. This would translate into a summation bias n for each super–poly variable as follows:

$$\Pr\left[\sum_{cube} \mathcal{P}_{M^0,C}(A,B) = \sum_{cube} \mathcal{P}_{M^1,C}(A,B)\right] \leq 2^{-896+n}$$

where A, B, C are chosen randomly (or set to the IV's of Shabal), M is chosen randomly except for the cube variables and one of the super–poly variables, for M^1 the super–poly variable is active and for M^0 inactive. Now n is the smallest real number such that the equation holds.
So now we have a B–forward bias of 3, which does not threaten any of the security claims made by the authors of Shabal.

3.6 Does Shabal differ from a random function

Gligoroski stated that the narrow–pipe candidates differ significantly from ideal random functions [36]. He used the gap in the padding rule of BLAKE to show that if only messages of a specific length are hashed, there exists a set in the image space for which all elements do not have a preimage of the specified size. As the padding for Shabal is only "a one followed by zeros", this might work for Shabal too.
A message is padded with a one and a proper amount of zeros. So any message of which the length is a multiple of the blocksize is padded with the same block.
Let us assume that we only want to hash messages of length 1536 (three blocks), so the padding consists of one block starting with a one followed by 511 zeros.
Let this message be $M_1||M_2||M_3||M_4$.

Compression without truncation

If we only look at the compression function (so the final rounds and the truncation are ignored) then the iteration would look like

$$\mathcal{R}\left(\mathcal{R}\left(\mathcal{R}\left(\mathcal{R}\left(A_{IV}, B_{IV}, C_{IV}, M_1\right), M_2\right), M_3\right), M_4\right)$$

Since M_4 is a constant for all 1536–bit messages, we can see the last step as a function from $\{0,1\}^{1408} \rightarrow \{0,1\}^{1408}$.
If we assume $\mathcal{R}$ is ideal we can use lemma 1 and proposition 1 of Gligoroski [36] to state that:

$$\Pr\left[\mathcal{R}^{-1}(y) = \emptyset\right] \approx \exp(-1)$$

But for an ideal random function $\mathcal{S}\{0,1\}^{1536} \rightarrow \{0,1\}^{1408}$ we have that

$$\Pr\left[\mathcal{S}^{-1}(y) = \emptyset\right] \approx \exp(-2^{128})$$

This implies that the compression function differs from a random function, but there is no obvious way to use this. Finding such a set is hard if not impossible. But if we have such a set, it is possible to build a distinguisher based on this set, which can distinguish the compression function from an ideal function with one query.
As the final rounds are identical to the last round except for the input, the above still holds.

Truncation

If we take the truncation into account, the full Shabal is a function $\{0,1\}^{1408} \rightarrow \{0,1\}^{512}$, so by only restricting the last message block, we will find

$$\Pr\left[\mathcal{R}^{-1}(y) = \emptyset\right] \approx \exp(-2^{896}).$$

And for an ideal random function $\mathcal{S}\{0,1\}^{1536} \rightarrow \{0,1\}^{512}$ we have that

$$\Pr\left[\mathcal{S}^{-1}(y) = \emptyset\right] \approx \exp(-2^{1024}).$$

So we would expect both functions to be surjective and we may conclude that the idea of Gligoroski indeed does not work for wide–pipe functions.
The method of Gligoroski has not yet lead to an attack, it distinguishes the function from an ideal random function, but not in feasible time. This idea depends highly on the MD structure, of which only the last step is considered. There does not seem to be a way to use this to attack the function and therefore it does not threaten the security of narrow–pipe functions. So in our opinion this does not show that wide–pipe functions are more secure than narrow–pipe functions.

3.7 Message extension Attack

The message rounds and the final rounds are so much alike, so a message extension attack on a reduced version might be applicable, in addition the padding consists only of a 1–bit followed by an appropriate amount of 0–bits. Let us look at a one–block message M such that the padded message $T = M||10\ldots$ is only one block.
Let H be Shabal deleted the counter. For H the message rounds and the final rounds are the same as long as the message block input is the same. So applying this reduced version of Shabal to a one–block message M and the concatenation of the padded message T with M, gives the same internal state for $H(M)$ at the end and for $H(T||M)$ before the last final round. But the output is only a small part of the internal state, l words of C, so applying another round to the output of $H(M)$ is not a trivial procedure (as the outputs A and B are unknown). If we consider Shabal with a 512–bit digest, so the output is C, this might lead to a probabilistic distinguisher. Recall that this applies to the reduced version of Shabal, the block counter will toughen this attack.

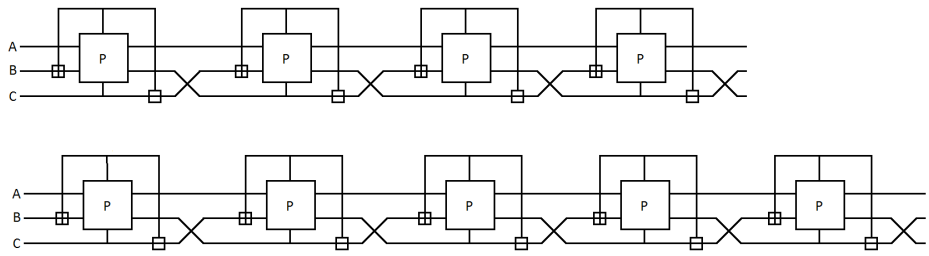

Figure 3.10: The Message extension attack on Shabal

The message extension attack will be of no use as long as A and B are completely unknown, since the probability to guess A and B correctly is $2^{-(l_a+l_m)}$.
After each round B and C switch places, so maybe a message extension attack with two extra rounds is more useful than with one extra round.
But it is possible to also retrieve B, since B is the C part of the next round, and the probability to guess A correctly is 2^{-l_a}.

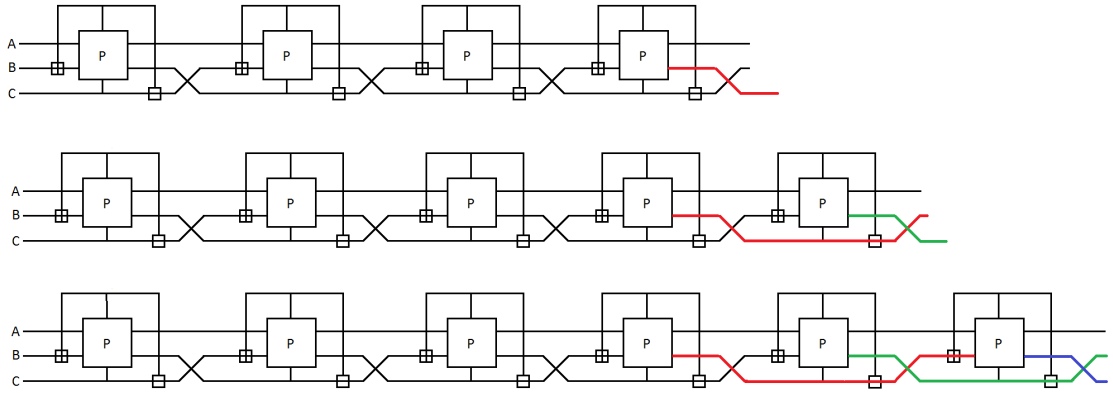

Figure 3.11: The Message extension attack on Shabal

With probability 2^{-l_a} the guess for A is correct and then Shabal can be distinguished from a random oracle. A message extension distinguisher of complexity 2^{-l_a} is now given.
Query M,$T||M$, $T||T||M$ to H to obtain b_1, b_2 and b_3. Then select a value for A at random and query (T, A, b_1, b_2) to the inner primitive to obtain b_4. Now compare b_3 and b_4, if these are equal, output that the system is Shabal, otherwise randomly select an output.
To improve this attack, we need to find some relations for A, or for the output after one final round given B, C and M.

3.8 The initial values

Many attacks are applicable only when the initial values can be manipulated. For example fixed points of the permutation [28] are found for specific values of A, B, C and M. One could use a message block of M to manipulate the value of A or C in the next round, or B for the second next round. But for fixed points or related key attack we need A, B and C to be adjusted at the beginning of one permutation round.
For the key collisions of Knudsen, Matusiewica and Thomsen [28], we need two tuples (A, B, C, m) and (A, B, C', m'). But if we use two message blocks to end in states where the C–part are C and C' then we will most probably also have differences in the A–part and B–part of the state.

3.8.1 Changing one of the internal variables

Starting from the initial values, with one message block (so one compression round), one of the variables can be changed for the next round. Changing the next B will take two rounds, so we start by looking at C and A. So assume we want to set C for the next round, where we will apply the permutation to the message block m.

Changing the next C

Changing the next C can be done by choosing a message block M^* such that $B - M^* + M$ is C^* the value you want for C.
Let $A_0, \ldots, A_{11}, B_0 \ldots, B_{15}$ and $C_0, \ldots, C_{15}$ be the initial words. We will write B_i^0 for the values of B after the rotation (step 1 of the permutation), $*_i^j$ for the values of A and B after the $j-$th inner round (of step 2 of the permutation) and A_i^F for the values of A after the final round. So the output of the permutation is (A_i^F, B_i^3).
Move forward through the permutation to find equations for all intermediate values expressed in words of m.

$$\begin{array}{rclrcl} A_0^1 & = & f_0(m_0^*) & B_0^1 & = & g_0(m_0^*) \\ A_1^1 & = & f_1(m_0^*, m_1^*) & B_1^1 & = & g_1(m_0^*, m_1^*) \\ & \vdots & & & \vdots & \\ A_{11}^4 & = & f_{47}(m_1^*, \ldots, m_{15}^*) & B_{16}^3 & = & g_{47}(m_1^*, \ldots, m_{15}^*). \end{array}$$

As we are only looking at B we can ignore the final transformation for A. Now reduce the equations to 16 equations for the B_i^3 in terms of m^*. Combine this with $B_i^3 = C_i^* - m_i^* + m_i$ to solve m^*. The complexity to find this system is one permutation of computational complexity and a lot of memory since the variable m will spread rapidly, i.e. each eqation will contain many variables. The system consists of 16 equations with 16 variables, if we express the variables in bits, we could solve the equations bitwise, starting with the least significant bit and building a solution tree.

3.9 T–functions

Definition 7 (T–function)**.** *A function $f : \{0,1\}^{m \times n} \to \{0,1\}^{l \times n}$ is called a T–function if the k^{th} column of the output $[f(x)]_{k-1}$ depends only on the first k columns of the input x.*

For a function on 32–bit words, this can be seen as, the function to find bit i depends only on the bits that are less significant than i.
T–functions can be used to build a solution graph for a set of equations.

If the permutation would contain only T–functions, the bits would not be properly mixed. In the permutation we find several rotations, which are not T–functions. But it is possible to define a solution graph for functions that do contain non–T–functions. A 1–bit rotation can be represented with a 1–bit shift and some replacement function for the last bit. The problem is that for rotation over 15 bits, one could use the shift over 15 bits and some replacement function, but the shift is 15–narrow, which makes the width of the solution tree very large.

It is not accurate to reduce the permutation by removing the rotation over 15 bits, since then one would assume that not only A_0 is rotational, but each new value of A is also rotational, which is hardly ever true[2].

[2]Since then A would be nine times 15–rotational, and thus 15–,30–,13–,28–,11–,26–,9–,24– and 7–rotational.

3.10 Linear Cryptanalysis

In linear cryptanalysis the non-linear operations are approximated by linear operations. Shabal has multiplication with constants (3,5) in step 2 and addition in step 3 of the permutation, and each round addition and substraction of the message block.
The permutation of Shabal has 48 multiplications by 3, 48 multiplications by 5, 48 AND's and 36 additions for each message block, so the probability that a message block input in the linearized version gives the same output as in the original permutation, is

$$\left(\frac{3}{4}\right)^{-(30\cdot 48+29\cdot 48+32\cdot 48+31\cdot 36)} = \left(\frac{3}{4}\right)^{-(5484)} \approx 2^{-2276}.$$

For an ideal random permutation on 896 bits this would be 2^{-896}, so the linearization does not help us in this case.
If we start with $A = C = 0$ then we can proceed through the first 12 multiplications by 5, the first 12 by 3 in step 2 and the additions in step 3 of the permutation with probability 1. So we find the probability of equality after one message round to be

$$\left(\frac{3}{4}\right)^{-(30\cdot 36+29\cdot 36+32\cdot 48)} = \left(\frac{3}{4}\right)^{-(3660)} \approx 2^{-1519}.$$

If we would also start with $B = 0$ we find probability 1 for the first 9 AND operations, in total we find

$$\left(\frac{3}{4}\right)^{-(30\cdot 36+29\cdot 36+32\cdot 39)} = \left(\frac{3}{4}\right)^{-(3372)} \approx 2^{-1400}.$$

A second possibility to increase the probability, is to work with words of a known maximum weight. If two words a, b are added modulo 2^n and the weight of the words is smaller than $k \leq l$ respectively then we find

$$\Pr[a + b = a \oplus b] \geq \left(\frac{3}{4}\right)^{l-1}.$$

Isobe and Shirai used this method (originally given by Lipmaa and Moriai [15]) to find a differential path for a pseudo–collision of Shabal, but they did not linearize the AND operation, but searched over all possibilities. We will elaborate on this in the following section.

3.11 Expanding the pseudo–collision attack

The pseudo–collision attack of Isobe and Shirai [17] is based on a pseudo–near–collision attack for the inverse compression function, $\mathcal{R}^{-1}$, of Shabal. This is a differential attack, and they use the differential properties given by Lipmaa and Moriai [15].
They use a linear approximation for the addition, substraction and multiplications of the compression function of Shabal, for these approximations Lipmaa and Moriai have given the success probabilities, dependent on the weight of the inputs. Due to this they could use an automatic search for a part of the path search. For the AND operation, they considered all possible outputs, this is useful to construct a high probability differential path. After the automatic search they used exhaustive search for the possible differences of the AND operation. With this construction the first 32 steps of $\mathcal{R}^{-1}$ are searched automatically, the probability for the AND operation with only a few differences is good enough. Then the last 16 steps of the path are searched manually.

With this technique they found a differential path for a pseudo–near–collision for $\mathcal{R}^{-1}$. The path starts with a 1–bit difference in the message and in C and a 2–bit difference in A. Then applies $\mathcal{R}^{-1}$ to find 15,29 and 1–bit differences in respectively A, B and C. This gives a pseudo–near–collision with probability 2^{-184}.
Using a two–block setting one could try to find a collision attack by using a near–collision and a pseudo–collision (or add intermediate free–start–near–collisions; multi–block setting). For this the result of Isobe and Shirai [17] could be used. They tried but did not succeed in finding a satisfying path. Besides, Wang has stated that finding the first block of a multi–block setting is the harder part, if this is found, the second part will be relatively easy [18]. So the first goal should be to find a near–collision starting from the initial values of Shabal.

3.11.1 Finding a near–collision

Finding a near–collision is generally harder than finding a pseudo–collision. A differential path could lead to near–collisions, but no such path is known for Shabal.
If the hash function would be continuous, one could use the Hill Climbing method, but, as we will see in the next section, for Shabal this is not very efficient.

Hill Climbing Method

Turan and Uyan used the hill climbing method to find near–collisions for round–reduced Blake, Fugue, Hamsi and JH [37]. The hill climbing method is mostly used in problems which have many near–optimal solutions, and it is very efficient if a function is continuous.
Turan and Uyan randomly select a chaining value, and search for other chaining values close to this one which minimize the Hamming distance of the mappings of the pair m_1, m_2.
We used the hill climbing algorithm starting from the initial values to try to find two message blocks m_1, m_2 which are mapped (by the permutation of Shabal) to two very close values, then we have a near–collision.
The algorithm starts with a random pair of messages m_1, m_2 and then tries to adjust m_2 to minimize the Hamming distance between the mappings of the pair. The Hamming distance HD of the m_2 and the adjusted message is small, k represents the maximum Hamming distance of this pair and $S_k(x)$ is the set of all elements for which the Hamming distance to x is smaller or equal to k. The reason for this is that if the distance is small, the search space is small[3]; thus this can be explored in feasible time.
A pair of messages m_1, m_2 is called *k–opt* if for all messages m_3 such that $HD(m_2, m_3) \leq k$, we have

$$HD\left[\mathcal{P}(IV, m_1), \mathcal{P}(IV, m_2)\right] \leq HD\left[\mathcal{P}(IV, m_2), \mathcal{P}(IV, m_3)\right].$$

If a pair is k–opt, a new message pair is randomly selected, the algorithm repeats this n times.

Algorithm: HillClimbing(k, n)
Randomly select m_1, m_2;
$dist_{best} = HD\left(\mathcal{P}(IV, m_1), \mathcal{P}(IV, m_2)\right)$;
while (m_1, m_2) is not k–opt
    $m_2 = x$ such that $x \in S_k(m_2)$ with $HD\left(\mathcal{P}(IV, m_1), \mathcal{P}(IV, x)\right) < dist_{best}$;
    $dist_{best} = HD\left(\mathcal{P}(IV, m_1), \mathcal{P}(IV, x)\right)$;
return $(m_1, m_2, dist_{best})$

[3] And for a linear function this would be a very efficient maximizing algorithm

We have run the algorithm with parameters $k = 2$ and $n = 3000$ which gave the following results. In the table p represents the number of times the first loop of step 2 of the permutation is used, e.g. if $p = 1$ there are 16 updates for A and B in the second step.

p	B	A&B		Message pair
1	179	337	M1:	0000757F 000070E3 00005699 00006D8B 00007910 000018FB
				00005E66 00005279 00003B26 0000768D 0000008B 00004E94
				000054F1 00006D02 000026FF 0000027A
			M2:	00001F6F 000051BA 0000516F 00006060 00007187 00004F5D
				000023F8 0000019F 00005563 0000221A 00004107 000046A3
				00000EB8 000052AD 0000349E 00006407
2	202	390	M1:	0000500B 00007802 00004C3A 00005401 00007B52 00003279
				00005F95 00007A52 00004B1F 000035DC 000064AF 0000427B
				00002D36 00004F10 00005221 00005526
			M2:	000059B5 000032D9 000036ED 00001F91 000042DA 000018B5
				00006EEE 00000276 000035C4 00002443 00004FE0 00005806
				00003613 00004B23 00005510 0000FBB1
3	204	404	M1:	00001C52 000072E4 00003AEF 000060A6 00001146 00003714
				00006DC3 00002879 00007EA8 0000167B 00005B84 00000DF3
				00002A74 0000248F 00005D1D 0000142A
			M2:	000054F4 00003AC7 00002377 000057D2 00000B0F 00005A81
				00000CEA 00004A56 00007BE2 000032AF 00004E43 00006A44
				0000405B 0000739B 000013C3 000058B5

Table 3.1: In the second column we have $HD(B, B')$ and in the third $HD(A||B, A'||B')$ where B has 512 bits and $A||B$ has 896 bits.

The Hamming distance of the outputs is not very small. The most important reason for this is that $\mathcal{P}$ is far from continues. A hash function should map two messages m and $m[i]$ that are almost equal[4] to outputs that are far from equal, preferably of Hamming distance close to $n/2$, so the inner primitive should not be continuous. Because of this, the hill climbing algorithm is not efficient on hash functions.

3.12 Rotational Attack

The rotational distinguisher given by Van Assche shows that the permutation is not ideal, we will first describe this distinguisher and then explain why this is not applicable to the compression function.
A rotational distinguisher for the permutation compares the outputs of x and x', where for all words we have $x'[i] = x[i] \lll_j$. As discussed in Appendix D, the rotational difference is invariant under the operations XOR and rotation. For the other operations $+, \mathcal{U}, \mathcal{V}$ we use the probability that the difference is invariant to find the probability that the outputs of x and x' are rotational. We use:

$$\Pr[x \lll_j + y \lll_j = (x + y) \lll_j] = \frac{1}{4}(1 + 2^{-j})(1 + 2^{j-n})$$

$$\Pr[\mathcal{U}(x \lll_1) = \mathcal{U}(x) \lll_1] = 2^{-1.585}$$

[4]Recall that $m[i]$ is the message m with the bit on position i complemented.

$$\Pr[\mathcal{V}(x \lll_1) = \mathcal{V}(x) \lll_1] = 2^{-1.737}.$$

For $j = 1$, e.g. 1–bit–rotation, we have probabilities $2^{-1.415}, 2^{-1.585}, 2^{-1.737}$ respectively.
The permutation contains 36 additions, and the operations $\mathcal{U}$ and $\mathcal{V}$ are applied 48 times, and thus the probability of a rotational pair to give a rotational output is

$$\Pr[\mathcal{P}(a,b,c,m) = \mathcal{P}(a',b',c',m')] \approx 2^{-(36\cdot 1.415+48\cdot 1.585+48\cdot 1.737)} = 2^{-210}.$$

If $\mathcal{P}$ would be an ideal permutation, this would be 2^{-896}, and thus we conclude that $\mathcal{P}$ is not ideal.
The main issue which makes the rotational attack of Van Assche [31] not applicable to the full Shabal is that the initial values for A, B and C are not rotational. The distinguisher requires the possibility to input x and $x' = x \lll_1$ and in the compression function only the message block can be chosen (not the start variables). One way to counterfeit this, is to first digest a (number of) message block(s) such that the internal state becomes (close to) rotational and then apply rotational analysis from there.
The next problem then will be the block counter which is XOR–ed to A. It only affects the first two words of A (and for small messages only one of the two), but it will spread quit fast due to the three rounds in step 2 of the permutation. So it might be useful to first try the above attack without using the counter.
Also the final rounds are expected to make the probability for a rotational pair smaller. This could be countered by finding a message block that has a high probability of maintaining the rotational property, and using this as the last block. But the final rounds digest the same message block and different internal states, so finding such a message block could be expensive. So again, the above analysis should first be tried without the final rounds.

3.13 Shift Attack

The shift attack compares the outputs of a tuple (x, x') where $x' = x \ll_i$.
The shift difference is invariant under all operations of $\mathcal{P}$, except for rotation. For rotation we have:

$$\Pr[(x \lll_r) \ll_s = (x \ll_s) \lll_r] = 2^{-2t}$$

where $t = \min(r, s, n-r, n-s)$, proved by Sokolowski [38].
Let us look at a shifted pair, x and $x \ll_1$. In the permutation the words of B are rotated over 17 positions in step 1 and in step 2 the words of B are rotated over 1 position and the words of A are rotated over 15 positions. The probability that the shifted input pair gives a shifted output pair is 2^{-2} for each of these rotations, as the shift value is the minimum. So in total, the probability for a shifted input pair to find a shifted output pair after the permutation is $2^{-2(16+3\cdot 16+3\cdot 16)} = 2^{-224}$.
For an ideal permutation $\mathcal{F}$ we find

$$\Pr[\mathcal{F}(x \ll_1) = (\mathcal{F}(x)) \ll_1] = 2^{-896}$$

and again we can conclude that the permutation of Shabal is not ideal.
This attack can be improved by choosing particular values for the B– and C–input. Choosing the values for B makes it possible to have probability 1 for step 1, that is for all words of B we have $(B[i] \lll_{17}) \ll_1 = (B[i] \ll_1) \lll_{17}$. Even more, we could choose B such that it also holds for the first round of step 2. In this case we would proceed through $16 + 16$ rotations with probability 1 and the probability to find a shifted output of $\mathcal{P}$ drops to $2^{-224+64} = 2^{-160}$.

The values of C help us pass through the first 16 rotations of A in step 2 with probability 1, C is such that for each word of A we have $(5 \cdot A[i] \lll_{15} \oplus C[j]) \ll_1 = 5 \cdot (A[i] \ll_1) \lll_{15} \oplus C[j]$. Then the probability of a shifted output pair drops even further to $2^{-160+32} = 2^{-128}$.

The advantage of a left shift[5] attack compared to a rotational attack is that the probability of maintaining a shifted pair under $\mathcal{U}$ and $\mathcal{V}$ is 1 (for a rotational pair this depends highly on the rotation amount, but for 1 it is approximated by Van Assche as $2^{-1.585}$ and $2^{-1.737}$ respectively [31]).

This attack can not be extended to the full hash function, as then it is not possible to choose the input pairs for A, B and C as shifted pairs.

3.14 Differential Attack

The first differential path for the permutation of Shabal has been given by Novotney [32]. This distinguisher shows that the diffusion is not sufficient for a given input difference, thus the output can be distinguished from the output of a random oracle. Novotney uses the following input differences in A and B:

$$\begin{aligned} \Delta A^0[10] &= \texttt{80000000} \\ \Delta B^{-1}[7] &= \texttt{00002000} \end{aligned}$$

Note that the difference of B after the first step of the permutation (rotation to the left by 17) is $\Delta B^0[7] = 40000000$. Recall that with B^{-1} we present the initial value of B, before step 1 of the permutation.
This pair of differences does not diffuse until round 26 with some probability. The diffusion is described in the following list. A list of all rounds can be found in Appendix B.

- **round 1** $\Delta A^1[1] = \Delta(B^0[10] \wedge \overline{B^0[7]})$, so $\Delta A^1[1] = 0$ if $\Delta B^0[10] \wedge \overline{B^0[7]} = 0$, this occurs with probability $1/2$, as there is only a 1–bit difference in $B^0[7]$. If we would restrict $B^{-1}[10] = 0$, the difference vanishes with probability 1. So we conclude $\Delta A^1[1] = \Delta B^1[1] = 0$.
- **round 7** $\Delta B^1[7] = \Delta B^0[7] \lll_1 = \texttt{80000000}$.
- **round 10** $\Delta A^1[10] = \Delta\left(3(A^0[10]) \oplus B^1[7]\right)$, now since $3\Delta A^1[10] = \Delta A^1[10] = \Delta B^1[7] = \texttt{80000000}$, the differences will cancel and thus $\Delta A^1[10] = \Delta B^1[10] = 0$.
- **round 14** $\Delta A^2[2] = \Delta(B^1[7] \wedge \overline{B^1[4]})$, so $\Delta A^2[1] = 0$ if $\Delta B^1[7] \wedge \overline{B^1[4]} = 0$, this occurs with probability $1/2$. So we conclude $\Delta A^1[1] = \Delta B^1[1] = 0$ with probability $1/2$.
- **round 17** $\Delta A^2[5] = \Delta(B^1[10] \wedge \overline{B^1[7]})$, so $\Delta A^2[5] = 0$ if $\Delta B^1[10] \wedge \overline{B^1[7]} = 0$, this occurs with probability $1/2$, as there is only a 1–bit difference in $B^1[7]$. So we conclude $\Delta A^2[5] = \Delta B^2[1] = 0$ with probability $1/2$.
- **round 23** $\Delta B^2[7] = \left(\Delta B^1[7]\right) \lll_1 = \texttt{00000001}$
- **round 26** $\Delta A^3[2] = \Delta B^2[7] = \texttt{00000001}$ and $\Delta B^2[10] = \Delta A^3[2] = \texttt{80000000}$.

At round 26 we find 1–bit differences in three words $A^3[2], B^2[7]$ and $B^2[10]$ with probability $1/8$, or probability $1/4$ if we are allowed to set $B^{-1}[10] = 0$.

[5] In the following we will use shift if we mean left shift, as this maintains many operations and the right shift does not. If we mean right shift we will explicitly mention it.

In round 32 the first output word $B^3[0]$ is set, thus there are only 6 rounds in which the differences can diffuse for this particular word. This appears not to be enough and it is thus possible to distinguish between the output of the permutation and the output of a random oracle. Novotney measured the bias of all bits of the output of the first word of B experimentally. This results in a distinguisher of complexity 2^{23} for bit 26 and if the value for $B^{-1}[10]$ can be chosen the complexity descends to 2^{21}.
A detailed description of this path is given in Appendix B.

3.14.1 Combination of attacks

Wang uses the combination of modular differences and XOR differences to break MD5 [18]. The advantage of this combination compared to only using XOR differences is that the active bits are determined. For example a two–bit XOR difference $(1,1)$ could mean $(0,0),(1,1)$ or $(1,0),(0,1)$ which can make a difference for example when these bits are multiplied in a later stage. If also the modular difference is known, then we can distinguish between the two cases above and the difference after the multiplication is also known.
The differential path used by Wang is most likely found by hand. When the path is given, the combination of the differences makes it easier to find conditions to enlarge the probability of the path. If a set of sufficient conditions exist, then the messages satisfying these conditions will, with high probability, lead to a collision.
Isobe and Shirai found a differential path for a pseudo–collision [17] for which the messages differ only in one bit. They used a computer program to decide whether the differences in B effect the multiplication, they chose the path where the differences of A and B are of smallest weight. Instead of using the combination they considered all possible difference patterns and thus the combinational attack will probably not improve the attack of Isobe and Shirai.
Fouque et. al. presented an algorithm to automatically improve a differential path [39], to minimize the number of conditions[6]. The algorithm is written for MD4, and they improved the results of Wang [40]. The algorithm starts from the output and finds a start point for that output. Thus, the first run might give a pseudo–collision. Then the algorithm is ran again to modify the path to lower the differences from the initial value.
The algorithm is trying to go through the whole search space in a smart way, i.e. if a step leads to high weight differences then this path is abandoned and cheaper corrections are favored to expensive corrections. The message block space of Shabal is the same as of MD–4 and MD–5. But the operations in the permutation of Shabal are more time consuming, therefore this attack is not efficient for Shabal.

3.15 Algebraic properties of Shabal

Tonien has described a way to use algebraic properties of hash functions for a collision test [41]. He wants to apply this to SHA–1, but it can be applied to any function for which each of the n output bits can be described by a multivariate polynomial.

[6] An example of such a condition: if $\Delta x_i = 0$ then $x_i = x_i' = 1$, and not $x_i = x_i' = 0$.

3.15.1 Algebraic collision test for $l \leq \log_2(r)$

For the full description we refer to the analysis of Tonien [41].
Let h_i be the i^{th} output bit for $i = 1, \ldots, n$ and m_i the i^{th} input bit for $i = 1, \ldots, j$. Let $\mathcal{F}$ be the mapping such that $h_i = \mathcal{F}_i(m_1, \ldots, m_j)$. This map exists, but is not easily discovered for each hash function, as we will see later on. Now there exists a $n \times 2^j$ matrix F for $\mathcal{F}$ such that:

$$\begin{pmatrix} h_1 \\ \ldots \\ h_n \end{pmatrix} = F \begin{pmatrix} 1 \\ m_1 \\ \ldots \\ m_2 \ldots m_j \\ m_1 m_2 \ldots m_j \end{pmatrix}$$

Now Tonien uses that a $k \times 2^l$ matrix has a left inverse if $k > 2^l$ and the matrix is of full rank. If one chooses l indices i_j where the input is nonzero, then the $n \times 2^l$ matrix F_l is such that:

$$\begin{pmatrix} h_1 \\ \ldots \\ h_n \end{pmatrix} = F_l \begin{pmatrix} 1 \\ m_{i_1} \\ \ldots \\ m_{i_2} \ldots m_{i_l} \\ m_{i_1} m_{i_2} \ldots m_{i_l} \end{pmatrix}$$

So we choose $l < \log_2(n)$ and find the rank of F_l to decide whether F_l has a left inverse. If F_l has a left inverse there is a one–to–one correspondence between input and output. So then there exist no two different messages, with active bits only in the given l places, which are mapped to the same bit–sequence. However if the matrix is not of full rank, colliding messages can be found by solving linear equations. The difference of the two messages must be in the nullspace of $\mathcal{F}$.

3.15.2 Collisions for $l > \log_2(r)$

If an $n \times 2^l$ matrix F_l has full rank, where $2^l > n$, then the matrix has a right inverse. We know that the dimension of the nullspace of F_l is equal to the dimension of the domain minus the rank of F_l. As here we have rank n strictly smaller than the dimension of the domain 2^l, the nullspace has dimension greater than 0. So there are non–zero elements in the nullspace of F_l.
F_l has a right inverse F_l^{-1}, so $F_l F_l^{-1} = I_n$ where I_n is $n \times n$ the identity matrix.
An inverse can be computed using $F_l^{-1} = F_l^T (F_l F_l^T)^{-1}$. We want to find $m \neq m'$ such that $F_l m = F_l m'$, now if we let $m' = F_l^{-1} F m$ then:

$$Fm' = F F_l^{-1} F m = Fm.$$

There are restrictions since m' should be of the form $(1, m'_{i_1}, \ldots, m'_{i_1} m'_{i_2} \ldots m'_{i_l})^T$. So we get a set of $2^l - l - 1$ equations. If this set of equations has a solution, we have found a pair of colliding messages.

Small example

Let

$$\begin{aligned} h_1 &= m_1 + m_2 \\ h_2 &= 1 + m_1 + m_2 \\ h_3 &= m_1 m_2 \end{aligned}$$

then we find

$$F = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

and

$$F^{-1}F = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

thus the set of 2 equations:

$$\begin{aligned} 1 + m_1 + m_2 &= 0 \\ m_1 m_2 &= 0. \end{aligned}$$

This system has two solutions $m = (1, 0)$ and $m' = (0, 1)$, so we find a colliding pair. This is not always possible and for more complex mappings this will be time consuming.

3.15.3 Application to Shabal

First we shall describe how the operations of the compression function of Shabal can be described as bit operations.
Rotation and *XOR* are bit–operations. The negation of x can be replaced by $1 - x$ and $x \wedge y$ operation by xy. Now we need to describe $\mathcal{U}$ and $\mathcal{V}$.
$\mathcal{U}(x) = 3x = x + x \ll_1$ can be bitwise defined by:

$$\begin{aligned} c_0 &= 0 \\ w_0 &= x_0 \\ c_1 &= x_0 x_1 \\ w_i &= c_i \oplus x_i \oplus x_{i-1} \\ c_{i+1} &= x_i x_{i-1} \oplus c_i x_i \oplus c_i x_{i-1} \end{aligned}$$

where w_i are the result bits and c_i are the so–called carry bits, for $i = 0, \ldots$,n.
And the same way $\mathcal{V}(x) = 3x = x + x \ll_1$ can be bitwise defined by:

$$\begin{aligned} c_0 &= 0 \\ w_0 &= x_0 \\ c_1 &= 0 \\ w_1 &= x_1 \\ c_2 &= x_0 x_2 \\ w_i &= c_i \oplus x_i \oplus x_{i-2} \\ c_{i+1} &= x_i x_{i-2} \oplus c_i x_i \oplus c_i x_{i-2} \end{aligned}$$

where w_i are the result bits and c_i are the so–called carry bits, for $i = 0, \ldots$.

Complexity

The bitwise definitions of $\mathcal{U}$ and $\mathcal{V}$ will raise the size of the internal state if a lot of the message bits are unknown. If for example only one word of the message is treated as unknown, these 32 variables will be inserted in two A in a particular step, in the next step this word of A is inside $\mathcal{U}$ and $\mathcal{V}$ and the result in this step will contain the 32 degree multivariate polynomial $x_0 x_1 \cdots x_{31}$. This part will be 0 unless all bits are 1–bits, so most probably this will have no effect. So the degree of the equations can be downsized by setting a limit on the weight of the words of the message, if a word has at most weight l, then any multiplication of more than l bits will be 0.

Find polynomials for low–weight messages

The polynomials for two active bit–positions, x_1 and x_2 can be found by evaluating $\mathcal{F}$ in 0, e_{x_1}, e_{x_2} and $e_{x_1} \oplus e_{x_2}$, where e represents the unit vector. Then the polynomials are given by:

$$F(x_1, x_2) = \mathcal{F}(0) + \mathcal{F}(e_{x_1})x_1 + \mathcal{F}(e_{x_2})x_2 + \mathcal{F}(e_{x_1} \oplus e_{x_2})x_1 x_2.$$

So find the complete polynomial, so for all bit–positions active this would require 2^{512} queries to the compression function of Shabal. If we want to apply the algebraic collision test to Shabal, we need $l \leq \log_2(512) = 9$. So we can test the existence of collisions of messages of weight at most 9. Finding the polynomial for an active 9–tuple, requires $\sum_{i=0}^{9} \binom{9}{i} = 2^9$ queries to the compression function. But there are $\binom{512}{9}$ possible 9–tuples, so to do this for all 9–tuples requires $\binom{512}{9}2^9$ queries.
Besides this, it is assumed that there will not be such a collision.

$l > 9$

To find the matrix for an l–tuple, $l > 9$, costs 2^l compression function queries. From this we could find the nullspace which costs a certain number of matrix operations. But then we find an overdetermined system, as we have that both message vectors must be of the form $(1, m_1, m_2, \ldots, m_1 m_2, \ldots, m_1 \ldots m_l)$ and the difference should be in the nullspace. So we have l degrees of freedom and 2^l constraints. Only in very special cases this system will have a solution. Since not all elements of the vector $(1, m_1, m_2, \ldots, m_1 m_2, \ldots, m_1 \ldots m_l)$ are independent.

3.16 Conclusions on the security of Shabal

Most results on Shabal are distinguishers. A distinguisher does not often lead to an attack on the full hash function, and thus distinguishers are not always seen as important. Indifferentiability and distinguishers are originally defined for ciphers, ciphers are supposed to behave like a random oracle and therefore it is important that they can not be distinguished from a random oracle.
For a hash function, the terms of security are not (yet) defined in terms of indifferentiability, and the distinguishers for Shabal can not be used to find preimages or collisions (yet).
The distinguishing property of distinguishers is not a reason to question the security of Shabal, none of the distinguishers applies to the full function, and they do not use the initial values of Shabal.
On the other hand is the indifferentiability proof of the authors of Shabal changed two times because of distinguishers. The proof was based on an incorrect assumption, that the permutation is ideal. Again there are a few assumptions in the proof, based on distinguishers, but a new, better distinguisher might reject these assumptions again.
At some point the authors also stated that the compression function is as secure as when the counter and the final rounds are removed, this proof also does not take the initial values into account. Many distinguishers, but in particular the rotational distinguisher of Van Assche will then work on the full hash function.
In the last proof this is somewhat taken into account, because p_C relies on whether there are final rounds or not. If the final rounds of Shabal are removed, the probability that ABORT_V will occur will increase to $N2^{-278}$ due to the last term. And, following the proof, Shabal will be indifferentiable from a random oracle up to 2^{278} requests.

We conclude that it is difficult to have trust in the indifferentiability proofs if they change often and still rely on assumptions that have not been proven to be right.

3.16.1 Ideas of further Analysis

The combination attack of modular addition and XOR differences could be useful to gain differential paths with high probability. The path search algorithm of Fougue et al. will not be convenient for Shabal because the search space is too large.

Yu et al. recently started to apply a rebound attack to an ARX hash function [42], this might work for Shabal too.

Chapter 4

BLAKE

BLAKE is one of the five finalists of the NIST SHA–3 competition. BLAKE is designed by Aumasson et al [43].
It was possible to adjust the function after it was selected for the next round. For BLAKE the number of rounds have changed and the names of the functions, no extreme changes. This implies that the function has been steady since the first round, and all the analysis is still applicable.
In this chapter we will first describe BLAKE, discuss the recent analysis and we will explain some discrepancies in the previous analysis.

4.1 The mode of operation

The submission document of Blake describes four different designs, which digest 224, 256, 384 an 512 bits respectively. As the functions are quite similar, in this thesis we will mostly elaborate on BLAKE–256.
The 224– and 256–bitlength digest are produced by the BLAKE–256 function, which operates on 512–bit message blocks. The 384– and 512–bitlength digest are produced by BLAKE–512, which operates on 1024–bit message blocks. The differences between these four functions are, the message block length, the padding, the initial values, the number of rounds and the truncation.
The hash of a message is constructed by the following steps; first the message is padded, then the compression function iteratively digests all message blocks, the output is the last chaining value (or a truncated chaining value), this is the hash of the message.
From now we will only consider BLAKE–256 and we will denote this with either BLAKE–256 or just BLAKE.
A message is padded so that the length is congruent 448 modulo 512. First a 1 bit is appended, then a sufficient number of zeros, followed by another 1. To end the padding we add a 64–bit unsigned big–endian representation of the bitlength of the message.

4.1.1 The compression function

BLAKE is a hash function based on the HAIFA structure, so the compression function takes four input values:

1. a chaining value h of 256 bits,
2. a message block m of 512 bits,

3. a salt s of 128 bits,
4. and a counter t of 64 bits.

The four variables contain 960 bits all together, and the output is the new chaining value. So the compression function is $\mathcal{C} : \{0,1\}^{960} \to \{0,1\}^{256}$ for BLAKE–256.
The internal state of the compression function is a 4×4–word matrix, this matrix contains 512 bits.
First the state is initialized as follows:

$$\begin{pmatrix} v_0 & v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 & v_7 \\ v_8 & v_9 & v_{10} & v_{11} \\ v_{12} & v_{13} & v_{14} & v_{15} \end{pmatrix} \leftarrow \begin{pmatrix} h_0 & h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 & h_7 \\ s_0 \oplus c_0 & s_1 \oplus c_1 & s_2 \oplus c_2 & s_3 \oplus c_3 \\ t_0 \oplus c_4 & t_0 \oplus c_5 & t_1 \oplus c_6 & t_1 \oplus c_7 \end{pmatrix}$$

Where h represents the chaining value, s the salt, t the counter and c a constant, each element of the matrix is a 32–bit word. In the first round the initial values are used as chaining variables. The initial values and the constants are given in Appendix C.

The Round function

The internal state of the compression function is twice the size of a chaining variable, so 512 bits for BLAKE–256. The compression function is based on an inner primitive, the round function $\mathcal{R} : \{0,1\}^{1024} \to \{0,1\}^{1024}$, which consists of addition, rotation and XOR (ARX).
Each round the functions $\mathcal{G}_0, \ldots, \mathcal{G}_7$ are applied to a subset of the state. We call the part where the first four are applied the column step:

$$\mathcal{G}_0(v_0, v_4, v_8, v_{12}),\ \mathcal{G}_1(v_1, v_5, v_9, v_{13}),\ \mathcal{G}_2(v_2, v_6, v_{10}, v_{14}),\ \mathcal{G}_3(v_3, v_7, v_{11}, v_{15})$$

and last four together are called the diagonal step:

$$\mathcal{G}_4(v_0, v_5, v_{10}, v_{15}),\ \mathcal{G}_5(v_1, v_6, v_{11}, v_{12}),\ \mathcal{G}_6(v_2, v_7, v_8, v_{13}),\ \mathcal{G}_7(v_3, v_4, v_9, v_{14}).$$

And $\mathcal{G}_i(a, b, c, d)$ in round r is defined as:

$$\begin{aligned} a &\leftarrow a + b + (m_{\sigma_r(2i)} \oplus c_{\sigma_r(2i+1)}) \\ d &\leftarrow (d \oplus a) \ggg_{16} \\ c &\leftarrow c + d \\ b &\leftarrow (b \oplus c) \ggg_{12} \\ a &\leftarrow a + b + (m_{\sigma_r(2i+1)} \oplus c_{\sigma_r(2i)}) \\ d &\leftarrow (d \oplus a) \ggg_{8} \\ c &\leftarrow c + d \\ b &\leftarrow (b \oplus c) \ggg_{7} . \end{aligned}$$

Here σ is a permutation, to make sure that the same word of a message block is not inserted on the same spot twice. We will omit the i for statements that do not depend on i and thus with $\mathcal{G}$ we denote any $\mathcal{G}_i$.
For $\mathcal{G}$, for the $\oplus$ differences we make the following distinctions:

1. $\Delta a, \Delta b, \Delta c, \Delta d$ denote the input differences,
2. $\Delta a^*, \ldots, \Delta d^*$ denote the differences after each element is updated once, and
3. $\Delta a', \ldots, \Delta d'$ denote the output differences.

σ_0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
σ_1	14	10	4	8	9	15	13	6	1	12	0	2	11	7	5	3
σ_2	11	8	12	0	5	2	15	13	10	14	3	6	7	1	9	4
σ_3	7	9	3	1	13	12	11	14	2	6	5	10	4	0	15	8
σ_4	9	0	5	7	2	4	10	15	14	1	11	12	6	8	3	13
σ_5	2	12	6	10	0	11	8	3	4	13	7	5	15	14	1	9
σ_6	12	5	1	15	14	13	4	10	0	7	6	3	9	2	8	11
σ_7	13	11	7	14	12	1	3	9	5	0	15	4	8	6	2	10
σ_8	6	15	14	9	11	3	0	8	12	2	13	7	1	4	10	5
σ_9	10	2	8	4	7	6	1	5	15	11	9	14	3	12	13	0

After the ten rounds the new chaining variable is generated as follows:

$$\begin{aligned}
h'_0 &\leftarrow h_0 \oplus s_0 \oplus v_0 \oplus v_8 \\
h'_1 &\leftarrow h_1 \oplus s_1 \oplus v_1 \oplus v_9 \\
h'_2 &\leftarrow h_2 \oplus s_2 \oplus v_2 \oplus v_{10} \\
h'_3 &\leftarrow h_3 \oplus s_3 \oplus v_3 \oplus v_{11} \\
h'_4 &\leftarrow h_4 \oplus s_0 \oplus v_4 \oplus v_{12} \\
h'_5 &\leftarrow h_5 \oplus s_1 \oplus v_5 \oplus v_{13} \\
h'_6 &\leftarrow h_6 \oplus s_2 \oplus v_6 \oplus v_{14} \\
h'_7 &\leftarrow h_7 \oplus s_3 \oplus v_7 \oplus v_{15}
\end{aligned}$$

4.1.2 Toy versions of BLAKE [1]

The authors provided four reduced versions of BLAKE. Now to analyze BLAKE, one can start with one of the reduced versions, and try to extend the attack to BLAKE or discuss why the missing property in the reduced version is important. This gives a uniform name to different methods of analyzing.

BLOKE

The permutations are replaced by the identity function.

FLAKE

The compression function makes no feedforward so the finalization of BLAKE-256 becomes

$$h'_i \leftarrow v_i \oplus v_{i+8}.$$

BLAZE

The addition of constants is removed, so replacing a becomes

$$a \leftarrow a + b + m_{\sigma_r}.$$

BRAKE

BRAKE is reduced to have the changes of all three BLOKE, FLAKE and BLAZE.

4.2 Recent Analysis

4.2.1 On the inner primitive $\mathcal{G}$

Aumasson, Guo, Knellwolf, Matusiewicz and Meier exploit the differential properties of the inner primitive $\mathcal{G}$ to find impossible differential paths [44]. The differences in the input of $\mathcal{G}$, $(\Delta a, \Delta b, \Delta c, \Delta d)$ and the output $(\Delta a', \Delta b', \Delta c', \Delta d')$ are restricted in several ways, they give some impossible classes and some probability 1 differential characteristics for the cases that there is a nonzero input difference.
Further they researched the existence of impossible differentials. Defined the inverse of $\mathcal{G}$ for 1.5 round, which gives a preimage attack of lower complexity than the one of Li and Liangyu [45]. They conjecture that a meet in the middle strategy for a preimage attack on BLAKE cannot apply to more than five rounds. Further they repeat what has been done before [46] and conclude that we need non–linear connectors to find collisions on more than four rounds. Last they give a bound on the probability for any DC.
Ming, Qiang and Zeng study the reversibility properties of $\mathcal{G}$ [47]. This paper exploits some flaws of the permutation σ and carries out a detailed analysis for differential characteristics of $\mathcal{G}$ and its inverse function. They analyze the security of BLAKE–32. The results show that BLAKE has a strong resistance against differential attacks.

4.2.2 On toy versions

Vidali, Nose and Pašalić found collisions for variants of BLAKE [48]. Collisions for full–round BLOKE can be found using fixed points of the inner primitive and internal collisions can be found for full–round BRAKE. So it can be concluded that the permutation and the message padding are important to the security of BLAKE.

4.2.3 On round–reduced versions

Li and Liangyu attacked 1.5, 2 and 2.5 rounds of BLAKE using message modification [45]. For the 1.5 round attack they guess the value of the message and of v_0 and v_4 after 1.5 rounds. Then they calculate forwardly and backwardly the value of the internal state after one round. In case of equality you will have a preimage, otherwise they modify 4 words of m. Repeat this until a

preimage is found. For the other two number of rounds the also apply a meet–in–the–middle attack. The complexity of the preimage attacks of BLAKE–256 are $2^{192}, 2^{224}$ and 2^{224} respectively.
Guo and Matusiewicz found near–collisions for BLAKE reduced to 4 rounds [46]. They noted that rotations over $7, 8, 12$ and 16 bits are used in the round function, three of those are divisible by 4. They used a difference that is rotational by 4 and tried to avoid that the differences go through the rotation by 7. The search space for chaining variables and messages is reduced to 2^{32} by these constraints[1], which is feasible. This can be reduced even more and via automatic search they have found a 24–bit near–collision for 4 rounds reduced BLAKE. This method can not be extended to more rounds as the search space ist too small. Secondly they state that if the constants of BLAKE would be equal, the hash output has a symmetry property and can be distinguished from a random output.
Su, Wu, Wu and Dong applied differential analysis to BLAKE [49]. They found near–collisions for 4 round–reduced BLAKE-256 and for 4 and 5 round–reduced BLAKE-512. This improves the result of Guo and Matusiewicz.
Turan and Uyan used the hill–climbing technique to find near–collisions for 1, 1.5 and 2 round–reduced versions of BLAKE [37]. The hill–climbing technique searches near–optimal solutions of a minimization problem. In this case we are minimizing the hamming weight of the difference of two digests. The algorithm starts with two random messages and searches better results close[2] to one of the two messages. Obviously this method will work best if the diffusion of differences is low, which is the case for BLAKE with a small number of rounds.

4.3 On the full function

Gligorsky stated that the narrow–pipe[3] candidates differ significantly from ideal random functions defined over big domains [36]. The size of a subset of the codomain of which the elements have no preimage, is different for the narrow–pipe hash functions than for an ideal random function. Especially when the padding is such, that the last block can contain no message bits, the probability of an element of the codomain to have no preimage is higher than for a ideal random function. There are no direct applications or threats to the security of narrow–pipe designs.

There are three more papers on the security of BLAKE which we did not go through[4]. We will briefly address them.
Biryukov, Nikolić and Roy apply the boomerang attack to BLAKE–32 [50]. They presented their attack at FSE 2011.
Aumasson et. al. define *tuple cryptanalysis* and apply this to BLAKE [51]. They suggest that tuple cryptanalysis can be a criterion to evaluate ARX algorithms.
Dunkelman and Khovratovich present differential attacks and distinguishers for round–reduced versions of BLAKE [52]. They argue that their results show that the belief that ARX algorithms are secure against differential attacks might not hold.

[1] I don't think this is true, as we can choose two message words and four chaining words, both of which four positions determine all other positions, so $2^{2\cdot4+4\cdot4} = 2^{24}$
[2] In this case close means that the hamming distance of the difference of the old and the new message is at most 2.
[3] BLAKE is narrow–pipe as the chaining value is half the size of the message blocks
[4] Since these papers were published close to the enddate of this project.

4.4 Properties

Recall that $\Delta a, \Delta b, \Delta c, \Delta d$ denote the input differences with respect to $\oplus$, $\Delta a^*, \ldots, \Delta d^*$ denote the $\oplus$ differences after each element is updated once by $\mathcal{G}$, and $\Delta a', \ldots, \Delta d'$ denote the output differences of $\mathcal{G}$ with respect to $\oplus$.
For now, we consider the constants c to be zero. In one round this can be seen as $m' = c \oplus m$, where m' is what we will call the message block, while m is the original message block. After one round the constants and message blocks are combined differently and therefore the above does no longer hold. In the latter case we will just evaluate the toy version BLAZE.

4.4.1 The round function is a permutation

First we will show that $\mathcal{G}_i$ is a permutation, and then we extend this to the round function. We consider m to be a constant.

$\mathcal{G}_i$ is a permutation

Let $\mathcal{Y} = \{0,1\}^{128}$ represent the set of all 4–tuples (a, b, c, d) of 32–bit words. Now $\mathcal{G}_i$ is a mapping from $\mathcal{Y}$ to $\mathcal{Y}$, if we consider m to be a constant.
Now for each element $y \in \mathcal{Y}$ we have $\mathcal{G}_i(y) \in \mathcal{Y}$ since the output of $\mathcal{G}_i$ is a 4–tuple of 32–bit words. So every element has an image.
Now for each element $y \in \mathcal{Y}$ we can compute the pre–image as follows:

$$\begin{aligned}
b &\leftarrow c \oplus (b \lll_7) \\
c &\leftarrow c - -d \\
d &\leftarrow a \oplus (d \lll_8) \\
a &\leftarrow a - -b - -(m_{\sigma_r(2i+1)} \oplus c_{\sigma_r(2i)}) \\
b &\leftarrow c \oplus (b \lll_{12}) \\
c &\leftarrow c - -d \\
d &\leftarrow a \oplus (d \lll_{16}) \\
a &\leftarrow a - -b - -(m_{\sigma_r(2i)} \oplus c_{\sigma_r(2i+1)}).
\end{aligned}$$

We conclude that every element has a precisely one pre–image, thus $\mathcal{G}_i$ is bijective.
A map from a set into itself, which is bijective is a permutation, thus $\mathcal{G}_i$ is a permutation.

Extension

Let $\mathcal{V} = \{0,1\}^{512}$ denote the set of all 16–tuples $(v_0, \ldots, v_{15})$ of 32–bit words. Now the round function $\mathcal{R}$ is a mapping from $\mathcal{V}$ to $\mathcal{V}$, again we consider m to be a constant.
We may consider $\mathcal{G}_i$ as a mapping from $\mathcal{V}$ to $\mathcal{V}$ as follows, $\mathcal{G}_0$ maps an element $(v_0, \ldots, v_3, c_4, \ldots, c_{15})$ to $(v'_0, \ldots, v'_3, c_4, \ldots, c_{15})$, $\mathcal{G}_1$ maps an element $(c_0, \ldots, c_3, v_4, \ldots, v_7, c_8, \ldots, c_{15})$ to $(v'_0, \ldots, v'_3, c_4, \ldots, c_{15})$, etcetera, where c_i represents a constant.

We have already shown that $\mathcal{G}_0$ is a permutation on the smaller space $\mathcal{Y}$. Now $\mathcal{Y}$ can be represented with the set of all 16–tuples $(a, b, c, d, c_4, \ldots, c_{15})$ where $c_4, \ldots, c_{15}$ are constants, let this set be $\mathcal{V}^{*0}$. Now for each $\mathcal{G}_i$ there exists a set $\mathcal{V}^{*i}$ such that $\mathcal{G}_i$ is a permutation on this set.
Now we state that applying $\mathcal{G}_3 \circ \mathcal{G}_2 \circ \mathcal{G}_1 \circ \mathcal{G}_0$ to $\mathcal{V}$ gives a permutation.
Here we only take $\mathcal{G}_0, \ldots, \mathcal{G}_3$ into account. Since for each $\mathcal{G}_i$ a different part of m is used, each $\mathcal{G}_i$ will map a different 4–tuple of the 16–tuple v independently from every other $\mathcal{G}_i$. Thus, each $\mathcal{G}_i$ permutes a 4–tuple of v, the 4–tuples are disjunct, so after the four mappings the whole state is permuted.
We conclude that $\mathcal{G}_3 \circ \mathcal{G}_2 \circ \mathcal{G}_1 \circ \mathcal{G}_0$ is a permutation on $\mathcal{V}$ for fixed m.
Similarly we can conclude that $\mathcal{G}_7 \circ \mathcal{G}_6 \circ \mathcal{G}_5 \circ \mathcal{G}_4$ is a permutation on $\mathcal{V}$.
As the concatenation of two permutations is also a permutation, we conclude that the round function $\mathcal{R}$ is a permutation on $\mathcal{V}$ for fixed m.

Consequences

Above we have shown that the round function is a permutation if m is fixed. Now to extend this to more rounds, we want the function to be a permutation of the message for a fixed starting state. Again we have $\mathcal{Y} = \{0,1\}^{512}$ represent the set of all the messages. Now a round is a mapping from $\mathcal{Y}$ to $\mathcal{Y}$, if we consider $m \in \mathcal{Y}$ to be the input and the starting state to be a constant.
Since we can find the message given the starting state and the output, we conclude that every output has a preimage in the form of a message. Therefore the round function is a permutation of the message.
In the next round we again let the state be fixed (the output of the previous round), digesting another 512–bit message block is again a permutation of the message block. Now, the parallel processing of two different 512–bit message blocks in this setting, will carry out two different states.
In the submission document it is stated that, because the round function is a permutation of the message blocks, one cannot find two distinct messages that produce the same internal state [43, P. 32].

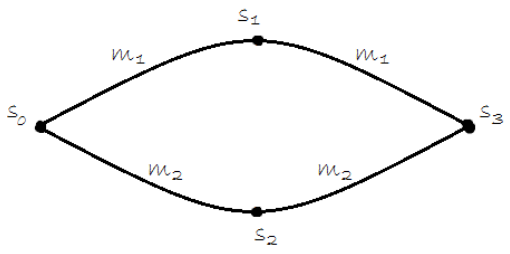

Figure 4.1: Two rounds applied to two different messages

Figure 4.1 shows that applying two rounds to two different messages $m_1 \neq m_2$ could give a collision. The statement that this does never occur, does not take into account that after one round, the states are different. Since the states after one round are different ($s_1 \neq s_2$) the permutations in the next round are not one and the same. The two different permutations (but defined on the same set) applied to the two different messages, might output the same state (s_3).

We conclude that the sequence of multiple rounds cannot be seen as a permutation of the message, as the internal states are not always the same, the permutations are not the same.

4.4.2 Differential Properties

This section is divided in three main parts.
First we will the possible states after one application of $\mathcal{G}$ to a state without differences. There are eight possible end states, one more than is claimed by Aumasson et. al. [44].
In the next section we proof that it is possible to reach the end state $(\Delta, 0, \Delta', 0)$, which is claimed otherwise by Aumasson et. al. [44].
Then we will see what happens if another $\mathcal{G}$ is added and the consequences of these possibilities.
Last we will discuss the consequences for the round function.

On one $\mathcal{G}_i$

First we will look at input states without differences. Now there will be no changes if both message block pairs do not have a difference, so we let at least one of them contain a difference. We split $\mathcal{G}_i$ in two halves and look at the two cases, $\Delta m_i = 0$ and $\Delta m_i \neq 0$.
Case 1: $\Delta m_i = 0$
There is no difference inserted in the first half, so $\Delta m_j \neq 0$ and this difference spreads in all four values, so in this case we find $\Delta a', \Delta b', \Delta c', \Delta d' \neq 0$.
Case 2: $\Delta m_i \neq 0$
In this case the difference is spread to all intermediate values, but it could be canceled by Δm_j. So assume $\Delta m_j \neq 0$, then we have the following output restrictions:

$$\begin{array}{lll|lll} \Delta a' = 0 & \Rightarrow & \Delta d' \neq 0 & \Delta c' = 0 & \Rightarrow & \Delta b' \neq 0 \quad \wedge \quad \Delta d' \neq 0 \\ \Delta b' = 0 & \Rightarrow & \Delta c' \neq 0 & \Delta d' = 0 & \Rightarrow & \Delta a' \neq 0 \quad \wedge \quad \Delta c' \neq 0. \end{array}$$

Proof. Recall that all intermediate value pairs have a nonzero difference, $\Delta a^* \neq 0$ etc. and $\Delta m_i \neq 0$.

$\Delta a' = 0$ then $\Delta d' = (\Delta d^* \oplus \Delta a') \ggg_8 = (\Delta d^*) \ggg_8 \neq 0$.

$\Delta b' = 0$ then $0 = \Delta b' = (\Delta b^* \oplus \Delta c') \ggg_7$ thus $\Delta c' = \Delta d^* \neq 0$.

$\Delta c' = 0$ then:
$\Delta b' = \Delta b^* + \Delta c' = \Delta b^* \neq 0$ and
$0 = \Delta c' = c^* + d'$ thus $\Delta c^* + \Delta d' = 0$ thus $\Delta d' \neq 0$.

$\Delta d' = 0$ then:
$\Delta c' = \Delta c^* + \Delta d' = \Delta c^* \neq 0$ and
$0 = \Delta d' = (d^* \oplus a') \ggg_8$ thus $\Delta d^* = \Delta a' \neq 0$.

□

So we can conclude that there must be at least two words with a non–zero output difference. Secondly, starting with a zero–differences state it is impossible to end in one of the following states

$$(\Delta, 0, 0, 0), (0, \Delta, 0, 0), (0, 0, \Delta, 0), (0, 0, 0, \Delta), (\Delta, \Delta', 0, 0), (\Delta, 0, \Delta', 0), (\Delta, 0, 0, \Delta'), (0, \Delta, \Delta', 0).$$

All but one of these impossible states follow directly from the output restrictions given previously (the full reasoning is given in Appendix C).

Why is the state $(\Delta, 0, \Delta', 0)$ impossible? In this section, we use $\mathcal{G}_{m_1,m_2}(a,b,c,d)$ to represent the round function $\mathcal{G}$ applied to (a,b,c,d) using the message blocks m_1 and m_2.
Aumasson et. al. conclude that it is not possible to start in a state without differences and end in $(\Delta, 0, \Delta', 0)$ after applying $\mathcal{G}$. This conclusion is not clearly explained by them. Further we will show that it is possible to have this transformation, but there are heavy restrictions on the input values.
First we take a look at our goal. Using Δm_1 and Δm_2 in $\mathcal{G}$ we find

$$\begin{aligned}
\Delta a^* &= \Delta m_1 && (4.1)\\
\Delta d^* &= \Delta m_1 \ggg_{16} && (4.2)\\
\Delta c^* &= \Delta m_1 \ggg_{16} && (4.3)\\
\Delta b^* &= \Delta m_1 \ggg_{28} && (4.4)\\
\Delta a' &= \Delta m_1 + \Delta m_1 \ggg_{28} + \Delta m_2 && (4.5)\\
\Delta d' &= (\Delta m_1 \ggg_{16} \oplus (\Delta m_1 + \Delta m_1 \ggg_{28} + \Delta m_2)) \ggg_8 . && (4.6)
\end{aligned}$$

To get to $\Delta d' = 0$ we need $\Delta m_1 \ggg_{16} \oplus (\Delta m_1 + \Delta m_1 \ggg_{28} + \Delta m_2) = 0$. This happens if

$$\Delta m_1 \ggg_{16} = (\Delta m_1 + \Delta m_1 \ggg_{28} + \Delta m_2).$$

This is our first assumption, thus we continue with $\Delta d' = 0$.

$$\begin{aligned}
\Delta c' &= \Delta m_1 \ggg_{16} && (4.7)\\
\Delta b' &= (\Delta m_1 \ggg_{28} \oplus \Delta m_1 \ggg_{16}) \ggg_7 . && (4.8)
\end{aligned}$$

To get to $\Delta b' = 0$ we need $\Delta m_1 \ggg_{28} \oplus \Delta m_1 \ggg_{16} = 0$. This happens if

$$\Delta m_1 \ggg_{28} = \Delta m_1 \ggg_{16} .$$

This is our second assumption.
It is easy to see that if a pair $(\Delta m_1, \Delta m_2)$ satisfies $\Delta m_1 \ggg_4 = \Delta m_1$ and $\Delta m_2 + \Delta m_1 = 0$ it will also satisfy the assumptions above.
Note that when a pair satisfies the two assumptions we might find an output where $\Delta b'$ or $\Delta d'$ are not zero. For example, this will happen when the difference of $a + m_1$ and $a + m_1'$ is not Δm_1. We will use this to estimate the success probability of each path.

For all the following lemma's in this paragraph, we use the following assumption. **Let $r \in \{0,1\}^4$ be random, let $m_1 - -m_1' = \Delta m_1 = r||\ldots||r$ and $m_2 - -m_2' = \Delta m_2 = 2^{32} - -\Delta m_1$.**

Lemma 3. *If all of the input values (a,b,c,d) are zero, then:*

$$\mathcal{G}_{m_1,m_2}(a,b,c,d) - -\mathcal{G}_{m_1',m_2'}(a,b,c,d) = (\Delta, 0, \Delta', 0)$$

with probability 1.

Proof.

$$\begin{array}{rcll}
\Delta a^* & = & \Delta m_1 & \text{since there is no carry due to } a = 0. \\
\Delta d^* & = & \Delta m_1 \ggg_{16} . & \\
\Delta c^* & = & \Delta m_1 \ggg_{16} & \text{since } c = 0. \\
\Delta b^* & = & \Delta m_1 \ggg_{28} . & \\
\Delta a' & = & \Delta m_1 + \Delta m_1 \ggg_{28} + \Delta m_2 = \Delta m_1 & \text{since } \Delta m_2 = 2^{32} - -\Delta m_1 \text{ and} \\
 & & & \Delta m_1 = \Delta m_1 \ggg_4 . \\
\Delta d' & = & (\Delta m_1 \ggg_{16} \oplus \Delta m_1) \ggg_8 = 0 & \text{since } \Delta m_1 = \Delta m_1 \ggg_4 . \\
\Delta c' & = & \Delta m_1 \ggg_{16} & \text{since } \Delta d' = 0. \\
\Delta b' & = & (\Delta m_1 \ggg_{28} \oplus \Delta m_1 \ggg_{16}) \ggg_7 = 0 & \text{since } \Delta m_1 = \Delta m_1 \ggg_4 .
\end{array}$$

So we have not more restrictions and we find probability 1. □

Lemma 4. *If precisely three of the four input values (a, b, c, d) are zero, and the fourth is a power of 2, then*

$$\mathcal{G}_{m_1,m_2}(a, b, c, d) - -\mathcal{G}_{m'_1,m'_2}(a, b, c, d) = (\Delta, 0, \Delta', 0)$$

with probability $1/2$.

Proof. We find

$$\mathcal{G}_{m_1,m_2}(a, b, c, d) - -\mathcal{G}_{m'_1,m'_2}(a, b, c, d) = (\Delta, 0, \Delta', 0)$$

if and only if Equations 4.1 to 4.8 hold.
Let $(a, b, c, d) = (2^j, 0, 0, 0)$, in total we have 32 different options, but there are only four options which give different restrictions. Since m_1 is rotatable over four, $j = 0$ and $j = 4$ will give the same restrictions.
We state that if $\Delta m_{1_{j \bmod 4}} = 0$ Equations 4.1 to 4.8 hold.

$$\begin{array}{rcll}
\Delta a^* & = & \Delta m_1 & \text{since there is no carry}^5 \text{ due to } \Delta m_{1_{j \bmod 4}} = 0. \\
\Delta d^* & = & \Delta m_1 \ggg_{16} . & \\
\Delta c^* & = & \Delta m_1 \ggg_{16} & \text{since } c = 0. \\
\Delta b^* & = & \Delta m_1 \ggg_{28} . & \\
\Delta a' & = & \Delta m_1 + \Delta m_1 \ggg_{28} + \Delta m_2 = \Delta m_1 & \text{since } \Delta m_2 = 2^{32} - -\Delta m_1 \text{ and} \\
 & & & \Delta m_1 = \Delta m_1 \ggg_4 . \\
\Delta d' & = & (\Delta m_1 \ggg_{16} \oplus \Delta m_1) \ggg_8 = 0 & \text{since } \Delta m_1 = \Delta m_1 \ggg_4 . \\
\Delta c' & = & \Delta m_1 \ggg_{16} & \text{since } \Delta d' = 0. \\
\Delta b' & = & (\Delta m_1 \ggg_{28} \oplus \Delta m_1 \ggg_{16}) \ggg_7 = 0 & \text{since } \Delta m_1 = \Delta m_1 \ggg_4 .
\end{array}$$

We conclude that we find $\Delta m_{1_{j \bmod 4}} = 0$ with probability $1/2$, and thus if $a = 2^j$ we find

$$\mathcal{G}_{m_1,m_2}(a, b, c, d) - -\mathcal{G}_{m'_1,m'_2}(a, b, c, d) = (\Delta, 0, \Delta', 0)$$

with probability $1/2$.
The cases were b, c or d are not zero can be proved similarly, this part can be found in Appendix C □

Lemma 5. *If precisely two of the four input values (a, b, c, d) are zero, and the other two are a power of 2, then*

$$\mathcal{G}_{m_1,m_2}(a, b, c, d) - -\mathcal{G}_{m'_1,m'_2}(a, b, c, d) = (\Delta, 0, \Delta', 0)$$

with probability at least $1/4$.

Proof. There are 6 choices for the two nonzero input values, again we will only discuss one case here, the others can be found in Appendix C.
We look at the case that a and b are the nonzero inputs. For $(a,b,c,d) = (2^i, 2^j, 0, 0)$ we distinguish two cases, $i = j$ and $i \neq j$.

Case: $i = j$ We state that Equation 4.1 to 4.8 hold if $(\Delta m_1)_i = (\Delta m_1)_{i+1} = 0$ for $a = b = 2^i$.

$$\begin{aligned}
\Delta a^* &= (a_1 + b_1 + m_1) \oplus (a_2 + b_2 + m_1') \\
&= (2^{i+1} + m_1) \oplus (2^{i+1} + m_1') \\
&= (2^{i+1} \oplus m_1) \oplus (2^{i+1} \oplus m_1') \qquad \text{if } (\Delta m_1)_{i+1} = 0 \\
&= \Delta m_1 \\
\Delta d^* &= a_1^* \ggg_{16} \oplus a_2^* \ggg_{16} \\
&= (\Delta m_1) \ggg_{16} \\
&= \Delta m_1 \\
\Delta c^* &= d_1^* \oplus d_2^* \\
&= \Delta m_1 \\
\Delta b^* &= (b_1 \oplus c_1^*) \ggg 12 \oplus (b_2 \oplus c_2^*) \ggg 12 \\
&= (b_1 \oplus c_1^* \oplus b_2 \oplus c_2^*) \ggg 12 \\
&= (m_1 \oplus m_1') \ggg_{12} \\
&= \Delta m_1 \\
\Delta a' &= (a_1^* + b_1^* + m_2) \oplus (a_2^* + b_2^* + m_2') \\
&= \big((2^{i+1} \oplus m_1) + (2^{i-12} \oplus 2^{i-29} \oplus m_1) + m_2\big) \oplus \big((2^{i+1} \oplus m_1') + (2^{i-12} \oplus 2^{i-29} \oplus m_1') + m_2'\big) \\
&= \big((2^{i+1} \oplus 2^{i-12} \oplus 2^{i-29}) + (2m_1 + m_2)\big) \oplus \big((2^{i+1} \oplus 2^{i-12} \oplus 2^{i-29}) + (2m_1' + m_2')\big) \\
&= \big((2^{i+1} \oplus 2^{i-12} \oplus 2^{i-29}) \oplus m_1\big) \oplus \big((2^{i+1} \oplus 2^{i-12} \oplus 2^{i-29}) \oplus 2m_1'\big) \qquad \text{if } (\Delta m_1)_i = 0 \\
&= \Delta m_1 \\
\Delta d' &= (d_1^* \oplus a_1') \ggg_8 \oplus (d_2^* \oplus a_2') \ggg_8 \\
&= (\Delta d^* \oplus \Delta a') \ggg_8 \\
&= (\Delta m_1 \oplus \Delta m_1) \ggg_8 \\
&= 0 \\
\Delta c' &= (c_1^* + d_1') \oplus (c_2^* + d_2') \\
&= \big((2^{i-17} \oplus m_1) + (2^{i+1} \oplus 2^{i-12} \oplus 2^{i-17} \oplus 2^{i-29})\big) \oplus \big((2^{i-17} \oplus m_1') + (2^{i+1} \oplus 2^{i-12} \oplus 2^{i-17} \oplus 2^{i-
\\
&= \big(2^{i-16} \oplus m_1 \oplus 2^{i+1} \oplus 2^{i-12} \oplus 2^{i-29}\big) \oplus \big(2^{i-16} \oplus m_1' \oplus 2^{i+1} \oplus 2^{i-12} \oplus 2^{i-29}\big) \\
&= \Delta m_1 \\
\Delta b' &= (b_1^* \oplus c_1') \ggg_7 \oplus (b_2^* \oplus c_2') \ggg_7 \\
&= \big(b_1^* \oplus b_2^* \oplus c_1' \oplus c_2'\big) \ggg_7 \\
&= (\Delta m_1 \oplus \Delta m_1) \ggg_7 \\
&= 0.
\end{aligned}$$

We conclude that if $(\Delta m_1)_i = (\Delta m_1)_{i+1} = 0$ the equations are satisfied. This happens with probability $1/4$.

Case: $i \neq j$ The proof that $1/4$ is a lower bound is quite similar as the proof for $i = j$ and so are the proofs for all five other couples of (a,b,c,d). All this can be found in Appendix C. □

Lemma 5 states that $1/4$ is a lower bound of the success probability of this path. In Appendix C it is shown that there are couples $a = 2^i, b = 2^i$ such that the success probability is $1/2$.

Lemma 6. *If precisely one of the four input values (a,b,c,d) are zero, and the other three are a power of*

2, then

$$\mathcal{G}_{m_1,m_2}(a,b,c,d) - -\mathcal{G}_{m_1',m_2'}(a,b,c,d) = (\Delta, 0, \Delta', 0)$$

with probability at least $1/8$.

The proof is given in Appendix C, there we also give some examples.

Lemma 7. *If none of the four input values* (a,b,c,d) *is zero, and all of them are a power of 2, then*

$$\mathcal{G}_{m_1,m_2}(a,b,c,d) - -\mathcal{G}_{m_1',m_2'}(a,b,c,d) = (\Delta, 0, \Delta', 0)$$

with probability at least $1/16$.

This proof is also given in Appendix C. There are particular cases that this probability is even $1/2$, some examples are given in the appendix.

Non–zero differences in input state Now that we have concluded that there are eight possible states to end after one $\mathcal{G}$ starting in a state without differences:

$$(\Delta_0, 0, \Delta_1, 0), (0, \Delta_0, 0, \Delta_1), (0, 0, \Delta_0, \Delta_1), (\Delta_0, \Delta_1, \Delta_2, 0),$$

$$(\Delta_0, \Delta_1, 0, \Delta_2), (\Delta_0, 0, \Delta_1, \Delta_2), (0, \Delta_0, \Delta_1, \Delta_2), (\Delta_0, \Delta_1, \Delta_2, \Delta_3)$$

For an internal collision we want to start without differences, apply one round and end without differences. That is, we apply $\mathcal{G}$ twice. So the goal is to go from any of the possible non–zero difference states to a zero difference state.

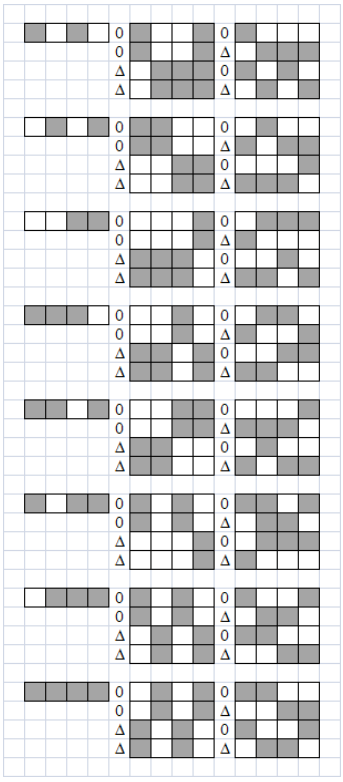

Figure 4.2: Options if all differences can cancel each other.

Figure 4.2 shows the progress of differences in the ideal case, i.e. each difference can cancel any other, under XOR and modular addition.

- The first block represents the start state, the white blocks represent no differences and the grey blocks represent a difference. The exact difference of the grey blocks is not known.
- The second block represents the possible difference states after the column step of $\mathcal{G}$ with either a difference in the message blocks (Δ) or no difference (0).
- The third block represents the possible difference states after the diagonal step of $\mathcal{G}$ with either a difference in the message blocks (Δ) or no difference (0).

What we see is that none of these go to a state without differences. What we can reach, are states that are easily converted to a zero difference state. For example, $(\Delta, 0, 0, 0)$ only needs one message block difference equal to Δ to reach a zero difference state. But in a round $\mathcal{G}$ is applied twice to each state. After one round, the same message blocks are used but in a different sequence due to the permutation σ, if it is possible to remove the last difference, then in the next step we will add differences again.

Collisions for a sequence of $\mathcal{G}$ When we start with two all zero states, and apply $\mathcal{G}$ three times we can get a collision. First we use the message blocks to get to state difference $(\Delta, 0, \Delta, 0)$. Then we want to reach intermediate state difference $(\Delta', 0, 0, \Delta'')$ which easily converts to $(\Delta''', 0, 0, 0)$. The next step we choose $\Delta m = 2^{32} - -\Delta'''$ and we reach intermediate state difference (0,0,0,0), which stays without differences if the message block is zero.
If the start states are not zero, but are equal, we find probabilities as for the first step, that is starting with $a = 2^i$ gives probability $1/2$, with $a = 2^i, b = 2^j$ gives $1/4$ etcetera.

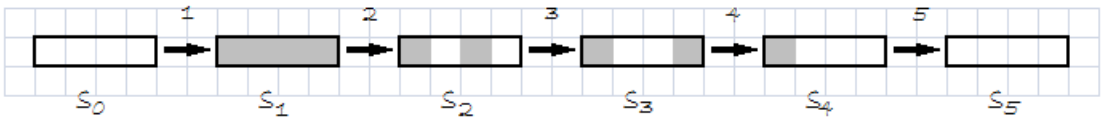

Figure 4.3: The first five message blocks are nonzero.

The states are represented by $s_i^j = (a_i^j, b_i^j, c_i^j, d_i^j)$ for $i = 0, \ldots, 5$ and $j = 1, 2$ denotes the first or the second state. The state s_i^j is mapped by a column step or diagonal step of $\mathcal{G}$ to s_{i+1}^j. The difference between the states is meant when j is omitted
We assume that $m_i^2 = 0$ for all i and thus the difference m_i is equal to m_i^1. These are given by:

$$
\begin{aligned}
m_1 &= r \cdot 2^0 + r \cdot 2^4 + \ldots + r \cdot 2^{24} + r \cdot 2^{28} && for \quad r \in 0, 15, \\
m_2 &= 2^{32} - -m_1, \\
m_3 &= \left(\left(c_2^2 + \left(d_2^2 \oplus (a_2^2 + b_2^2)\right) \ggg_{16} - - c_2^1\right) \lll_{16} \oplus d_2^1\right) - -a_2^1 - -b_2^1, \\
m_4 &= \left(d_3^1 \oplus d_3^2 \oplus (a_3^2 + b_3^2)\right) - -a_3^1 - -b_3^1, \\
m_5 &= a_4^2 - -a_4^1.
\end{aligned}
$$

When the start states (a, b, c, d) are restricted as in paragraph 4.4.2 this path has the same success probability as the first step, i.e. for one nonzero block which is of the form 2^j we find $(0, 0, 0, 0)$ with probability $1/2$.
This does not automatically work for the definition of the round function, as this consists of 1.5 rounds, the block m_5 must be equal to some block in the first round, the permutation gives which block this is. And therefore, the next block (m_6) can not be zero, and will thus introduce

a new difference in the states after 2 rounds.
Even if we would look at three rounds, which consists of 6 half rounds and thus we could apply the above twice, we would still have the restrictions $m_5 = m_1$ and $m_6 = m_2$ etcetera. This complicates the attack in the original function.

On the full round function $\mathcal{R}$

Assume no differences introduced in the column step[6] of $\mathcal{G}$, then the introduction of a difference in the diagonal step will distribute to at least to words of the internal value, by the previous result.
Now assume there are differences introduced in the column step, to make those differences disappear we must have $\Delta b = \Delta c = 0$. Thus after the column step the differences must be of the form $(\Delta, 0, 0, \Delta')$, but the results in the previous paragraph state that $\Delta b' = 0 \Rightarrow \Delta c' \neq 0$. So there will be no two values mapped to the same output, thus the round function must be unique, and so injective.

4.4.3 Fixed points of $\mathcal{G}$

The authors of BLAKE state that there is only one fixed point for the linear approximation of $\mathcal{G}$ with $m \oplus c$ always zero, and that is the all zero state [43, P. 31]. We denote a fixed point with $m \oplus c = 0$ with 0–fixed point. We denote the linear approximation of $\mathcal{G}$ with $\bar{\mathcal{G}}$. For a 0–fixed point (a, b, c, d) we should have $\bar{\mathcal{G}}(a, b, c, d) = (a, b, c, d)$. Thus the authors state that it is not possible to satisfy the following equations:

$$\begin{aligned} a^* &= a \oplus b \\ d^* &= (d \oplus a^*) \ggg_{16} \\ c^* &= c \oplus d^* \\ b^* &= (b \oplus c^*) \ggg_{12} \\ a &= a^* \oplus b^* \\ d &= (d^* \oplus a) \ggg_{8} \\ c &= c^* \oplus d \\ b &= (b^* \oplus c) \ggg_{7} \end{aligned}$$

with any other 4–tuple than $(0, 0, 0, 0)$.
And therefore they conclude that it is not possible to obtain preservation of differences by the linearization. If it would be possible to preserve differences, then this difference would be a 0–fixed point of $\bar{\mathcal{G}}$.

Fixed points of the linearization of $\mathcal{G}$

We state that for each pair (b, b^*) there exists a fixed point of $\bar{\mathcal{G}}$. The fixed point can be found using the following algorithm:

[6] first half

Algorithm: FIXED POINT OF $\bar{\mathcal{G}}$

Randomly select b, b^*;

$$\begin{aligned} c &= (b \lll_7) \oplus b^* \\ c^* &= b \oplus (b^* \lll_{12}) \\ d &= c \oplus c^* \\ d^* &= c \oplus c^* \\ a &= d(\lll_8) \oplus d^* \\ a^* &= d \oplus (d^* \lll_{16}) \\ m_1 &= a^* \oplus a \oplus b \\ m_2 &= a^* \oplus a \oplus b^* \end{aligned}$$

return (a, b, c, d, m_1, m_2)

The construction of this Algorithm is described in Appendix C.
Now with the use of the fixed points we can construct couples of inputs for which the differences are preserved under $\bar{\mathcal{G}}$. Some examples of fixed points and difference preserving couples are given in Appendix C.

Fixed points of $\mathcal{G}$

As for the linearization of $\mathcal{G}$, we give an algorithm to find fixed points for a couple (b, b^*). Now in this situation it is not trivial to find difference preserving states, as $\mathcal{G}$ is not linear and therefore

$$\mathcal{G}_{m_1,m_2}[a \oplus \delta_1, b \oplus \delta_2, c \oplus \delta_3, d \oplus \delta_4] \oplus \mathcal{G}_{m_1,m_2}[a, b, c, d] = (\delta_1, \delta_2, \delta_3, \delta_4)$$

almost never holds. Obviously this does hold for the trivial couples of any fixed point and the all zero input.
A fixed point can be found using the following algorithm:

Algorithm: FIXED POINT OF $\mathcal{G}$

Randomly select b, b^*;

$$\begin{aligned} c &= (b \lll_7) \oplus b^* \\ c^* &= b \oplus (b^* \lll_{12}) \\ d &= c - -c^* \mod 2^{32} \\ d^* &= c^* - -c \mod 2^{32} \\ a &= (d \lll_8) \oplus d^* \\ a^* &= d \oplus (d^* \ggg_{16}) \\ m_1 &= a^* - -a - -b \mod 2^{32} \\ m_2 &= a - -a^* - -b^* \mod 2^{32} \end{aligned}$$

return (a, b, c, d, m_1, m_2)

the construction is given in Appendix C.

4.4.4 On one round

If the input for a round consists of four equal states (a, b, c, d) for which the message blocks that preserve the state are known, then we can preserve the state of one round. For example, in the first round, the permutation σ is the identity, we use the fixed point (a, b, c, d) which is fixed

under the message blocks m_1, m_2. So we set all even numbered blocks $m_{2i} = m_1$ and all odd numbered blocks $m_{2i+1} = m_2$. We set all input states $v_0, \ldots, v_{15}$ as $a, b, c, d, \ldots, a, b, c, d$. Now applying the column step of the round function will preserve the states, then the input for the diagonal step is equal to the input. Therefore, as we use the message blocks m_1, m_2 we will end in the same state. Thus we find a fixed point for the round function.
In the second round, σ_1 is used, and then four message blocks which should be inserted in the column step, will be inserted in the diagonal step. This could be obtained by finding a fixed point with $m_1 = m_2$. Now our algorithm is defined to choose b and b^*, we can not start with choosing m_1 and m_2. Therefore, we ran a program to find such message blocks, but the program did not give any result.

4.4.5 On the compression function

The authors state that a fixed point for the compression function of BLAKE–256 can be found with effort approximately 2^{192} instead of 2^{256} ideally. For this they use that one can compute h such that:

$$\begin{aligned}
h_0 &\leftarrow h_0 \oplus s_0 \oplus v_0 \oplus v_8 \\
h_1 &\leftarrow h_1 \oplus s_1 \oplus v_1 \oplus v_9 \\
h_2 &\leftarrow h_2 \oplus s_2 \oplus v_2 \oplus v_{10} \\
h_3 &\leftarrow h_3 \oplus s_3 \oplus v_3 \oplus v_{11} \\
h_4 &\leftarrow h_4 \oplus s_0 \oplus v_4 \oplus v_{12} \\
h_5 &\leftarrow h_5 \oplus s_1 \oplus v_5 \oplus v_{13} \\
h_6 &\leftarrow h_6 \oplus s_2 \oplus v_6 \oplus v_{14} \\
h_7 &\leftarrow h_7 \oplus s_3 \oplus v_7 \oplus v_{15}.
\end{aligned}$$

If we consider the case where no salt is used ($s = 0$), then we need to find v such that $v_0 = v_8, v_1 = v_9, \ldots, v_7 = v_{15}$, which gives 2^{256} possible matrices. We also need that the third line is (c_0, c_1, c_2, c_3) since $s = 0$ and since we use t_0 and t_1 twice, we need $v_{12} = v_{13} \oplus c_4 \oplus c_5$ and $v_{14} = v_{15} \oplus c_6 \oplus c_7$. So if we compute the round function backward, we find a fixed point with effort

$$2^{4 \cdot 32} \cdot 2^{2 \cdot 32} = 2^{128} 2^{64} = 2^{192}.$$

This method does not provide a distinguisher, since for a distinguisher we should approach the compression function as a blackbox, and here we use the internal mechanics of the compression function.

4.4.6 Bounds on the probability of DC's for BLAKE

Aumasson et. al. also worked on the bounds on the probability of differential characteristics (DC's) for ARX algorithms [44]. They use the notations of Lipmaa and Moriai [15], we will summarize all the notations we reuse.
The differential probability of addition modulo 2^n is defined as

$$DP^+[\delta] = DP^+[\alpha, \beta \rightarrow \gamma] = \Pr_{x,y}[(x + y) \oplus ((x \oplus \alpha) + (y \oplus \beta)) = \gamma].$$

Given the differences α and β we define the maximum differential probability over all γ as

$$DP^{+}_{\max}[\alpha, \beta] = \max_{\gamma}(DP^{+}[\alpha, \beta \to \gamma]).$$

Given only β we define the maximum differential probability over all α and γ as

$$DP^{+}_{2\max}[\alpha] = \max_{\beta,\gamma}(DP^{+}[\alpha, \beta \to \gamma]).$$

Lipmaa and Moriai give efficient algorithms to compute these three values.
Now we use this to find bounds for differential characteristics of BLAKE. For this we estimate the maximum probabilities that modular addition can be replaced by XOR.
First, if only one of the terms has a difference.
For each Δ under one addition we find

$$DP^{+}[\Delta, 0 \to \Delta] = DP^{+}_{\max}[0, \Delta] = 2^{--||\Delta||}.$$

It is easy to see that it is possible to achieve $\gamma = \Delta$. That this has probability is $2^{--||\Delta||}$ can be concluded since modular addition can be replaced by XOR if there are no carry's, but this does not hold for the most significant bit (MSB), so for now we let the MSB of all differences be zero[7]. That linearization of the modular addition is optimal follows from the results of Lipmaa and Moriai [15], we follow Algorithm 3 and we see that $\gamma = \Delta$ attains the maximum.

On one $\mathcal{G}$

In one $\mathcal{G}$ there are only two message blocks inserted, m_i, m_j. We can distinguish the following four cases for the possible differences in the message blocks. We consider the original $\mathcal{G}$.

- $\Delta_i \neq 0 \wedge \Delta_j = 0$ The difference is introduced in the first step, preferably the difference is canceled in this step, since otherwise it would proceed to step three which will decrease the probability. Therefore, we choose $\Delta a = \Delta m_i$, now we have two active additions, $a + b$ and $(a + b) + m_i$. Therefore we find highest probability $2^{--2||\Delta_i||}$.
- $\Delta_i = 0 \wedge \Delta_j \neq 0$ The differences are introduced in the second half of $\mathcal{G}$, we want to cancel these differences. So we need that Δa^* cancels Δ_j. But the difference in a should not spread to the other variables, since a difference proceeding through step 3 will decrease the probability. Therefore we choose $\Delta a = \Delta d = \Delta_j$. There are 4 active additions, and the highest probability is $2^{--4||\Delta_i||}$.
- $\Delta_i = \Delta_j \neq 0$ The highest probability is achieved when $\Delta d = \Delta_i$. Now in the first step we only have one active addition since $a + b$ is computed first, without active addition and then $(a + b) + m_i$ which is the first active addition. Then in the second half of $\mathcal{G}$ the computation of a contains two active additions. In total three active additions and therefore we find highest probability $2^{--3||\Delta_i||}$.
- $0 \neq \Delta_i \neq \Delta_j \neq 0$ Aumasson et. al. state that, for the original $\mathcal{G}$, if $\Delta d = \Delta_i$ the success probability is
$$2^{--2||\Delta_i||--||\alpha||} \cdot DP^{+}_{\max}[\Delta_i, \Delta_j]$$
where α is the difference the maximizes $DP^{+}_{\max}[\Delta_i, \Delta_j]$.
Since in step 1 we use $DP^{+}_{\max}[0, \Delta_i \to \Delta_i] = 2^{--||\Delta_i||}$, in step 2 the difference will vanish, so

[7] If the MSB of Δ is 1, we replace $2^{--||\Delta||}$ with $2^{--||\Delta||+1}$

the next active addition is found in step 5. Here we first use $DP^{+}[\Delta_i, 0 \rightarrow \Delta_i] = 2^{--||\Delta_i||}$ for $(a+b)$ and then $DP^{+}[\Delta_i, \Delta_j \rightarrow \alpha] = DP^{+}_{\max}[\Delta_i, \Delta_j]$ for $(a+b)+m_j$. Then in step 7 we use $DP^{+}_{\max}[0, \alpha \ggg_8 \rightarrow \alpha \ggg_8] = 2^{--||\alpha \ggg_8||} = 2^{--||\alpha||}$.
With this method we end in a state where $\Delta a \neq 0, \Delta b \neq 0, \Delta c \neq 0, \Delta d \neq 0$ unless we are in a special case[8]. The highest probability to end in a zero difference state (which also was the case in the above three cases), is achieved with $\Delta a = \Delta m_j - -\Delta_i$ and $\Delta d = \Delta_j$. Then we have four active additions:

$$\begin{aligned} a &= (a+b) + m_i \\ a &= (a+b) + m_j \end{aligned}$$

for two of these there are two elements with differences, since both a and the message blocks contain a difference. The additions in step 3 and 7 are not active since the differences are canceled in step 2 and 5. Now we need

$$DP^{+}[\Delta_j - -\Delta_i, \Delta_i \rightarrow \Delta_j] = 2^{--||\Delta_i||--||\Delta_j--\Delta_i||}.$$

The proof can be found in Appendix C.
Thus we find highest success probability $2^{--||\Delta_i||--2||\Delta_j--\Delta_i||--3||\Delta_j||}$.

On one round

Each round we apply $\mathcal{G}$ eight times. First we will discuss the column step, then the diagonal step. Note that r represents the round, $r = 0, \ldots, 9$.

Column Step In the column step we use eight message blocks for the four applications of $\mathcal{G}$. For each time $\mathcal{G}$ is applied we can bound the success probability using the results of Section 4.4.6. So for $\mathcal{G}_i$, where $i = 0, \ldots, 3$, applied to the states using message blocks $m_{\sigma_r(2i)}, m_{\sigma_r(2i+1)}$, we find the bound bc_i for the success probability:

$$bc_i = \begin{cases} 2^{--2||\Delta_{\sigma_r(2i)}||} & \text{if} \quad ||\Delta_{\sigma_r(2i+1)}|| = 0 \\ 2^{--4||\Delta_{\sigma_r(2i+1)}||} & \text{if} \quad ||\Delta_{\sigma_r(2i)}|| = 0 \\ 2^{--2||\Delta_{\sigma_r(2i)}||} DP^{+}_{\max}[\sigma_r(2i), \sigma_r(2i+1)] & \text{else.} \end{cases}$$

Note that we have used the following two properties to estimate the bound when both message blocks have a difference

$$\begin{aligned} \Delta_i = \Delta_j &\Rightarrow DP^{+}_{\max}[\Delta_i, \Delta_j] = 2^{--||\Delta_i||} \\ \Delta_i \neq \Delta_j &\Rightarrow 2^{--||\alpha||} DP^{+}_{\max}[\Delta_i, \Delta_j] \leq DP^{+}_{\max}[\Delta_i, \Delta_j]. \end{aligned}$$

Diagonal Step In the diagonal step we use the other eight message blocks for the four applications of $\mathcal{G}$. For each time $\mathcal{G}$ is applied we can bound the success probability using the results of Section 4.4.6. So for $\mathcal{G}_i$, where $i = 4, \ldots, 7$, applied to the states using message blocks

[8] For example when the difference in a is canceled, i.e. $\Delta_i = 2^{32} - -\Delta_j$.

$m_{\sigma_r(2i)}, m_{\sigma_r(2i+1)}$, we find the bound bd_i for the success probability:

$$bd_i = \begin{cases} 2^{--2||\Delta_{\sigma_r(2i)}||} & \text{if} \quad ||\Delta_{\sigma_r(2i+1)}|| = 0 \\ 2^{--4||\Delta_{\sigma_r(2i+1)}||} & \text{if} \quad ||\Delta_{\sigma_r(2i)}|| = 0 \\ 2^{--2||\Delta_{\sigma_r(2i)}||} DP^{+}_{\max}[\sigma_r(2i), \sigma_r(2i+1)] & \text{else.} \end{cases}$$

Product Now that we have a bound for the column step and for the diagonal step we can combine this to find a bound $b(1)$ for one round.

$$b(1) = bc_i \cdot bd_i$$

and this can be extended to r rounds:

$$b(r) = \prod_{i=0}^{r} bc_i \cdot bd_i.$$

This bound is arguably loose, in example, we assumed that the additions in step 3 and 7 are never active and that the input values always have ideal differences, which both might be rare events.

4.5 Conclusion

The analysis on BLAKE so far does not indicate that this hash function is not secure. There are some assumptions made on BLAKE (by its authors), which are shown to be incorrect in this thesis. Neither one of these changes threatens the security of the full function at this time.
For example, for the fact that the state $(\Delta, 0, \Delta', 0)$ can be reached after one $\mathcal{G}$ it is proven that it will not lead to an attack on the full function. Secondly, the fixed points do not give a distinguisher, so they can not (yet) be used to attack BLAKE.
The simplicity of the design makes it easier to analyze. Therefore the explanations that an attack can not be extended to more rounds, or the full function, are reasonable to be right.

Chapter 5

Comparison between Blake and Shabal

5.1 Hardware requirements & speed

Since our implementations are not suitable for comparing amounts of ROM and RAM, this sections is based on the round reviews of NIST [53, 26].
Both Shabal and BLAKE are among the best performers of the SHA-3 contest concerning the hashing of long messages. For very short messages BLAKE performs the best of all contenders, Shabal is not as good as the others in this respect.
Concerning speed, Shabal is among the top performers, and in constrained environments, BLAKE is too. BLAKE–512, which produces the 512–bit digest, is optimized for 64–bit processors and performs significantly worse on a 32–bit processor.
Both Shabal and BLAKE require low amounts of ROM and provide average performance in throughput–to–area ratio. However, BLAKE has a structure that allows for flexible designs.

5.2 Security

Both Shabal and BLAKE were subjected to different types of cryptanalyses. We will first summarize some of the differences and agreements. Afterwards we will discuss whether the cryptanalysis can be compared.

- BLAKE is an ARX algorithm[1] and Shabal also contains multiplication by 3 ($\mathcal{U}$), multiplication by 5 ($\mathcal{V}$) and the AND operator.
- Shabal is a so–called wide–pipe algorithm, BLAKE is a narrow-pipe algorithm. The difference is that for Shabal the output is a small part of the internal state, and for BLAKE the output the internal state compressed to half the size, thus the latter contains all the information.
- Both algorithms are based on a permutation, the permutation of Shabal is keyed with a message block and part of the inner state, the permutation of BLAKE is keyed with two message blocks.
- Both Shabal and BLAKE can be reduced to less complex versions, which makes it easy to see if a property is very important for the security of the function.

[1] ARX algorithm means that the algorithm consists only of the operations: Addition, Rotation and eXclusive OR

Indifferentiability The most recent proof of the authors of Shabal takes into account that the permutation is biased. We do not totally understand the proof, but let us assume it is correct. Then it does show that Shabal is secure, with the differentiators that have been proven to exist, until now. Thus, the permutation has a large responsibility, and during the last years more and more irregularities were found, so the question is, if there will be better differentiators in the future.
Indifferentiability has not yet proven to be useful for hash functions, and proofs are always based on many assumptions on the inner primitive.
On the other hand, differentiators are making their way into cryptanalysis of hash functions. Mainly because the irregularities might be a start for another attack (e.g. a preimage attack).
For Shabal there exist distinguishers on the inner primitive, the permutation, but these have not yet led to an attack.
The authors of BLAKE state that the counter is important, since it assures to consecutive compressions differ. Therefore the inputs of the iteration mode are prefix–free and this guarantees that BLAKE is indifferentiable from a random oracle as long as the compression function is assumed ideal [8]. The conclusion is that if BLAKE is in some way not secure, it must be due to the compression function.

Fixed Points For both the inner primitive of Shabal and BLAKE, the permutation respectively the round function, there are fixed points that are easily generated. In both cases this does not apply for the full function, as the starting state may not be chosen.
The authors of Shabal claim that there exist no trivial fixed points for the compression fucntion due to the counter.
The authors of BLAKE state that fixed points for the compression function can be found with effort 2^{192} instead of 2^{256} ideally, but this does not lead to an attack.

Preimages For both functions it is possible to find a preimage of the inner primitive, since both are a permutation and they do not truncate information. To extend this to the full function makes both problems infeasible. For Shabal this is due to the truncation, the output is only the C part of the inner state after the three final rounds. In a reduced version it is possible to find the B and the C part, but still the A part is unknown and this is guessed correctly with probability 2^{-384}.
In BLAKE the digest is a compressed version of the inner state. A correct guess, such that a preimage can be computed, occurs with probability 2^{-512} (since the previous chaining variable and the inner state are unknown), assuming that the salt is known to the attacker (otherwise the probability decreases to $2^{-512-4\cdot 32} = 2^{-640}$).

Length Extension Attack For both functions the counter is a sufficient method to protect the function against length extension attacks.

NIST NIST decided to progress BLAKE to the final round of the competition, due to its simplicity, its speed, and its security margin.
Shabal did not progress to the final round mainly due to security concerns, the differentiators with low complexity, that have been found for the permutation, raised concerns. The latest proof of indifferentiability did not convince NIST that the function would remain secure.

Opinion NIST chose five functions out of the fourteen second round candidates to advance to the next round. We only analyzed two of the second round candidates and we state that if we needed to choose between those two, Shabal and BLAKE, we would choose BLAKE.
Shabal's inner primitive, the permutation has been attacked many times, mostly by distinguishers, but there are also many fixed points. The security of Shabal depends highly on the counter, the initial values and the final rounds. Although every functions gains some security from the initial values and possibly from the counter, the compression function itself should provide more security than the compression function of Shabal does.
The inner primitive of BLAKE is simple compared to the inner primitive of Shabal and therefore it is less expected to have properties that have not proven to bring down the security margin, will be found in the near future.
For BLAKE the cryptanalysis is still advancing, while the analysis still concludes that BLAKE is very strong against differential attacks, the latest publication shows that there might be a reason to believe that low probability differentials not necessarily imply a high security margin against differential attacks. This does not threaten the security of BLAKE but it might raise concerns. On the other side, this is not only relevant for BLAKE, also for the final round candidate Skein.
There are more questions about the security of BLAKE now, than one year ago. Many cryptanalysts are focussing on the five remaining contenders, none of which has been broken or is computationally broken. This implies that all five will remain secure at least a little while longer. Never before has there been so much analysis on a hash function that is not yet in use. This induces confidence in whichever function will become the new world wide standard.

Acknowledgements

First I would like to say that I really enjoyed working on this project, the hardest part was to stop working on the project and start writing down what I did do. So along the way, I had a lot of support from my supervisor Benne in this respect. On the contents of my thesis I had some nice and very useful discussions with him as well.
Thanks Benne, for all the help when I was in Australia and in particular when I was back in the Netherlands.
I really enjoyed the fact that I could work on this project at Macquarie University in Sydney. It was nice to see that also there, they have an informal structure as we do in Eindhoven. I enjoyed the weekly meetings. It was nice to share my ideas and get comments. Thanks for that, Josef, Ron, Przemek, Sareh and Malin. Also, Sareh and Przemek, I would like to thank you both for the help, the conversations and discussions outside the weekly meetings.
I would like to thank Henk, who asked Josef whether it was possible for me to join his group during this project.
Off course I also want to thank my parents for making this trip (financially) possible and for supporting my choices. And I would like to thank the dear friends I made in Australia who made my time there so perfect. I would also like to thank my friends who supported me from the Netherlands and provided me with gossip at, mostly strange, local times.

Nieke

Bibliography

[1] J.-P. Aumasson, L. Henzen, W. Meier, and R. C.-W. Phan, "Toy versions of BLAKE." published on website, 2007.

[2] NIST, "Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA–3) Family," *Federal Register*, vol. 72, pp. 62212–62220, November 2007.

[3] J. Kelsey and B. Schneier, "Second preimages on n–bit hash functions for much less than 2^n work." Cryptology ePrint Archive, Report 2004/304, 2004. http://eprint.iacr.org/.

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

[5] E. Fleischmann, C. Forler, and M. Gorski, "Classification of the SHA–3 Candidates," 2009.

[6] R. C. Merkle, "One way hash functions and DES," in *CRYPTO '89: Proceedings on Advances in cryptology*, (New York, NY, USA), pp. 428–446, Springer–Verlag New York, Inc., 1989.

[7] I. Damgård, "A Design Principle for Hash Functions," in *CRYPTO '89: Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, (London, UK), pp. 416–427, Springer–Verlag, 1990.

[8] J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya, "Merkle–Damgård revisited: How to construct a hash function," in *Adcances in Cryptology – CRYPTO 2005, Lecture Notes in Computer Science*, vol. 3621/2005, pp. 430–448, Springer–Verlag, 2005.

[9] U. Maurer, R. Renner, and C. Holenstein, "Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology," in *Theory of Cryptography Conference – TCC 2004* (Moni Naor, ed.), vol. 2951 of *Lecture Notes in Computer Science*, pp. 21–39, Springer–Verlag, Feb. 2004.

[10] M. Matsui and A. Yamagishi, "A New Method for Known Plaintext Attack of FEAL Cipher.," in *Advances in Cryptology – EUROCRYPT '92*, vol. 658/1993, pp. 81–91, 1992.

[11] M. Matsui, "Linear Cryptanalysis Method for DES Cipher," in *Advances in Cryptology – EUROCRYPT '93*, vol. 765/1994, 1993.

[12] E. Biham and A. Shamir, "Differential Cryptanalysis of DES–like Cryptosystems," in *CRYPTO'91*, 1991.

[13] D. Coppersmith, "The Data Encryption Standard (DES) and its strength against attacks," *IBM J. Res. Dev.*, vol. 38, pp. 243–250, May 1994.

[14] M. Daum, "Cryptanalysis of Hash Functions of the MD4–Family," 2005.

[15] H. Lipmaa and S. Moriai, "Efficient algorithms for computing differential properties of addition," in *Properties of Addition, FSE 2001*, pp. 336–350, Springer–Verlag, 2001.

[16] S. Indesteege and B. Preneel, "Practical collisions for EnRUPT," in *LNCS*, vol. 5665, pp. 246–259, Springer–Verlag, 2009.

[17] T. Isobe and T. Shirai, "Low–weight Pseudo Collision Attack on Shabal and Preimage Attack on Reduced Shabal–512." Cryptology ePrint Archive, Report 2010/434, 2010.

[18] X. Wang and H. Yu, "How to Break MD5 and Other Hash Functions," in *EUROCRYPT*, pp. 19–35, 2005.

[19] D. Wagner, "The Boomerang Attack," in *LNCS*, vol. 1636, pp. 156–170, Springer, 1999.

[20] I. Dinur and A. Shamir, "Cube Attacks on Tweakable Black Box Polynomials." Cryptology ePrint Archive, Report2008/385, 2008.

[21] M. Vielhaber, "Breaking ONE.FIVIUM by AIDA an Algebraic IV Differential Attack." Cryptology ePrint Archive, Report 2007/413, 2007. http://eprint.iacr.org/.

[22] M. Vielhaber, "Shamir's "cube attack": A Remake of AIDA, The Algebraic IV Differential Attack." Note on the Cube Attack, 2009.

[23] E. Biham and R. Chen, "Near–Collisions of SHA–0," in *LNCS*, vol. 3152, pp. 290–305, Springer, 2004.

[24] B. Zhu, W. Yu, and T. Wang, "A Practical Platform for Cube–Attack–like Cryptanalyses." Cryptology ePrint Archive, Report 2010/644, 2010.

[25] E. Bresson, A. Canteaut, B. Chevallier-Mames, C. Clavier, T. Fuhr, A. Gouget, T. Icart, J.-F. Misarsky, M. Naya-Plasencia, P. Paillier, T. Pornin, J.-R. Reinhard, C. Thuillet, and M. Videau, "Shabal, a Submission to NIST's Cryptographic Hash Algorithm Competition." Submission to NIST, 2008.

[26] NIST, "Status report on the second round of the SHA–3 Cryptographic Hash Competition," February 2011.

[27] J.-P. Aumasson, "On the pseudorandomness of Shabal's keyed permutation," 2009.

[28] L. R. Knudsen, K. Matusiewicz, and S. S. Thomsen, "Observations on the Shabal keyed permutation," 2009.

[29] J.-P. Aumasson, A. Mashatan, and W. Meier, "More on Shabal's permutation," 2009.

[30] E. Bresson, A. Canteaut, B. Chevallier-Mames, C. Clavier, T. Fuhr, A. Gouget, T. Icart, J.-F. Misarsky, M. Naya-Plasencia, P. Paillier, T. Pornin, J.-R. Reinhard, C. Thuillet, and M. Videau, "Indifferentiability with Distinguishers: Why Shabal Does Not Require Ideal Ciphers." Cryptology ePrint Archive, Report 2009/199, 2009.

[31] G. V. Assche, "A rotational distinguisher on Shabal's keyed permutation and its impact on the security proofs," 2010.

[32] P. Novotney, "Distinguisher for Shabal's Permutation Function." Cryptology ePrint Archive, Report 2010/398, 2010.

[33] J.-P. Aumasson, "Observation on Shabal." NIST mailing list, 2010.

[34] E. Bresson, A. Canteaut, B. Chevallier-Mames, C. Clavier, T. Fuhr, A. Gouget, T. Icart, J.-F. Misarsky, M. Naya-Plasencia, P. Paillier, T. Pornin, J.-R. Reinhard, C. Thuillet, and M. Videau, "Internal Distinguishers in Indifferentiable Hashing: The Shabal Case." , 2010.

[35] M. Naya-Placensia, "Chiffrement par flot et fonctions de hachage: conception et cryptanalyse." PhD thesis, 2009.

[36] D. Gligoroski, "Narrow–pipe SHA–3 candidates differ significantly from ideal random functions defined over big domains." NIST mailing list, 2010, 2010.

[37] M. S. Turan and E. Uyan, "Practical Near–Collisions for Reduced Round Blake, Fugue, Hamsi and JH." Second SHA–3 Candidate Conference, 2010.

[38] P. Sokolowski, "Rotational and Shift Cryptanalysis of (Modified) Versions of SHA–3 Candidates," 2010.

[39] P. alain Fouque, G. Leurent, and P. Nguyen, "Automatic Search of Differential Path in MD4." Cryptology ePrint Archive: Report 2007/206, 2007.

[40] X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu, "Cryptanalysis of the Hash Functions MD4 and RIPEMD.," in *EUROCRYPT*, pp. 1–18, 2005.

[41] D. Tonien, "Some Algebraic properties of SHA–1 and collision test for iterative hash functions." Not published.

[42] H. Yu, J. Chen, K. jia, , and X. Wang, "Near–Collision Attack on the Step–Reduced Compression Function of Skein–256," 2011.

[43] J.-P. Aumasson, L. Henzen, W. Meier, and R. C. W. Phan, "SHA–3 proposal"BLAKE." Submission to NIST, 2008.

[44] J.-P. Aumasson, J. Guo, S. Knellwolf, K. Matusiewicz, and W. Meier, "Differential and invertibility properties of BLAKE (full version)." Cryptology ePrint Archive, Report 2010/043, 2010.

[45] L. Ji and X. Liangyu, "Attacks on Round–Reduced BLAKE." Cryptology ePrint Archive, Report 2009/238, 2009.

[46] J. Guo and K. Matusiewicz, "Round–Reduced Near–Collisions of BLAKE–32." sent over NIST SHA–3 mailing list, 2009.

[47] M. Ming, H. Qiang, and S. Zeng, "Security analysis of BLAKE–32 based on differential properties (abstract)." ICCIS 2010, 2010.

[48] J. Vidali, P. Nose, and E. Pašalic, "Collisions for variants of the BLAKE hash function." In Proceedings of Information Processing Letters, 2010.

[49] B. Su, W. Wu, S. Wu, and L. Dong, "Near–Collisions on the Reduced–Round Compression Functions of Skein and BLAKE," 2010.

[50] A. Biryukov, I. Nikolic, and A. Roy, "Boomerang attacks on BLAKE–32." (slides) FSE 2011, 2011.

[51] J.-P. Aumasson, G. Leurent, W. Meier, F. Mendel, N. Mouha, R. C.-W. Phan, Y. Sasaki, and P. Susil, "Tuple cryptanalysis of ARX with application to BLAKE and Skein.." ECRYPT2 Hash Workshop 2011, 2011.

[52] O. Dunkelman and D. Khovratovich, "Iterative differentials, symmetries, and message modification in BLAKE–256." ECRYPT2 Hash Workshop 2011, 2011.

[53] A. Regenscheid, R. Perlner, S. jen Chang, J. Kelsey, M. Nandi, and S. Paul, "Status report on the first round of the SHA–3 Cryptographic Hash Competition," February 2011.

Appendix A

Notations

$\|\|$	Concatenation ($x\|\|y$: append y to x)
$\oplus$	XOR
$\ggg$	Right-Rotation
$\lll$	Left-Rotation
$\gg$	Right-Shift
$\ll$	Left-Shift
$+$	Modular Addition (mostly $\mod 2^{32}$)
$\vee$	OR operator
$\wedge$	AND operator
$\overline{x}$	Negation of x

Shabal

- $\mathcal{R}$ is the Compression/Round Function.
- $\mathcal{P}$ is the Permutation.
- A, B, C, m are the inputs for the permutation.
- $A[i]$ The i^{th} word of the array A.
- $A^j[i]$ The j^{th} version of the i^{th} word, we start with $A^0 = A$, after changing it in step 2 of the permutation, we call it A^1.
- $B[i]$ The i^{th} word of the array B.
- $B^j[i]$ The j^{th} version of the i^{th} word, we start with $B^{-1} = B$, after changing it in step 1 of the permutation, we call it B^0.

BLAKE

- $\mathcal{C}$ is the Compression Function.
- $\mathcal{R}$ is the Round Function.
- $\mathcal{G}$ is the Inner Primitive.
- h, m, s, a is the input for the compression function.

- The 4×4 word matrix is the input for the round function.
- a, b, c, d, i, r is the input for the inner primitive, where i represents the step within the round and r the round.
- $\mathcal{G}_{m_1,m_2}(a, b, c, d)$ represents the inner primitive applied to (a, b, c, d) using message blocks m_1 and m_2.

Appendix B

on Shabal

B.1 Differential Attack of Novotney

The rounds of the second step of the permutation of Shabal are given below. In the first 26 steps the active elements are colored red. Recall that $A^0[10]$ and $B^0[7]$ start with a difference. At round 26 we have differences only in $A^3[2]$, $B^2[7]$ and $B^2[10]$ with probability $1/8$

round			
0	$A^1[0]$	$=$	$3(A^0[0] \oplus 5(A^0[11] \lll_{15}) \oplus C[8]) \oplus B^0[13] \oplus (B^0[9] \wedge \overline{B^0[6]}) \oplus m[0]$
	$B^1[0]$	$=$	$B^0[0] \oplus \overline{A^1[0]}$
1	$A^1[1]$	$=$	$3(A^0[1] \oplus 5(A^1[0] \lll_{15}) \oplus C[9]) \oplus B^0[14] \oplus (B^0[10] \wedge \overline{B^0[7]}) \oplus m[1]$
	$B^1[1]$	$=$	$B^0[1] \oplus \overline{A^1[1]}$
2	$A^1[2]$	$=$	$3(A^0[2] \oplus 5(A^1[1] \lll_{15}) \oplus C[10]) \oplus B^0[15] \oplus (B^0[11] \wedge \overline{B^0[8]}) \oplus m[2]$
	$B^1[2]$	$=$	$B^0[2] \oplus \overline{A^1[2]}$
3	$A^1[3]$	$=$	$3(A^0[3] \oplus 5(A^1[2] \lll_{15}) \oplus C[11]) \oplus B^1[0] \oplus (B^0[12] \wedge \overline{B^0[9]}) \oplus m[3]$
	$B^1[3]$	$=$	$B^0[3] \oplus \overline{A^1[3]}$
4	$A^1[4]$	$=$	$3(A^0[4] \oplus 5(A^1[3] \lll_{15}) \oplus C[12]) \oplus B^1[1] \oplus (B^0[13] \wedge \overline{B^0[10]}) \oplus m[4]$
	$B^1[4]$	$=$	$B^0[4] \oplus \overline{A^1[4]}$
5	$A^1[5]$	$=$	$3(A^0[5] \oplus 5(A^1[4] \lll_{15}) \oplus C[13]) \oplus B^1[2] \oplus (B^0[14] \wedge \overline{B^0[11]}) \oplus m[5]$
	$B^1[5]$	$=$	$B^0[5] \oplus \overline{A^1[5]}$
6	$A^1[6]$	$=$	$3(A^0[6] \oplus 5(A^1[5] \lll_{15}) \oplus C[14]) \oplus B^1[3] \oplus (B^0[15] \wedge \overline{B^0[12]}) \oplus m[6]$
	$B^1[6]$	$=$	$B^0[6] \oplus \overline{A^1[6]}$
7	$A^1[7]$	$=$	$3(A^0[7] \oplus 5(A^1[6] \lll_{15}) \oplus C[15]) \oplus B^1[4] \oplus (B^1[0] \wedge \overline{B^0[13]}) \oplus m[7]$
	$B^1[7]$	$=$	$B^0[7] \oplus \overline{A^1[7]}$
8	$A^1[8]$	$=$	$3(A^0[8] \oplus 5(A^1[7] \lll_{15}) \oplus C[0]) \oplus B^1[5] \oplus (B^1[1] \wedge \overline{B^0[14]}) \oplus m[8]$
	$B^1[8]$	$=$	$B^0[8] \oplus \overline{A^1[8]}$
9	$A^1[9]$	$=$	$3(A^0[9] \oplus 5(A^1[8] \lll_{15}) \oplus C[1]) \oplus B^1[6] \oplus (B^1[2] \wedge \overline{B^0[15]}) \oplus m[9]$
	$B^1[9]$	$=$	$B^0[9] \oplus \overline{A^1[9]}$
10	$A^1[10]$	$=$	$3(A^0[10] \oplus 5(A^1[9] \lll_{15}) \oplus C[2]) \oplus B^1[7] \oplus (B^1[3] \wedge \overline{B^3[0]}) \oplus m[10]$
	$B^1[10]$	$=$	$B^0[10] \oplus \overline{A^1[10]}$

round			
11	$A^1[11]$	$=$	$3(A^0[11] \oplus 5(A^1[10] \lll_{15}) \oplus C[3]) \oplus B^1[8] \oplus (B^1[4] \wedge \overline{B^1[1]}) \oplus m[11]$
	$B^1[11]$	$=$	$B^0[11] \oplus \overline{A^1[11]}$
12	$A^2[0]$	$=$	$3(A^1[0] \oplus 5(A^1[11] \lll_{15}) \oplus C[4]) \oplus B^1[9] \oplus (B^1[5] \wedge \overline{B^1[2]}) \oplus m[12]$
	$B^1[12]$	$=$	$B^0[12] \oplus \overline{A^2[0]}$
13	$A^2[1]$	$=$	$3(A^1[1] \oplus 5(A^2[0] \lll_{15}) \oplus C[5]) \oplus B^1[10] \oplus (B^1[6] \wedge \overline{B^1[3]}) \oplus m[13]$
	$B^1[13]$	$=$	$B^0[13] \oplus \overline{A^2[1]}$
14	$A^2[2]$	$=$	$3(A^1[2] \oplus 5(A^2[1] \lll_{15}) \oplus C[6]) \oplus B^1[11] \oplus (B^1[7] \wedge \overline{B^1[4]}) \oplus m[14]$
	$B^1[14]$	$=$	$B^0[14] \oplus \overline{A^2[2]}$
15	$A^2[3]$	$=$	$3(A^1[3] \oplus 5(A^2[2] \lll_{15}) \oplus C[7]) \oplus B^1[12] \oplus (B^1[8] \wedge \overline{B^1[5]}) \oplus m[15]$
	$B^1[15]$	$=$	$B^0[15] \oplus \overline{A^2[3]}$
16	$A^2[4]$	$=$	$3(A^1[4] \oplus 5(A^2[3] \lll_{15}) \oplus C[8]) \oplus B^1[13] \oplus (B^1[9] \wedge \overline{B^1[6]}) \oplus m[0]$
	$B^2[0]$	$=$	$B^1[0] \oplus \overline{A^2[4]}$
17	$A^2[5]$	$=$	$3(A^1[5] \oplus 5(A^2[4] \lll_{15}) \oplus C[9]) \oplus B^1[14] \oplus (B^1[10] \wedge \overline{B^1[7]}) \oplus m[1]$
	$B^2[1]$	$=$	$B^1[1] \oplus \overline{A^2[5]}$
18	$A^2[6]$	$=$	$3(A^1[6] \oplus 5(A^2[5] \lll_{15}) \oplus C[10]) \oplus B^1[15] \oplus (B^1[11] \wedge \overline{B^1[8]}) \oplus m[2]$
	$B^2[2]$	$=$	$B^1[2] \oplus \overline{A^2[6]}$
19	$A^2[7]$	$=$	$3(A^1[7] \oplus 5(A^2[6] \lll_{15}) \oplus C[11]) \oplus B^2[0] \oplus (B^1[12] \wedge \overline{B^1[9]}) \oplus m[3]$
	$B^2[3]$	$=$	$B^1[3] \oplus \overline{A^2[7]}$
20	$A^2[8]$	$=$	$3(A^1[8] \oplus 5(A^2[7] \lll_{15}) \oplus C[12]) \oplus B^2[1] \oplus (B^1[13] \wedge \overline{B^1[10]}) \oplus m[4]$
	$B^2[4]$	$=$	$B^1[4] \oplus \overline{A^2[8]}$
21	$A^2[9]$	$=$	$3(A^1[9] \oplus 5(A^2[8] \lll_{15}) \oplus C[13]) \oplus B^2[2] \oplus (B^1[14] \wedge \overline{B^1[11]}) \oplus m[5]$
	$B^2[5]$	$=$	$B^1[5] \oplus \overline{A^2[9]}$
22	$A^2[10]$	$=$	$3(A^1[10] \oplus 5(A^2[9] \lll_{15}) \oplus C[14]) \oplus B^2[3] \oplus (B^1[15] \wedge \overline{B^1[12]}) \oplus m[6]$
	$B^2[6]$	$=$	$B^1[6] \oplus \overline{A^2[10]}$
23	$A^2[11]$	$=$	$3(A^1[11] \oplus 5(A^2[10] \lll_{15}) \oplus C[15]) \oplus B^2[4] \oplus (B^2[0] \wedge \overline{B^1[13]}) \oplus m[7]$
	$B^2[7]$	$=$	$B^1[7] \oplus \overline{A^2[11]}$
24	$A^3[0]$	$=$	$3(A^2[0] \oplus 5(A^2[11] \lll_{15}) \oplus C[0]) \oplus B^2[5] \oplus (B^2[1] \wedge \overline{B^1[14]}) \oplus m[8]$
	$B^2[8]$	$=$	$B^1[8] \oplus \overline{A^3[0]}$
25	$A^3[1]$	$=$	$3(A^2[1] \oplus 5(A^3[0] \lll_{15}) \oplus C[1]) \oplus B^2[6] \oplus (B^2[2] \wedge \overline{B^1[15]}) \oplus m[9]$
	$B^2[9]$	$=$	$B^1[9] \oplus \overline{A^3[1]}$
26	$A^3[2]$	$=$	$3(A^2[2] \oplus 5(A^3[1] \lll_{15}) \oplus C[2]) \oplus B^2[7] \oplus (B^2[3] \wedge \overline{B^1[0]}) \oplus m[10]$
	$B^2[10]$	$=$	$B^1[10] \oplus \overline{A^3[2]}$
27	$A^3[3]$	$=$	$3(A^2[3] \oplus 5(A^3[2] \lll_{15}) \oplus C[3]) \oplus B^2[8] \oplus (B^2[4] \wedge \overline{B^2[1]}) \oplus m[11]$
	$B^2[11]$	$=$	$B^1[11] \oplus \overline{A^3[3]}$
28	$A^3[4]$	$=$	$3(A^2[4] \oplus 5(A^3[3] \lll_{15}) \oplus C[4]) \oplus B^2[9] \oplus (B^2[5] \wedge \overline{B^2[2]}) \oplus m[12]$
	$B^2[12]$	$=$	$B^1[12] \oplus \overline{A^3[4]}$
29	$A^3[5]$	$=$	$3(A^2[5] \oplus 5(A^3[4] \lll_{15}) \oplus C[5]) \oplus B^2[10] \oplus (B^2[6] \wedge \overline{B^2[3]}) \oplus m[13]$
	$B^2[13]$	$=$	$B^1[13] \oplus \overline{A^3[5]}$
30	$A^3[6]$	$=$	$3(A^2[6] \oplus 5(A^3[5] \lll_{15}) \oplus C[6]) \oplus B^2[11] \oplus (B^2[7] \wedge \overline{B^2[4]}) \oplus m[14]$
	$B^2[14]$	$=$	$B^1[14] \oplus \overline{A^3[6]}$

round			
31	$A^3[7]$	$=$	$3(A^2[7] \oplus 5(A^3[6] \lll_{15}) \oplus C[7]) \oplus B^2[12] \oplus (B^2[8] \wedge \overline{B^2[5]}) \oplus m[15]$
	$B^2[15]$	$=$	$B^1[15] \oplus \overline{A^3[7]}$
32	$A^3[8]$	$=$	$3(A^2[8] \oplus 5(A^3[7] \lll_{15}) \oplus C[8]) \oplus B^2[13] \oplus (B^2[9] \wedge \overline{B^2[6]}) \oplus m[0]$
	$B^3[0]$	$=$	$B^2[0] \oplus \overline{A^3[8]}$
33	$A^3[9]$	$=$	$3(A^2[9] \oplus 5(A^3[8] \lll_{15}) \oplus C[9]) \oplus B^2[14] \oplus (B^2[10] \wedge \overline{B^2[7]}) \oplus m[1]$
	$B^3[1]$	$=$	$B^2[1] \oplus \overline{A^3[9]}$
34	$A^3[10]$	$=$	$3(A^2[10] \oplus 5(A^3[9] \lll_{15}) \oplus C[10]) \oplus B^2[15] \oplus (B^2[11] \wedge \overline{B^2[8]}) \oplus m[2]$
	$B^3[2]$	$=$	$B^2[2] \oplus \overline{A^3[10]}$
35	$A^3[11]$	$=$	$3(A^2[11] \oplus 5(A^3[10] \lll_{15}) \oplus C[11]) \oplus B^3[0] \oplus (B^2[12] \wedge \overline{B^2[9]}) \oplus m[3]$
	$B^3[3]$	$=$	$B^2[3] \oplus \overline{A^3[11]}$
36	$A^4[0]$	$=$	$3(A^3[0] \oplus 5(A^3[11] \lll_{15}) \oplus C[12]) \oplus B^3[1] \oplus (B^2[13] \wedge \overline{B^2[10]}) \oplus m[4]$
	$B^3[4]$	$=$	$B^2[4] \oplus \overline{A^4[0]}$
37	$A^4[1]$	$=$	$3(A^3[1] \oplus 5(A^4[0] \lll_{15}) \oplus C[13]) \oplus B^3[2] \oplus (B^2[14] \wedge \overline{B^2[11]}) \oplus m[5]$
	$B^3[5]$	$=$	$B^2[5] \oplus \overline{A^4[1]}$
38	$A^4[2]$	$=$	$3(A^3[2] \oplus 5(A^4[1] \lll_{15}) \oplus C[14]) \oplus B^3[3] \oplus (B^2[15] \wedge \overline{B^2[12]}) \oplus m[6]$
	$B^3[6]$	$=$	$B^2[6] \oplus \overline{A^4[2]}$
39	$A^4[3]$	$=$	$3(A^3[3] \oplus 5(A^4[2] \lll_{15}) \oplus C[15]) \oplus B^3[4] \oplus (B^3[0] \wedge \overline{B^2[13]}) \oplus m[7]$
	$B^3[7]$	$=$	$B^2[7] \oplus \overline{A^4[3]}$
40	$A^4[4]$	$=$	$3(A^3[4] \oplus 5(A^4[3] \lll_{15}) \oplus C[0]) \oplus B^3[5] \oplus (B^3[1] \wedge \overline{B^2[14]}) \oplus m[8]$
	$B^3[8]$	$=$	$B^2[8] \oplus \overline{A^4[4]}$
41	$A^4[5]$	$=$	$3(A^3[5] \oplus 5(A^4[4] \lll_{15}) \oplus C[1]) \oplus B^3[6] \oplus (B^3[2] \wedge \overline{B^2[15]}) \oplus m[9]$
	$B^3[9]$	$=$	$B^2[9] \oplus \overline{A^4[5]}$
42	$A^4[6]$	$=$	$3(A^3[6] \oplus 5(A^4[5] \lll_{15}) \oplus C[2]) \oplus B^3[7] \oplus (B^3[3] \wedge \overline{B^3[0]}) \oplus m[10]$
	$B^3[10]$	$=$	$B^2[10] \oplus \overline{A^4[6]}$
43	$A^4[7]$	$=$	$3(A^3[7] \oplus 5(A^4[6] \lll_{15}) \oplus C[3]) \oplus B^3[8] \oplus (B^3[4] \wedge \overline{B^3[1]}) \oplus m[11]$
	$B^3[11]$	$=$	$B^2[11] \oplus \overline{A^4[7]}$
44	$A^4[8]$	$=$	$3(A^3[8] \oplus 5(A^4[7] \lll_{15}) \oplus C[4]) \oplus B^3[9] \oplus (B^3[5] \wedge \overline{B^3[2]}) \oplus m[12]$
	$B^3[12]$	$=$	$B^2[12] \oplus \overline{A^4[8]}$
45	$A^4[9]$	$=$	$3(A^3[9] \oplus 5(A^4[8] \lll_{15}) \oplus C[5]) \oplus B^3[10] \oplus (B^3[6] \wedge \overline{B^3[3]}) \oplus m[13]$
	$B^3[13]$	$=$	$B^2[13] \oplus \overline{A^4[9]}$
46	$A^4[10]$	$=$	$3(A^3[10] \oplus 5(A^4[9] \lll_{15}) \oplus C[6]) \oplus B^3[11] \oplus (B^3[7] \wedge \overline{B^3[4]}) \oplus m[14]$
	$B^3[14]$	$=$	$B^2[14] \oplus \overline{A^4[10]}$
47	$A^4[11]$	$=$	$3(A^3[11] \oplus 5(A^4[10] \lll_{15}) \oplus C[7]) \oplus B^3[12] \oplus (B^3[8] \wedge \overline{B^3[5]}) \oplus m[15]$
	$B^3[15]$	$=$	$B^2[15] \oplus \overline{A^4[11]}$

Appendix C

on BLAKE

C.1 Initial Values and Constants

BLAKE–256

BLAKE–256 starts with the same initial values as SHA–2:

```
IV0 = 6A09E667  IV1 = BB67AE85
IV2 = 3C6EF372  IV3 = A54FF53A
IV4 = 510E527F  IV5 = 9B05688C
IV6 = 1F83D9AB  IV7 = 5BE0CD19.
```

BLAKE-256 uses the 16 constants:

```
c0  = 243F6A88  c1  = 85A308D3
c2  = 13198A2E  c3  = 03707344
c4  = A4093822  c5  = 299F31D0
c6  = 082EFA98  c7  = EC4E6C89
c8  = 452821E6  c9  = 38D01377
c10 = BE5466CF  c11 = 34E90C6C
c12 = C0AC29B7  c13 = C97C50DD
c14 = 3F84D5B5  c15 = B5470917.
```

BLAKE–224

BLAKE–224 uses the same constants as BLAKE–256 and starts with the following initial values:

```
IV0 = C1059ED8  IV1 = 367CD507
IV2 = 3070DD17  IV3 = F70E5939
IV4 = FFC00B31  IV5 = 68581511
IV6 = 64F98FA7  IV7 = BEFA4FA4.
```

BLAKE–512

The initial value ofBLAKE-512 is the same as for SHA-512:

IV0	=	6A09E667F3BCC908	IV1	=	BB67AE8584CAA73B
IV2	=	3C6EF372FE94F82B	IV3	=	A54FF53A5F1D36F1
IV4	=	510E527FADE682D1	IV5	=	9B05688C2B3E6C1F
IV6	=	1F83D9ABFB41BD6B	IV7	=	5BE0CD19137E2179.

BLAKE-512 uses the 16 constants:

c0	=	243F6A8885A308D3	c1	=	13198A2E03707344
c2	=	A4093822299F31D0	c3	=	082EFA98EC4E6C89
c4	=	452821E638D01377	c5	=	BE5466CF34E90C6C
c6	=	C0AC29B7C97C50DD	c7	=	3F84D5B5B5470917
c8	=	9216D5D98979FB1B	c9	=	D1310BA698DFB5AC
c10	=	2FFD72DBD01ADFB7	c11	=	B8E1AFED6A267E96.
c12	=	BA7C9045F12C7F99	c13	=	24A19947B3916CF7
c14	=	0801F2E2858EFC16	c15	=	636920D871574E69

BLAKE–384

BLAKE–384 uses the same constants as BLAKE–512 and starts with the following initial values:

IV0	=	CBBB9D5DC1059ED8	IV1	=	629A292A367CD507
IV2	=	9159015A3070DD17	IV3	=	152FECD8F70E5939
IV4	=	67332667FFC00B31	IV5	=	8EB44A8768581511
IV6	=	DB0C2E0D64F98FA7	IV7	=	47B5481DBEFA4FA4.

C.2 Impossible States

Recall that we proved:

$$\begin{array}{lllll|lllll} \Delta a' = 0 & \Rightarrow & \Delta d' \neq 0 & & & \Delta c' = 0 & \Rightarrow & \Delta b' \neq 0 & \wedge & \Delta d' \neq 0 \\ \Delta b' = 0 & \Rightarrow & \Delta c' \neq 0 & & & \Delta d' = 0 & \Rightarrow & \Delta a' \neq 0 & \wedge & \Delta c' \neq 0. \end{array}$$

for the process of one $\mathcal{G}$ and the start states are equal.
Now we easily conclude that we will never have only one of (a, b, c, d) with a difference. And from the equations above we conclude:

$$\begin{array}{lll} (\cdot, \cdot, 0, 0) & \Rightarrow & c \neq 0, d \neq 0 \\ (\cdot, 0, 0, \cdot) & \Rightarrow & a \neq 0, d \neq 0 \\ (0, \cdot, \cdot, 0) & \Rightarrow & b \neq 0, c \neq 0. \end{array}$$

C.3 Proofs on output $(\Delta, 0, \Delta', 0)$

We stated that if the following equations are satisfied, we will find output $(\Delta, 0, \Delta', 0)$.

$$\begin{aligned}
\Delta a^* &= \Delta m_1 && \text{(C.1)}\\
\Delta d^* &= \Delta m_1 \ggg_{16} && \text{(C.2)}\\
\Delta c^* &= \Delta m_1 \ggg_{16} && \text{(C.3)}\\
\Delta b^* &= \Delta m_1 \ggg_{28} && \text{(C.4)}\\
\Delta a' &= \Delta m_1 + \Delta m_1 \ggg_{28} + \Delta m_2 && \text{(C.5)}\\
\Delta d' &= (\Delta m_1 \ggg_{16} \oplus (\Delta m_1 + \Delta m_1 \ggg_{28} + \Delta m_2)) \ggg_8 && \text{(C.6)}\\
\Delta c' &= \Delta m_1 \ggg_{16} && \text{(C.7)}\\
\Delta b' &= (\Delta m_1 \ggg_{28} \oplus \Delta m_1 \ggg_{16}) \ggg_7 . && \text{(C.8)}
\end{aligned}$$

Note that we use $m_1 \ggg_4 = m_1$ and $m_1 + m_2 = 2^{32}$ in all the following, and $\mathcal{G}_{m_1,m_2}(a,b,c,d)$ denotes the round function $\mathcal{G}$ applied to (a,b,c,d) using the message blocks m_1 and m_2.

When three input variables are 0

Lemma 8. *If precisely three of the four input values (a,b,c,d) are zero, and the fourth is a power of 2, then*

$$\mathcal{G}_{m_1,m_2}(a,b,c,d) - \mathcal{G}_{m_1',m_2'}(a,b,c,d) = (\Delta, 0, \Delta', 0)$$

with probability $1/2$.

Proof. We find

$$\mathcal{G}_{m_1,m_2}(a,b,c,d) - \mathcal{G}_{m_1',m_2'}(a,b,c,d) = (\Delta, 0, \Delta', 0)$$

if and only if Equations 4.1 to 4.8 hold.
Let $(a,b,c,d) = (2^j, 0, 0, 0)$, in total we have 32 different options, but there are only four options which give different restrictions. Since m_1 is rotatable over four, $j = 0$ and $j = 4$ will give the same restrictions.
We state that if $\Delta m_{1_{j \bmod 4}} = 0$ Equations 4.1 to 4.8 hold.

Δa^*	=	Δm_1	since there is no carry[1] due to $\Delta m_{1_{j \bmod 4}} = 0$.
Δd^*	=	$\Delta m_1 \ggg_{16}$.	
Δc^*	=	$\Delta m_1 \ggg_{16}$	since $c = 0$.
Δb^*	=	$\Delta m_1 \ggg_{28}$.	
$\Delta a'$	=	$\Delta m_1 + \Delta m_1 \ggg_{28} + \Delta m_2 = \Delta m_1$	since $\Delta m_2 = 2^{32} - \Delta m_1$ and $\Delta m_1 = \Delta m_1 \ggg_4$.
$\Delta d'$	=	$(\Delta m_1 \ggg_{16} \oplus \Delta m_1) \ggg_8 = 0$	since $\Delta m_1 = \Delta m_1 \ggg_4$.
$\Delta c'$	=	$\Delta m_1 \ggg_{16}$	since $\Delta d' = 0$.
$\Delta b'$	=	$(\Delta m_1 \ggg_{28} \oplus \Delta m_1 \ggg_{16}) \ggg_7 = 0$	since $\Delta m_1 = \Delta m_1 \ggg_4$.

We conclude that we find $\Delta m_{1_{j \bmod 4}} = 0$ with probability $1/2$, and thus if $a = 2^j$ we find

$$\mathcal{G}_{m_1,m_2}(a,b,c,d) - \mathcal{G}_{m_1',m_2'}(a,b,c,d) = (\Delta, 0, \Delta', 0)$$

with probability $1/2$.

Case: $(a,b,c,d) = (0,2^j,0,0)$ We find

$$\mathcal{G}_{m_1,m_2}(a,b,c,d) - \mathcal{G}_{m_1',m_2'}(a,b,c,d) = (\Delta,0,\Delta',0)$$

if and only if Equations C.1 to C.8 hold.
Let $(a,b,c,d) = (0,2^j,0,0)$ and we state that if $m_{1_{j \bmod 4}} = 0$ Equations C.1 to C.8 hold.

$$\begin{array}{rcll}
\Delta a^* & = & \Delta m_1 & \text{since there is no carry due to } m_{1_{j \bmod 4}} = 0. \\
\Delta d^* & = & \Delta m_1 \ggg_{16} . & \\
\Delta c^* & = & \Delta m_1 \ggg_{16} & \text{since } c = 0. \\
\Delta b^* & = & \Delta m_1 \ggg_{28} . & \\
\Delta a' & = & \Delta m_1 + \Delta m_1 \ggg_{28} + \Delta m_2 = \Delta m_1 & \text{since } \Delta m_2 = 2^{32} - \Delta m_1 \text{ and} \\
 & & & \Delta m_1 = \Delta m_1 \ggg_4 . \\
\Delta d' & = & (\Delta m_1 \ggg_{16} \oplus \Delta m_1) \ggg_8 = 0 & \text{since } \Delta m_1 = \Delta m_1 \ggg_4 . \\
\Delta c' & = & \Delta m_1 \ggg_{16} & \text{since } \Delta d' = 0. \\
\Delta b' & = & (\Delta m_1 \ggg_{28} \oplus \Delta m_1 \ggg_{16}) \ggg_7 = 0 & \text{since } \Delta m_1 = \Delta m_1 \ggg_4 .
\end{array}$$

We conclude that we find $m_{1_{j \bmod 4}} = 0$ with probability $1/2$, and thus if $b = 2^j$ we find

$$\mathcal{G}_{m_1,m_2}(a,b,c,d) - \mathcal{G}_{m_1',m_2'}(a,b,c,d) = (\Delta,0,\Delta',0)$$

with probability $1/2$.

Case: $(a,b,c,d) = (0,0,2^j,0)$ We find

$$\mathcal{G}_{m_1,m_2}(a,b,c,d) - \mathcal{G}_{m_1',m_2'}(a,b,c,d) = (\Delta,0,\Delta',0)$$

if and only if Equations C.1 to C.8 hold.
Let $(a,b,c,d) = (0,0,2^j,0)$ and we state that if $m_{1_{j \bmod 4}} = 0$ Equations C.1 to C.8 hold.

$$\begin{array}{rcll}
\Delta a^* & = & \Delta m_1 & \text{since } a = b = 0. \\
\Delta d^* & = & \Delta m_1 \ggg_{16} . & \\
\Delta c^* & = & \Delta m_1 \ggg_{16} & \text{since there is no carry due to } m_{1_{j \bmod 4}} = 0. \\
\Delta b^* & = & \Delta m_1 \ggg_{28} . & \\
\Delta a' & = & \Delta m_1 + \Delta m_1 \ggg_{28} + \Delta m_2 = \Delta m_1 & \text{since } \Delta m_2 = 2^{32} - \Delta m_1 \text{ and} \\
 & & & \Delta m_1 = \Delta m_1 \ggg_4 . \\
\Delta d' & = & (\Delta m_1 \ggg_{16} \oplus \Delta m_1) \ggg_8 = 0 & \text{since } \Delta m_1 = \Delta m_1 \ggg_4 . \\
\Delta c' & = & \Delta m_1 \ggg_{16} & \text{since } \Delta d' = 0. \\
\Delta b' & = & (\Delta m_1 \ggg_{28} \oplus \Delta m_1 \ggg_{16}) \ggg_7 = 0 & \text{since } \Delta m_1 = \Delta m_1 \ggg_4 .
\end{array}$$

We conclude that we find $m_{1_{j \bmod 4}} = 0$ with probability $1/2$, and thus if $c = 2^j$ we find

$$\mathcal{G}_{m_1,m_2}(a,b,c,d) - \mathcal{G}_{m_1',m_2'}(a,b,c,d) = (\Delta,0,\Delta',0)$$

with probability $1/2$.

Case: $(a,b,c,d) = (0,0,0,2^j)$ We find

$$\mathcal{G}_{m_1,m_2}(a,b,c,d) - \mathcal{G}_{m_1',m_2'}(a,b,c,d) = (\Delta,0,\Delta',0)$$

if and only if Equations C.1 to C.8 hold.
Let $(a,b,c,d) = (0,0,0,2^j)$ and we state that if $m_{1_{j \bmod 4}} = 0$ Equations C.1 to C.8 hold.

$$
\begin{array}{rcll}
\Delta a^* & = & \Delta m_1 & \text{since } a = b = 0. \\
\Delta d^* & = & \Delta m_1 \ggg_{16} . & \\
\Delta c^* & = & \Delta m_1 \ggg_{16} & \text{since } c = 0. \\
\Delta b^* & = & \Delta m_1 \ggg_{28} . & \\
\Delta a' & = & \Delta m_1 + \Delta m_1 \ggg_{28} + \Delta m_2 = \Delta m_1 & \text{since } \Delta m_2 = 2^{32} - \Delta m_1 \text{ and} \\
 & & & \Delta m_1 = \Delta m_1 \ggg_4 . \\
\Delta d' & = & (\Delta m_1 \ggg_{16} \oplus \Delta m_1) \ggg_8 = 0 & \text{since } \Delta m_1 = \Delta m_1 \ggg_4 . \\
\Delta c' & = & \Delta m_1 \ggg_{16} & \text{since } \Delta d' = 0. \\
\Delta b' & = & (\Delta m_1 \ggg_{28} \oplus \Delta m_1 \ggg_{16}) \ggg_7 = 0 & \text{since } \Delta m_1 = \Delta m_1 \ggg_4 .
\end{array}
$$

We conclude that we find $m_{1_{j \mod 4}} = 0$ with probability $1/2$, and thus if $d = 2^j$ we find

$$\mathcal{G}_{m_1,m_2}(a,b,c,d) - \mathcal{G}_{m_1',m_2'}(a,b,c,d) = (\Delta, 0, \Delta', 0)$$

with probability $1/2$.

□

Conclusion

We showed that for all cases the probability of success is $1/2$ and therefore the theorem
Lemma 9. *If precisely three of the four input values (a,b,c,d) are zero, and the fourth is a power of 2, then*

$$\mathcal{G}_{m_1,m_2}(a,b,c,d) - \mathcal{G}_{m_1',m_2'}(a,b,c,d) = (\Delta, 0, \Delta', 0)$$

with probability $1/2$.

is proven.

When two input variables are 0

Lemma 10. *If precisely two of the four input values (a,b,c,d) are zero, and the other two are a power of 2, then*

$$\mathcal{G}_{m_1,m_2}(a,b,c,d) - \mathcal{G}_{m_1',m_2'}(a,b,c,d) = (\Delta, 0, \Delta', 0)$$

with probability at least $1/4$.

Case: $i = j$ We state that Equation C.1 to C.8 hold if $(\Delta m_1)_i = (\Delta m_1)_{i+1} = 0$ for $a = b = 2^i$.

$$
\begin{aligned}
\Delta a^* &= (a_1 + b_1 + m_1) \oplus (a_2 + b_2 + m'_1) \\
&= (2^{i+1} + m_1) \oplus (2^{i+1} + m'_1) \\
&= (2^{i+1} \oplus m_1) \oplus (2^{i+1} \oplus m'_1) \qquad \text{if } (\Delta m_1)_{i+1} = 0 \\
&= \Delta m_1 \\
\Delta d^* &= a_1^* \ggg_{16} \oplus a_2^* \ggg_{16} \\
&= (\Delta m_1) \ggg_{16} \\
&= \Delta m_1 \\
\Delta c^* &= d_1^* \oplus d_2^* \\
&= \Delta m_1 \\
\Delta b^* &= (b_1 \oplus c_1^*) \ggg 12 \oplus (b_2 \oplus c_2^*) \ggg 12 \\
&= (b_1 \oplus c_1^* \oplus b_2 \oplus c_2^*) \ggg 12 \\
&= (m_1 \oplus m'_1) \ggg_{12} \\
&= \Delta m_1 \\
\Delta a' &= (a_1^* + b_1^* + m_2) \oplus (a_2^* + b_2^* + m'_2) \\
&= \Big((2^{i+1} \oplus m_1) + (2^{i-12} \oplus 2^{i-29} \oplus m_1) + m_2\Big) \oplus \Big((2^{i+1} \oplus m'_1) + (2^{i-12} \oplus 2^{i-29} \oplus m'_1) + m'_2\Big) \\
&= \Big((2^{i+1} \oplus 2^{i-12} \oplus 2^{i-29}) + (2m_1 + m_2)\Big) \oplus \Big((2^{i+1} \oplus 2^{i-12} \oplus 2^{i-29}) + (2m'_1 + m'_2)\Big) \\
&= \Big((2^{i+1} \oplus 2^{i-12} \oplus 2^{i-29}) \oplus m_1\Big) \oplus \Big((2^{i+1} \oplus 2^{i-12} \oplus 2^{i-29}) \oplus 2m'_1\Big) \qquad \text{if } (\Delta m_1)_i = 0 \\
&= \Delta m_1 \\
\Delta d' &= (d_1^* \oplus a'_1) \ggg_8 \oplus (d_2^* \oplus a'_2) \ggg_8 \\
&= (\Delta d^* \oplus \Delta a') \ggg_8 \\
&= (\Delta m_1 \oplus \Delta m_1) \ggg_8 \\
&= 0 \\
\Delta c' &= (c_1^* + d'_1) \oplus (c_2^* + d'_2) \\
&= \Big((2^{i-17} \oplus m_1) + (2^{i+1} \oplus 2^{i-12} \oplus 2^{i-17} \oplus 2^{i-29})\Big) \oplus \Big((2^{i-17} \oplus m'_1) + (2^{i+1} \oplus 2^{i-12} \oplus 2^{i-17} \oplus 2^{i-29})\Big) \\
&= \Big(2^{i-16} \oplus m_1 \oplus 2^{i+1} \oplus 2^{i-12} \oplus 2^{i-29}\Big) \oplus \Big(2^{i-16} \oplus m'_1 \oplus 2^{i+1} \oplus 2^{i-12} \oplus 2^{i-29}\Big) \\
&= \Delta m_1 \\
\Delta b' &= (b_1^* \oplus c'_1) \ggg_7 \oplus (b_2^* \oplus c'_2) \ggg_7 \\
&= \big(b_1^* \oplus b_2^* \oplus c'_1 \oplus c'_2\big) \ggg_7 \\
&= (\Delta m_1 \oplus \Delta m_1) \ggg_7 \\
&= 0
\end{aligned}
$$

We conclude that if $(\Delta m_1)_i = (\Delta m_1)_{i+1} = 0$ the equations are satisfied. This happens with probability $1/4$.

Case: $i \neq j$ We state that Equation C.1 to C.8 hold for $a = 2^i, b = 2^j$ with probability at least $1/4$.

$$
\begin{array}{rcll}
a^* & = & 2^j + 2^i + m_1 & \\
\Delta a^* & = & \Delta m_1 & \text{if } (\Delta m_1)_j = (\Delta m_1)_i = 0 \\
d^* & = & (2^j \oplus 2^i \oplus m_1) \ggg_{16} & \\
\Delta d^* & = & \Delta m_1 & \\
c^* & = & 2^{j-16} \oplus 2^{i-16} \oplus m_1 \ggg_{16} & \\
\Delta c^* & = & \Delta m_1 & \\
b^* & = & (2^{j-16} \oplus 2^{i-16} \oplus m_1 \ggg_{16} \oplus 2^i) \ggg_{12} & \\
\Delta b^* & = & \Delta m_1 & \\
a' & = & 2^j + 2^i + m_1 + (2^{j-28} \oplus 2^{i-28} \oplus m_1 \ggg_{28} \oplus 2^{i-12}) + m_2 & \\
 & = & (2^i \oplus 2^j) + (2^{j-28} \oplus 2^{i-28} \oplus m_1 \ggg_{28} \oplus 2^{i-12}) & \\
\Delta a' & = & \Delta m_1 & \text{if } \left\{ \begin{array}{rcll} i & = & j+4 & \Rightarrow \ (\Delta m_1)_{i+1} = 0 \\ i & = & j+12 & \Rightarrow \ (\Delta m_1)_{i+1} = 0 \\ i & = & j+28 & \Rightarrow \ (\Delta m_1)_{i+1} = 0 \end{array} \right. \\
d' & = & \Big((2^i \oplus 2^j) + (2^{j-28} \oplus 2^{i-28} \oplus m_1 \ggg_{28} \oplus 2^{i-12}) & \\
 & & \oplus 2^{j-16} \oplus 2^{i-16} \oplus m_1 \ggg_{16} \Big) \ggg_8 & \\
 & = & 2^{i-8} \oplus 2^{j-8} \oplus 2^{i-4} \oplus 2^{j-4} \oplus 2^{i-20} \oplus 2^{i-24} \oplus 2^{j-24} & \\
\Delta d' & = & 0 & \\
c' & = & (2^{i-8} \oplus 2^{j-8} \oplus 2^{i-4} \oplus 2^{j-4} \oplus 2^{i-20} \oplus 2^{i-24} \oplus 2^{j-24}) + & \\
 & & (2^{j-16} \oplus 2^{i-16} \oplus m_1 \ggg_{16}) & \\
\Delta c' & = & \Delta m_1 & \text{if } \left\{ \begin{array}{rcll} i & = & j+8 & \Rightarrow \ (\Delta m_1)_{i+1} = 0 \\ i & = & j+20 & \Rightarrow \ (\Delta m_1)_{i+1} = 0 \\ i & = & j+24 & \Rightarrow \ (\Delta m_1)_{i+1} = 0 \end{array} \right. \\
b' & = & (c' \oplus b^*) \ggg_7 & \\
\Delta b' & = & (\Delta c' \oplus \Delta b^*) \ggg_7 & \\
 & = & (\Delta m_1 \oplus \Delta m_1) \ggg_7 = 0. &
\end{array}
$$

We conclude that if $i \neq j \mod 4$ we find probability $1/4$ since $(\Delta m_1)_i = (\Delta m_1)_j = 0$ are the only two restrictions. If $i = j \mod 4, i \neq j$ we find probability $1/4$ if $i = j + z$ for $z = 4, 8, 12, 20, 24, 28$ since there are two restrictions, $(\Delta m_1)_i = 0$ and $(\Delta m_1)_{i+1} = 0$. Now there is one possibility left, that is $i = j + 16$, here we only find one restriction, $(\Delta m_1)_i = 0$ and therefore the success probability is $1/2$.

Now we have shown that for any couple $a = 2^i, b = 2^j$ the success probability is at least $1/4$. The same holds for the other couples, we will briefly address all these cases.

Case: $b = d = 0$ We state that Equation C.1 to C.8 hold for $a = 2^i, c = 2^j$ with probability at least $1/4$.

$$
\begin{array}{rcll}
a^* & = & 2^i + m_1 & \\
\Delta a^* & = & \Delta m_1 & \text{if } (\Delta m_1)_i = 0 \\
d^* & = & (2^i \oplus m_1) \ggg_{16} & \\
\Delta d^* & = & \Delta m_1 & \\
c^* & = & (2^{i-16} \oplus m_1 \ggg_{16}) + 2^j & \\
\Delta c^* & = & \Delta m_1 & \text{if } \left\{ \begin{array}{rcrcl} i & = & j+16 & \Rightarrow & (\Delta m_1)_{i+1} = 0 \\ i & \neq & j+16 & \Rightarrow & (\Delta m_1)_i = 0 \end{array} \right. \\
b^* & = & (2^{i-16} \oplus m_1 \ggg_{16} \oplus 2^j) \ggg_{12} & \\
\Delta b^* & = & \Delta m_1 & \\
a' & = & 2^i + m_1 + (2^{i-28} \oplus m_1 \ggg_{28} \oplus 2^{j-12}) + m_2 & \\
 & = & 2^i + (2^{i-28} \oplus m_1 \ggg_{28} \oplus 2^{j-12}) & \\
\Delta a' & = & \Delta m_1 & \text{if } i = j + 20 \Rightarrow (\Delta m_1)_{i+1} = 0 \\
d' & = & \Big(2^i \oplus 2^{i-28} \oplus m_1 \ggg_{28} \oplus 2^{j-12} & \\
 & & \oplus 2^{i-16} \oplus m_1 \ggg_{16}\Big) \ggg_8 & \\
 & = & 2^{i-8} \oplus 2^{i-4} \oplus 2^{j-20} \oplus 2^{i-24} & \\
\Delta d' & = & 0 & \\
c' & = & (2^{i-8} \oplus 2^{i-4} \oplus 2^{j-20} \oplus 2^{i-24} \oplus) + & \\
 & & (2^{i-16} \oplus m_1 \ggg_{16} \oplus 2^j) & \\
\Delta c' & = & \Delta m_1 & \text{if } \left\{ \begin{array}{rcrcl} i & = & j+4 & \Rightarrow & (\Delta m_1)_{i+1} = 0 \\ i & = & j+8 & \Rightarrow & (\Delta m_1)_{i+1} = 0 \\ i & = & j+24 & \Rightarrow & (\Delta m_1)_{i+1} = 0 \\ i & = & j+28 & \Rightarrow & (\Delta m_1)_{i+1} = 0 \end{array} \right. \\
b' & = & (c' \oplus b^*) \ggg_7 & \\
\Delta b' & = & (\Delta c' \oplus \Delta b^*) \ggg_7 & \\
 & = & (\Delta m_1 \oplus \Delta m_1) \ggg_7 = 0. &
\end{array}
$$

We conclude that if $i \neq j \mod 4$ we find probability $1/4$ since $(\Delta m_1)_i = (\Delta m_1)_j = 0$ are the only two restrictions. If $i = j \mod 4$ we find probability $1/4$ if $i = j + z$ for $z = 4, 8, 16, 20, 24, 28$ since there are two restrictions, $(\Delta m_1)_i = 0$ and $(\Delta m_1)_{i+1} = 0$. Now there are two options left, that is $i = j$ and $i = j + 12$, here we only find one restriction[2], $(\Delta m_1)_i = 0$ and therefore the success probability is $1/2$.

[2]Since $(\Delta m_1)_i = (\Delta m_1)_j$ for $i = j \mod 4$.

Case: $b = c = 0$ We state that Equation C.1 to C.8 hold for $a = 2^i, d = 2^j$ with probability at least $1/4$.

$$
\begin{array}{rcll}
a^* & = & 2^i + m_1 & \\
\Delta a^* & = & \Delta m_1 & \text{if } (\Delta m_1)_i = 0 \\
d^* & = & (2^j \oplus 2^i \oplus m_1) \ggg_{16} & \\
\Delta d^* & = & \Delta m_1 & \\
c^* & = & 2^{j-16} \oplus 2^{i-16} \oplus m_1 \ggg_{16} & \\
\Delta c^* & = & \Delta m_1 & \\
b^* & = & (2^{j-16} \oplus 2^{i-16} \oplus m_1 \ggg_{16}) \ggg_{12} & \\
\Delta b^* & = & \Delta m_1 & \\
a' & = & 2^i + m_1 + (2^{j-28} \oplus 2^{i-28} \oplus m_1 \ggg_{28}) + m_2 & \\
 & = & 2^i + (2^{j-28} \oplus 2^{i-28} \oplus m_1 \ggg_{28}) & \\
\Delta a' & = & \Delta m_1 & \text{if } i = j+4 \Rightarrow (\Delta m_1)_{i+1} = 0 \\
d' & = & \Big((2^i \oplus 2^{j-28} \oplus 2^{i-28} \oplus m_1 \ggg_{28}) & \\
 & & \oplus 2^{j-16} \oplus 2^{i-16} \oplus m_1 \ggg_{16}\Big) \ggg_8 & \\
 & = & 2^{i-8} \oplus 2^{i-4} \oplus 2^{j-4} \oplus 2^{i-24} \oplus 2^{j-24} & \\
\Delta d' & = & 0 & \\
c' & = & (2^{i-8} \oplus 2^{i-4} \oplus 2^{j-4} \oplus 2^{i-24} \oplus 2^{j-24}) + & \\
 & & (2^{j-16} \oplus 2^{i-16} \oplus m_1 \ggg_{16}) & \\
\Delta c' & = & \Delta m_1 & \text{if } \begin{cases} i \neq j \mod 4 & \Rightarrow \quad (\Delta m_1)_j = 0 \\ i = j+8 & \Rightarrow \quad (\Delta m_1)_{i+1} = 0 \\ i = j+12 & \Rightarrow \quad (\Delta m_1)_{i+1} = 0 \\ i = j+20 & \Rightarrow \quad (\Delta m_1)_{i+1} = 0 \\ i = j+24 & \Rightarrow \quad (\Delta m_1)_{i+1} = 0 \end{cases} \\
b' & = & (c' \oplus b^*) \ggg_7 & \\
\Delta b' & = & (\Delta c' \oplus \Delta b^*) \ggg_7 & \\
 & = & (\Delta m_1 \oplus \Delta m_1) \ggg_7 = 0. &
\end{array}
$$

We conclude that if $i \neq j \mod 4$ we find probability $1/4$ since $(\Delta m_1)_i = (\Delta m_1)_j = 0$ are the only two restrictions. If $i = j \mod 4$ we find probability $1/4$ if $i = j + z$ for $z = 4, 8, 12, 20, 24$ since there are two restrictions, $(\Delta m_1)_i = 0$ and $(\Delta m_1)_{i+1} = 0$. There are some exceptions to this result, for example, we find probability $1/2$ for $a = 2^{15}, d = 2^3$, since in the second update of c they will produce a carry, but the carry of the most significant bit is $2^{32} = 0$.

$$c' = (2^7 \oplus 2^{11} \oplus 2^{31} \oplus 2^{11} \oplus 2^{23}) + (2^{19} \oplus 2^{31} \oplus m_1 \ggg_{16})$$

The probability for each case is at least $1/4$.
For the cases $z = 0, 16, 28$ only restriction $(\Delta m_1)_i = 0$ is active, and therefore they have probability $1/2$ to be successful.

Case: $a = d = 0$ We state that Equation C.1 to C.8 hold for $b = 2^i, c = 2^j$ with probability at least $1/4$.

$$
\begin{array}{rcll}
a^* & = & m_1 + 2^i & \\
\Delta a^* & = & \Delta m_1 & \text{if } (\Delta m_1)_i = 0 \\
d^* & = & m_1 \ggg_{16} \oplus 2^{i-16} & \\
\Delta d^* & = & \Delta m_1 & \\
c^* & = & 2^j + (m_1 \oplus 2^{i-16}) & \\
\Delta c^* & = & \Delta m_1 & \text{if } \begin{cases} i \neq j \mod 4 \Rightarrow (\Delta m_1)_j = 0 \\ i = j + 16 \Rightarrow (\Delta m_1)_{i+1} = 0 \end{cases} \\
b^* & = & (2^j \oplus m_1 \oplus 2^{i-16} \oplus 2^i) \ggg_{12} & \\
\Delta b^* & = & \Delta m_1 & \\
a' & = & (m_1 \oplus 2^i) + (2^{j-12} \oplus m_1 \oplus 2^{i-28} \oplus 2^{i-12}) + m_2 & \\
 & = & 2^i + (2^{j-12} \oplus m_1 \oplus 2^{i-28} \oplus 2^{i-12}) & \\
\Delta a' & = & \Delta m_1 & \text{if } i = j + 20 \Rightarrow (\Delta m_1)_{i+1} = 0 \\
d' & = & (2^i \oplus 2^{j-12} \oplus m_1 \oplus 2^{i-28} \oplus 2^{i-12} \oplus m_1 \ggg_{16} \oplus 2^{i-16}) \ggg_8 & \\
 & = & 2^{i-8} \oplus 2^{j-20} \oplus 2^{i-4} \oplus 2^{i-20} \oplus 2^{i-24} & \\
\Delta d' & = & 0 & \\
c' & = & (2^{i-8} \oplus 2^{j-20} \oplus 2^{i-4} \oplus 2^{i-20} \oplus 2^{i-24}) + & \\
 & & (2^j \oplus m_1 \oplus 2^{i-16}) & \\
\Delta c' & = & \Delta m_1 & \text{if } \begin{cases} i = j + 4 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = j + 8 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = j + 24 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = j + 28 \Rightarrow (\Delta m_1)_{i+1} = 0 \end{cases} \\
b' & = & (c' \oplus b^*) \ggg_7 & \\
\Delta b' & = & (\Delta c' \oplus \Delta b^*) \ggg_7 & \\
 & = & (\Delta m_1 \oplus \Delta m_1) \ggg_7 = 0. &
\end{array}
$$

We conclude that if $i \neq j \mod 4$ we find probability $1/4$ since $(\Delta m_1)_i = (\Delta m_1)_j = 0$ are the only two restrictions. If $i = j \mod 4$ we find probability $1/4$ if $i = j + z$ for $z = 4, 8, 16, 20, 24, 28$ since there are two restrictions, $(\Delta m_1)_i = 0$ and $(\Delta m_1)_{i+1} = 0$. This does not cover all cases, e.g. if $b = 2^{15}, c = 2^{31}$ we find a carry of 2^{32} in the first update of c:

$$c^* = 2^{31} + (m_1 \oplus 2^{31}) = m_1.$$

Again all cases have probability at least $1/4$.
For the other cases $z = 0, 12$ only restriction $(\Delta m_1)_i = 0$ is active, and therefore they have probability $1/2$ to be successful.

Case: $a = c = 0$ We state that Equation C.1 to C.8 hold for $b = 2^i, d = 2^j$ with probability at least $1/4$.

$$
\begin{array}{rcll}
a^* & = & 2^i + m_1 & \\
\Delta a^* & = & \Delta m_1 & \text{if } (\Delta m_1)_i = 0 \\
d^* & = & m_1 \ggg_{16} \oplus 2^{i-16} \oplus 2^{j-16} & \\
\Delta d^* & = & \Delta m_1 & \\
c^* & = & m_1 \ggg_{16} \oplus 2^{i-16} \oplus 2^{j-16} & \\
\Delta c^* & = & \Delta m_1 & \\
b^* & = & (2^{j-16} \oplus m_1 \oplus 2^{i-16} \oplus 2^i) \ggg_{12} & \\
\Delta b^* & = & \Delta m_1 & \\
a' & = & (m_1 \oplus 2^i) + (2^{j-28} \oplus m_1 \oplus 2^{i-28} \oplus 2^{i-12}) + m_2 & \\
 & = & 2^i + (2^{j-28} \oplus m_1 \oplus 2^{i-28} \oplus 2^{i-12}) & \\
\Delta a' & = & \Delta m_1 & \text{if } i = j+4 \Rightarrow (\Delta m_1)_{i+1} = 0 \\
d' & = & \Big(2^i \oplus 2^{j-28} \oplus m_1 \oplus 2^{i-28} \oplus 2^{i-12} \oplus m_1 \ggg_{16} & \\
 & & \oplus 2^{i-16} \oplus 2^{j-16}\Big) \ggg_8 & \\
 & = & 2^{i-8} \oplus 2^{j-4} \oplus 2^{i-4} \oplus 2^{i-20} \oplus 2^{i-24} \oplus 2^{j-24} & \\
\Delta d' & = & 0 & \\
c' & = & (2^{i-8} \oplus 2^{j-4} \oplus 2^{i-4} \oplus 2^{i-20} \oplus 2^{i-24} \oplus 2^{j-24}) + & \\
 & & (2^{j-16} \oplus m_1 \oplus 2^{i-16}) & \\
\Delta c' & = & \Delta m_1 & \text{if } \begin{cases} i \neq j \mod 4 \Rightarrow (\Delta m_1)_j = 0 \\ i = j+8 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = j+12 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = j+20 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = j+24 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = j+28 \Rightarrow (\Delta m_1)_{i+1} = 0 \end{cases} \\
b' & = & (c' \oplus b^*) \ggg_7 & \\
\Delta b' & = & (\Delta c' \oplus \Delta b^*) \ggg_7 & \\
 & = & (\Delta m_1 \oplus \Delta m_1) \ggg_7 = 0. &
\end{array}
$$

We conclude that if $i \neq j \mod 4$ we find probability $1/4$ since $(\Delta m_1)_i = (\Delta m_1)_j = 0$ are the only two restrictions. If $i = j \mod 4$ we find probability $1/4$ if $i = j + z$ for $z = 4, 8, 12, 20, 24$ since there are two restrictions, $(\Delta m_1)_i = 0$ and $(\Delta m_1)_{i+1} = 0$. This does not cover all cases, e.g. if $b = 2^{15}, c = 2^3$ we find a carry of 2^{32} in the second update of c:

$$c' = (2^{i-8} \oplus \mathbf{2^{31}} \oplus 2^{i-4} \oplus 2^{i-20} \oplus 2^{i-24} \oplus 2^{j-24}) + (2^{j-16} \oplus m_1 \oplus \mathbf{2^{31}}).$$

Again all cases have probability at least $1/4$.
For the other cases $z = 0, 16, 28$ only restriction $(\Delta m_1)_i = 0$ is active, and therefore they have probability $1/2$ to be successful.

Case: $a = b = 0$ We state that Equation C.1 to C.8 hold for $c = 2^i, d = 2^j$ with probability at least $1/4$.

$$
\begin{array}{rcll}
a^* & = & m_1 & \\
\Delta a^* & = & \Delta m_1 & \\
d^* & = & 2^{j-16} \oplus m_1 \ggg_{16} & \\
\Delta d^* & = & \Delta m_1 & \\
c^* & = & (2^{j-16} \oplus m_1) + 2^i & \\
\Delta c^* & = & \Delta m_1 & \text{if } \begin{cases} (\Delta m_1)_i = 0 \\ i = j + 16 \Rightarrow (\Delta m_1)_{i+1} = 0 \end{cases} \\
b^* & = & (2^{j-16} \oplus m_1 \oplus 2^i) \ggg_{12} & \\
\Delta b^* & = & \Delta m_1 & \\
a' & = & m_1 + (2^{j-28} \oplus m_1 \oplus 2^{i-12}) + m_2 & \\
 & = & 2^{j-28} \oplus m_1 \oplus 2^{i-12} & \\
\Delta a' & = & \Delta m_1 & \\
d' & = & (2^{j-16} \oplus m_1 \oplus 2^{j-28} \oplus m_1 \oplus 2^{i-12}) \ggg_8 & \\
 & = & 2^{j-24} \oplus 2^{j-4} \oplus 2^{i-20} & \\
\Delta d' & = & 0 & \\
c' & = & (2^{j-24} \oplus 2^{j-4} \oplus 2^{i-20}) + (2^{j-16} \oplus m_1 \oplus 2^i) & \\
\Delta c' & = & \Delta m_1 & \text{if } \begin{cases} i \neq j \mod 4 \Rightarrow (\Delta m_1)_j = 0 \\ i = j + 4 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = j + 8 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = j + 28 \Rightarrow (\Delta m_1)_{i+1} = 0 \end{cases} \\
b' & = & (c' \oplus b^*) \ggg_7 & \\
\Delta b' & = & (\Delta c' \oplus \Delta b^*) \ggg_7 & \\
 & = & (\Delta m_1 \oplus \Delta m_1) \ggg_7 = 0. &
\end{array}
$$

We conclude that if $i \neq j \mod 4$ we find probability $1/4$ since $(\Delta m_1)_i = (\Delta m_1)_j = 0$ are the only two restrictions. If $i = j \mod 4$ we find probability $1/4$ if $i = j + z$ for $z = 4, 8, 16, 24$ since there are two restrictions, $(\Delta m_1)_i = 0$ and $(\Delta m_1)_{i+1} = 0$. This does not cover all cases, e.g. if $b = 2^{31}, c = 2^{15}$ we find a carry of 2^{32} in the first update of c:

$$c^* = (2^{31} \oplus m_1) + 2^3 1 = m_1$$

Again all cases have probability at least $1/4$.
For the other cases $z = 0, 12, 20, 28$ only restriction $(\Delta m_1)_i = 0$ is active, and therefore they have probability $1/2$ to be successful.

Conclusion

We conclude that we have proven all separate cases and therefore
Lemma 11. *If precisely two of the four input values (a, b, c, d) are zero, and the other two are a power of 2, then*

$$\mathcal{G}_{m_1,m_2}(a, b, c, d) - \mathcal{G}_{m'_1,m'_2}(a, b, c, d) = (\Delta, 0, \Delta', 0)$$

with probability at least $1/4$.

has been proven.

When one input variable is zero.

Lemma 12. *If precisely one of the four input values (a, b, c, d) are zero, and the other three are a power of 2, then*

$$\mathcal{G}_{m_1,m_2}(a, b, c, d) - \mathcal{G}_{m'_1,m'_2}(a, b, c, d) = (\Delta, 0, \Delta', 0)$$

with probability at least $1/8$.

We will again look at the separate cases, $a = 0, b = 0, c = 0$ or $d = 0$.

Case $a = 0$ We state that Equation C.1 to C.8 hold for $b = 2^i, c = 2^j, d = 2^k$ with probability at least $1/8$.

$$
\begin{array}{rcll}
a^* & = & 2^i + m_1 & \\
\Delta a^* & = & \Delta m_1 & \text{if } (\Delta m_1)_i = 0 \\
d^* & = & (2^k \oplus 2^i \oplus m_1) \ggg_{16} & \\
\Delta d^* & = & \Delta m_1 & \\
c^* & = & 2^j + (2^{k-16} \oplus 2^{i-16} \oplus m_1) & \\
\Delta c^* & = & \Delta m_1 & \text{if } \begin{cases} i \neq j \mod 4 \Rightarrow (\Delta m_1)_j = 0 \\ i = j + 16 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ j = k + 16 \Rightarrow (\Delta m_1)_{i+1} = 0 \end{cases} \\
b^* & = & (2^j \oplus 2^{k-16} \oplus 2^{i-16} \oplus m_1 \oplus 2^i) \ggg_{12} & \\
\Delta b^* & = & \Delta m_1 & \\
a' & = & 2^i + (2^{j-12} \oplus 2^{k-28} \oplus 2^{i-28} \oplus m_1 \oplus 2^{i-12}) & \\
\Delta a' & = & \Delta m_1 & \text{if } \begin{cases} i = j + 20 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = k + 4 \Rightarrow (\Delta m_1)_{i+1} = 0 \end{cases} \\
d' & = & \Big(2^{k-16} \oplus 2^{i-16} \oplus m_1 \oplus 2^i \oplus 2^{j-12} & \\
 & & \oplus 2^{k-28} \oplus 2^{i-28} \oplus m_1 \oplus 2^{i-12}\Big) \ggg_8 & \\
 & = & 2^{k-24} \oplus 2^{i-24} \oplus 2^{i-8} \oplus 2^{j-20} \oplus & \\
 & & 2^{k-4} \oplus 2^{i-4} \oplus 2^{i-20} & \\
\Delta d' & = & 0 & \\
c' & = & (2^j \oplus 2^{k-16} \oplus 2^{i-16} \oplus m_1) + & \\
 & & \Big(2^{k-24} \oplus 2^{i-24} \oplus 2^{i-8} \oplus 2^{j-20} \oplus & \\
 & & 2^{k-4} \oplus 2^{i-4} \oplus 2^{i-20}\Big) & \\
\Delta c' & = & \Delta m_1 & \text{if } \begin{cases} i \neq k \mod 4 \bigwedge j \neq k \mod 4 \Rightarrow (\Delta m_1)_k = 0 \\ i = j + 28 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = k + 8 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = k + 24 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ j = k + 4 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ j = k + 8 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ j = k + 28 \Rightarrow (\Delta m_1)_{i+1} = 0 \end{cases} \\
b' & = & (c' \oplus b^*) \ggg_7 & \\
\Delta b' & = & (\Delta c' \oplus \Delta b^*) \ggg_7 & \\
 & = & (\Delta m_1 \oplus \Delta m_1) \ggg_7 = 0. &
\end{array}
$$

If $i \neq j \mod 4$, $i \neq k \mod 4$ and $j \neq k \mod 4$ then we find probability $1/8$ as we need

$$(\Delta m_1)_i = (\Delta m_1)_j = (\Delta m_1)_k = 0.$$

If one of the three equalities above does not hold, the probability is decreased with $1/2$, if two of these do not hold, by $1/4$. Now we consider $i = j + z$, $i = k + \tilde{z}$ and $j = k + y$. Now if $i = j = k \mod 4$ we need $(\Delta m_1)_i = 0$, which has probability $1/2$. Now if $z = 16, 20, 28$ the probability is decreased with a factor $1/2$, then if $\tilde{z} = 4, 8, 24$ the probability is again decreased with a factor $1/2$. Last, if $y = 4, 8, 16, 28$ the probability is again decreased with a factor $1/2$.
For example, we find that $a = 0, b = 2^0, 2^0, 2^0$ has probability $1/2$. And if we have $i \neq j \mod 4$, $i \neq k \mod 4$ and $j = k \mod 4$ and $y = 4$ we find probability $1/8$, unless $j = 21, k = 15$ (since then the carry is 2^{32}).
Thus, in the worst case we find success probability $1/8$.
The other cases are similar, and we will only state the proofs, not the conclusions.

Case $b = 0$ We state that Equation C.1 to C.8 hold for $a = 2^i, c = 2^j, d = 2^k$ with probability at least $1/8$.

$$
\begin{array}{rcll}
a^* & = & 2^i + m_1 & \\
\Delta a^* & = & \Delta m_1 & \text{if } (\Delta m_1)_i = 0 \\
d^* & = & (2^k \oplus 2^i \oplus m_1) \ggg_{16} & \\
\Delta d^* & = & \Delta m_1 & \\
c^* & = & 2^j + (2^{k-16} \oplus 2^{i-16} \oplus m_1) & \\
\Delta c^* & = & \Delta m_1 & \text{if } \left\{ \begin{array}{l} i \neq j \mod 4 \Rightarrow (\Delta m_1)_j = 0 \\ i = j + 16 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ j = k + 16 \Rightarrow (\Delta m_1)_{i+1} = 0 \end{array} \right. \\
b^* & = & (2^j \oplus 2^{k-16} \oplus 2^{i-16} \oplus m_1) \ggg_{12} & \\
\Delta b^* & = & \Delta m_1 & \\
a' & = & 2^i + (2^{j-12} \oplus 2^{k-28} \oplus 2^{i-28} \oplus m_1) & \\
\Delta a' & = & \Delta m_1 & \text{if } \left\{ \begin{array}{l} i = j + 20 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = k + 4 \Rightarrow (\Delta m_1)_{i+1} = 0 \end{array} \right. \\
d' & = & \Big(2^{k-16} \oplus 2^{i-16} \oplus m_1 \oplus 2^i \oplus & \\
 & & 2^{j-12} \oplus 2^{k-28} \oplus 2^{i-28} \oplus m_1\Big) \ggg_8 & \\
 & = & 2^{k-24} \oplus 2^{i-24} \oplus 2^{i-8} \oplus 2^{j-20} \oplus & \\
 & & 2^{k-4} \oplus 2^{i-4} & \\
\Delta d' & = & 0 & \\
c' & = & (2^j \oplus 2^{k-16} \oplus 2^{i-16} \oplus m_1) + & \\
 & & \Big(2^{k-24} \oplus 2^{i-24} \oplus 2^{i-8} \oplus & \\
 & & 2^{j-20} \oplus 2^{k-4} \oplus 2^{i-4}\Big) & \\
\Delta c' & = & \Delta m_1 & \text{if } \left\{ \begin{array}{l} i \neq k \mod 4 \bigwedge j \neq k \mod 4 \Rightarrow (\Delta m_1)_k = 0 \\ i = j + 8 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = j + 28 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = k + 12 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = k + 24 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ j = k + 4 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ j = k + 8 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ j = k + 28 \Rightarrow (\Delta m_1)_{i+1} = 0 \end{array} \right. \\
b' & = & (c' \oplus b^*) \ggg_7 & \\
\Delta b' & = & (\Delta c' \oplus \Delta b^*) \ggg_7 & \\
 & = & (\Delta m_1 \oplus \Delta m_1) \ggg_7 = 0. &
\end{array}
$$

Case $c = 0$ We state that Equation C.1 to C.8 hold for $a = 2^i, b = 2^j, d = 2^k$ with probability at least $1/8$.

$$
\begin{aligned}
a^* &= 2^i + 2^j + m_1 \\
\Delta a^* &= \Delta m_1 && \text{if } \begin{cases} i = j \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i \neq j \Rightarrow (\Delta m_1)_i = 0 \bigwedge (\Delta m_1)_j = 0 \end{cases} \\
d^* &= (2^k \oplus 2^i \oplus 2^j \oplus m_1) \ggg_{16} \\
\Delta d^* &= \Delta m_1 \\
c^* &= 2^{k-16} \oplus 2^{i-16} \oplus 2^{j-16} \oplus m_1 \\
\Delta c^* &= \Delta m_1 \\
b^* &= (2^j \oplus 2^{k-16} \oplus 2^{i-16} \oplus 2^{j-16} \oplus m_1) \ggg_{12} \\
\Delta b^* &= \Delta m_1 \\
a' &= (2^i \oplus 2^j) + (2^{j-12} \oplus 2^{k-28} \oplus 2^{i-28} \oplus 2^{j-28} \oplus m_1) \\
\Delta a' &= \Delta m_1 && \text{if } \begin{cases} i = j+4 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = j+20 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = j+28 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = k+4 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ j = k+4 \Rightarrow (\Delta m_1)_{i+1} = 0 \end{cases} \\
d' &= \Big(2^{k-16} \oplus 2^{i-16} \oplus 2^{j-16} \oplus m_1 \oplus 2^i \oplus \\
&\quad 2^j \oplus 2^{j-12} \oplus 2^{k-28} \oplus 2^{i-28} \oplus m_1\Big) \ggg_8 \\
&= 2^{k-24} \oplus 2^{i-24} \oplus 2^{j-24} \oplus 2^{i-8} \oplus \\
&\quad 2^{j-8} \oplus 2^{j-20} \oplus 2^{k-4} \oplus 2^{i-4} \\
\Delta d' &= 0 \\
c' &= (2^j \oplus 2^{k-16} \oplus 2^{i-16} \oplus m_1) + \\
&\quad \Big(2^{k-24} \oplus 2^{i-24} \oplus 2^{j-24} \oplus 2^{i-8} \oplus \\
&\quad 2^{j-8} \oplus 2^{j-20} \oplus 2^{k-4} \oplus 2^{i-4}\Big) \\
\Delta c' &= \Delta m_1 && \text{if } \begin{cases} i \neq k \mod 4 \bigwedge j \neq k \mod 4 \Rightarrow (\Delta m_1)_k = 0 \\ i = j+8 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = j+24 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = k+8 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = k+12 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = k+20 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = k+24 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ j = k+8 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ j = k+24 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ j = k+28 \Rightarrow (\Delta m_1)_{i+1} = 0 \end{cases} \\
b' &= (c' \oplus b^*) \ggg_7 \\
\Delta b' &= (\Delta c' \oplus \Delta b^*) \ggg_7 \\
&= (\Delta m_1 \oplus \Delta m_1) \ggg_7 = 0.
\end{aligned}
$$

Case $d = 0$ We state that Equation C.1 to C.8 hold for $a = 2^i, b = 2^j, c = 2^k$ with probability at least $1/8$.

$$
\begin{aligned}
a^* &= 2^i + 2^j + m_1 \\
\Delta a^* &= \Delta m_1 && \text{if } \begin{cases} i = j \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i \neq j \Rightarrow (\Delta m_1)_i = 0 \bigwedge (\Delta m_1)_j = 0 \end{cases} \\
d^* &= (2^i \oplus 2^j \oplus m_1) \ggg_{16} \\
\Delta d^* &= \Delta m_1 \\
c^* &= 2^k + (2^{i-16} \oplus 2^{j-16} \oplus m_1) \\
\Delta c^* &= \Delta m_1 \text{if } \begin{cases} i = k \mod 4 \Rightarrow (\Delta m_1)_k = 0 \\ i = k + 16 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ j = k + 16 \Rightarrow (\Delta m_1)_{k+1} = 0 \end{cases} \\
b^* &= (2^j \oplus 2^k \oplus 2^{i-16} \oplus 2^{j-16} \oplus m_1) \ggg_{12} \\
\Delta b^* &= \Delta m_1 \\
a' &= (2^i \oplus 2^j) + (2^{j-12} \oplus 2^{k-12} \oplus 2^{i-28} \oplus 2^{j-28} \oplus m_1) \\
\Delta a' &= \Delta m_1 && \text{if } \begin{cases} i = j \mod 4 \bigwedge k = j \mod 4 \Rightarrow (\Delta m_1)_j = 0 \\ i = j + 4 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = j + 20 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = j + 28 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = k + 20 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ j = k + 20 \Rightarrow (\Delta m_1)_{i+1} = 0 \end{cases} \\
d' &= \Big(2^{i-16} \oplus 2^{j-16} \oplus m_1 \oplus 2^i \oplus 2^j \oplus \\
&\quad 2^{j-12} \oplus 2^{k-12} \oplus 2^{i-28} \oplus 2^{j-28} \oplus m_1\Big) \ggg_8 \\
&= 2^{i-24} \oplus 2^{j-24} \oplus 2^{i-8} \oplus 2^{j-8} \oplus \\
&\quad 2^{j-20} \oplus 2^{k-20} \oplus 2^{i-4} \oplus 2^{j-4} \\
\Delta d' &= 0 \\
c' &= (2^k \oplus 2^{i-16} \oplus 2^{j-16} \oplus m_1) + \\
&\quad \Big(2^{i-24} \oplus 2^{j-24} \oplus 2^{i-8} \oplus 2^{j-8} \oplus \\
&\quad 2^{j-20} \oplus 2^{k-20} \oplus 2^{i-4} \oplus 2^{j-4}\Big) \\
\Delta c' &= \Delta m_1 && \text{if } \begin{cases} i \neq k \mod 4 \bigwedge j \neq k \mod 4 \Rightarrow (\Delta m_1)_k = 0 \\ i = j + 8 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = j + 12 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = j + 24 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = k + 8 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = k + 12 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = k + 24 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ j = k + 4 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ j = k + 8 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ j = k + 24 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ j = k + 28 \Rightarrow (\Delta m_1)_{i+1} = 0 \end{cases} \\
b' &= (c' \oplus b^*) \ggg_7 \\
\Delta b' &= (\Delta c' \oplus \Delta b^*) \ggg_7 \\
&= (\Delta m_1 \oplus \Delta m_1) \ggg_7 = 0.
\end{aligned}
$$

Conclusion

We conclude that we have proven all separate cases and therefore

Lemma 13. *If precisely one of the four input values (a, b, c, d) are zero, and the other three are a power of 2, then*

$$\mathcal{G}_{m_1,m_2}(a, b, c, d) - \mathcal{G}_{m'_1,m'_2}(a, b, c, d) = (\Delta, 0, \Delta', 0)$$

with probability at least $1/8$.

has been proven.

No zero input variables

Lemma 14. *If precisely none of the input variables is zero and all of them are a power of 2, then*

$$\mathcal{G}_{m_1,m_2}(a,b,c,d) - \mathcal{G}_{m_1',m_2'}(a,b,c,d) = (\Delta, 0, \Delta', 0)$$

with probability at least $1/16$.

We state that Equation C.1 to C.8 hold for $a = 2^i, b = 2^j, c = 2^k, d = 2^l$ with probability at least $1/16$.

$$\begin{aligned}
a^* &= 2^i + 2^j + m_1 \\
\Delta a^* &= \Delta m_1 \qquad \text{if} \begin{cases} i = j \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i \neq j \Rightarrow (\Delta m_1)_i = 0 \bigwedge (\Delta m_1)_j = 0 \end{cases} \\
d^* &= (2^l \oplus 2^i \oplus 2^j \oplus m_1) \ggg_{16} \\
\Delta d^* &= \Delta m_1 \\
c^* &= 2^k + (2^{l-16} \oplus 2^{i-16} \oplus 2^{j-16} \oplus m_1) \\
\Delta c^* &= \Delta m_1 \qquad \text{if} \begin{cases} i \neq k \mod 4 \bigwedge j \neq k \mod 4 \Rightarrow (\Delta m_1)_k = 0 \\ i = k+16 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ j = k+16 \Rightarrow (\Delta m_1)_{j+1} = 0 \\ k = l+16 \Rightarrow (\Delta m_1)_{k+1} = 0 \end{cases} \\
b^* &= (2^j \oplus 2^k \oplus 2^{l-16} \oplus 2^{i-16} \oplus 2^{j-16} \oplus m_1) \ggg_{12} \\
\Delta b^* &= \Delta m_1 \\
a' &= 2^i + 2^j + m_1 + (2^{j-12} \oplus 2^{k-12} \oplus 2^{l-28} \oplus 2^{i-28} \oplus 2^{j-28} \oplus m_1) + m_2 \\
&= 2^i + 2^j + (2^{j-12} \oplus 2^{k-12} \oplus 2^{l-28} \oplus 2^{i-28} \oplus 2^{j-28} \oplus m_1) \\
\Delta a' &= \Delta m_1 \qquad \text{if} \begin{cases} i = j+4 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = j+20 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = j+28 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = k+20 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = l+4 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ j = k+20 \Rightarrow (\Delta m_1)_{j+1} = 0 \\ j = l+4 \Rightarrow (\Delta m_1)_{k+1} = 0 \end{cases} \\
d' &= \big(2^i \oplus 2^j \oplus 2^{j-12} \oplus 2^{k-12} \oplus 2^{l-28} \oplus 2^{i-28} \oplus 2^{j-28} \oplus m_1 \\
&\quad 2^{l-16} \oplus 2^{i-16} \oplus 2^{j-16} \oplus m_1\big) \ggg_8 \\
&= \big(2^i \oplus 2^j \oplus 2^{j-12} \oplus 2^{k-12} \oplus 2^{l-28} \oplus 2^{i-28} \oplus 2^{j-28} \\
&\quad 2^{l-16} \oplus 2^{i-16} \oplus 2^{j-16}\big) \ggg_8 \\
\Delta d' &= 0 \\
c' &= (2^k \oplus 2^{l-16} \oplus 2^{i-16} \oplus 2^{j-16} \oplus m_1) + \big(2^{l-24} \oplus 2^{i-24} \oplus 2^{j-24} \oplus \\
&\quad 2^{i-8} \oplus 2^{j-8} \oplus 2^{j-20} \oplus 2^{k-20} \oplus 2^{l-4} \oplus 2^{i-4} \oplus 2^{j-4}\big) \\
\Delta c' &= \Delta m_1 \qquad \text{if} \begin{cases} i = j+8 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = j+12 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = j+24 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = k+4 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = k+8 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = k+24 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = k+28 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = l+8 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = l+12 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = l+20 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ i = l+24 \Rightarrow (\Delta m_1)_{i+1} = 0 \\ j = k+4 \Rightarrow (\Delta m_1)_{j+1} = 0 \\ j = k+8 \Rightarrow (\Delta m_1)_{j+1} = 0 \\ j = k+24 \Rightarrow (\Delta m_1)_{j+1} = 0 \\ j = k+28 \Rightarrow (\Delta m_1)_{j+1} = 0 \\ j = l+8 \Rightarrow (\Delta m_1)_{k+1} = 0 \\ j = l+12 \Rightarrow (\Delta m_1)_{k+1} = 0 \\ j = l+20 \Rightarrow (\Delta m_1)_{k+1} = 0 \\ j = l+24 \Rightarrow (\Delta m_1)_{k+1} = 0 \\ k = l+4 \Rightarrow (\Delta m_1)_{k+1} = 0 \\ k = l+8 \Rightarrow (\Delta m_1)_{k+1} = 0 \\ k = l+28 \Rightarrow (\Delta m_1)_{k+1} = 0 \end{cases} \\
b' &= (c' \oplus b^*) \ggg_7 \\
\Delta b' &= (\Delta c' \oplus \Delta b^*) \ggg_7 \\
&= (\Delta m_1 \oplus \Delta m_1) \ggg_7 = 0.
\end{aligned}$$

In total we find at least probability $1/16$ if at least one of i, j, k, l is zero. We will give a few examples of special cases.

Probability $1/16$ Obviously if none of i, j, k, l is equal to another modulo 4, we find that

$$(\Delta m_1)_i = (\Delta m_1)_j = (\Delta m_1)_k = (\Delta m_1)_l = 0$$

so $m_1 = m_2 = 0$ is the only possibility and this has probability $1/16$.

Probability $1/8$ If precisely one of i, j, k, l is equal to another modulo 4, say $i = j \mod 4$, we find that

$$(\Delta m_1)_i = (\Delta m_1)_j = (\Delta m_1)_k = (\Delta m_1)_l = 0$$

with probability $1/8$, as there is one of the four bits in the repeated set of m_1 that can be either 0 or 1. There are some exceptions in this case, i.e. if one of the restrictions given above is satisfied. Therefore, we will give an example for which none of the restrictions hold: $a = 2^1, b = 2^2, c = 2^3, d = 2^5$ gives probability $1/8$.

Probability $1/4$ If $i = k \mod 4$ and $j = l \mod 4$, there are cases for which

$$(\Delta m_1)_i = (\Delta m_1)_j = (\Delta m_1)_k = (\Delta m_1)_l = 0$$

with probability $1/4$, as there are two of the four bits in the repeated set of m_1 that can be either 0 or 1. There are many exceptions in this case, i.e. if one of the restrictions given above is satisfied. Therefore, we will give an example for which none of the restrictions hold: $a = 2^1, b = 2^2, c = 2^5, d = 2^2$ gives probability $1/4$

Probability $1/2$ For $l = 0$ we found 14 starting states which have probability $1/2$ to end in $(\Delta, 0, \Delta', 0)$. All of these have at least $i = 0$ or $k = 0$, unless the caries are 2^{32}, which happens for $i = j = k = 31$.
The fourteen states for $l = 0$ are:

i	j	k	l
12	0	0	0
20	0	0	0
4	0	4	0
16	0	4	0
16	0	16	0
16	0	24	0
0	4	4	0
0	12	12	0
0	16	4	0
0	16	16	0
0	16	24	0
0	20	8	0
0	20	20	0
31	31	31	0

Conclusion

We conclude that
Lemma 15. *If precisely none of the input variables is zero and all of them are a power of 2, then*

$$\mathcal{G}_{m_1,m_2}(a, b, c, d) - \mathcal{G}_{m_1',m_2'}(a, b, c, d) = (\Delta, 0, \Delta', 0)$$

with probability at least $1/16$.

has been proven.

C.4 Construction of Fixed Point Algorithms

Here we will explain how to construct the fixed point algorithms and we will give some examples of fixed points.

Linearization $\bar{\mathcal{G}}$

The eight steps of the linearization of $\bar{\mathcal{G}}$ are given by

$$\begin{array}{llll} 1. & a^* = a \oplus b \oplus m_1 & 5. & a = a^* \oplus b^* \oplus m_2 \\ 2. & d^* = (a^* \oplus d) \ggg_{16} & 6. & d = (a \oplus d^*) \ggg_8 \\ 3. & c^* = c \oplus d^* & 7. & c = c^* \oplus d \\ 4. & b^* = (b \oplus c^*) \ggg_{12} & 8. & b = (b^* \oplus c) \ggg_7 . \end{array}$$

Now following these eight steps we construct the algorithm, for fixed b, b^*.

$$\begin{array}{llll} 8. & c = b^* \oplus b \lll_7 & 4. & c^* = b^* \lll_{12} \oplus b \\ 7. & d = c^* \oplus c & 3. & d^* = c^* \oplus c \\ 6. & a = d^* \oplus d \lll_8 & 2. & a* = d^* \lll_{16} \oplus d \\ 5. & m_2 = a^* \oplus a \oplus b^* & 1. & m_1 = a^* \oplus a \oplus b \end{array}$$

Examples

b	b^*	a	b	c	d	m_1	m_2
0	0	0	0	0	0	0	0
0	1	1052929	0	1	4097	269549824	269549825
1	0	33153	1	128	129	8487169	8487168
2^{16}	0	2172715008	65536	8388608	8454144	2164326529	2164260993
0	2^{16}	285278224	0	65536	268500992	16781329	16846865

Original $\mathcal{G}$

The eight steps of $\bar{\mathcal{G}}$ are given by

$$\begin{array}{llll} 1. & a^* = a + b + m_1 & 5. & a = a^* + b^* + m_2 \\ 2. & d^* = (a^* \oplus d) \ggg_{16} & 6. & d = (a \oplus d^*) \ggg_8 \\ 3. & c^* = c + d^* & 7. & c = c^* + d \\ 4. & b^* = (b \oplus c^*) \ggg_{12} & 8. & b = (b^* \oplus c) \ggg_7 . \end{array}$$

Now following these eight steps we construct the algorithm, for fixed b, b^*.

$$\begin{array}{llll} 8. & c = b^* \oplus b \lll_7 & 4. & c^* = b^* \lll_{12} \oplus b \\ 7. & d = c - c^* & 3. & d^* = c^* - c \\ 6. & a = d^* \oplus d \lll_8 & 2. & a* = d^* \lll_{16} \oplus d \\ 5. & m_2 = a - a^* - b^* & 1. & m_1 = a^* - a - b. \end{array}$$

Examples

b	b^*	a	b	c	d	m_1	m_2
0	0	0	0	0	0	0	0
0	1	4293922304	0	1	4294963201	4027638273	267329022
1	0	4294934657	1	128	127	4286742270	8225025
2^{16}	0	2155937792	65536	8388608	8323072	2147352449	2147549311
0	2^{16}	251592944	0	65536	4026597376	3775008527	519893233

C.5 Proof on DC's

Lemma 16.

$$DP^{+}[\Delta_j - \Delta_i, \Delta_i \to \Delta_j] = 2^{-||\Delta_i|| - ||\Delta_j - \Delta_i||}.$$

Proof. Again we assume that the MSB's are 0, otherwise we should multiply the solution by two.

$$DP^{+}[\Delta_j - \Delta_i, \Delta_i \to \Delta_j] = \Pr_{x,y}[(x+y) \oplus ((x \oplus (\Delta_j - \Delta_i)) + (y \oplus \Delta_i)) = \Delta_j].$$

Let $\Delta_i^{\text{MSB}} = \Delta_j^{\text{MSB}} = 0$. Now if $(x+y)$ and

$$((x \oplus (\Delta_j - \Delta_i)) + (y \oplus \Delta_i)).$$

have the same set of carry's we have equality. So for each bit k we have

$$\begin{aligned} (\Delta_j - \Delta_i)^k = \Delta_i^k = 0 \quad &\text{if} \quad x^k = y^k = 1 \\ (\Delta_j - \Delta_i)^k = 0 \quad &\text{if} \quad y^k = 1 \\ \Delta_i^k = 0 \quad &\text{if} \quad x^k = 1. \end{aligned}$$

Therefore, we have two out of four options for x^k, y^k if $(\Delta_j - \Delta_i)^k$ and Δ_i^k are given and therefore we have the same carry for each bit with probability $1/2$ for randomly chosen x, y. When the bits $(\Delta_j - \Delta_i)^k$ and Δ_i^k are zero, we have no restrictions for x and y. We conclude that if either $(\Delta_j - \Delta_i)^k = 1$ or $\Delta_i^k = 1$ we loose half of the options. Therefore

$$DP^{+}[\Delta_j - \Delta_i, \Delta_i \to \Delta_j] = 2^{-||\Delta_i|| - ||\Delta_j - \Delta_i||}.$$

Note that we again excluded the most significant bits, since the carry of the MSB's will disappear due to the modular addition. So to be precise, we have

$$DP^{+}[\Delta_j - \Delta_i, \Delta_i \to \Delta_j] = 2^{-||\Delta_i|| - ||\Delta_j - \Delta_i|| + \Delta_i^{\text{MSB}} + (\Delta_j - \Delta_i)^{\text{MSB}}}.$$

□

Appendix D

Proofs on combinations of operations

Note that all these proofs are covered by Sokolowski [38] or Daum [14].

under (left–)Shift

Let $x = x_0||x_1$ and $y = y_0||y_1$, where x_0, y_0 consist of j bits and x_1, y_1 of $n - j = k$ bits.

Addition

$\Pr[x \ll_j + y \ll_j = (x + y) \ll_j] = 1$

Proof.

$$\begin{aligned}
\Pr[(x+y) \ll_j = x \ll_j + y \ll_j] &= \Pr[(x+y) \ll_j = 2^j x_0 \mod 2^n + 2^j x_1 \mod 2^n] \\
&= \Pr[(x_0 \cdot 2^j + x_1 + y_0 \cdot 2^j + y_1) \ll_j = (2^j x_0 + 2^j x_1) \mod 2^n] \\
&= \Pr[(x_1 + y_1) \ll_j = (2^j x_0 + 2^j x_1) \mod 2^n] \\
&= \Pr[2^j (x_1 + y_1) \mod 2^n = (2^j x_0 + 2^j x_1) \mod 2^n] \\
&= \Pr[2^j x_1 + 2^j y_1 \mod 2^n = (2^j x_0 + 2^j x_1) \mod 2^n] \\
&= 1
\end{aligned}$$

□

Multiplication

$\Pr[\mathcal{U}(x \ll_j) = \mathcal{U}(x) \ll_j] = 1$

Proof.

$$\begin{aligned}
\Pr[\mathcal{U}(x \ll_j) = \mathcal{U}(x) \ll_j] &= \Pr[3(x \ll_j) \mod 2^n = 3x \ll_j \mod 2^n] \\
&= \Pr[3(2^j x) \mod 2^n = 2^j \cdot 3x \mod 2^n] \\
&= \Pr[2^j \cdot 3x \mod 2^n = 2^j \cdot 3x \mod 2^n] \\
&= 1
\end{aligned}$$

□

$\Pr[\mathcal{V}(x \ll_j) = \mathcal{V}(x) \ll_j] = 1$

Proof.

$$\begin{aligned}\Pr[\mathcal{V}(x \ll_j) = \mathcal{V}(x) \ll_j] &= \Pr[5(x \ll_j) \mod 2^n = 5x \ll_j \mod 2^n] \\ &= \Pr[5(2^j x) \mod 2^n = 2^j \cdot 5x \mod 2^n] \\ &= \Pr[2^j \cdot 5x \mod 2^n = 2^j \cdot 5x \mod 2^n] \\ &= 1\end{aligned}$$

□

Bitwise Operations

$\Pr[x \ll_j \wedge \overline{y \ll_j} = (x \wedge \overline{y}) \ll_j] = 1$

Proof.

$$\begin{aligned}\Pr[x \ll_j \wedge \overline{y \ll_j} = (x \wedge \overline{y}) \ll_j] &= \Pr[\forall i : x_i \wedge \overline{y_i} = x_i \wedge \overline{y_i}] \\ &= 1\end{aligned}$$

□

under (right–)Shift

Recall that $x = x_0||x_1$ and $y = y_0||y_1$. Let c_0 the carry of $x_0 + y_0$, that is, $c_0 \cdot 2^j + (x_0 + y_0 \mod 2^j) = x_0 + y_0$ and c_1 is the carry of $x_1 + y_1$.

Addition

$$\Pr[x \gg_k + y \gg_k = (x + y) \gg_k] = \frac{1}{4}(1 + 2^{-k})(1 + 2^{k-n})$$

Proof. We distinguish three cases: $c_0 = 1$, $c_0 = 0 \wedge c_1 = 1$ and $c_0 = 0 \wedge c_1 = 0$.
Case 1:$c_0 = 1$
In this case we never have equality due to the carry of $x_0 + y_0$:

$$\begin{aligned}x \gg_k + y \gg_k &= x_0 + y_0 \\ &\geq x_0 + y_0 - c_0 \cdot 2^k + c_1 \\ &\geq x_0 + y_0 + c_1 \mod 2^k \\ &= (x + y) \gg_k\end{aligned}$$

Case 2:$c_0 = 0 \wedge c_1 = 1$
In this case we never have equality due to the carry of $x_1 + y_1$.

$$\begin{aligned}x \gg_k + y \gg_k &= x_0 + y_0 \\ &\neq x_0 + y_0 + c_1 \mod 2^j \\ &= (x + y) \gg_k\end{aligned}$$

Case 3: $c_0 = 0 \wedge c_1 = 0$
In this case we have equality with probability 1:

$$\begin{aligned} x \gg_k + y \gg_k &= x_0 + y_0 \\ &= x_0 + y_0 + c_1 \mod 2^j \\ &= (x+y) \gg_k \end{aligned}$$

So we have

$$\Pr[x \gg_k + y \gg_k = (x+y) \gg_k] = \Pr[c_0 = 0 \wedge c_1 = 0]$$

$$\begin{aligned} \Pr[c_0 = 0 \wedge c_1 = 0] &= \Pr[x_0 + y_0 < 2^j \wedge x_1 + y_1 < 2^k] \\ &= \Pr[x_0 + y_0 < 2^j] \Pr[x_1 + y_1 < 2^k] \end{aligned}$$

And

$$\begin{aligned} \Pr[x_0 + y_0 < 2^j] &= 2^{-2j} \cdot \sum_{i_1=0}^{2^j} \sum_{i_2=0}^{2^j} \mathbb{1}[i_1 + i_2 < 2^j] \\ &= 2^{-2j} \cdot \sum_{i=0}^{2^j} 2^j - i \\ &= 2^{-2j} \cdot 2^{2j} + 2^j - 1/2 \left(2^j(1+2^j)\right) \\ &= 2^{-2j} \cdot \tfrac{1}{2}(2^{2j} + 2^j) \\ &= \tfrac{1}{2}(1 + 2^{-j}) \end{aligned}$$

And thus we find

$$\begin{aligned} \Pr[x \gg_k + y \gg_k = (x+y) \gg_k] &= \Pr[c_0 = 0 \wedge c_1 = 0] \\ &= \tfrac{1}{2}(1 + 2^{-j}) \cdot \tfrac{1}{2}(1 + 2^{-k}) \\ &= \tfrac{1}{4}(1 + 2^{-k})(1 + 2^{k-n}) \end{aligned}$$

□

under Rotation

Addition

$\Pr[x \lll_j + y \lll_j = (x+y) \lll_j] = \frac{1}{4}(1 + 2^{-j})(1 + 2^{j-n})$

Proof.

$$\begin{aligned} \Pr[x \lll_j + y \lll_j = (x+y) \lll_j] &= \Pr[x \ll_j + x \gg_{n-j} + y \ll_j + y \gg_{n-j} = (x+y) \ll_j + (x+y) \gg_{n-j}] \\ &= \Pr[x \gg_{n-j} + y \gg_{n-j} = (x+y) \gg_{n-j}] \\ &= \tfrac{1}{4}(1 + 2^{-j})(1 + 2^{j-n}) \end{aligned}$$

□

Index