Master's Thesis

Repairing Boomerang Characteristics

Wolfgang Wieser
0831773

Graz, 2013

Institute for Applied Information Processing and Communications
Graz University of Technology

Assessor: Florian Mendel

Advisor: Martin Schläffer

Abstract

Differential cryptanalysis is one of the two most widely used attacks on block ciphers and hash functions. Therefore, the algorithm is applied on pairs of messages with a certain difference for instance. Then the differences on output of the algorithm can be used to determine information about the secret key. The boomerang attack extends the classic differential cryptanalysis by using differences of differences. There exist related attacks like the amplified boomerang attack and the rectangle attack. These attacks belong to the family of second-order cryptanalysis. With these techniques round reduced versions of various cryptographic functions have been broken. It even has been possible to break the full eight rounds of the KASUMI block cipher. In the last years several boomerang characteristics for a round reduced version of SHACAL-2, which is a block cipher based on SHA-2, were published. For all of them contradictions were found - manually. In this thesis we describe a tool which can automatically check the consistency of differential boomerang characteristics. We developed different methods to check the consistency of boomerang characteristics automatically.

Differential characteristics can be seen as a huge set of constraints. If a characteristic is inconsistent, this means that a contradiction is somewhere in the set of constraints. We propose debugging techniques to find these contradictions and to repair them. In this thesis we describe four algorithms to find inconsistencies in boomerang characteristics. We describe different techniques we tested to improve the results. Different detail-levels can be used for the checks while repairing the characteristics. We tried to find a good trade-off between speed and quality of the result. These algorithms and techniques can also be used to repair first-order differential characteristics.

The repairing algorithms provide truncated boomerang characteristics which can be used as a starting point to determine the characteristic and search confirming messages. Finally, we describe how to search for boomerang characteristics with a high probability. With this tool valid characteristics can be found automatically which helps to increase the number of attacked rounds.

Zusammenfassung

Differentielle Kryptoanalyse ist eine der zwei häufigsten Verfahren um kryptographische Blockchiffre und Hashfunktion anzugreifen. Dafür wird der Algorithmus zum Beispiel auf Paare von Nachrichten mit einer bestimmten Differenz angewendet, um aus der Differenz am Ausgang Informationen über den geheimen Schlüssel zu berechnen. Die Boomerang Attacke erweitert die klassische differentielle Kryptoanalyse indem sie Differenzen von Differenzen verwendet. Außer der Boomerang Attacke gibt es noch weitere Methoden, wie etwa die Amplified Boomerang Attacke und die Rectangle Attacke. Diese Angriffsmethoden gehören zur Kryptoanalyse zweiter Ordnung. Mit ihnen wurden bereits rundenreduzierte Varianten von einigen kryptographische Funktionen gebrochen. Die KASUMI Blockchiffre konnte sogar vollständig gebrochen werden.

In den letzten Jahren wurden einige Boomerang Charakteristiken für eine rundenreduzierte Variante von SHACAL-2, einer Blockchiffre basierend auf SHA-2, veröffentlicht. Allerdings konnte in allen Charakteristiken ein Fehler gefunden werden - allerdings händisch. Um die Überprüfung solcher Charakteristiken zu vereinfachen, haben wir Methoden entwickelt, um die Überprüfung zu automatisieren, die in dieser Arbeit beschrieben werden.

Differentielle Charakteristiken können als große Menge von Bedingungen gesehen werden. Widersprechen sich manche dieser Bedingungen, ist die Charakteristik inkonsistent. Daher verwenden wir Techniken die bereits zur Fehlersuche in Soft- und Hardware bentzt werden, um diese Widersprüche zu finden und sie zu beheben. In dieser Arbeit beschreiben wir vier Algorithmen die dazu verwendet werden können. Außerdem beschreiben wir zusätzliche Methoden um die damit erzielten Ergebnisse zu verbessern. Dabei können die dafür benötigten Überprüfungen der Charakteristik mit verschiedenen Genauigkeitsstufen durchgeführt werden. Wir haben versucht einen guten Kompromiss zwischen Geschwindigkeit und Qualität zu finden. Die vorgestellten Techniken können auch zum Reparieren von Charakteristiken erster Ordnung verwendet werden.

Nach dem die Charakteristiken repariert wurden, sind einige ihrer Differenzen unbestimmt. Diese Werte können mit Hilfe einer Suche bestimmt werden. Mit der Suche können auch passende Nachrichten gefunden werden. Dabei sollte die Charakteristik am Ende eine möglichst hohe Wahrscheinlichkeit besitzen. In dieser Arbeit beschreiben wir wie Charakteristiken mit einer möglichst hohen Wahrscheinlichkeit automatisch gefunden werden können. Die vorgestellten Algorithmen und Methoden sollen helfen in Zukunft Boomerang Attacken auf mehr Runden durchführen zu können.

Acknowledgments

At first I would like to thank my supervisor Martin Schläffer to guide me through a huge part of my studies starting with my bachelor thesis. I would like to thank him for your many ideas, hints and showing me different exciting aspects of cryptography. Many thanks also to my second supervisor Florian Mendel for his suggestions which helped me a lot during this thesis. I would also like to thank Prof. Andreas Felfernig, Prof. Franz Wotawa and Birigit Hofer from the Institute for Software Technology for introducing me to debugging in their lectures and exercises, which gave me the idea to use debugging to repair differential characteristics.

Equally, I would like to thank my family for their support enabling me to concentrate entirely on my studies. I would like to thank all my fellow students for making the studying pleasant even in stressful times and supporting me in group works and learning for exams. Special thanks for great team work and joining me in most exercises go to Simon Außerlechner, Raphael Spörk and Clemens Mühlbacher. Last but not least I want to thank my girlfriend Fabia for supporting and pushing me to finish my thesis and her understanding in all the stressful times.

Contents

1 Introduction

Everyone uses cryptography but only a few people notice. All current smartphones have a complex security system to protect the user data and the conversations. Another practical examples for cryptography in everyday life are debit cards and online banking. Today even contracts can be signed digitally. To guarantee the authenticity of such contracts or to protect user data many different cryptographic algorithms have been developed. In cryptanalysis these algorithms are analyzed to find vulnerabilities and learn from them to create new stronger structures and algorithms. This is necessary because according to Moore's law computational power grows exponentially and so does the ability to do more complex attacks.

One of the most powerful forms of cryptanalysis is differential cryptanalysis, which can be divided into first-order and higher-order differential-cryptanalysis. For higher-order differential cryptanalysis several attacks are known, one of them is the boomerang attack. This attack uses four messages to find collisions or to distinguish an algorithm from a random function. In this thesis we apply the boomerang attack on SHACAL-2. Target of the differential cryptanalysis is to find differentials. Differentials predict the difference between two outputs of a function depending on the input difference with a certain probability, which should be as high as possible. Since recent cryptographic algorithms are designed in a way to counter differential cryptanalysis, finding differentials and characteristics with a high probability is a challenging task.

However, it is not only a hard task to find a good characteristic, it is also difficult to check if a differential characteristic is valid. For a boomerang attack even four characteristics are necessary. These characteristics do not only need to be consistent within themselves. They have to fulfill additionally constraints because they are strongly related to each other. We developed a tool to automate the consistency check of a whole boomerang characteristic.

In case of a contradiction we have to localize the constraints which lead to the conflict and change them to repair the characteristic. Therefore we use different debugging algorithms to find conflict sets or diagnoses and to change the contained constraints. Once a valid characteristic with a reasonable probability has been constructed, messages fulfilling these characteristics must be found. The higher the probability of a characteristic the easier it is to find a message.

In this thesis we implement a tool to automate the boomerang attack. Therefore, we developed algorithms to check whether a given boomerang characteristic is valid. If inconsistencies can be found they can be repaired automatically using different debugging algorithms. When the characteristic is repaired confirming messages can be searched to ensure that the characteristic is indeed valid.

In Chapter 2, we give an introduction into symmetric cryptography. One group of symmetric algorithms are block ciphers which are used to encrypt a bulk of data. One of them is SHACAL-2, which is based on the hash function SHA-2, an international standard hash function designed by the U.S. National Security Agency (NSA). Old hash functions

are based on block ciphers. Therefore, the block cipher is used in different constructions to form a non-invertible function. Since block ciphers are only suitable for encrypting a message with a fixed length, several structures were developed to securely encrypt longer messages. Additionally, we describe some basic attacks.

In Chapter 3, we describe the concept of differential cryptanalysis. It was developed to attack block ciphers and hash functions but can also be applied on other cryptographic functions. Differential cryptanalysis looks at differences of messages and analyze their influence on the output. For some input differences some differences on the output are more likely then the average. This can be used to generate differentials and characteristics. In this chapter, we also describe some techniques which can be used to improve the probability of characteristics in some attack settings.

In Chapter 4, the boomerang attack is introduced. The boomerang attack is a method of higher-order differential cryptanalysis and therefore analyzes differences of differences. There exist related attacks like the amplified boomerang attack, the rectangle attack and the boomerang attack on hash functions.

In Chapter 5, we introduce the usage of debugging algorithms to repair differential characteristics. Since differential characteristics can be seen as a huge set of constraints, debugging algorithms can help to find inconsistencies. Lots of different debugging algorithms are known, we selected ReplayXPlain, QuickXPlain and FastDiag and developed SequentialSearch. In this chapter we describe these algorithms and how they can be used to find diagnoses which can be used to repair characteristics.

Chapter 6, describes how first-order differential characteristics can be updated, checked and searched on. Conditions describe the difference between the values in two messages at the same position. Generalized conditions can be used to store possible combinations of differences. Thus, generalized multi-bit conditions represent the possible differences for more than one bit value such as a result of an addition. The relation between two conditions can be described with two-bit conditions.

In Chapter 7, we extend these concepts to the boomerang attack and implemented them in a tool. We describe how a change in one part of the boomerang characteristics can be propagated to the other parts and how the validity of boomerang characteristics can be proven. Moreover, we explain how boomerang characteristics can be repaired using the debugging algorithms described before. These basic debugging strategies can be extended by a statistical approach and by debugging all possible rotations of the boomerang characteristics. This chapter also describes how we can search on boomerang characteristics and how this can be used as extended consistency checks.

In Chapter 8, we apply the technique described before on recently published boomerang characteristics for a round reduced variant of SHACAL-2. All of them have been proven to have inconsistencies. Hence, we check all the characteristics with our tool to test if it can find the inconsistencies automatically and how long these checks need. Then we choose one characteristic and repair it with different debugging algorithms and various setups to compare the performance and to find the best setup. Finally, we search a new boomerang characteristic.

In Chapter 9, we summarize the results and discuss open problems.

2 Symmetric Cryptography

There are two classes of algorithms in cryptography: public-key and private-key, also called symmetric-key, algorithms. Public-key algorithms use a public encryption key and a secret decryption key. This has the big advantage that the encryption key must not be kept secret and therefore everyone can generate the ciphertext but only the recipient can decrypt it. A big disadvantage of those algorithms it that they are very slow, because they have to work on they whole message at once and thus have to work on very big numbers. Therefore, for encryption of a bulk of data symmetric-key algorithms are used. These algorithms can divide the data into blocks of arbitrary size. As a result they are much faster. However, they use for encryption and decryption the same key. Therefore, the key for encryption must transmitted on a secure channel and stored in a way that no unauthorized party can gain access to the key. To solve this problem in practice hybrid cryptosystem are used. Such systems use public-key cryptography to transfer the secret key and use the transfered one-time key to encrypt and decrypt the messages with a symmetric-key algorithm.

2.1 Block Cipher

Beside stream ciphers, block ciphers belong to symmetric-key algorithms. Hence, for encryption and decryption the same key must be used. Symmetric key algorithms are widely used to encrypt bulk data, because they are faster than asymmetric key algorithms. Block ciphers are deterministic algorithms working on fixed-length groups of bits, so called blocks. Therefore the input message also called plaintext P is divided into several blocks $p_1, \ldots, p_n$ with length l. If the message length is not a multiple of l the last block must be padded.

Simplified can be said that each message block is substituted by a random looking but deterministically calculated value. Therefore on each block the transformation function is applied. Typically the transformation function iterates a weaker round function several times and each iteration is called a round. The round function itself is a function of the output of the previous round and a subkey generated from the master key using a key scheduling algorithm. More generally said, a block cipher $E(\cdot)$ can be seen as mapping from all possible plaintexts P and all possible keys K to all possible ciphertexts C so that $E : P \times K \to C$. When p, c and k are specific elements of P, C and K, then the encryption is defined as $c = E_k(p)$ and the decryption is defined as $p = E_k^{-1}(c)$.

One widespread implementation of block ciphers is the Feistel network, which becomes well known due to its usage in DES [Des77]. Another common structure is the substitution-permutation network, used by several common algorithms, including AES [NIS01].

If each block would be calculated independently, identical plaintext blocks would result in identical ciphertext blocks. When the ciphertext contains a pattern, the resulting ciphertext also contains a pattern and an attacker thereby may deduce some information about the plaintext. Therefore, the encryption of each block uses information from the previous encryption to randomize the output in a deterministic way to hide patterns. Many modes of operations have been suggested, in the following we shortly describe the four most common modes.

Electronic codebook (ECB): In this mode each block is encrypted separately. The only advantage is that encryption and decryption can be completely parallelized. Encryption is defines as $c_i = E_k(p_i)$ and decryption by $p_i = E_k^{-1}(c_i)$.

Cipher–block chaining (CBC): In this mode each block of plaintext is XORed with the previous block of ciphertext. For the first block the initial vector IV is used instead. Because the encryption of the previous message block is needed to calculate the next one, encryption cannot be parallelized. In return each ciphertext block depends on all previous blocks. Decryption can be done parallel for all blocks because already all ciphertext blocks are known. A block of the ciphertext can be calculated by $c_i = E_k(p_i \oplus c_{i-1})$ and decrypted by $p_i = E_k^{-1}(c_i) \oplus c_{i-1}$, where $c_0 = IV$.

Cipher feedback (CFB): In this mode only the previous block of ciphertext is used as input for encryption and then the result is XORed with the block of plaintext. Thus a block of ciphertext is defined as $c_i = E_k(c_{i-1}) \oplus p_i$ and can be decrypted by $p_i = E_k(c_{i-1}) \oplus c_i$. Thus for decryption no inverse implementation of the encryption function is necessary. As in CBC decryption can be parallelized while encryption cannot.

Output feedback (OFB): In this mode the output of the encryption function is used as input for the next encryption. Thus the chaining values do not depend on the plaintext and can be precalculated. Moreover, encryption and decryption works identically. Encryption is defined as $c_i = p_i \oplus o_i$ and decryption is defined as $p_i = c_i \oplus o_i$, where $o_i = E_k(i_{i-1})$ and $o_0 = IV$. The output blocks can be seen as key stream. Thus, in this mode a block cipher works very similar as an stream cipher.

2.2 Hash Function

A hash function H is a deterministic algorithm that maps an input message m of any length to a fixed length output message $h = H(m)$, the so called hash value. Because the set of possible inputs is larger than the set of possible output, a hash function is surjective. This means the same hash value can be produced by more than one input. Thus, a hash function is not invertible. Moreover, for a good hash function the output should look random. This means, if a single bit flips in the input, the hash value should be very different from the previous hash value. Because the hash value changes strongly on a little change on the input, hash functions can be used for checksums, as pseudo random number generator or to calculate an index in a hash table. Other applications are message authentication, digital signatures and password protection.

Hash functions often have a similar structure as block ciphers. Moreover, they are often even based on block ciphers. Thus, methods of cryptanalysis for block ciphers can also be applied on hash functions in a very similar way. Hence, for a long time hash functions were attacked like block ciphers, although hash functions do not have a secret key.

2.2.1 Security Notions

Because the mapping from input to output should be evenly distributed, every hash value should be generated with the same probability. Therefore it should be hard to find two different input messages mapping to the same hash value. From these facts, three properties can be derived: preimage resistance, second-preimage resistance and collision resistance. Let $H(\cdot,\cdot)$ be an iterated hash function generating an n-bit hash value h_i depending on the previous hash value h_{i-1} and a l-bit message block m_i. Then the compression function is defined by $h_i = H(h_{i-1}, m_i)$. For the first iteration h_{i-1} is the initial vector IV defined by the specification of the hash function. Following definitions for compression function attacks are based on the definitions given in [LM93, LIS12].

Collision attack: Since the hash value has a fixed length and the message can have any length, many messages map to the same hash value. Thus, a collision attack in terms of a hash function means finding two messages (m_i, m_i'), yielding the same hash value from given IV, so that $m_i \neq m_i'$ but $H(IV, m_i) = H(IV, m_i')$.

Semi-free-start collision attack: This type of collision attack allows additionally to choose h_{i-1} so that $m_i \neq m_i'$ but $H(h_{i-1}, m_i) = H(h_{i-1}, m_i')$.

Free-start or pseudo collision attack: For a pseudo collision attack the hash function is called with different h_{i-1} and/or different messages result in the same hash value. Thus, $(h_{i-1}, m_i) \neq (h_{i-1}', m_i')$ but $H(h_{i-1}, p_i) = H(h_{i-1}', p_i')$.

Preimage attack: While for collision attacks hash value can be chosen freely, for preimage attacks also the hash value has a fixed value $h_{i,fixed}$. Thus, for a preimage attack a message m_i must be found, so that $H(IV, m_i) = h_{i,fixed}$.

Pseudo preimage attack: For a pseudo preimage the hash function is called with different h_{i-1} and/or different messages result in the same fixed hash value. Thus, $(h_{i-1}, m_i) \neq (h_{i-1}', m_i')$ but $H(h_{i-1}, p_i) = H(h_{i-1}', p_i') = h_{i,fixed}$.

Partial preimage attack: For a partial preimage attack only some bits of the hash value are fixed. Thus, for this attack a second message m_i' must be found, so that these fixed bits of the calculated hash value have the same value as in $h_{i,fixed}$. The rest of the hash value $h_i' = H(IV, m_i)$ can have random values.

Second-Preimage attack: In this attack a collision for a fixed hash value is searched, so that $m_i \neq m_i'$ but $H(IV, m_i) = H(IV, m_i') = h_{i,fixed}$.

Since finding collisions cannot be avoided, a cryptographic hash function is called ideal if the generic bounds are met. The generic bounds are defined by the number of message

which must be hashed until the attack was succesfull. For a preimage or second-preimage attack at least 2^n and for a collision attack at least $2^{n/2}$ messages must be tried, where n is the length of the hash value.

2.2.2 One-Way Compression functions

As mentioned before hash functions are often based on block ciphers. There exists three common variants of chaining.

Davies–Meyer: This one-way compression functions uses the message block m_i as key and the previous output h_{i-1} as input for the encryption function $E(\cdot)$. Additionally the new output is XORed with the previous output so that $h_i = E_{m_i}(h_{i-1}) \oplus h_{i-1}$. Using this construction in each iteration k message bits can be processed, where k is the key size of the encryption function [Win84].

Matyas–Meyer–Oseas: It uses the message block m_i as input for the encryption function and the output of the previous iteration h_{i-1} as key. Additionally, the ciphertext is XORed with the message block so that $h_i = E_{h_{i-1}}(m_i) \oplus m_i$. This construction can process n message bits in each iteration, where n is the block size of the used cipher. If the cipher uses different block and key lengths, h_{i-1} would not fit as key. Therefore, this construction uses an additional function $g(\cdot)$ to convert the size accordingly [MMO85].

Miyaguchi–Preneel: This one-way compression function extends the Matyas–Meyer construction by an additional XOR operation. The ciphertext is additional XORed with the output of the previous encryption h_{i-1} so that $h_i = E_{h_{i-1}}(m_i) \oplus m_i \oplus h_{i-1}$. This construction also needs an additional function $g(\cdot)$ to convert the size of h_{i-1} to fit the key size of the used encryption function [MIO89, PGV94].

In Figure 1 from left to right plain encryption, Davies–Meyer, Matyas–Meyer–Oseas, Miyaguchi–Preneel are shown. For example SHA-2 works very similarly as SHACAL-2 in Davies–Meyer mode, which will be described in Section 2.4.

These compression functions can generate only hash values with a length up to the length of the ciphertext of the used encryption function. Thus they are called single-block-length compression functions. There are also other compression functions which can generate hash value with a multiple length of the ciphertext. They are called multi-block-length compression functions. To generate a larger hash value they do in each iteration multiple encryptions and combine the results. One example is MDC-2, which is shown in Figure 2 [BCH+90].

One-way compression functions often are used in Merkle-Damgård construction. This construction builds hash functions from one-way compression functions. Widely known algorithms using the Merkle-Damgård construction are MD5, SHA-1 and SHA-2.

To use an encryption function as hash function in a Merkle-Damgård construction, the

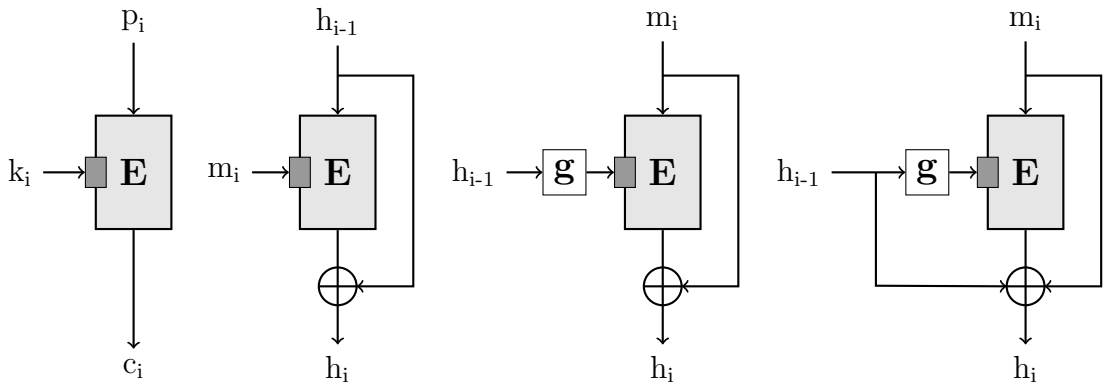

Figure 1: A encryption function in different constructions

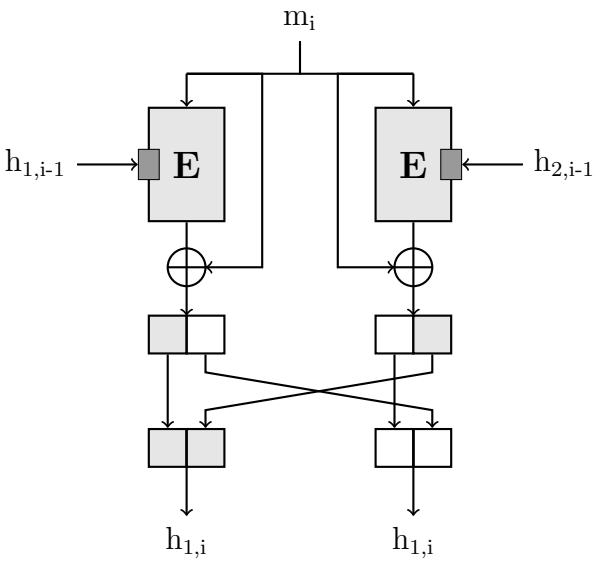

Figure 2: MDC-2 compression function

algorithm has to define an initial vector IV which is used as h_0. The input message with length l is divided into blocks of length n. If l is not a multiple of n, the message must be padded. Therefore, different padding schemes are known, typically the message is padded with zeros. To prevent for instance two message M_1 ="message" and M_2 ="message0" resulting in the same hash value, the length of the message is added to message within padding process [Mer79]. For each block the one-way compression function C is applied. After the last round a final function F may be is executed. An illustration of the construction is shown in Figure 3. For this construction several attacks like multi-collisions and length-extension are known [Jou04, CDMP05].

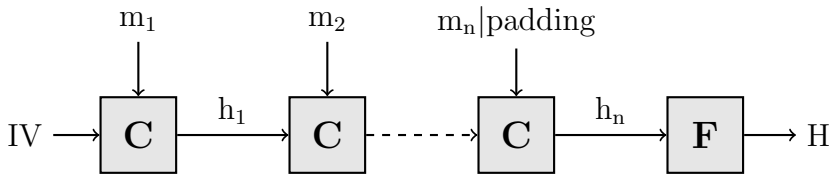

Figure 3: Merkle-Damgård construction

Hash functions based on block ciphers are typically slower than specially designed hash functions. This is because of the key scheduling in encryption function. Additionally, the security is based on the security of the encryption function. This includes the key scheduling and the block size. A weak key scheduling can lead to fix points or key collisions and a small block size reduces the complexity of attacks. As a result in the last years new special designed hash functions were published and a new standard for hash functions was searched in the SHA-3 competition.

2.3 Attack Techniques

In this section, we summarize two basic cryptographic attacks and two alternatives to differential cryptanalysis which will be presented in Section 3.

2.3.1 Birthday Attack

The generic complexity of finding a collision in an ideal random function is $2^{n/2}$, where n is the size of the output of the function. This complexity results from the birthday attack: The cryptographic function is applied on q randomly distributed messages and the results are stored in a list. Because of the birthday paradox after $q \approx 1.18\sqrt{n}$ messages a collision can be found [Yuv79]. The birthday paradox states that the probability p of finding two persons with the same birthday, increases non linearly with the number N of asked persons [FO90]:

$$p(N) \approx 1 - e^{-\frac{N^2}{2^{n+1}}}. \tag{1}$$

It is important that the colliding day is not fixed. Thus, in the birthday attack the collision is searched for any output of the function and not for a specific. Note that, searching a collision for a fixed output corresponds to a second-preimage attack.

2.3.2 Meet-In-The-Middle Attack

For a meet-in-the-middle attack a cryptographic function E_k using a key k is divided into two sub functions so that $E_k = E_{k,bw} \circ E_{k,fw}$. The aim of the attack is to find a pair of plaintext and ciphertext (p, c) resulting in the same intermediate value $v = E_{k,fw}(p) = E^{-1}_{k,bw}(c)$ is valid. For such a pair $E_k(p) = c$ is valid. In the first step of the attack all possible values for v are calculated, iterating over all possible plaintexts or keys. Then in

the second step, for all possible ciphertexts or keys again v is calculated until a match is found. Going forwards and backwards through a function at the same time can reduce the complexity of an attack dramatically [DH77].

Zhu and Gong introduced a multidimensional version in [ZG11]. This attack can be used if the cryptographic function consists of several sub-functions using different keys so that $c = E_{k_n}(\dots E_{k_2}(E_{k_1}(p))\dots)$. In the simplest case two different keys k_1 and k_2 are used. The intermediate value g is the value between $E_{k_1} = E_{k_1,bw} \circ E_{k_1,fw}$ and $E_{k_2} = E_{k_2,bw} \circ E_{k_2,fw}$. For all possible values of g the intermediate values $v_1' = E_{k_1,bw}^{-1}(g)$ and $v_2' = E_{k_2,fw}(c)$ Then for all possible keys or plaintexts and ciphertexts the values for $v_1 = E_{k_1,fw}(p)$ and $v_2 = E_{k_2,bw}^{-1}(c)$ are calculated until a match is found. Figure 4 shows the structure of both variants.

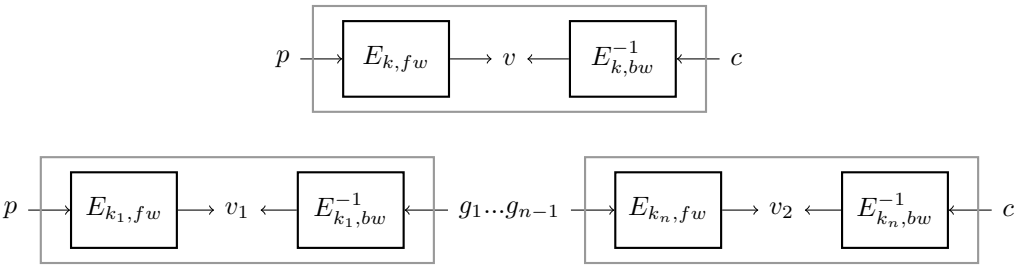

Figure 4: Structure of the single and multi dimensional meet-in-the-middle-attack

2.3.3 Linear Cryptanalysis

Linear cryptanalysis was first applied on a block cipher in [MY93]. To be fast most encryption algorithms have linear parts using operations like XOR. However, a block cipher also consists of non linear parts like an s-box. An s-box substitutes an input value by a fixed value and is often implemented as lookup table. Such a function should disguise the relation between the key and the ciphertext. Thus, the mapping should be as random as possible and is therefore nonlinear. Linear cryptanalysis tries to find some linear approximations which hold with a high probability for such nonlinear functions. In most cases this is not possible for complete words but often single bits have nearly linear relation. This relations may be used to reveal some information about the secret key.

2.3.4 Integral Cryptanalysis

This attack is also known as Square attack because it was first proposed to attack the Square block cipher [DKR97]. Integral Cryptanalysis can be applied on cryptographic functions based on substitution-permutation networks. The saturation attack is a more generalized form and can also be applied on functions having a Feistel network structure [Luc00]. In contrast to differential cryptanalysis integral cryptanalysis does not use pairs of messages but sets of messages. The messages within this set differ only in a specific

part, the rest is constant for all messages. In addition, it is balanced if the XOR-sum of all contained messages is 0. Then for instance the XOR-sums of the corresponding set of ciphertexts can be used to deduce information whether a guessed key was correct [DKR97].

2.4 SHACAL-2

SHACAL-2 was designed 2000 by Handschuh and Naccache [HN00]. It was selected by NESSIE (New European Schemes for Signatures, Integrity and Encryption). NESSIE is a European research project with the target to provide proved and secure cryptographic algorithms. The algorithm is a symmetric block cipher and is based on the SHA-2 hash function. The compression function of SHA-2 is invertible, thus its structure can also be used for decryption. SHACAL-2 uses the SHA-2 compression function without feed-forward in encryption mode. Therefore, the key is used as message and the plaintext is used as initial value, by ignoring the final addition with the initial values. Thus, SHACAL-2 is a 256-bit block cipher and its key can have up to 512 bits and should have a minimum length of 128 bit. If the key is shorter then 512, it is padded with zeros.

2.4.1 Definition of SHA-2

The structure of SHACAL-2 is defined as follows:

Message Expansion: Each 512-bit message input block gets divided into 16 words M_i where $i = 0, ...15$. They are used to calculate 64 expanded message words W_i as follows:

$$W_i = \begin{cases} M_i & 0 \leq i<16 \\ \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15} + W_{i-16} & 16 \leq i<64 \end{cases} \tag{2}$$

Thereby, modular additions modulo 2^{32} are used. The functions $\sigma_0(X)$ and $\sigma_1(X)$ are given following equation, whereby $X \ggg n$ means a rotation of word X of size w by n bits $(X \ggg n = (X \gg n) \vee (X \ll (w - n)))$:

$$\begin{aligned} \sigma_0(X) &= (X \gg 3) \oplus (X \ggg 7) \oplus (X \ggg 18) \\ \sigma_1(X) &= (X \gg 10) \oplus (X \ggg 17) \oplus (X \ggg 19) \end{aligned} \tag{3}$$

State Update Transformation: Before each round the state h_i is initialized with a new block 256-bit block of the message. Then 64 rounds of the step function are performed to update the state. The state consists of eight 32-bit words $A_i, ..., H_i$, where i indicates the step. In the step function a constant K_i is used. The definition of the step function is as

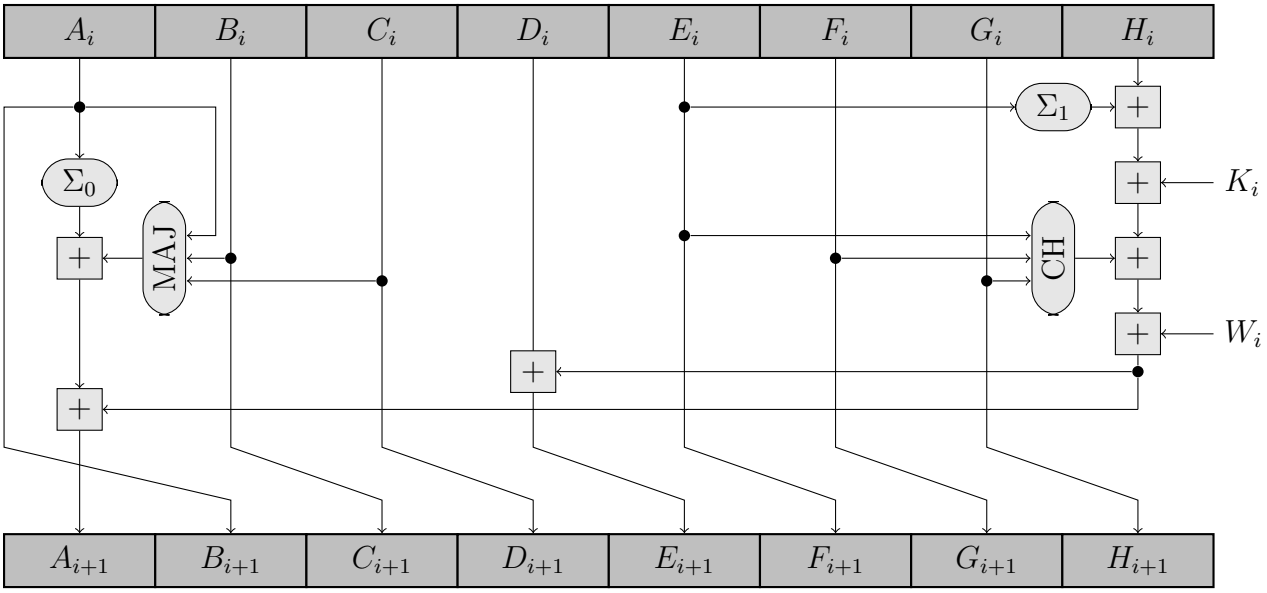

Figure 5: Step function of SHACAL-2

follows:

$$
\begin{aligned}
&MAJ(X,Y,Z) = X \wedge Y \oplus Y \wedge Z \oplus X \wedge Z \\
&CH(X,Y,Z) = X \wedge Y \oplus \neg X \wedge Z \\
\\
&\Sigma_0 = (X \ggg 2) \oplus (X \ggg 13) \oplus (X \ggg 22) \\
&\Sigma_1 = (X \ggg 6) \oplus (X \ggg 11) \oplus (X \ggg 25) \\
\\
&T_1 = H_i + \Sigma_1(E_i) + CH(E_i, F_i, G_i) + K_i + W_i \\
&T_2 = \Sigma_0(A_i) + MAJ(A_i, B_i, C_i) \\
\\
&A_{i+1} = T_1 + T_2, \quad B_{i+1} = A_i, \quad C_{i+1} = B_i, \quad D_{i+1} = C_i \\
&E_{i+1} = D_i + T_1, \quad F_{i+1} = E_i, \quad G_{i+1} = F_i, \quad H_{i+1} = G_i.
\end{aligned}
\tag{4}
$$

Figure 5 shows the step function of SHACAL-2. In contrast to SHA-2 the initial state value for SHACAL-2 is not added to the output state after the last step of the state update transformation. More detailed descriptions of SHA-2 and SHACAL-2 are given in [Nat08, HN00].

2.4.2 Current Attacks

In the last years several boomerang characteristics for a round reduced version of SHACAL-2 were published [KKL+05, LKKD06, Wan07, LK08, FGL09]. But for all of them inconsis-

tencies have been found in the switch manually [BLMN11]. See Tables 21-23 in Appendix A for the definition of these characteristics. More details about the terms used below can be found in Section 3.

It must be considered that the used boomerang characteristics are not independent. Thus, for some bits, the conditions cannot be satisfied in both characteristics at the same time. In [LK08], bit 13 in E_{25} in one characteristic has a difference. Thus, there must be the same signed difference in the same bit in F_{25} and G_{25}. In the next step F_{25} becomes G_{26} and G_{25} becomes H_{26}. As a result bit 13 of G_{26} and H_{26} have the same signed difference. The characteristics requires these two differences canceling out each other, which is not possible, because their difference have the same sign. In this way we get a contradiction. For the characteristics defined in [LKKD06, Wan07, FGL09], the conflict also lies in bit 13 of the switch. In a similar way, a contradiction for the characteristics defined in [KKL$^+$05] can be found [BLMN11].

In [HKK$^+$03] an impossible differential attack on 30 rounds of SHACAL-2 was published. The authors extend an 11-round impossible differential characteristic to a 14-round distinguisher. Using this distinguishers, they are able to attack 30-round of SHACAL-2 with a data complexity of 744 chosen plaintexts and a time complexity of $2^{495.1}$ encryptions. For more details see [HKK$^+$03]. There exist also other attacks for reduced versions of SHACAL-2. For a complete list see Table 1.

Table 1: Comparison of attacks on SHACAL-2

Attack	Rounds	Data	Time	Memory
Square [SKK$^+$04]	28	$463 \cdot 2^{32}$ CP	$2^{494.1}$	$2^{45.9}$
Impossible Differential [HKK$^+$03]	30	744 CP	$2^{495.1}$	$2^{14.5}$
Differential [SKK$^+$04]	32	$2^{43.4}$ CP	$2^{504.2}$	$2^{48.4}$
RK Differential [KKL$^+$05]	35	$2^{42.32}$ RK-CP	$2^{452.1}$	$2^{47.32}$
RK Rectangle †[KKL$^+$05]	37	$2^{235.16}$ RK-CP	$2^{486.95}$	$2^{240.16}$
RK Boomerang †[FGL09]	39	$2^{3.5}$ RK-CPCC	$2^{483.5}$	$2^{8.5}$
RK Rectangle †[LKKD06]	40	$2^{243.38}$ RK-CP	$2^{448.43}$	$2^{247.38}$
RK Rectangle †[LKKD06]	42	$2^{243.38}$ RK-CP	$2^{488.37}$	$2^{247.38}$
RK Rectangle †[Wan07]	43	$2^{240.38}$ RK-CP	$2^{480.4}$	$2^{245.38}$
RK Rectangle †[LK08]	44	2^{233} RK-CP	$2^{497.2}$	2^{238}

CP . . . needed pairs of ciphertexts
RK . . . related-key
CC . . . chosen ciphertexts
†. . . the used characteristic contains conflicts

3 Differential Cryptanalysis

The word cryptanalysis derives from the Greek words kryptós (hidden) and análysis (to loosen) [WB01]. Thus, it stands for the science of studying the hidden aspects of information systems. Cryptanalysis is used to break cryptographic systems and obtain information of secret messages and the used cryptographic keys. To reach this target, different forms of attacks are known. Some of them are linear cryptanalysis, integral cryptanalysis or side channel attacks. A further form is the differential cryptanalysis, which is one of the most powerful attack strategies.

First-order differential cryptanalysis or short differential cryptanalysis can be applied on block ciphers, stream ciphers and hash functions. These attacks can be used to distinguish an unknown algorithm from possible candidates, to deduce information about the secret key or to find collisions. Differential cryptanalysis is typically a statistical attack using chosen plaintexts. It was first published by Eli Biham and Adi Shamir 1991 in [BS91]. However, it was already known to NSA and IBM during the development of DES 1975 [Cop94]. Thus, the S-box of DES was chosen in a way to counter differential cryptanalysis. The basic idea of differential cryptanalysis is to analyze the effect of differences in plaintext pairs on the resulting output pair [BS91].

Differential cryptanalysis was first applied on a hash function in 1993 to attack MD5 [dBB94]. The target of most attacks on hash functions using differential cryptanalysis is finding a collision. In this case, two different messages should result in the same output. An advantage compared to differential cryptanalysis on block ciphers is, that for hash functions message modification can be used (see Section 3.3.1).

3.1 Preliminaries

Difference: In most cases a difference of two elements x and x' is defined by an XOR difference $x \oplus x'$. Generally the difference for any group is defined as $x \bullet x'^{-1}$, where x'^{-1} is the inverse of x' and $\bullet$ the group operator [LMM91]. But there are also other types of differences. In designs using modular addition a modular difference ($x \boxminus x'$) is used instead [Dob98]. In [WY05] signed bit differences were presented. Generalizing signed bit differences allow to use generalized conditions [DCR06] which will be presented in Section 6.1.1. In the remainder of this thesis the notation $\Delta(x, x')$ stands for the difference of x and x'. If $x \neq x'$, the according bit is called active, otherwise it is called inactive.

For non linear functions, like an S-box, for some input differences the probability for a certain output difference deviates from average. This fact can be used to predict a certain difference on the output of the non linear function. In addition, sometimes some differences have a probability of zero, thus they are impossible. Such impossible differences are used in impossible differential cryptanalysis [Knu98].

In differential cryptanalysis usually differences over several number of rounds are used.

Differences over several rounds can be represented by characteristics or differentials.

Characteristic: A characteristic is a sequence of predicted differences after each round of the cryptographic function. The sequence starts with the difference of the plaintexts. Each entry corresponds to a one-round characteristic. For each of them the probability can be estimated. The probability p of the whole characteristic is the product of the probabilities of all involved one-round characteristics. This calculation is based on the assumption that the involved one-round characteristics are independent from each other. For most cases, this assumption is not valid. However, it is a good approximation if the characteristic is sparse, thus has few active bits. Furthermore, the complexity of the attack can be estimated by $\mathcal{O}(p^{-1})$. Differential characteristics sometimes are also called differential trail or differential path.

Differential: A differential is a collection of characteristics, which share the same input and output differences. Thus, a differential typically defines only the difference of the plaintext and the ciphertext. If X is an input difference and Y is an output difference after i rounds of a function, then the pair (X, Y) called is a differential. As a result, in general the probability of a differential is higher than the probability of a characteristic. On the other hand, a characteristic can be seen as sequence of one round differentials.

3.2 The Attack

In this section, we explain the concept of a cryptographic attack using differential cryptanalysis. Therefore, we assume a block cipher $E_{k_1||\ldots||k_n}(\cdot)$ using n sub-keys $k_1, \ldots, k_n$ to encrypt a plaintext p. The block cipher consists of n steps and for each step a step function $F(\cdot)$ is applied on the former state s_{i-1} and defines the new state s_i where s_0 is the plaintext. In each step a subkey k_i derived from a master key with a key scheduling algorithm is used. Thus, the resulting ciphertext c is defined as $c = E_{k_1||\ldots||k_n}(p) = F_{k_n}(\ldots F_{k_2}(F_{k_1}(p)))$. Because we use differential cryptanalysis we use a pair of plaintexts (p_1, p_2) which result in a pair of ciphertexts (c_1, c_2). The structure of the example is shown in Figure 6.

To find the keys, we introduce some differences. The difference on the input is defined by $\Delta p = p_0 \oplus p_1$. We also define differences of the states $\Delta s_i = s_{0,i} \oplus s_{1,i}$. These differences are part of the differential characteristic.

Then the attack consists of two parts. In the first step we search a differential characteristic with a probability as high as possible and suitable pairs of plaintexts are generated. In the second step, keys are guessed and filtered by looking if they fulfill the probability of the characteristic.

Searching a differential characteristic: The probability for a difference on the output of the step function depends on difference on the input. For an ideal step function all output differences are equally likely. However, for real step functions for certain input differences some output differences are more or less likely than the average. For illustrattion we look

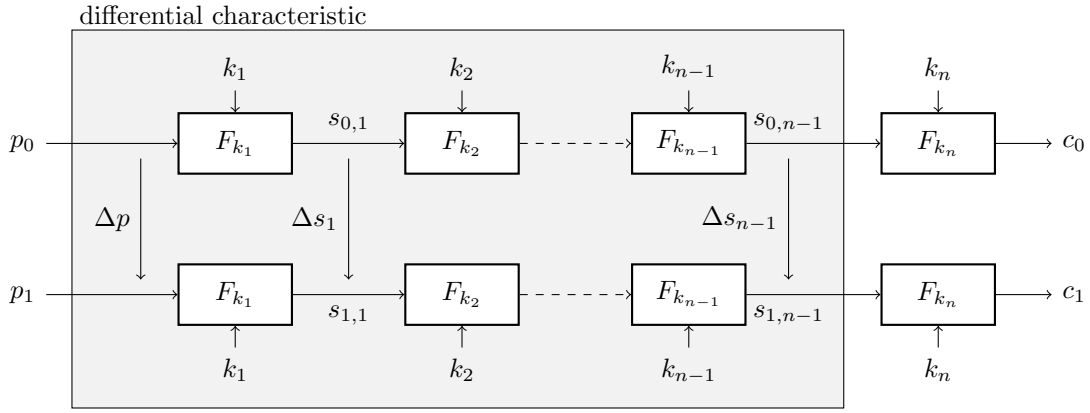

Figure 6: Exemplary structure for differential attack

at the difference distribution table of the 4-bit s-box used by PRESENT [BKL+07] which is shown in Table 2. A difference distribution table can be generated by sending all possible input differences with all possible values through the s-box and count the differences on output. The result is shown in Table 3. Each line represents an input difference and each column an output difference. As one can see, if there is no difference on the input for all possible values no difference occur on the output. The maximum number in the table is four, thus the maximum probability for a output difference for a specific input difference is $p_{max} = \frac{4}{16} = \frac{1}{4}$. For instance for the input difference 0xF the probability to get an output difference of 0xF is $\frac{1}{4}$. If we assume for our exemplary encryption function the same difference distribution table, we could use a characteristic $\{0xF \rightarrow 0xF \ldots 0xF \rightarrow 0xF\}$ which would result in a total probability of $p = (\frac{1}{4})^{n-1}$. Aim of this step is finding the characteristic with the highest probability. Once a good characteristic is found several pairs of plaintext with the initial difference of the characteristic must be generated and encrypted to get ciphertext pairs belonging to them.

Table 2: PRESENT S-Box

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
S(x)	6	4	c	5	0	7	2	e	1	f	3	d	8	a	9	b

Filter key candidates: In the second phase we guess a key k_n and calculate from all pairs of ciphertexts one step backwards to determine their difference $\Delta s_{n-1} = F_{k_n}^{-1}(c_0) \oplus F_{k_n}^{-1}(c_1)$. Thereby we have to count how often the final difference of the characteristic is matched. If the percentage of matching pairs corresponds to the expected probability the guessed key is a good candidate for being the correct key. This must be done for all possible keys. When more than one key candidate remains then either more pairs of plaintext must be

Table 3: Difference distribution table of PRESENT s-box [KYK10]

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	4	0	0	0	4	0	4	0	0	0	4	0	0
2	0	0	0	2	0	4	2	0	0	0	2	0	2	2	2	0
3	0	2	0	2	2	0	4	2	0	0	2	2	0	0	0	0
4	0	0	0	0	0	4	2	2	0	2	2	0	2	0	2	0
5	0	2	0	0	2	0	0	0	0	2	2	2	4	2	0	0
6	0	0	2	0	0	0	2	0	2	0	0	4	2	0	0	4
7	0	4	2	0	0	0	2	0	2	0	0	0	2	0	0	4
8	0	0	0	2	0	0	0	2	0	2	0	4	0	2	0	4
9	0	0	2	0	4	0	2	0	2	0	0	0	2	0	4	0
A	0	0	2	2	0	4	0	0	2	0	2	0	0	2	2	0
B	0	2	0	0	2	0	0	0	4	2	2	2	0	2	0	0
C	0	0	2	0	0	4	0	2	2	2	2	0	0	0	2	0
D	0	2	4	2	2	0	0	2	0	0	2	2	0	0	0	0
E	0	0	2	2	0	0	2	2	2	2	0	0	2	2	0	0
F	0	4	0	0	4	0	0	0	0	0	0	0	0	0	4	4

generated or this step is repeated with another characteristic until one single key candidate remains.

If a key was found the attack can be repeated on $n-1$ steps to get key k_{n-1} and so on. Additionally it can be done upside down to get key k_0. For this attack often less encryptions are needed than for a brute force attack. However, this requires a characteristic with an practical probability. The design of modern cryptographic functions considers this attack and is chosen in a way to minimize the probability to find a good characteristic.

3.3 Techniques in Differential Cryptanalysis

There are some related techniques, which can be used to improve the probability of differential characteristics. A short description of some of them can be found in this sections. We also summarize two techniques for first-order differential cryptanalysis, which have some similarities with the boomerang attack (see Chapter 4), in Section 3.3.6.

3.3.1 Message Modification

For most attacks it is essential to use characteristics with a probability as high as possible. The probability of a characteristic can be improved by selecting the messages accordingly. Often it is possible to influence the state of some rounds directly. In many cases messages can be chosen to correct single bits in the characteristic. If a bit of a message word for a

certain round depends only on one bit of the message, this is quite straight forward. But often it depends on several bits of the message. Then advanced modification techniques must be used to handle the complexity. These advanced techniques depend strongly on the attacked algorithm [WY05]. For instance, a detailed description of message modification for SHA-0 can be found in [WYY05b].

3.3.2 Neutral Bits

When the value of a neutral bit in a message changes the resulting message pair still conforms the same characteristic. In this context the term neutral bit is taken in its information theoretical sense and may be a group of several elementary bits which are all flipped simultaneously [BC04]. Knowing neutral bits reduces the complexity of searching valid characteristics and validating messages for the differential.

3.3.3 Truncated Differentials

A differential that predicts not all of the bits is called a truncated differential. Formally a truncated difference is defined as (X', Y'), if (X, Y) is a differential and X' is a subsequence of X and Y' is a subsequence of Y. Using generalized conditions we can write a truncated difference (see Section 6.1.1) using '?', because this conditions allow any value. For instance in [????x?-?] only the second and the fourth bit are fixed, the others are free. This technique allows differentials to be less strict and so it improves their probability [Knu95].

3.3.4 Related Key Attack

In a related key attack the attacker can observe the operation of a block cipher, using several unknown but mathematical related keys. Therefore weak key scheduling algorithms can be exploited. If the relation of some keys is known, this information can be used to choose the keys in a way that they cancel out differences in the messages. As a result fewer differences go through the block cipher which improve the probability of the characteristic and reduce the complexity of other attacks [Knu93, Bih94]. A famous example is WEP using RC4, which allows to recover the key [FMS01].

3.3.5 Differential Multi-Collision

For a differential multi-collision a difference in plaintexts (ΔP) as well as a difference in the keys (ΔK) is used. A set of q pairs of plaintexts (p_i) and keys (k_i) for a cryptographic function $E_k(\cdot)$ is called a differential q-multi-collision, if

$$E_{k_1}(p_1) \oplus E_{k_1 \oplus \Delta K}(p_1 \oplus \Delta P) = ... = E_{k_q}(p_q) \oplus E_{k_q \oplus \Delta K}(p_q \oplus \Delta P). \tag{5}$$

If the keys are different, this attack can only generate pseudo-collisions. This attack can also be used to generate a chosen-key distinguisher. Using this attack also weakness in the key scheduling can be found [BKN09].

3.3.6 Inside-Out Attack

In this attack no input and output differences are selected but an intermediate difference Δ_1 of the block cipher $E = E_1 \circ E_0$. Then we search for pairs of plain texts with desired intermediate difference and an input difference Δ_0. On input the difference $\Delta_1 \xrightarrow{E_0^{-1}} \Delta_0$ and on output the difference $\Delta_1 \xrightarrow{E_1} \Delta_2$ should occur (see Figure 7). After analyzing a certain amount of pairs of plaintexts at least one pair with wanted differentials $\Delta_0 \xrightarrow{E_0} \Delta_1$ and $\Delta_1 \xrightarrow{E_1} \Delta_2$ should be found [Wag99].

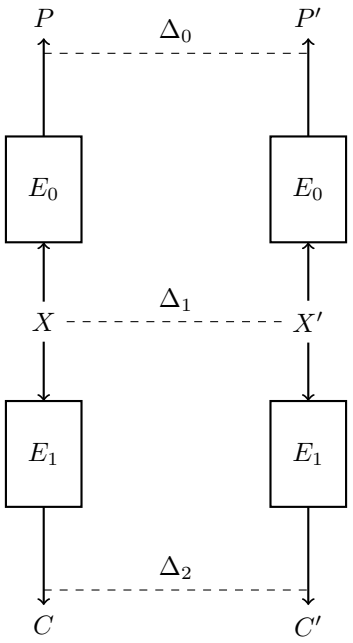

Figure 7: Structure of the inside-out attack

Getting the needed difference in the middle of the function randomly enables differential attacks, even if there are no differentials with high probability through the whole cipher. Therefore, this idea can be used to deal with differentials with lower probabilities. The attack can also be used to distinguish a certain cryptographic function from a ideal and random function. For this one has to calculate the expected probability that a randomly chosen pair of plaintexts fits the required conditions for the inside-out attack. If the expected probability differs from the measured probability, then the analyzed algorithm distinguishes from expected algorithm [KKS01].

3.3.7 Rebound Attack

For the rebound attack the cryptographic function E is divided into three sub-functions $E = E_{fw} \circ E_{in} \circ E_{bw}$. The attack can be divided into two phases. It consists of an inbound

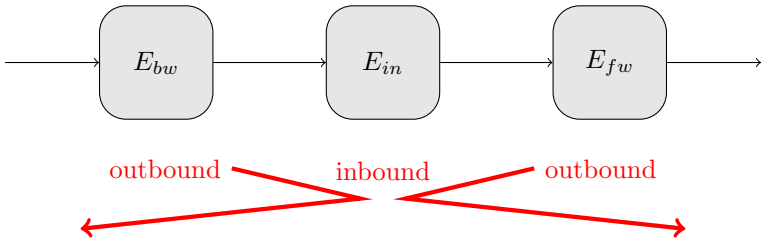

Figure 8: Structure of the rebound attack

and and outbound phase (see Figure 8). In the inbound phase a meet-in-the-middle attack on E_{in} is implemented. The attack was designed for the cryptanalysis of hash functions. Thus, the attacker can use freedom which are available when analyzing hash functions. In the outbound phase differentials are used in forward- and backward directions through E_{fw} and E_{bw} to find collisions. When these differentials have a too low probability, the inbound phase can be repeated. More details can be found in [MRST09].

4 Second-Order Differential Cryptanalysis

Higher order differential cryptanalysis is a generalization of differential cryptanalysis and was first published in [Lai94]. This concept considers differences of differences. In this section we give some basic definitions for higher-order, in particular for second-order differential cryptanalysis, which are needed to describe the boomerang attack.

Abelian group: An abelian group (G, $\bullet$) is a defined by a group of numbers G and an group operator $\bullet$ with certain properties. Two of them are that the result of applying the operation to two elements of the group does not depend on their order and the result is also in G. Thus, for instance the result of modular addition of two 32-bit numbers denoted as $(\mathbb{N}_{32}, +)$ is also a 32-bit number. Further properties are that there exists an identity element e so that $a \bullet e = a$ and for all group members exists an inverse element i so that $a \bullet i = e$. Additional associativity is given so that $(a \bullet b) \bullet c = a \bullet (b \bullet c)$.

Higher order differentials are recursively defined as follows [Lai94, Knu95]:

Derivative: Let (S,+) and (T,+) be an abelian group. For a function $f : S \to T$, the derivative at a point $a_1 \in S$ is defined as

$$\Delta_{(a_1)} f(y) = f(y + a_1) - f(y). \tag{6}$$

Higher derivative: The i-th derivative of f at $(a_1, a_2, ..., a_i)$ is recursively defined as

$$\Delta_{(a_1,...,a_i)} f(x) = \Delta_{(a_i)}(\Delta_{(a_1,...,a_{i-1})} f(y)). \tag{7}$$

Thus, the 2^{nd} derivative of f at point (a_1, a_2) is defined as

$$\begin{aligned}\Delta_{(a_1,a_2)}f(x) &= \Delta_{(a_2)}(\Delta_{(a_1)}(f(x)) \\ &= \Delta_{(a_2)}(f(x+a_1) - f(x)) \\ &= (f(x+a_1+a_2) - f(x+a_2)) - (f(x+a_1) - f(x)) \\ &= f(x+a_1+a_2) - f(x+a_1) - f(x+a_2) + f(x).\end{aligned} \tag{8}$$

Higher-order differential: A differential of order i is an (i+1)-tuple $(\alpha_1, ..., \alpha_i, \beta)$ so that

$$\Delta_{(a_1,...,a_i)}f(x) = b. \tag{9}$$

Higher-order collision: If the output difference of a higher-order differential is zero it is called a collision.

$$\Delta_{(a_1,...,a_i)}f(x) = 0. \tag{10}$$

A second-order differential collision is given if $f(x+a_1+a_2)-f(x+a_1)-f(x+a_2)+f(x) = 0$.

4.1 Boomerang Attack

The boomerang attack is one attack using the concept of second-order differential cryptanalysis and was first presented in [Wag99]. Since we use differences of differences we have to consider four plaintexts (P, P', Q, Q') and four ciphertexts (C, C', D, D'). To attack the target function $E(\cdot)$ the basic attack works as follows:

- Define an input difference Δ for $P \oplus P'$
- Define an output difference ∇ for $C \oplus C'$
- Choose a plaintext P and calculate $P' = P \oplus \Delta$
- Apply function $E(\cdot)$ on P and P' to get $C = E(P)$ and $C' = E(P')$
- Calculate $D = C \oplus \nabla$ and $D' = C' \oplus \nabla$
- Apply inverse function $E^{-1}(\cdot)$ on D and D' to get $Q = E^{-1}(D)$ and $Q' = E^{-1}(D')$

For illustration of this algorithm see Figure 9. Because we choose plaintexts and ciphertexts, the boomerang attack is a chosen plaintext - adaptive chosen ciphertext attack. The fact that the transmitted difference $(P \oplus P')$ should be equal to the returning difference $(Q \oplus Q')$ induced Wagner to his choice of the name boomerang attack: "This is why we call it the boomerang attack: When you send it properly, it always comes back to you" [Wag99].

In more detail, the cryptographic function $E(\cdot)$ can be divided into two sub functions $E = E_1 \circ E_0$, which can also be written as $E(X) = E_0(E_1(X))$. Thereby E_0 represents the first part and E_1 the second part of the function $E(\cdot)$. The intermediate result between these two parts is (X, X', Y, Y'). The differences in this so called switch are Δ^* and ∇^*.

$$\begin{aligned} X = E_0(P) = E_1^{-1}(C), \quad & X' = E_0(P') = E_1^{-1}(C'), \\ Y = E_0(Q) = E_1^{-1}(D), \quad & Y' = E_0(Q') = E_1^{-1}(D'). \end{aligned} \tag{11}$$

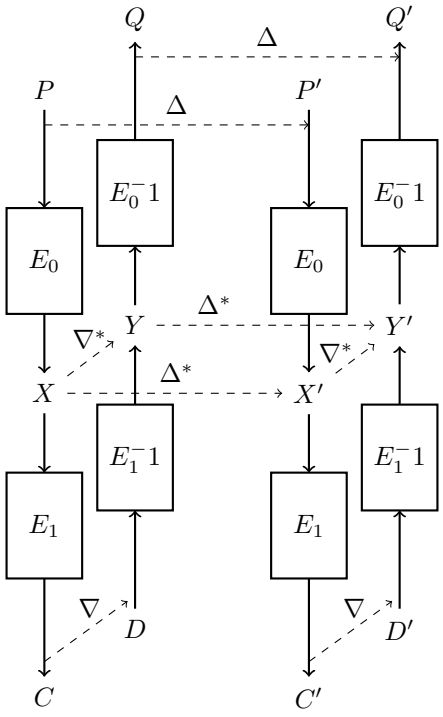

Figure 9: Structure of the boomerang attack

For these intermediate differences in the switch the following equation must hold [Wag99]:

$$\begin{aligned} E_0(Q) \oplus E_0(Q') &= E_0(P) \oplus E_0(P') \oplus E_0(P) \oplus E_0(Q) \oplus E_0(P') \oplus E_0(Q') \\ &= E_0(P) \oplus E_0(P') \oplus E_1^{-1}(C) \oplus E_1^{-1}(D) \oplus E_1^{-1}(D') \oplus E_1^{-1}(D') \\ &= \Delta^* \oplus \nabla^* \oplus \nabla^* = \Delta^*. \end{aligned} \tag{12}$$

If the following conditions are fulfilled, (P, P', Q, Q') is called a right quartet [Mur09]:

$$\begin{aligned} P \oplus P' = Q \oplus Q' = \Delta, \qquad X \oplus X' = Y \oplus Y' = \Delta^*, \\ X \oplus Y = X' \oplus Y' = \nabla^*, \qquad C \oplus D = C' \oplus D' = \nabla. \end{aligned} \tag{13}$$

For this attack the output differences $C \oplus C'$ and $D \oplus D'$ can have any values. That applies also for the differences $P \oplus Q$ and $P' \oplus Q'$. Therefore, almost only the characteristics $\Delta \to \Delta^*$ and $\nabla \to \nabla^*$ are important. Thus, differentials for a boomerang attack can be a combination of two differentials for half of the attacked rounds - one in forward and one in backward direction. Hence it is easier to get such high probability differentials using the boomerang attack than a high probability differential for a first-order attack on E. If we assume that the differentials are independent, a simple approximation of the probability p for a right quartet is [Wag99]:

$$p \geq Pr[\Delta \xrightarrow{E_0} \Delta^*]^2 \cdot Pr[\nabla \xrightarrow{E_1^{-1}} \nabla^*]^2 = p_0^2 \cdot p_1^2. \tag{14}$$

However, when using this probability it is expected to find one solution after $p_0^{-2} \cdot p_1^{-2}$ iterations. This probability does not hold if truncated characteristics are used. The probability calculated in Equation 14 assumes that $Pr[\Delta \xrightarrow{E_0^{-1}} \Delta^*] = Pr[\Delta^* \xrightarrow{E_0^{-1}} \Delta]$. For truncated differentials this is not valid in general. Thus, a more accurate formula for the probability p of a successful boomerang attack using truncated differentials is [Wag99]:

$$\begin{aligned} p &\approx \sum_{w \oplus x \oplus y \oplus z = 0} Pr[\Delta \xrightarrow{E_0} w] \cdot Pr[\nabla \xrightarrow{E_1^{-1}} x] \cdot Pr[\nabla \xrightarrow{E_1^{-1}} y] \cdot Pr[z \xrightarrow{E_0^{-1}} \Delta] \\ &\approx Pr[\Delta \xrightarrow{E_0} \Delta^*] \cdot Pr[\nabla \xrightarrow{E_1^{-1}} \nabla^*]^2 \cdot \Pr[\Delta^* \xrightarrow{E_0^{-1}} \Delta] \cdot \\ &\quad Pr[w \oplus x \oplus y \in \Delta^* | w \in \Delta^*, x, y, \in \nabla^*]. \end{aligned} \tag{15}$$

4.2 Related Attacks

Beside the classic boomerang attack there exist some variants and also other higher-order differential cryptanalysis attacks. Three of them are the amplified boomerang attack, the rectangle attack and the boomerang attack using auxiliary differentials on hash functions.

4.2.1 Amplified Boomerang Attack

Sometimes it is not possible to apply a chosen ciphertext attack as needed for the boomerang attack. Then the amplified boomerang attack may work because it is a chosen plaintext variant of the boomerang attack. For this attack, input pairs are generated until two of them, (P, P') and (Q, Q'), fulfill the boomerang properties (see Figure 10):

$$\begin{aligned} &P \oplus P' = Q \oplus Q' = \Delta \\ &X = E_0(P), X' = E_0(P'), Y = E_0(Q), Y' = E_0(Q') \\ &X \oplus X' = Y \oplus Y' = \Delta^* \\ & \\ &X \oplus Y = X' \oplus Y' = \nabla^* \\ &C = E_1(X), C' = E_1(X'), D = E_1(Y), D' = E_1(Y') \\ &C \oplus D = C' \oplus D' = \nabla \end{aligned} \tag{16}$$

The main difference compared to the original boomerang attack is that no differences go backwards through the cryptographic function. This can be an advantage because using this attack only in one end of the function a key must be guessed. Hence it is not necessary to repeat the requests on the function for several guessed keys. Another advantage is that the boomerang amplifier can also be used on k-tuples of texts not only on pairs. In addition it is possible to add truncated differential for the remaining rounds. This allows using truncated differentials only for a small part of the function. The attack is called amplified boomerang attack because the boomerang structure amplifies the effect of a low-probability event ($X \oplus X' = Y \oplus Y' = \Delta_0$) enough that it can be easily detected [KKS01].

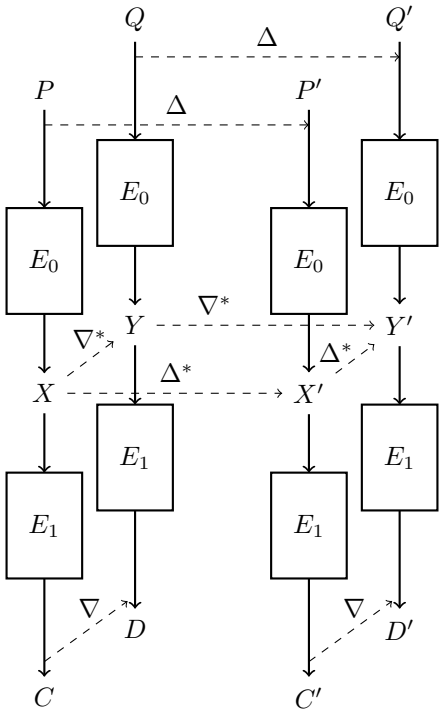

Figure 10: Structure of the amplified boomerang attack

4.2.2 Rectangle Attack

The rectangle attack is an extension of the amplified boomerang attack. The structure is shown in Figure 10. This attack allows even pairs of plaintexts resulting in wrong intermediate difference Δ^*. The different input pairs get sorted by their intermediate difference Δ^*. This approves all quartets which results in the output difference ∇. Thus, the probability to find a correct quartet is higher. Therefore, for all characteristics for which $\nabla^* \rightarrow \nabla$ and $\nabla^* \oplus \Delta^* \rightarrow \nabla$ is valid the probability for a correct quartet is higher than for the original boomerang attack [BDK01].

4.2.3 Boomerang Attack using Auxiliary Differentials

Leurent and Roy extended in [LR12] the idea of using auxiliary paths for message modifications to the boomerang setting when attacking hash functions. The idea of auxiliary paths divides the problem of finding a characteristic for a specific differential into smaller sub problems. Then the biggest clique of auxiliary differential paths is searched to place them into the characteristic. Therefore the auxiliary paths should be as independent as possible.

This idea can also be applied to the boomerang attack. We consider a cryptographic function E that can be decomposed into three sub-functions $E = E_c \circ E_b \circ E_a$, then

- for f_a a differential $\alpha \to \alpha^*$ holding with probability p_a is used
- for f_b a set $\mathcal{B}$ of b differentials $\beta_i \to \beta_i^*$ holding with probability p_b is used
- for f_c a differential $\gamma \to \gamma^*$ holding with probability p_c is used

The attack starts with a boomerang quartet $(U_0, U_1, U_2, U_3) \to (V_0, V_1, V_2, V_3)$ over f_b, with

$$U_1 = U_0 \oplus \alpha^*, \quad U_3 = U_2 \oplus \alpha^*, \qquad V_2 = V_0 \oplus \delta, \quad V_3 = V_1 \oplus \delta. \tag{17}$$

In the next step an auxiliary path $\beta_i \to \beta_i'$ is used to generate $U_i' = U_i \oplus \beta_i$. With probability p_b^4 this results in a new quartet $(U_0', U_1', U_2', U_3') \to (V_0', V_1', V_2', V_3')$ over f_b, where $V_i' = V_i \oplus \beta_i^*$. Thus, for this new quartet must also hold the following equation:

$$U_1' = U_0' \oplus \alpha^*, \quad U_3' = U_2' \oplus \alpha^*, \qquad V_2' = V_0' \oplus \delta, \quad V_3' = V_1' \oplus \delta. \tag{18}$$

Then the complexity to find a correct quartet for the full function E is defined by [LR12]:

$$p = \frac{1}{p_a^2 p_c^2} \left(\frac{C}{b \cdot p_b} + 1 \right). \tag{19}$$

where C is the computational cost to find a correct quartet for f_b. The structure of the attack is shown in Figure 11.

5 Finding Inconsistencies

In this chapter, we discuss a completely different topic than before, namely debugging. Debugging is a process to find errors or conflicts in a set of data, which does not work as expected. It has its main application in computer programming and designing hardware. However, it can be used to find conflicts in any set of data, which can be proven to be correct or not too. A differential characteristic can also be seen as such a set.

There are lots of possibilities for debugging. In general, it can be distinguished between static and dynamic debugging. Static debugging only uses the information whether a specific subset of the inconsistent data set is valid or not. On the contrary, dynamic debugging uses information about the validation process like a execution trace in case of debugging source code. Using this additional information, they may be able to trace the reason of a conflict very precisely. In this thesis, we use only static debugging algorithms, because for differential characteristics it is not always possible to trace the cause of a fault. Such algorithms are also called non-intrusive algorithms [Jun01]. In the following, we give a couple of basic definitions. They are based on the definitions given in [Rei87].

Constraint: A constraint c_i is a single condition, which either can be fulfilled or not. Typically several constraints relate some variables to each other. An exemplary set of constraints would be $c_1 : x_1 < x_2, \;\; c_2 : x_2 = x_3 - x_4$. In this thesis, we define a constraint as a generalized bit condition which should have the value defined in the characteristic.

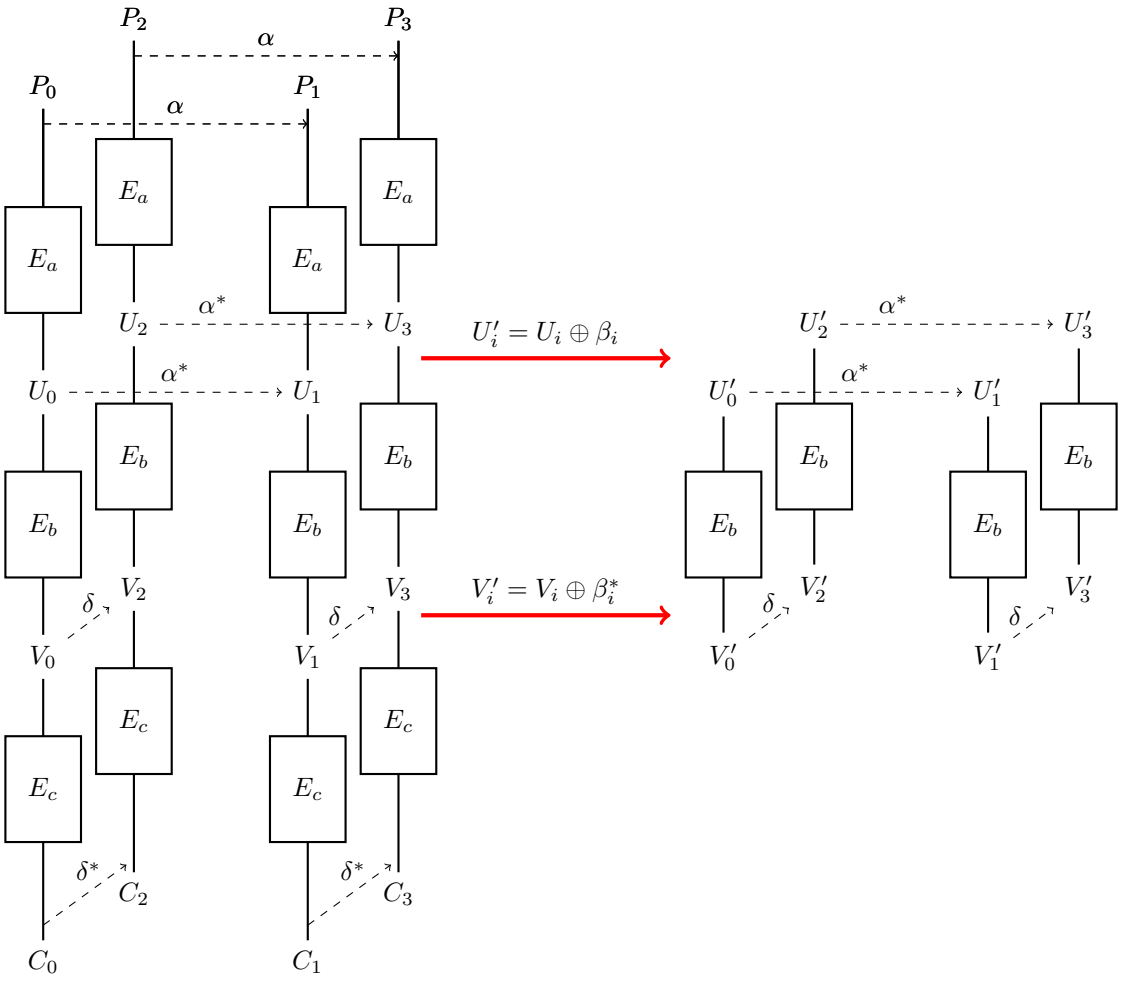

Figure 11: Structure of the boomerang attack using auxiliary differentials

Candidate or Test case: In a test case T some constraints are active and some are inactive. Thus, it is as sub set $\{c_1, \ldots, c_k\}$ of all n available constraints C. Active constraints are set on the values defined in the characteristic. Inactive constraints are set on '?', so they allow any value and do not lead to a conflict. Thus, a test case in terms of a differential characteristic is a characteristic in which some of the conditions are ignored.

Conflict: A conflict C is given if not all constraints in the test case can be fulfilled at the same time. If we can find a conflict for a characteristic, then this characteristic is impossible.

Conflict Set: A conflict set CS is defined as a set of constraints which are inconsistent. This means that the constrains of a conflict set are inconsistent. An example would be

$c_1 : x_1 < x_2$, $c_2 : x_1 = x_2$, $c_3 : x_1 > x_2$. But this conflict set is not minimal, because each pair of those constraints is a conflict set already. Thus, a minimal conflict set contains a minimal set of constraints which are inconsistent. A minimal conflict set CS is defined as $\nexists CS'$ so that $CS' \subset CS$. For the former example, one possible minimal conflict set would be $c_1 : x_1 < x_2$, $c_3 : x_1 > x_2$.

Diagnosis: If the constraints contained in a diagnosis Δ are removed from C, then the remaining constraints must be consistent. Moreover, a diagnosis Δ is determined by a set of constraints, which must be removed from C to make $C - \Delta$ consistent. More formal, a diagnosis $\Delta \subseteq C : C - \Delta \cup \{c_i\}$ is consistent $\forall c_i \in C$. For a set of constraints more than one diagnosis can exist which also can have different sizes.

The selected algorithms basically work in a the divide and conquer principle. They divide the given set of constraints into subsets, therefore a test case, and check the consistency of this set. Depending on the result of these checks, constraints are added or removed from the current subset. Thereby they can pursue two different targets. Either search a small set of inconsistent constraints, a conflict set. Or they search a big set of consistent constraints. This correlates to the search of a diagnosis. Differential characteristics are well suited to apply divide and conquer principle on them, because each condition separately can be deactivated and no syntax or semantic gets violated.

Checking the consistency of a differential characteristic can end up in searching a fulfilling pair of inputs. Thus, checking a differential characteristic can be very expensive. Therefore, debugging algorithms should do as few consistency checks as possible. A set of constraints can have an exponential number of conflicts. Thus, a very naive method would be to test each possible combinations of constraints, until a conflict set or a diagnosis is found. In the worst case this would also take an exponential number of checks.

We chose two algorithms which calculate conflict sets and two further algorithms which calculate diagnoses. In this way we can test which concept is more suited for finding and repairing differential characteristics. We chose very generic debugging algorithms where we had made some experience already. Moreover, the selected algorithms are well suited, because they are completely independent from the underlying structure of the constraints. To show the differences between the presented algorithms we will apply them all on the same example.

Example 1: We define a set of constraints $\{c_1, c_2, c_3, c_4\}$ and two minimal conflicts sets $CS_1 = \{c_2, c_4\}, CS_2 = \{c_3\}$. Thus, possible diagnoses are $\Delta_1 = \{c_2, c_3\}, \Delta_2 = \{c_4, c_3\}$.

In the following sections, we describe four debugging algorithms we implemented (Sections 5.1-5.4) and the usage of them to find a diagnosis to repair boomerang characteristics (Section 5.5).

5.1 SequentialSearch

For a first try of debugging differential characteristics we implemented a very simple debugging algorithm. We created an algorithm which returns a diagnosis. Thus, after removing the constraints in this diagnosis the remaining constraints are consistent and the characteristic is repaired. This algorithm consists of two phases. In the first phase, it removes constraints sequentially until the remaining constraints are consistent. Once having a consistent set of constraints we can test for each removed constraints if it make the set inconsistent and thus, is part of the diagnosis. Therefore, in the second phase, the algorithm adds each removed constraint. If the new set is still consistent, this constraint is not part of the diagnosis and remains in the actual set of constraints. Otherwise, this constraint is part of the diagnosis and is removed again to keep the working set of constraints consistent. The complete algorithm can be found in Algorithm 1.

SequentialSearch always determines and returns a diagnosis if any can be found. If all constraints are removed from C without solving the conflict, in line 12 the algorithm terminates and returns $\varnothing$. Otherwise, C must be consistent at the beginning of the second loop in line 15. C remains consistent, because all c_i which make C inconsistent again are removed and added to X. Hence, per definition X must be a diagnosis.

In worst case almost each constraint can be removed once and added once again. This can happen if the last constraint is a diagnosis. Thus, in worst case $\mathcal{O}(n)$ consistency checks must be done.

Table 4 shows the algorithm applied on Example 1.

Table 4: Exemplary execution of SequentialSearch

iteration	C	D	R	check	comment	retval
1.1	c_2, c_3, c_4	c_1	$\varnothing$	check($\{\mathbf{c_2}, \mathbf{c_3}, \mathbf{c_4}\}$) $\lightning$		
1.2	c_3, c_4	c_1, c_2	$\varnothing$	check($\{\mathbf{c_3}, c_4\}$) $\lightning$		
1.3	c_4	c_1, c_2, c_3	$\varnothing$	check($\{c_4\}$) ✓	$\rightarrow$ break upper loop	
2.1	c_4	c_1, c_2, c_3	$\varnothing$	check($\{c_1, c_4\}$) ✓		
2.2	c_1, c_4	c_1, c_2, c_3	$\varnothing$	check($\{c_1, \mathbf{c_2}, \mathbf{c_4}\}$) $\lightning$		
2.3	c_1, c_4	c_1, c_2, c_3	c_2	check($\{c_1, \mathbf{c_3}, c_4\}$) $\lightning$	$\lvert T\rvert$ is 0 $\rightarrow$ ret R	c_2, c_3

5.2 ReplayXPlain

The ReplayXPlain algorithm was presented in [Jun01]. It is an iterative algorithm based on [DSP88]. The algorithm is described in Algorithm 2. The algorithm starts with an empty set of constraints B which is therefore consistent (line 5). Then it adds the given constraints $c_1, \ldots, c_n$ until B is inconsistent (loop in lines 9-12). Because the last constraint c_k added led to a inconsistency this constraint must be part of the conflict. Thus, c_k is added to the conflict set X (line 16). Then the algorithm makes a backtracking and loads only the constraints in X into C (line 16). If this set still is consistent not the whole conflict set is found. Therefore the remaining constraints again are added to C until a

conflict occurs. These iterations go on until X is inconsistent and thus is a conflict set (check in line 7). Moreover, it must be a minimal conflict set, because each of the contained constraints led to a conflict.

Algorithm 1 SequentialSearch

Require: Set of n inconsistent constraints $C = \{c_1, \ldots, c_n\}$
Ensure: A diagnosis for C

```
 1: procedure SequentialSearch(C)
 2:     if size of C = 0 then
 3:         return ∅
 4:     end if
 5:     D ← ∅
 6:     repeat
 7:         c ← first element of C
 8:         add c to D
 9:         remove c from C
10:     until C is consistent
11:     if size of C = 0 then
12:         return ∅
13:     end if
14:     R ← ∅
15:     for all constraints d in D do
16:         if C ∪ d is consistent then
17:             add d tp C
18:         else
19:             add d to R
20:         end if
21:     end for
22:     return R
23: end procedure
```

The number of checks depends on the size k of the resulting conflict set. If all constraints of the conflict are the k last constraints, for all of them almost n checks have to be done before they are added to C and thus, C gets inconsistent. Additionally for each element of X a check must be done do determine if the conflict set is complete. Thus, in the worst case $\mathcal{O}(k \cdot n + k)$ checks must be done.

Originally, this algorithm allows to define preferred constraints. We do not use this possibility, because we typically cannot rank the constraints. Algorithm 2 is a slightly modified version of the algorithm presented in [Jun01].

Algorithm 2 ReplayXPlain

Require: Set of n inconsistent constraints $C = \{c_1, \ldots, c_n\}$
Ensure: A conflict set for C

```
 1: procedure ReplayXPlain(C)
 2:     if size of C = 0 then
 3:         throw exception 'no conflict'
 4:     end if
 5:     B ← ∅
 6:     X ← ∅
 7:     while B is consistent do
 8:         k ← 0
 9:         while B is consistent and k < n do
10:             k ← k + 1
11:             add c_k to B
12:         end while
13:         if B is consistent then
14:             throw exception 'no conflict'
15:         end if
16:         add c_k to X
17:         B ← X
18:     end while
19:     return X
20: end procedure
```

Table 5 shows the algorithm applied on Example 1.

Table 5: Exemplary execution of SequentialSearch

iteration	B	C	X	check	comment	retval
1	$\varnothing$	c_1, c_2, c_3, c_4	$\varnothing$	check($\varnothing$) ✓		
1.1	$\varnothing$	c_1, c_2, c_3, c_4	$\varnothing$	check($\varnothing$) ✓		
1.2	c_1	c_1, c_2, c_3, c_4	$\varnothing$	check($\{c_1\}$) ✓		
1.3	c_1, c_2	c_1, c_2, c_3, c_4	$\varnothing$	check($\{c_1, c_2\}$) ✓		
1.4	c_1, c_2, c_3	c_1, c_2, c_3, c_4	$\varnothing$	check($\{c_1, c_2, \mathbf{c_3}\}$) ↯	→ break inner loop	
2	c_3	c_1, c_2, c_3, c_4	c_3	check($\{\mathbf{c_3}\}$) ↯	→ break outer loop	c_3

5.3 QuickXPlain

QuickXPlain improves ReplayXPlain by recursively partitioning the problem into subproblems of half size. It is described in Algorithm 3. Therefore the set of constraints is divided into two subsets (lines 14-16). Subsets which do not contain an element of the conflict can be skipped without further checks (line 10). Otherwise, some constraints have to be added

to the other subset (lines 14-16). In this way, the inconsistent problem is divided until one constraint remains which must be part of the conflict set (line 12). The algorithm always terminates and returns a minimal conflict. In worst case $\mathcal{O}(2k \cdot log_2(\frac{n}{k}) + 2k)$ consistency checks must be done [Jun04].

The original algorithm allows to define a ranking of more or less preferred constraints. We do not use this possibility, because we do not quantify the importance of the constraints. Thus, algorithm 3 is a slightly modified version of the algorithm presented in [Jun04].

Algorithm 3 QuickXPlain

Require: Set of n inconsistent constraints $C = \{c_1, \ldots, c_n\}$
Ensure: A conflict set for C

```
 1: procedure QuickXPlain(C)
 2:     if size of C = 0 or C is consistent then
 3:         return ∅
 4:     else
 5:         return QX(∅, ∅, C)
 6:     end if
 7: end procedure

 8: procedure QX(B, Δ, C)
 9:     if size of Δ > 0 and B is inconsistent then
10:         return ∅
11:     else if C = {c_α} then
12:         return c_α
13:     end if
14:     k ← ⌊n/2⌋
15:     C_1 ← {c_1, ..., c_k}
16:     C_2 ← {c_{k+1}, ··· , c_n}
17:     δ_2 ← QX(B ∪ C_1, C_1, C_2)
18:     δ_1 ← QX(B ∪ δ_2, δ_2, C_1)
19:     return δ_1 ∪ δ_2
20: end procedure
```

Table 6 shows the algorithm applied on Example 1.

Table 6: Exemplary execution of QuickXPlain

recursion	B	Δ	C	δ_1	δ_2	check	comment	retval
1.0	$\varnothing$	$\varnothing$	c_1, c_2, c_3, c_4	c_1, c_2	c_3, c_4	no check		c_3
2.0	c_1, c_2	c_1, c_2	c_3, c_4	c_3	c_4	check($\{c_1, c_2\}$) ✓		c_3
3.0	c_1, c_2, c_3	c_3	c_4			check($\{c_1, c_2, \mathbf{c_3}\}$) ↯	→ ret $\varnothing$	$\varnothing$
3.1	c_1, c_2	$\varnothing$	c_3			no check	$\lvert C\rvert$ is 1→ ret C	c_3
2.1	c_3	c_3	c_1, c_2			check($\{\mathbf{c_3}\}$) ↯	→ ret $\varnothing$	$\varnothing$

5.4 FastDiag

This algorithm works similarly as QuickXPlain, but it searches for a diagnosis instead of a conflict set. The description of the algorithm can be found in Algorithm 4. In first step FastDiag also divides the set of constraints into two subsets (lines 14-16). If one of them is a diagnosis and thus, AC is consistent, then the other set can be omitted, because it cannot contain constraints of the diagnosis (line 10). The set of constraints is divided until one constraint remains and the set is still inconsistent. Then this constraint must be part of the diagnosis (line 12). The algorithm is complete in the sense that if C contains one minimal diagnosis, then it will be found. If there are more than one minimal diagnosis then one of them is found. As for QuickXPlain in worst case $\mathcal{O}(2k \cdot log_2(\frac{n}{k}) + 2k)$ consistency checks must be done [FS10].

The original version of FastDiag allows to define a ranking of more or less preferred constraints as in QuickXPlain. We again do not use this possibility. Thus, Algorithm 4 is a slightly modified version of the algorithm presented in [FS10].

Algorithm 4 FastDiag

Require: Set of n inconsistent constraints $C = \{c_1, \ldots, c_n\}$
Ensure: A diagnosis for C

```
 1: procedure FastDiag(C)
 2:     if size of C = 0 then
 3:         return ∅
 4:     else
 5:         return FD(∅, C, C)
 6:     end if
 7: end procedure

 8: procedure FD(D, C, AC)
 9:     if size of D > 0 and AC is consistent then
10:         return ∅
11:     else if C = {c_α} then
12:         return c_α
13:     end if
14:     k ← ⌊n/2⌋
15:     C_1 ← {c_1, ..., c_k}
16:     C_2 ← {c_{k+1}, ··· , c_n}
17:     D_1 ← FD(C_2, C_1, AC \ C_2)
18:     D_2 ← FD(D_1, C_2, AC \ D_1)
19:     return D_1 ∪ D_2
20: end procedure
```

Table 7 shows the algorithm applied on Example 1.

Table 7: Exemplary execution of FastDiag

recursion	D	C	AC	C_1	C_2	check	comment	retval
1.0	$\varnothing$	c_1, c_2, c_3, c_4	c_1, c_2, c_3, c_4	c_1, c_2	c_3, c_4	no check		c_3, c_4
2.0	c_3, c_4	c_1, c_2	c_1, c_2			check($\{c_1, c_2\}$) ✓	$\rightarrow$ ret $\varnothing$	$\varnothing$
2.1	$\varnothing$	c_3, c_4	c_1, c_2, c_3, c_4	c_3	c_4	no check		c_3, c_4
3.0	c_4	c_3	c_1, c_2, c_3			check($\{c_1, c_2, \mathbf{c_3}\}$) $\lightning$	$\lvert C\rvert$ is 1$\rightarrow$ ret C	c_3
3.1	c_3	c_4	c_1, c_2, c_4			check($\{c_1, \mathbf{c_2}, \mathbf{c_4}\}$) $\lightning$	$\lvert C\rvert$ is 1$\rightarrow$ ret C	c_4

5.5 Finding diagnoses

To repair a set of constraints we have to remove the constraints which are contained in the diagnosis. Therefore we have to differ if the used debugging algorithm returns a conflict set or already a diagnosis. The algorithms SequentialSearch and FastDiag return a diagnosis, thus no further steps are necessary. Conversely, QuickXPlain and ReplayXPlain return constraint sets.

Typically a conflict set $CS_i = \{c_{i,1}, \ldots, c_{i,n}\}$ is not a valid diagnosis. Therefore in most cases more conflict sets must be searched. An new conflict set CS_{i+1} can be found by removing CS_1 from C and restart the debugging algorithm. This must repeated until $\Delta = \bigcup_{i=1}^{n} CS_i$ is a valid diagnosis.

In a next step this diagnosis can be minimized. Therefore the hitting set algorithm presented by Reiter in [Rei87] can be used. With the hitting set algorithm all minimal diagnoses based on given conflict sets can be found. Simplified it can be said that therefore the hitting set algorithm takes one constraint from each conflict set. Removing one constraint from each conflict set solves the conflicts, therefore they must be a diagnosis. In our implementation we do not minimize the diagnosis, because a diagnosis containing more constraints increases the probability to find a valid characteristic (see Section 7.2).

Sometimes one diagnosis is not sufficient. Having several diagnoses has the advantage that one can select, for instance, the smallest one. Therefore, we avoid that the used algorithm can find the same diagnosis again. Assuming an algorithm has found a diagnosis $\Delta_i = \{c_{i,1}, \ldots, c_{i,n}\}$, than an new diagnosis $\Delta_{i+1} = \{c_{i+1,1}, \ldots, c_{i+1,n}\}$ should be found, so that Δ_i and Δ_{i+1} differ in at least one constraint. This can be formulated as $\exists c_{i,j} | c_{i,j} \notin \Delta_{i+1}$ and $\exists c_{i+1,j} | c_{i+1,j} \notin \Delta_i$. Therefore, a constraint $c_{i,j}$ must be removed from C before executing the algorithm again. To avoid that the algorithm returns $\Delta_{i+1} = \Delta_i \setminus c_{i,j}$, we have to set $c_{i,j}$ statical active before. Thus, $\Delta_i \setminus c_{i,j}$ is not a valid diagnosis. All possible diagnoses can be found by removing each constraint once from each found diagnosis until no new diagnosis can be found. This technique requires the possibility to set constraints statically active so that they are even active even when they are not in the tested set of constraints. This can be done by using a global set of active constraints C_A which is added temporarily to the tested set of constraints before they are checked.

6 Automatic Tool

In this thesis, we extended the tool presented in [MNS11] to handle bommerang characteristics. We use all the functionalities to update, check and search first-order differential characteristics provided in this tool to also handle boomerang characteristics. In this chapter, we describe the provided functionalities, our extensions are described detailed in Chapter 7.

6.1 Conditions and Characteristics

To specify, store and handle differential characteristics, generalized conditions and multi-bit conditions are used. In this section, we describe the principle of conditions and extend it to multi-bit conditions and characteristics. We also define linear two-bit conditions which can describe the relation between two bits.

6.1.1 Generalized Conditions

If we look at characteristics, a pair of bits (x_i, x'_i) in a pair of messages can either be equal or different. Moreover, we can even use a signed difference. Thus, it makes a difference if the pair of bits has a negative difference (1,0) or a positive difference (0,1). We can also consider the bit values, when both bits are equal. So we distinguish between (0,0) and (1,1). For non-linear functions like an IF this has influence on the output [WYY05a]. We call these four differences $\{(0,0),(1,0),(0,1),(1,1)\}$ the four *basic differences*. The bit difference can be even more generalized using generalized conditions. These conditions allow to represent a set of possible basic differences. In cases like when some input bits of a non-linear function are unknown this generalization is necessary. This can be the case when truncated differences (see Section 3.3.3) are used or the bits derive from an unknown key. Then it is often only possible to calculate a set of possible output differences. If some differences are not possible on the output, we can use this information on the input of the next function to eliminate there some differences too. To represent all possible differences for a certain pair of bits we can use generalized conditions which are listed in Table 8 [DCR06].

The generalized condition can be any combination of the four basic differences. They vary from '?', which allows every difference, to one of the four basic differences individually with '0', 'u', 'n' or '1'. If a contradiction occurs and no possible difference remains, we mark the bit with '#'. These abbreviation can be used to write differences very compact. For instance

$$\nabla X = \{X^2 | x_7 \cdot x'_7 = 0,\ x_i = x'_i \text{ for } 2 \leq i \leq 6,\ x_1 \neq x'_1,\ x_0 = x'_0 = 0\} \tag{20}$$

can be written as $\nabla X = [7? - - - - x0]$, as described in [DCR06].

Table 8: Generalized bit conditions

(x_i, x'_i)	(1,1)	(0,1)	(1,0)	(0,0)	equation
#	-	-	-	-	$\lightning$
0	-	-	-	✓	$\neg X \wedge \neg X'$
u	-	-	✓	-	$X \wedge \neg X'$
3	-	-	✓	✓	$\neg X'$
n	-	✓	-	-	$\neg X \wedge X'$
5	-	✓	-	✓	$\neg X$
x	-	✓	✓	-	$X \oplus X'$
7	-	✓	✓	✓	$\neg X \vee \neg X'$
1	✓	-	-	-	$X \wedge X'$
-	✓	-	-	✓	$\neg(X \oplus X')$
A	✓	-	✓	-	X
B	✓	-	✓	✓	$X \vee \neg X'$
C	✓	✓	-	-	X'
D	✓	✓	-	✓	$\neg X \vee X'$
E	✓	✓	✓	-	$X \vee X'$
?	✓	✓	✓	✓	$\varnothing$

6.1.2 Generalized Multi-Bit Conditions

Sometimes it is useful to have conditions on more than one bit. This is particularly true if the cryptographic function uses modular addition because the result of the addition of two bits consists of a sum and a carry bit. Moreover, adding four single bits needs already two carry bits and therefore three bits are needed to store the result. Thus, the number of bits of the result of an operation on bits can be higher than one. To store the resulting information, generalized multi-bit conditions can be used. Leurent introduces multi-bit constraints in [Leu12a]:

- A 1.5-bit constraint is a set involving (x_i, x'_i, x_{i-1})
- A 2.0-bit constraint is a set involving $(x_i, x'_i, x_{i-1}, x'_{i-1})$
- A 2.5-bit constraint is a set involving $(x_i, x'_i, x_{i-1}, x'_{i-1}, x_{i-2})$

These constraints link the sign of an active bit to the sign of the previous bit. Thus, a 1.5-bit constraint encodes the concrete values x_i and x'_i and whether the x_{i-1} bit is equal or unequal to x_i. , a 1.5-bit constraint can store the following relation: $x'_i \neq x_i = x_{i-1}$. However, 1.5-bit constraints do not use information about x'_{i-1}. This principle is only efficient when the difference (x_i, x'_i) is known, otherwise information gets lost. Therefore, 2.0-bit constraints can be used to safe this information. As Leurentshowed, these constraints can exactly represent the modular difference. Thus, on the one hand, the 1.5-bit constraints are constructed in order to capture information about the carry when the sign of the

difference is unknown. On the other hand, the 2.0-bit constraints can capture exactly the modular difference, but the sign of the difference must be known. The 2.5-bit constraints are a combination of both and can store the modular difference even when the sign is not known [Leu12a].

We use a slightly different concept. We store the result of an addition with several input bits into one condition. Therefore, we use generalized 2-bit conditions which allow to store the result of an addition of up to 3 bits and generalized 3-bit conditions which can handle additions of up to 7 input variables. Doing several additions at once has the advantage that fewer intermediate variables are necessary. This increases performance and lowers complexity. In the following description we explain the concept of generalized 2-bit conditions. However, it works rather similarly for generalized 3-bit conditions.

Generalized 2-bit conditions consist of one sum bit and one carry bit. Because it is a generalized 2-bit condition it is defined as a set of possible 2-bit differences. For a partial list of all possible values see Table 9. In all the following the notation x_i^2 indicates a two bit value at position i and x_i^3 a three bit value respectively.

Table 9: Partial list of generalized 2-bit conditions

$(x_i^2, x_i^{2\prime})$	(3,3)	(2,3)	(1,3)	(0,3)	...	(3,0)	(2,0)	(1,0)	(0,0)
0x0000	-	-	-	-	...	-	-	-	-
0x0001	-	-	-	-	...	-	-	-	✓
0x0002	-	-	-	-	...	-	-	✓	-
0x0003	-	-	-	-	...	-	-	✓	✓
0x0004	-	-	-	-	...	-	✓	-	-
0x0005	-	-	-	-	...	-	✓	-	✓
0x0006	-	-	-	-	...	-	✓	✓	-
0x0007	-	-	-	-	...	-	✓	✓	✓
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0xFFF8	✓	✓	✓	✓	...	✓	-	-	-
0xFFF9	✓	✓	✓	✓	...	✓	-	-	✓
0xFFFA	✓	✓	✓	✓	...	✓	-	✓	-
0xFFFB	✓	✓	✓	✓	...	✓	-	✓	✓
0xFFFC	✓	✓	✓	✓	...	✓	✓	-	-
0xFFFD	✓	✓	✓	✓	...	✓	✓	-	✓
0xFFFE	✓	✓	✓	✓	...	✓	✓	✓	-
0xFFFF	✓	✓	✓	✓	...	✓	✓	✓	✓

Each bit of the value of the generalized condition represents a different condition. For instance, when bit 13 is set this denotes that the 2-bit difference can have the difference (1,3). The 16-bit number of a 2-bit generalized condition can also be represented in form of a matrix. Therefore, each bit has an own cell and the bit index increases per row from left to right. Thus, x increases with the row index and with ascending column index x' rises.

The resulting matrix can be divided into four regions and each of those regions consists of four fields. The sequence of differences of the least significant bit in those four fields is repeated within all four regions. The same sequence can be found for the second bit, whereby the difference of the second bit is constant within each region. In Table 10 one can see the resulting conditions for each bit. For instance the difference (1,3) represents 'n1'. In Table 11 the resulting conditions for the generalized 3-bit conditions are listed.

Table 10: Generalized 2-bit condition matrix

X' \ X	$0 \stackrel{\wedge}{=} (00)_2$	$1 \stackrel{\wedge}{=} (01)_2$	$2 \stackrel{\wedge}{=} (10)_2$	$3 \stackrel{\wedge}{=} (11)_2$
$0 \stackrel{\wedge}{=} (00)_2$	00	0u	u0	uu
$1 \stackrel{\wedge}{=} (01)_2$	0n	01	un	u1
$2 \stackrel{\wedge}{=} (10)_2$	n0	nu	10	1u
$3 \stackrel{\wedge}{=} (11)_2$	nn	n1	1n	11

Table 11: Generalized 3-bit condition matrix

X' \ X	$(000)_2$	$(001)_2$	$(010)_2$	$(011)_2$	$(100)_2$	$(101)_2$	$(110)_2$	$(111)_2$
$(000)_2$	000	00u	0u0	0uu	u00	u0u	uu0	uuu
$(001)_2$	00n	001	0un	0u1	u0n	u01	uun	uu1
$(010)_2$	0n0	0nu	010	01u	un0	unu	u10	u1u
$(011)_2$	0nn	0n1	01n	011	unn	un1	u1n	u11
$(100)_2$	n00	n0u	nu0	nuu	100	10u	1u0	1uu
$(101)_2$	n0n	n01	nun	nu1	10n	101	1un	1u1
$(110)_2$	nn0	nnu	n10	n1u	1n0	1nu	110	11u
$(111)_2$	nnn	nn1	n1n	n11	1nn	1n1	11n	111

6.1.3 Linear Two-Bit Conditions

In a characteristic often two or more bits depend directly on each other. Such extended conditions occur mostly in the propagation of differences through boolean functions. These relations can be modeled with linear two-bit conditions. If for instance the characteristic requires that in the j-th step the bit $c_{j,i} = c_{j-1,i} \oplus c_{j-2,i}$ is 0, then $c_{j-1,i} \stackrel{!}{=} c_{j-2,i}$. Two-bit conditions cannot be used to propagate information through the characteristic but they can be used to find conflicts. Therefore, sets of linear two-bit conditions can be formed. Two-bit conditions can connect bits of different steps and bits of the same step can be connected depending on the update function. In this way, cycles can occur in the est of equations which may be inconsistent [WLF+05, MNS11].

6.2 Searching for a Characteristic and Messages

Finding differential characteristics with high probabilities is the most important and also most difficult part in differential cryptanalysis. In this thesis, we use the searching technique described in [MNS11]. The idea for this technique is taken from resolution algorithm in logic, which is implemented in SAT solvers [Rob65]. The searching algorithm can be divided into three steps: decision, deduction and backtracking.

Decision: In this step a bit of the characteristic is guessed. Therefore, a random undetermined condition is chosen and set on one of its possible differences. To improve efficiency of the searching algorithms the selection of the guessed bit may not be completely random. When it turns out that the guess on the selected bit leads to a conflict, this bit is marked as critical. Guessing first on these critical bits, allows faster pruning of the search tree. Additionally, searching bits only in a specified area allows to search in the most critical part of a characteristic.

Deduction: In this step the changed condition is propagated over the whole characteristic. Optionally the resulting characteristic can be checked for inconsistencies. See Section 6.3 and 6.4 for more details. If in this step a conflict is detected, then backtracking must be done. Otherwise another decision can be made.

Backtracking: When all possible guesses for the actual constraint led to an inconsistent characteristic, the guess before has to be undone. If for this bit other values are possible try them in the decision step. Otherwise jump back to earlier decisions.

A simplified description can be found in Algorithm 5.

Even when the characteristic has passed all checks, it does not need to be valid or the probability to find confirming messages can be unacceptable low or even zero. Thus, in a final step a message pair satisfying the characteristic must be found to confirm the validity of the characteristic. Including this into the search is much more efficient because message bits have big influence on the other bits. This allows to detect invalid characteristics earlier. However, it should not be done too early because fixing a message restricts the search of characteristic.

Typically only bits with certain generalized conditions are tested with some particular values. It is possible to divide the search into some phases. In each phase on bits with a certain generalized conditions can be guessed. Note that differences cause conflicts more likely than equalities and guessing more equalities than differences increases the probability of the resulting characteristic. Considering these facts the authors of [MNS11] proposed to guess '-' for '?' and 'u' or 'n' for 'x' in the first searching phase. The remaining bits are determined in a second searching phase.

Because in the decision phase a random condition is chosen, the search path can look for each run differently. Thus, the resulting characteristics and the runtime vary.

Algorithm 5 Search

Require: Characteristic C
Ensure: **true** or **false**

```
1: procedure SEARCH(C)
2:     define list L of all undetermined conditions in C
3:     if size of L = 0 then
4:         return true
5:     end if
6:     c ← random condition in L
7:     define list D of all possible differences of c
8:     for all Difference d in D do
9:         set c in C on d
10:        if Update C fails or Checking C fails then
11:            continue
12:        else if Search(C) returns true then
13:            return true
14:        end if
15:    end for
16:    mark c as critical bit
17:    return false
18: end procedure
```

6.3 Updating a Characteristic

After guessing a bit in the search, the characteristic must be updated. The complexity of propagation conditions increases exponentially with the number of input bits and used additions. Thus, we split up the computation of the step function and calculate the sub-functions σ, Σ, CH and MAJ of the SHACAL-2 step function seperately as proposed in [MNS11]. Due to this fact these step sub functions have only up to 3 input bits their result for each input can be precomputed. The bitslices of the modular additions combining these results have up to 5 input bits of generalized conditions. Thus, precomputing all possible result for the additions is infeasible. Instead, a size-limited hash map stores already computed results. Using more sub-functions with fewer input bits would cause too many intermediate results which would negatively influence the performance of the search.

In the tool each bit difference of a characteristic is represented as generalized 1-bit, 2-bit or 3-bit condition. Thus, the characteristic is represented by an array of conditions, the so called state. The bits belonging to one word are also grouped. A step of the cryptographic function is divided into several sub-step functions as mentioned before. These sub-step functions know the needed input words and the word where the result should be stored. Therefore, sometimes intermediate words are defined. An example for a sub-step function can be the implementation of the MAJ function of SHACAL-2. The accuracy of these sub-step functions is important. With this structure each bit can be connected with its related

bits and it is straightforward to update all connected bits when a single bit difference changes. To update the characteristic all actual changes are propagated over the whole state until it remains stable or a conflict occurs. A conflict occurs when through updating a generalized conditions no more possible differences remain.

In cryptographic functions typically not the whole state changes in each round of the step function. For instance in SHACAL-2 the state consists of eight 32-bit words $A_i, ..., H_i$, where i indicates the round. In each step only A_i and E_i must be calculated, the other words can be directly taken from former steps: $B_{i+1} = A_i, C_{i+1} = B_i, \ldots$. For the calculation of A_i and E_i also the message word W_i is necessary, which also must be calculated for $i \geq 16$. Considering this fact only A_i, E_i and W_i must be calculated and stored in the state.

6.4 Advanced Consistency Checks

Guessing conditions and propagating them over the characteristic is an expensive task. In most cases in some point of the search an obvious contradiction occurs. Typically the contradiction was hidden already in the characteristic some guesses before the conflict occurs. Thus, these last guesses wasted computational power. To improve the efficiency of the search it is important to find conflicts as soon as possible. Due to the high complexity of modern cryptographic functions a trade-off between quality and time for checks is necessary. We use two checks described in [MNS11].

Two-Bit Condition Check: Linear two-bit conditions describe the linear relation between two bits. They either can be equal ($c_{i,j} = c_{i',j'}$) or unequal ($c_{i,j} \neq c_{i',j'}$). The concept of linear two-bit conditions was presented in Section 6.1.3. Therefore cliques of linear two-bit conditions are searched and their consistencies checked. Contradictions in cliques of linear two-bits can be found by applying Gaussian elimination on the resulting linear system of equations. To illustrate this we assume the following three linear two-bit conditions:

- $c_{1,9} = c_{0,4} \oplus c_{0,8}$ should be $0 \rightarrow c_{0,4} \overset{!}{=} c_{0,8}$
- $c_{2,5} = c_{0,8} \oplus c_{1,5}$ should be $0 \rightarrow c_{0,8} \overset{!}{=} c_{1,5}$
- $c_{2,1} = c_{0,4} \oplus c_{1,5}$ should be $1 \rightarrow c_{0,4} \overset{!}{\neq} c_{1,5}$

This leads to a conflict because its not possible to satisfy $c_{0,4} = c_{0,8} = c_{1,5}$ and $c_{0,4} \neq c_{1,5}$.

Complete Condition Check: To find messages which fulfill the characteristics the bits finally must be zero or one. Hence, each conditions must be able to be at least one of the four basic differences. Thus, we can test each active condition being a negative difference 'u' and a positive difference 'n' and each inactive condition can be tested being '0' or '1'. For a valid characteristic at least for one option no conflict must be found. If both options conflict we know the characteristic is inconsistent. If one option works, we can update that bit condition, because it must have this value.

This check is quite expensive. Thus, it is done only on conditions which are restricted by two-bit conditions. Because these bits have to fullfil more constraints, the probability to find conflicts on such conditions is higher than for the others. This check can also be done by testing even sets of bits at the same time. Such tests can detect more conflicts but are even more inefficient.

6.5 Related Work

We found two frameworks which are able to analyze differential characteristics. Both tools can handle only first-order differential characteristics and cannot repair inconsistent characteristics.

S-function Tool: In [MVDCP11] a general and efficient framework to determine the differential properties of S-functions was presented. An S-function ("state function") takes a list of n states $S_0, \ldots, S_{n-1}$, k n-bit input words $(a_0, \ldots, a_k)$ and produces an n-bit output b. Thereby the i-th bit of the output is defined as:

$$(b[i], S_{i+1}) = f(a_1[i], \ldots, a_k[i], S_i), \quad 0 \leq i < n, \quad S_0 = 0. \tag{21}$$

This means that each bit $b[i]$ of the output word and the state S_{i+1} can be computed directly from bit i of the input words and the state S_i. Examples of S-functions are modular addition, subtraction, multiplication, bitwise logical operations like XOR and shifting operations. Some recent cryptographic functions forgo the concept of the s-box and use only S-functions. This concept is the so called ARX design what stands for Additions ($a \boxplus b$), Rotations ($a \ggg i$) and XORs ($a \oplus b$). Blake and Skein, two of the SHA-3 finalists, follow this design strategy.

S-boxes typically work on bytes and therefore a difference distribution table can be calculated. Building a difference distribution table for an S-functions is infeasible, since they work typically with 32-bit or 64-bit words. Thus, it is important to analyze such functions efficiently. The presented framework can be used to calculate the probability that a given input difference leads to a given output difference. In addition it is able to count the number of output differences having a non-zero probability. Therefore it uses methods based on graph theory and matrix multiplications. More details can be found in [MVDCP11].

ARX-Toolkit: Leurent presents in [Leu12b] three tools to analyze ARX systems.

He did not manage to automate the search of differential paths, but his tools can help to build and check characteristics. Using this tools he also found flaws in the differential characteristics used in some recent boomerang attacks.

The first tool is designed for the analysis of S-functions and is based on the ideas of the S-function tool. Additionally it can construct the transition matrix for the S-function completely automated. For instance this tool can be used to find out which XOR-differences are possible for an addition.

With the second tool the differential properties of ARX designs can be studied. It allows to check whether the differences are consistent across the rotations and propagate constraints by detecting incompatibilities. It works similarly as the approach we use from [DCR06], but with a different set of constraints from [Leu12a] which were presented in Section 6.1.2. However, the tool is not able to build differential characteristics automatically as our tool can.

The third tool is a graphical tool. It displays differential characteristic and allows the user to add and remove constraints. This tool automatically propagates the new constraints and if a contradiction is found, it highlights the specific constraints causing the contradiction.

These tools can be downloaded under http://www.di.ens.fr/∼leurent/arxtools.html.

7 Automatic Boomerang Tool

In the former section, we described an automatic tool for finding first-order differential characteristics. Now we extend that tool to handle second-order differential characteristics. First, we describe how we represent the characteristics needed for the boomerang attack. Then we explain some methods we use to update these characteristics and check their consistency. Afterwards we show how we use the debugging techniques described in Section 5 to repair the boomerang characteristics. Finally we illustrate how new characteristics are searched.

7.1 Implementing Boomerang State

As shown in Figure 9 a boomerang attack requires four differential characteristics. In the following, we call these four characteristics the *boomerang state* and each characteristic is called *substate*. Furthermore, we will look only at one bit of each characteristic where the bit has the same position in all characterisitcs. Such a bit quartet is shown in Figure 12. This figure shows the arrangement of four conditions c_0, c_1, c_2, c_3 of the substates C_0, C_1, C_2, C_3 and their four bit values b_0, b_1, b_2, b_3. Each condition shares a bit with each of its neighbors. For instance the conditions c_0 and c_1 share the bit b_0.

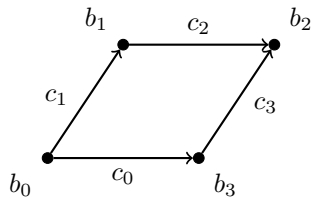

Figure 12: A bit quartet in a boomerang state

We extend the tool in a way that allows us to combine four characteristics. Therefore

we store four separate characteristics and each characteristic can be independently updated and checked as described in Section 5. In a first step each characteristic is updated and checked independently. Additionally, the four characteristics have to fulfill the boomerang properties described in Section 4.1. Therefore, we implement some further checks and propagation rules described in the following section.

7.2 Repair Boomerang Characteristics

The update and consistency check methods described before can be used to find conflicts in boomerang characteristics. For the validity checks in the debugging algorithms different detail-levels for the check can be used. We distinguish three levels:

- fast: updates substates (Section 6.3) and boomerang state (Sections 7.3.1, 7.3.2-7.3.4)
- intermediate: checks condition in substates and boomerang state (Sections 6.4, 7.3.5)
- deep check: does additionally a search on the boomerang state (Sections 6.2, 7.4)

The higher the used level the higher is the probability to find a boomerang characteristic for which a confirming message can be found. On the other hand also the computing efforts increase rapidly.

In the following, we describe how we try to fix conflicts in the boomerang characteristic using the debugging techniques described in Section 5. Furthermore, we describe two other approaches. These techniques can also be used to repair first-order differential characteristics.

Using Debugging Algorithms: For this approach we use different debugging algorithms to find a diagnosis. Therefore the debugging algorithms use a list of conditions and return a list of positions which are in the diagnosis. Because the debugging algorithms are deterministic we randomize the input list to find different diagnoses. Finally the characteristic is fixed by setting the conditions in the diagnosis on '?'.

Rotating Words: In this method we rotate the words of the characteristics before we apply a debugging algorithm. Thereby we rotate the words of two neighboring characteristics independently and of two opposing characteristics equally. Applied on SHACAL-2 which has a word length of 32-bit this results in $32 * 32 = 1024$ different rotations. If we assume that the characteristics are rotation invariant, we have to rotate only one state which results in 32 different rotations. For each rotation we apply a debugging algorithm and use the size of the resulting diagnosis as measure of quality. This measure is only an approximation, because it is based on the size of a random diagnosis and not the smallest diagnosis. In addition, the size of the diagnosis does not have to be proportional to the probability of the repaired characteristic. Anyway, this approach is very time-consuming. However, rotating the characteristics may result in smaller diagnoses. Therefore, the properties of initial differential can be preserved better.

Statistical Approach: For this approach we collect and evaluate the diagnoses of different debugging runs. Thus, we can find the most common conflicting conditions. Then we set them on '?' and rerun the debugging algorithm until the boomerang state is valid.

To improve the performance the debugging algorithm first work on a set of words instead on a set of conditions. Thus, the algorithm has to work on less constraints. Moreover, often several bits of one word are part of the diagnosis anyway. When a diagnosis is found, for each bit in the contained words can be tested if it leads to a conflict when it is removed from the diagnosis. This filtering step decreases the size of the diagnosis.

7.3 Update and Check Boomerang Characteristics

If a condition in one substate changes, then this may have effects on the other substates. Thus, we have to propagate the new information over the whole boomerang state. In this chapter, we describe how to update and check the consistency of a boomerang state.

7.3.1 Transfer Conditions

If a condition is active, then the opposing condition must be also active to fulfill the properties of the boomerang attack. The same applies, if a condition is inactive. However, they do not need to have the same difference which can be seen in Figure 13. Thus, we can only transfer information whether a bit condition is active or not. If a condition can be active and inactive, we cannot transfer any information to the opposing substate. This is the case when the condition is '3', '5', '7', 'A', 'B', 'C', 'D', 'E' or '?'. This principle is demonstrated in Figure 14. Anyway, we can use this transfer of information to find contradictions in the boomerang state. If by propagation a condition must be active but the opposing condition must not, this leads to a contradiction.

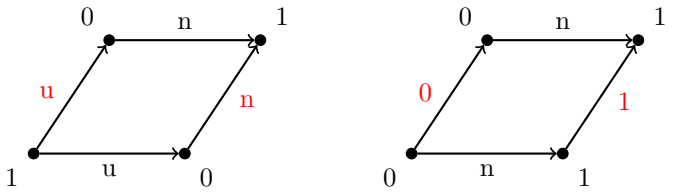

Figure 13: Example for different opposing differences

7.3.2 Update Neighboring Substates

Two neighboring conditions share a bit. If the bit is fixed for one condition, it is also fixed for the other one. We can use this knowledge to transfer information from a substate to its neighboring substates. Therefore, we have to extract the information for the shared bit

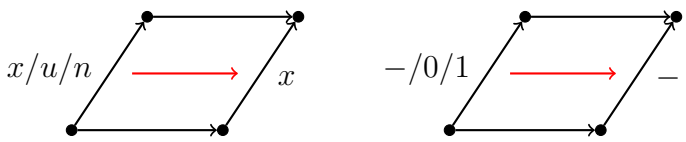

Figure 14: Transferring conditions

Algorithm 6 TransferCondition

Require: BoomerangState S, List of all Conditions C of S
Ensure: **true** or **false**

```
1: for all Condition c in C do
2:     if c = 'x' or c = 'u' or c = 'n' then
3:         set c.opposing() in S on 'x';
4:     else if c = '-' or c = '0' or c = '1' then
5:         set c.opposing() in S on '-';
6:     end if
7: end for
8: return S.validate()
```

and update the corresponding condition in the neighboring substate. An example is shown in Figure 15.

The difference of a condition is directed and consists of bit0 at the base and bit1 at the endpoint. As you can see in Figure 16, bit0 of a condition updates either bit0 or bit1 of the neighboring condition, depending to which condition it belongs. The same applies for bit1. Thus, we have to care if we update the base bit or the bit of the endpoint of the neighboring condition. To distinguish the conditions we name them as shown in Figure 16.

If we handle the generalized condition c as a four bit number, then two bits indicate the same value of bit0 and bit1 each. In Figure 17, one can see that for instance bits0 and bit2 both indicate whether bit0 can be 0 or not. Moreover, if one of both bits is set, then bit0 can be 0. We can determine if at least one of these two bits is set, if $c \wedge 0b0101$ is not 0.

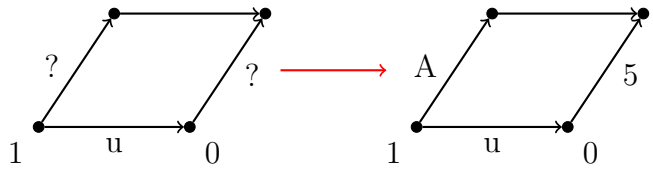

Figure 15: Example of updating neighboring substates

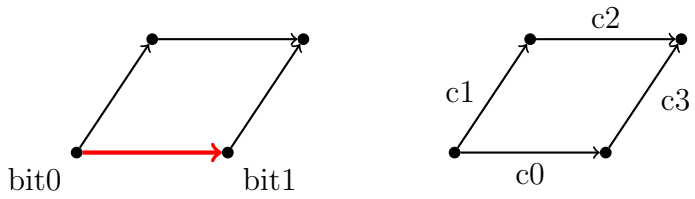

Figure 16: Directed difference and naming of the substates

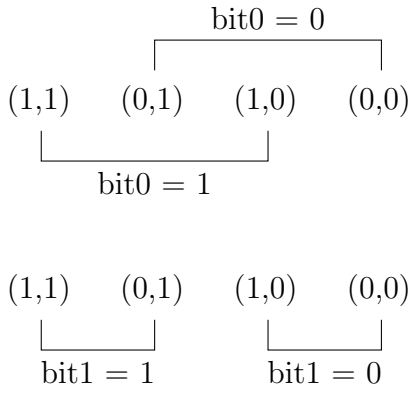

Figure 17: Two bits indicate the same value for bit0 and bit1 each

This can be done similarly for the other values:

- If $c \wedge 0b0101 > 0$ then bit0 can be 0
- If $c \wedge 0b1010 > 0$ then bit0 can be 1
- If $c \wedge 0b0011 > 0$ then bit1 can be 0
- If $c \wedge 0b1100 > 0$ then bit1 can be 1

Similar masks can be used to update the neighboring conditions. However, we have to consider witch condition updates which one as mentioned before. As an illustration here are two examples:

- If $c0 \wedge 0b0101 = 0 \rightarrow$ bit0 of $c0$ and bit0 of $c1$ cannot be $0 \rightarrow c1' = c1 \wedge 0b1010$
- If $c2 \wedge 0b0101 = 0 \rightarrow$ bit0 of $c2$ and bit1 of $c1$ cannot be $0 \rightarrow c1' = c1 \wedge 0b1100$

Although in these two examples bit0 of the updating condition cannot be 0, the neighboring substate has to be updated with different masks. Comparing the two examples above, in the second example the bits 1 and 2 of the updating mask are swapped. In Table 12 all needed updating masks are listed. Depending on the value of condition c the updating

Table 12: Updating masks

c	c	M0	M1	M0'	M1'
#	0b0000	0b0000	0b0000	0b0000	0b0000
0	0b0001	0b0101	0b0101	0b0011	0b0011
u	0b0010	0b1010	0b0101	0b1100	0b0011
3	0b0011	0b1111	0b0101	0b1111	0b0011
n	0b0100	0b0101	0b1010	0b0011	0b1100
5	0b0101	0b0101	0b1111	0b0011	0b1111
x	0b0110	0b1111	0b1111	0b1111	0b1111
7	0b0111	0b1111	0b1111	0b1111	0b1111
1	0b1000	0b1010	0b1010	0b1100	0b1100
-	0b1001	0b1111	0b1111	0b1111	0b1111
A	0b1010	0b1010	0b1111	0b1100	0b1111
B	0b1011	0b1111	0b1111	0b1111	0b1111
C	0b1100	0b1111	0b1010	0b1111	0b1100
D	0b1101	0b1111	0b1111	0b1111	0b1111
E	0b1110	0b1111	0b1111	0b1111	0b1111
?	0b1111	0b1111	0b1111	0b1111	0b1111

masks for the neighboring conditions can be selected from that table. Algorithm 7 shows a description of this princple.

7.3.3 Transfer Multi-Bit Conditions

We update also the generalized multi-bit conditions presented in Section 6.1.2. The domain of multi-bit conditions is too large to update them in the same way as single-bit conditions. Alternatively, the conditions can be divided into a list of possible tuples of basic differences and operations on this list can be done.

As mentioned before, opposing conditions share some properties. If a condition is a difference, then the opposing condition must be also a difference and the other way round. The same applies for multi-bit conditions. Thereby the bits of a condition must be updated independently.

Because we look at tuples of basic differences, they can have only values from {'0', 'u', 'n', '1'}. If a bit of the condition is a difference - either 'u' or 'n' - the same bit in the opposing condition can be one of these two values. Further, if a bit of the condition has no difference, and thus is '0' or '1', then the same bit of the opposing condition can be one of these two values again. If for instance a two-bit generalized condition allows to be '0u', the opposing condition can be {'1u','0n','1n'}. The other way round, in the opposing condition '0u' can be set, if at least one of '0u', '1u', '0n' or '1n' is set. Each possible difference corresponds to a certain bit in the generalized condition. Therefore for each possible difference in the opposing condition several bits

Algorithm 7 UpdatingNeighbors

Require: BoomerangState S and Arrays M0, M0', M1, M1' containing the update masks
Ensure: **true** or **false**

```
 1: list for States s[] = S.getSubStates()
 2: for  i = 0 to 3 do
 3:     for all Condition c in s[i] do
 4:         if i < 2 then
 5:             Mask m0 = M0[c]
 6:             Mask m1 = M1[c]
 7:         else
 8:             Mask m0 = M0'[c]
 9:             Mask m1 = M1'[c]
10:         end if
11:          update c in s[1 - (i % 2)] using m0
12:          update c in s[3 - (i % 2)] using m1
13:     end for
14: end for
15: return S.validate()
```

must be tested. For two-bit conditions four positions and for three-bit conditions seven positions must be tested. The relative position of these related conditions depends on the position of the corresponding bit. Using the condition matrix the related bit position b_i depends on the row and column of the actual bit position b. For an example see Table 13. For reasons of clarity in the following formulas we use the ternary operator which is defined as $< condition > ? \ < if - branch > : < else - branch >$. For bit position b the related positions are defined as following:

2-bit conditions:

- $b_0 =$ b
- $b_1 =$ (b/4 (mod 2)) ? (b (mod 2) ? b-5 : b-3) : (b (mod 2) ? b+3 : b+5)
- $b_2 =$ (b/8 (mod 2)) ? (b/2 (mod 2) ? b-10 : b-6) : (b/2 (mod 2) ? b+6 : b+10)
- $b_3 = b_1 + b_2$ - b_0

2-bit conditions:

- $b_0 =$ b
- $b_1 =$ (b/8 (mod 2)) ? (b (mod 2) ? b-9 : b-7) : (b (mod 2) ? b+7 : b+9)
- $b_2 =$ (b/16 (mod 2)) ? (b/2 (mod 2) ? b-18 : b-14) : (b/2 (mod 2) ? b+14 : b+18)
- $b_3 =$ (b/32 (mod 2)) ? (b/4 (mod 2) ? b-36 : b-28) : (b/4 (mod 2) ? b+28 : b+36)
- $b_4 = b_1 + b_2$ - b_0
- $b_5 = b_2 + b_3$ - b_0

- $b_6 = b_1 + b_2 + b_3$ - $2 \cdot b_0$

Table 13: Example for related 2-bit conditions

X' \ X	0 .. 00	1 .. 01	2 .. 10	3 .. 11
0 .. 00	00	0u	u0	uu
1 .. 01	0n	01	un	u1
2 .. 10	n0	nu	10	1u
3 .. 11	nn	n1	1n	11

For illustration we calculate the related bit positions of $(1n)$, whereby index(LSB) = 0:

- $b_0 =$ 14
- $b_1 = (14/4 \pmod 2)$? ($14 \pmod 2$? 9 : **11**) : ($14 \pmod 2$? 17 : 19) = 11
- $b_2 = (14/8 \pmod 2)$? ($14/2 \pmod 2$? **4** : 8) : ($14/2 \pmod 2$? 20 : 24) = 4
- $b_3 = 11 + 4 - 14 = 1$

An further description can be found in Algorithm 8.

7.3.4 Update Neighboring Multi-Bit Conditions

To update neighboring multi-bit conditions the possible two-bit differences for neighboring multi-bit conditions have to be stored in two separated lists. For each entry in those lists the corresponding multi-bit values, either for bit0 or bit1 are calculated, depending on which two conditions are checked. See Section 7.3.2 for more details about bit0 and bit1. In the next step all possible combinations of these multi-bit values are tested. If no valid combination can be found, these two multi-bit conditions are inconsistent. For all valid combinations, we know that these values are still possible. Thus, we can store them into the updated condition. This must be done for all pairs of neighboring conditions.

Here is an example for two-bit conditions:

- $c_0^2 = 0x0210$
- $c_3^2 = 0x0802$

Updating those two conditions we have to update bit0 for both of their values as shown in Figure 17. For these values it can be checked which bits are set. Each set bit corresponds to a specific two-bit difference (see Tables 9 and 10):

- $c_0^2 = 0x0210 = 2^4 + 2^9 \quad \rightarrow \{(0,1),(1,2)\} = \{(0n),(nu)\}$

Algorithm 8 Transfer2BitCondition

Require: BoomerangState S, List of all 2-bit Conditions C of S
Ensure: **true** or **false**

for all 2-bit Condition c in C **do**
  $c_{new} \leftarrow 0$
  for $b_0 = 0$ **to** 63 **do**
    $b_1 \leftarrow ((b_0 \gg 2) \pmod 2)\ ?\ ((\ b_0 \pmod 2))\ ?\ b_0 - 5 : b_0 - 3) :$
    $((b_0 \pmod 2))\ ?\ b_0 + 3 : b_0 + 5)$
    $b_2 \leftarrow ((b_0 \gg 3) \pmod 2)\ ?\ (((b_0 \gg 1) \pmod 2))\ ?\ b_0 - 10 : b_0 - 6) :$
    $(((b_0 \gg 1) \pmod 2))\ ?\ b_0 + 6 : b_0 + 10)$
    $b_3 \leftarrow b_1 + b_2 - b_0$
    if c & $((1 \ll \text{b0}) \mid (1 \ll \text{b1}) \mid (1 \ll \text{b2}) \mid (1 \ll \text{b3}))$ **then**
      $c_{new} \leftarrow c_{new} \mid (1 \ll b_0)$
    end if
  end for
  if $c_{new} = 0$ **then**
    return false
  else
    set c on c_{new}
  end if
end for
return true

- $c_3^2 = 0x0802 = 2^1 + 2^{11} \rightarrow \{(1,0),(3,2)\} = \{(0u),(1u)\}$

From these lists of possible two-bit differences we calculate the needed two-bit values according to Table 12:

- $c_0^2 = 0x0210 = 2^4 + 2^9 \rightarrow \{(0,1),(1,2)\} = \{(0n),(nu)\} \rightarrow$ bit1: $\{(01),(10)\}$
- $c_3^2 = 0x0802 = 2^1 + 2^{11} \rightarrow \{(1,0),(3,2)\} = \{(0u),(1u)\} \rightarrow$ bit0: $\{(01),(11)\}$

In Table 14 we check all possible combinations of these two-bit values if they match.

Table 14: Checking all combinations of possible two-bit values

c_0^2	c_1^2	match
(01)	(01)	✓
(01)	(10)	-
(11)	(01)	-
(11)	(10)	-

Because one of the combinations is valid, the two conditions do not contradict. The previous check shows that for each conditions only one value is possible. Thus, the conditions can be updated. Therefore we have to store from which two-bit difference we get which two-bit value. From c_0^2 we got (01) from (0u) and from c_1^2 we got (01) from (0n). Then we can set the corresponding bits in the updated two-bit conditions:

- c_0^2': $\{(0n)\} = \{(0,1)\} \rightarrow c_1^2$'$= 2^4 = 0x0010$
- c_3^2': $\{(0u)\} = \{(1,0)\} \rightarrow c_0^2$'$= 2^1 = 0x0002$

7.3.5 Test Conditions

To find a message each bit must be assigned to a concrete value. Thus, each difference must be one of the four basic differences ('0', 'u', 'n', '1'). That implies that a condition with value 'x' finally must become 'u' or 'n', and respectively a condition with value '-' finally must become '0' or '1'. This fact can be used to do another test on the boomerang state. In Algorithm 9 the principle is shown. We set all conditions which are 'x', sequentially and temporarily first on 'u' (line 4) and then on 'n' (line 5). For both possibilities we update and check the boomerang state and store whether the state is consistent or not. If both changes lead to a conflict, it is not possible to find a valid assignment for the bits of the condition. Therefore, the state is invalid (line 11). If only one of these two changes leads to a conflict, we can set the according condition permanently on the other value, which leads to no conflict (lines 14, 16, 20, 22). Otherwise, we gain no new information and proceed doing this test with the next 'x' condition. The same concept can be applied on conditions with value '-'. Those conditions can be set sequentially and temporarily first on '0' (line 7)

and then on '1' (line 8). This test could also be done for all other generalized conditions which allow more than one difference.

Algorithm 9 ConditionTest

Require: BoomerangState S, List C of all Conditions of S
Ensure: **true** or **false**

```
1: for all Condition c in C do
2:     c_orig ← c
3:     if c = 'x' then
4:         set c in S on 'u'; v1 ← S.validate(); undo changes on S
5:         set c in S on 'n'; v2 ← S.validate(); undo changes on S
6:     else if c ='-' then
7:         set c in S on '0'; v1 ← S.validate(); undo changes on S
8:         set c in S on '1'; v2 ← S.validate(); undo changes on S
9:     end if
10:    if v1 = false and v2 = false then
11:        return false
12:    else if v1 = false then
13:        if c_orig = 'x' then
14:            set condition c in S on 'n'
15:        else
16:            set condition c in S on '1'
17:        end if
18:    else if v2 = false then
19:        if c_orig = 'x' then
20:            set condition c in S on 'u'
21:        else
22:            set condition c in S on '0'
23:        end if
24:    end if
25: end for
26: return true
```

7.4 Search a Boomerang Characteristic

Searching boomerang characteristics works in the same way as the search for first-order differential characteristics as described in Section 6.2. Therefore we treat the boomerang characteristics as one big characteristic. To improve the efficiency we guess only on C_0 and C_1 C_2 and C_3 will be determined using the updating algorithms. The search can again be applied on a subset of words which can be defined for each state independently.

We use the search also as an extended consistency check. If for a set of words no valid values can be found such that the characteristic is valid, the characteristic must be invalid.

The principle behind this approach is comparable with the condition test in Section 7.3.5. Searching on the complete characteristic would take too long since all possible values need to be tried until a valid fully determined characteristic is found. Because the switch is the most critical part of a boomerang characteristic, it is very likely to find inconsistencies in this part. Therefore, we search only on a small part of characteristics in the switching area. If for each condition in a set of words a concrete value can be found so that the resulting partially determined characteristic is valid, this need not imply that the characteristic has no hidden inconsistencies.

We distinguish between two groups of setups: the block search and the sequential search. In the block search we search on words of several steps in one search phase. On the contrary in a sequential search in each search phase we guess conditions only on words of one step. In Section 8, we compare the performance of both approaches.

However, if this technique is used, no concrete conflict can be detected. To gain still some information we create a statistic about how often guessing on a certain bit leads to an invalid characteristic. Bits which lead more often than others may be part of a conflict.

8 Attacks and Results

We applied our checking algorithms to four published boomerang characteristics for round reduced SHACAL-2. For all four characteristics contradictions were found manually before [BLMN11]. With our tool we can find these inconsistencies automatically. In the next step we use our tool to generate repaired characteristics. However, these characteristics still can have some hidden conflicts or an unacceptable low probability. Therefore, we apply a partially search on these characteristics. In this section, we present our results on checking, repairing and searching boomerang characteristics. For our attack we use a multiprocessor system with 16 Intel Xeon X5550 with 2.67GHz.

8.1 Searching Setups

For checking, repairing and searching boomerang characteristics we use several searching setups. In this section we shortly describe how a searching setup is defined and which setups we use.

Since our tool stores only the state words A and E and the message words W, a setup can contain only these words. Thus a setup to search for instance on first step would look like $\{A_0, E_0, W_0\}$. For the same reason we have to set for example A_{-1} to define the value of B_0. Thus, for SHACAL-2 it also possible to guess on $A_{(-4)-(-1)}$ and $E_{(-4)-(-1)}$. Additionally, we can define several searching phases and the values of conditions we guess on. For searching first on '?' in step five in state C_0, then on '?' in step five and six in state C_1 and finally on '?' and 'x' in step five and six in both states we define a searching setup as follows:

-P1: $\{A_{0,4}, \quad E_{0,4}, \quad W_{0,4} \quad \}$: {'?'}

-P2: $\{A_{1,4-5}, E_{1,4-5}, W_{1,4-5}\}$: {'?'}
-P3: $\{A_{4-5}, E_{4-5}, W_{4-5}\}$: {'?','x'}

For our attacks we use several searching setups. All of them search at least in the switching area of the characteristic. To evaluate whether it is better to search on all words in a single phase or use several phases we create searching setups for both cases. In $B1_n$ and $B2_n$ we define only one phase to search on n words. Contrarily, in $S1_n$ and $S2_n$ we divide the search into n phases. Additionally, we use mixed setups M_n and R_n with n phases in which we do first a sequential search and then a search on several steps in one phase. Moreover, we analyze if it is better to search on only one state or to search on two neighboring states. Therefore, the searching setups $B1$ and $S1$ includes only words of the first state, $B2$, $S2$ and $M2$ include also include words of the second state. Additionally, we vary the size of the setups. This means that they search in different numbers of steps. In all setups we search on '?' and 'x' as proposed for SHA-2 in [MNS11] for the first searching phase. To search a new boomerang characteristic we defined also a group of setups M and R, which guess additionally on '-' as proposed in [MNS11] for the second searching phase. All used setups can be found in Appendix B.

8.2 Check Boomerang Characteristics

In this section, we use our tool to check some recently published characteristics [KKL+05, LKKD06, Wan07, LK08, FGL09] for a round reduced SHACAL-2. These characteristics can be found in Tables 21-23. For three of them ([LKKD06, Wan07, FGL09]) we can find an inconsistency by doing a neighbor-state-update. For simplicity we call this group of characteristics $\mathbb{G}_1$. This conflict can be found in less than a second on a state-of-the-art home PC. The found conflict is in bit 13 of state word E after step 24 which corresponds to the results in [BLMN11]. Figure 18 shows the conditions which lead to the conflict.

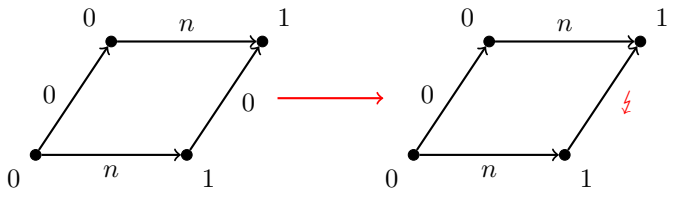

Figure 18: A bit quartet in a boomerang state

To determine that the other two characteristics ([KKL+05, LK08]) are invalid, we have to use a partial search. Therefore we try several searching setups. For the characteristic defined in [KKL+05] we search on words A, E and the message W in steps 24-26. This check is more complex and takes about 17 times as long as the check for $\mathbb{G}_1$. For the

characteristic from [LK08] we have to search on words A, E and W in steps 24-27. The check for this characteristic takes about 20 times as long as the check for $\mathbb{G}_1$.

Because we need three different detailed checks for the characteristics, we divide them into three groups of characteristics. Table 15 shows the needed time to prove the characteristics as inconsistent and the grouping. The value for the running times in this table are averaged over 100 executions.

Table 15: Running times to check characteristics

characteristic	group	running time [s]
$\mathbb{C}_1$ [KKL$^+$05]	$\mathbb{G}_2$	5.96
$\mathbb{C}_2$ [LKKD06]	$\mathbb{G}_1$	0.35
$\mathbb{C}_3$ [Wan07]	$\mathbb{G}_1$	0.35
$\mathbb{C}_4$ [LK08]	$\mathbb{G}_3$	7.19
$\mathbb{C}_5$ [FGL09]	$\mathbb{G}_1$	0.35

Doing all the checks for this chapter, we also created a statistic by which method how many conflicts were found. The results can be found in Table 16. If a method is not listed it has never found a conflict. The reason why a method has never found a conflict can also be the order in which the checks are done.

Table 16: Number of found conflicts per checking method

Check	Absolute	Percentage
Update single characteristic (Section 6.3)	8310451	32.5%
Transfer conditions (Section 7.3.1)	1903036	7.4%
Update neighboring states (Section 7.3.2)	13716602	53.6%
Transfer neighboring multi-bit Conditions (Section 7.3.4)	66485	0.3%
Test conditions (Section 7.3.5)	1602201	6.3%

In total for this statistic 147537077 checks were performed. Only 25598775 (17.4%) of them detected a conflict. According to this statistic about a third of all conflicts are found by updating the states of the boomerang characteristic independently. Almost all other conflicts can be found by updating the neighboring states of the boomerang characteristic. On the contrary, checks on multi-bit conditions has been able to detect conflicts only in 0.3% of all cases if no other check has been able to find a conflict before. Thus, for more speed checks on multi-bit conditions may be skipped. In relation to the computational effort with transfer conditions a reasonable amount of conflicts can be found (7.4%). However, the condition test is a very time consuming operation and it can find only 6.3% of the conflicts. Therefore, this test should not be done in every check.

8.3 Repair a Boomerang Characteristic

In this section, we compare the implemented debugging algorithms in combination with different setups. For the comparison we repaired only the characteristic described in [FGL09]. We decided to use a characteristic from group $\mathbb{G}_1$ because this characteristic can be checked faster than the others. To repair a characteristic many consistency checks are necessary, therefore this has a big influence on the running time.

Used abbreviations:

- -S: SequentialSearch (Section 5.1)
- -R: ReplayXPlain (Section 5.2)
- -Q: QuickXPlain (Section 5.3)
- -F: FastDiag (Section 5.4)
- -Rot: Rotate with QuickXPlain (Section 7.2)
- -Stat: Statistical (Section 7.2)
- -IC: intermediate check without searching
- -$B1_{04} \ldots S2_{07}$: see Appendix B

In order to compare the different debugging algorithms and setups we used them to generate repaired characteristics. Some of the setups are very fast, others take several hours to generate one boomerang characteristic. To collect a representative amount of statistical information we tried to generate at least 50 files for each setup. However, for instance ReplayXPlain is so slow that we decided to generate files only by doing fast check. Repairing with the statistical approach or with rotation using the setups $B1_{07}$ and $B2_{07}$ are also very slow. Hence, we generated also only a few files with them. Table 17 shows how many files were generated with which debugging algorithm in combination with which setup.

Table 17: Number of repaired characteristics

Setup	Q	F	S	R	Rot	Stat
IC	100	100	100	100	225	18
$B1_{04}$	50	50	50	0	0	0
$B1_{07}$	38	35	24	0	0	0
$B2_{04}$	50	50	50	0	0	0
$B2_{07}$	6	5	11	0	0	0
$S1_{04}$	50	50	50	0	0	0
$S1_{07}$	50	50	50	0	0	0
$S2_{04}$	50	50	50	0	0	0
$S2_{07}$	50	50	50	0	0	0

The bigger a diagnosis is the more conditions are set on '?' in the repaired boomerang characteristic. Thereby properties of the original characteristic are lost. To find characteristics with a high probability it is good to have areas without any active bits because

the probability for such partial differentials is 1. Thus, by setting conditions without a difference on '?', this property is lost as they can be set on a condition with a difference when we apply the search on them. We tried to repair the characteristic by removing only conditions with differences, but then the debugging algorithms were not able to repair the characteristic. Additionally, the complexity of searching confirming messages increases with the number of removed constraints since they have to be determined again. Therefore, small diagnoses are preferred.

In Table 18 the average size of the diagnoses of all repaired files can be found. It shows that the average size of diagnoses found with QuickXPlain is always bigger than the size of diagnoses found with FastDiag or SequentialSearch. This can be explained by the fact that QuickXPlain searches conflict sets and not a diagnosis. As described in Section 5.5 we combine them to a diagnosis but do not minimize it. For IC ReplayXPlain and QuickXPlain, the resulting diagnoses on average have the same size. That is not surprising since it is almost the same algorithm but without recursion. Also the statistical approach and the method using rotation return comparatively large diagnoses. Comparing FastDiag with SequentialSearch FastDiag yields smaller diagnosis in most cases. Diagnoses found by using SequentialSearch are on average 11.8% larger. Using a searching setup including more steps results also in a larger diagnosis. This is because a deep check on more steps is harder to fulfill than one on less steps, thus more conflicts are found. It is remarkable that for sequential searching setups the size of the diagnosis is larger than for searching on the same steps in only one phase. However, it makes no big difference if on one or two neighboring states are searched. Figure 19 shows a graphical comparison of the average diagnosis size of QuickXPlain, FastDiag and SequantialSearch for all setups.

Table 18: Average size of diagnosis

Setup	Q	F	S	R	Rot	Stat
IC	507	42	45	507	248	558
$B1_{04}$	1525	319	322	-	-	-
$B1_{07}$	1694	467	446	-	-	-
$B2_{04}$	1602	514	562	-	-	-
$B2_{07}$	2190	773	642	-	-	-
$S1_{04}$	2194	1376	1528	-	-	-
$S1_{07}$	2194	1389	1545	-	-	-
$S2_{04}$	1781	983	1179	-	-	-
$S2_{07}$	2384	1157	1577	-	-	-

If we get a well repaired characteristic which we can use as starting point to search a determined characteristic, time for reparation is not that important. Nevertheless, we want to take a look on the running times of the different algorithms. The first thing that is remarkable is the difference between ReplayXPlain and the other three debugging algorithms. ReplayXPlain needs at about ten times as long as QuickXPlain and even seventy times as long as FastDiag. The statistical approach is an iterative approach and

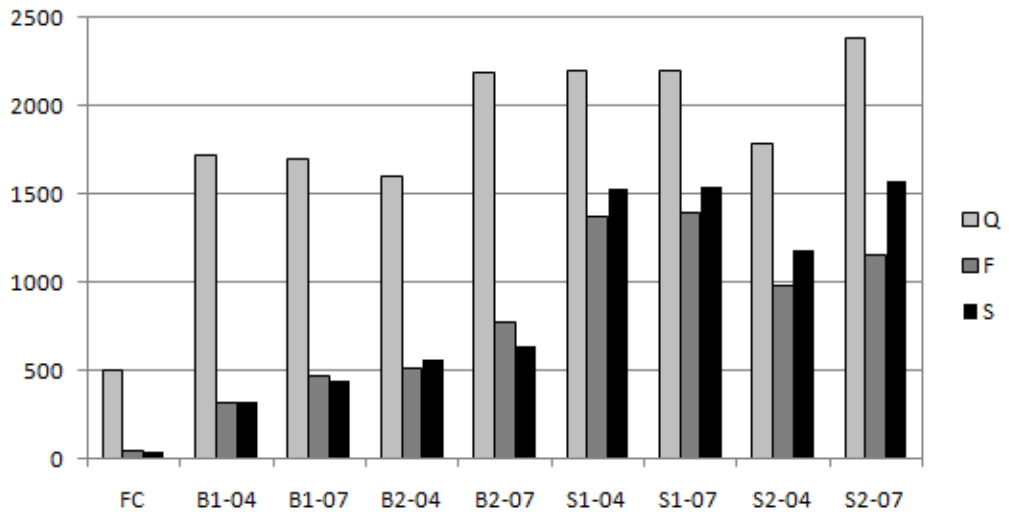

Figure 19: Average size of diagnosis

therefore much slower than the others. To debug one rotation with QuickXPlain we need about the same time as using QuickXPlain directly. I takes a bit longer, because we have to reinitialize the whole boomerang state before each iteration. It takes much longer to iterate over all possible $32 \cdot 32 = 1024$ rotations for SHACAL-2. Thus, we did not use ReplayXPlain, the statistical approach and the rotation method with other setups. The other debugging algorithms are quite comparable, whereby QuickXPlain is slower in most cases. On average SequentialSearch is faster than FastDiag. A repair with deep check using a setup which searches on more steps, is slower than a repair with a smaller setup. Moreover, using a sequential setup is much faster than searching on all steps in one phase, especially for larger setups. All averaged values can be found in Table 19.

Table 19: Average time in seconds to repair a characteristic

Setup	Q	F	S	R	Rot	Stat
IC	29	4	15	281	30*	1518
$B1_{04}$	1793	679	811	-	-	-
$B1_{07}$	9861	10492	11780	-	-	-
$B2_{04}$	5681	3877	3577	-	-	-
$B2_{07}$	45682	53849	18551	-	-	-
$S1_{04}$	1195	769	701	-	-	-
$S1_{07}$	1438	776	743	-	-	-
$S2_{04}$	1412	676	732	-	-	-
$S2_{07}$	2780	898	1273	-	-	-

Remark: * value for one specific rotation

The time which is needed to repair a characteristic is mainly determined by the number of checks and the time they need to find a conflict. Each debugging algorithm uses a differently debugging strategy and thus for each algorithm checking the test cases can vary in difficulty. In Table 20, the average number of checks for all algorithms and setups can be found. Figure 20 shows a comparison of QuickXPlain, FastDiag and SequentialSearch. It is remarkable that for these three algorithms for each setup the relation of needed checks is similar. In all cases FastDiag is the fastest and QuickXPlain the slowest algorithm. ReplayXPlain and the statistical approach need much more checks. To repair using QuickXPlain with rotation needs about as much as pure QuickXPlain, but this number of checks is valid only for one specific rotation and must be done for all possible rotations.

Table 20: Average number of checks to repair a characteristic

Setup	Q	F	S	R	Rot	Stat
IC	79	12	43	741	13	4823
$B1_{04}$	155	44	70	-	-	-
$B1_{07}$	149	53	67	-	-	-
$B2_{04}$	135	58	87	-	-	-
$B2_{07}$	188	87	104	-	-	-
$S1_{04}$	214	121	134	-	-	-
$S1_{07}$	214	122	134	-	-	-
$S2_{04}$	172	95	120	-	-	-
$S2_{07}$	278	112	163	-	-	-

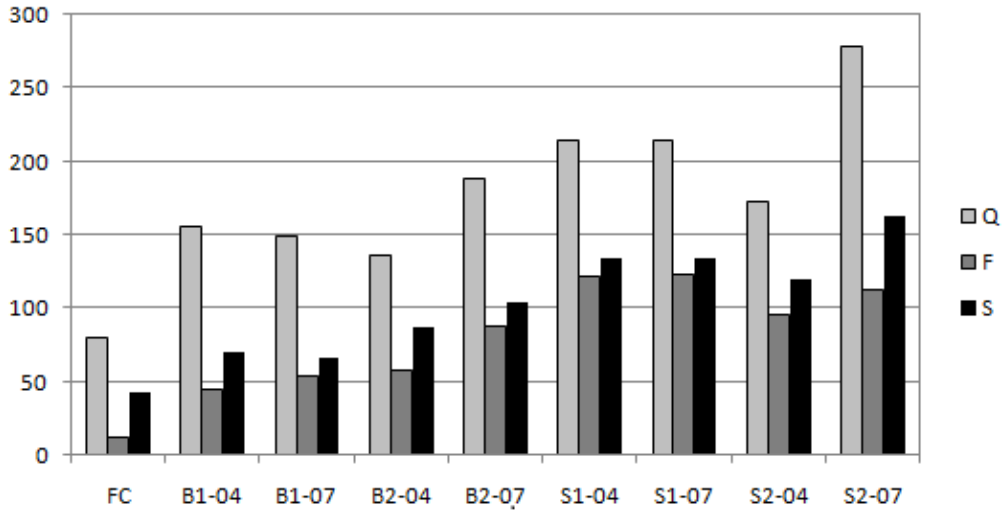

Figure 20: Average number of checks to repair a characteristic

After creating repaired boomerang characteristics we want to analyze their quality. The measurement of the quality is hard and finally can be only specified when the probability for the fully determined characteristic is known. Additionally the resulting characteristic depends strongly on the searching setup. However, we use the number of steps which can be determined while searching on them in a certain time as measure for the quality. Therefore, we search on all repaired characteristics with the setups $M1_{07}$ - $M1_{11}$, which contain more steps with increasing index. Figure 21 shows the percentage of files repaired by doing only an intermediate check which finished each setup sorted by the used algorithm. It shows that with the statistical approach partial more results can be found than for the others. The numbers of results for QuickXPlain and ReplayXPlain are comparable. Also a similar number of files repaired with FastDiag, SequentialSearch and the statistical approach can finish the search within time. Figure 22 shows the averaged percentage over all repaired characteristics for QuickXPlain, FastDiag and SequentialSearch. It shows again that characteristics repaired with QuickXPlain are more likely to finish the search in a certain time than the others. For FastDiag and SequentialSearch a similar number of results can be found.

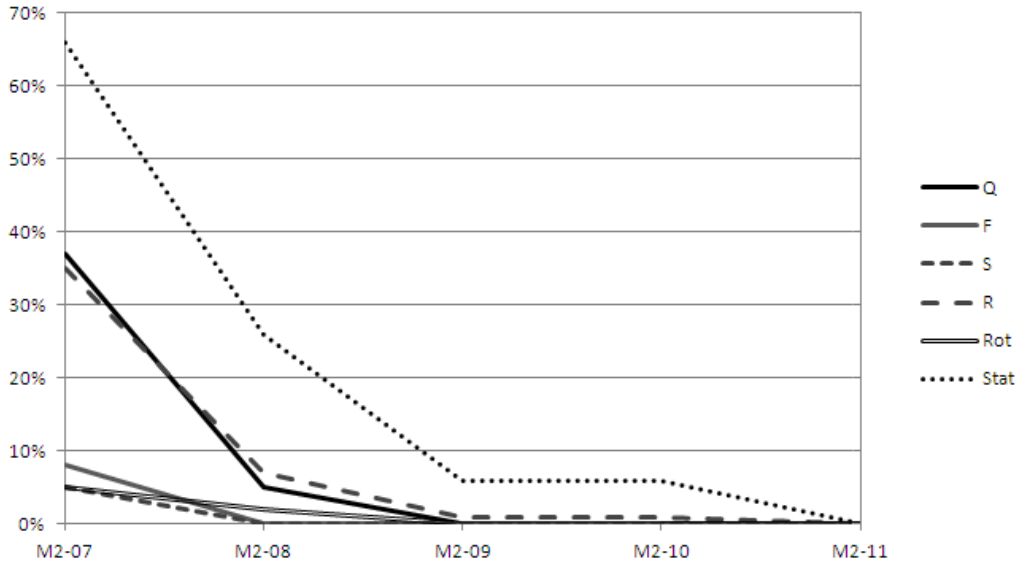

Figure 21: Percentage of finished searching setups - only IC

Summarizing it can be said that the statistical approach and the use of rotation can improve the debugging results. However, they need much more time. Comparing the debugging algorithms itself ReplayXPlain stands out through its bad time performance. In the other categories it is quite similar to QuickXPlain. Therefore, we prefer QuickXPlain against ReplayXPlain. SequentialSearch and FastDiag are very similar in all ratings. Using an algorithm which searches a diagnosis (FastDiag and SequentialSearch) result in smaller diagnoses than using an algorithm which searches a conflict set (QuickXPlain and

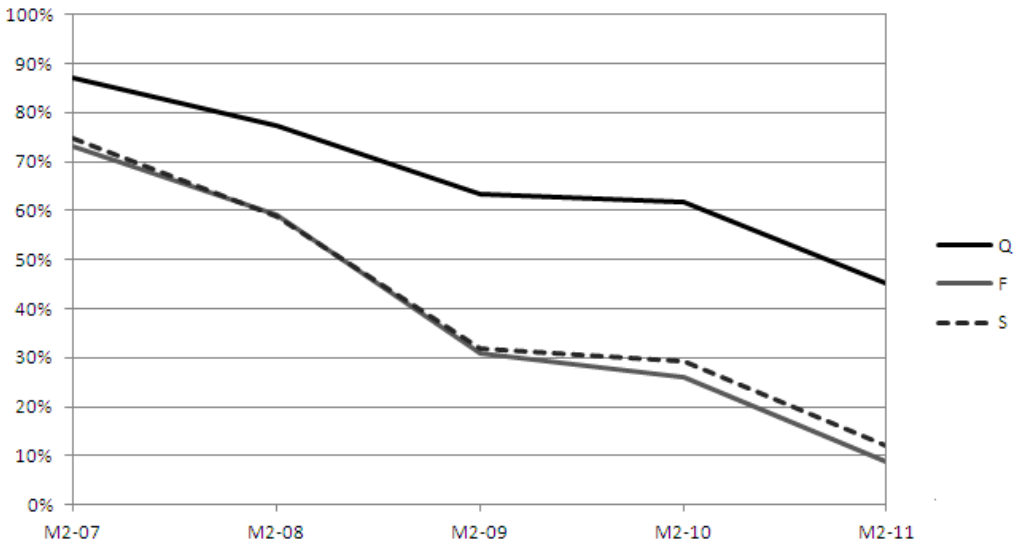

Figure 22: Averaged percentage of finished searching setups

ReplayXPlain) However, for characteristics repaired with larger diagnoses in search a result can be found faster, though they will be more different from the original characteristic.

8.4 Search of Boomerang Characteristic

After repairing a boomerang characteristic the removed constraints must be determined again. Therefore we use the boomerang search. In this section we will show that a good starting characteristic is the most crucial part of searching. Therefore we compare the results of searching on an automatically repaired characteristic and a manually repaired characteristic.

As mentioned before it is important that a repaired characteristic still shares some properties of the original characteristic. Mainly it is essential that high probability parts stay unchanged. We could not achieve this with automatically debugging currently. However, for a characteristic repaired with a searching setup containing many steps it is relatively easy to determine also many steps in the search. In return the final characteristic is very dense and at some point it gets really hard to determine more steps. We tried many variants, Table 26 shows the best characteristic we found. It is based on the characteristic published in [FGL09]. First, we rotated the words of the second characteristic four positions to the left. With this modification we wanted to reduce the number of active conditions at the same bit position. The rotated characteristic is shown in Table 24. Then we repaired it with QuickXPlain using the searching setup $S1_{07}$. From this result, we removed all differences from the message of the second characteristic because we did not want additional differences in this message (see Table 25). Finally, we searched on the repaired characteristic using setup R_{17}. The resulting boomerang characteristic is way too

dense, therefore we repaired a characteristic manually.

Knowing the critical areas from analyzing the automatically repaired characteristics we set all conditions of the words in the most critical areas on '?'. This manually repaired characteristic is shown in Table 27. Then we applied search with setup R_{19} on it and got the characteristic shown in Table 28. This time we found the sparsest characteristic without rotation. However, we could not determine all words.

Repairing characteristics currently seems to result in better results. Though, automatic reparation can be used to get a feeling for critical parts of the characteristics.

9 Conclusion

In this thesis, we presented the first automatic implementation of the boomerang attack and applied it on recently published boomerang characteristics for SHACAL-2. Therefore, we presented methods to check the consistency of boomerang characteristics, to repair them when the check fails and to search new characteristics and confirming messages.

To prove the consistency of a boomerang characteristic we developed algorithms to check with different levels of accuracy. First, a fast check just updates the boomerang characteristic by transferring single-bit and multi-bit conditions to opposing and neighboring states. Second, an intermediate check proves the consistency of the boomerang characteristic by setting single generalized conditions temporarily on all of its possible differences. If all possible differences lead to a conflict, the characteristic is invalid. Third, in the deep check we extend this approach by setting several conditions simultaneously in the characteristic search. If no valid assignment can be found, the characteristic must also be inconsistent. Using these methods we were able to find conflicts in recently published characteristics within seconds.

In order to repair boomerang characteristic we proposed the usage of debugging algorithms, since differential characteristics can be seen as set of constraints. First, with SequentialSearch we developed an own debugging algorithm. Then we also used the three debugging algorithms ReplayXPlain, QuickXPlain and FastDiag. Moreover, we proposed two extended repairing methods. First, a statistical approach, that does several iterations of debugging in which only the most frequent occurring conditions of the diagnoses are used to repair the characteristic until it is consistent. The other technique debugs the characteristic in all possible rotations and looks for the best one. Especially, with rotation we achieved better results than using pure debugging, but they need much more time. To evaluate the performance of the different debugging algorithms and methods we applied them on the characteristics published in [FGL09]. Therefore, we used also different searching setups to find a setup with a good running time to quality trade-off.

As we showed, automatically debugging needs improvements to find a good starting characteristic to search on it. Nevertheless, the presented debugging algorithms are able to consider a rating of the constraints. Therefore, a rating algorithm for constraints can be developed as future work. For instance, rating could protect high probability parts of the characteristics from being repaired. Therefore, with such an algorithm it would be possible

to repair characteristics as we did manually. As a next step repairing and searching could be combined. Then it is possible to repair a characteristic a bit if no solution can be found while searching on it. Additionally, the performance could be improved by using information about the location of a detected conflict in the debugging algorithm.

Searching characteristics is challenging task and needs much experience to choose the searching setup in a proper way. Future work is to improve the tool in way that conflicts can be found even earlier, so that repairing and searching new characteristics can be done faster. For instance, the linear two-bit conditions could be generated globally to relate bits of different substates. Additionally, the four conditions representing the same position in the four boomerang characteristics could be combined into one boomerang condition. This would reduce the complexity in updating all characteristics when one condition has changed in one of them.

With this tool boomerang characteristics can be checked and repaired. Furthermore, valid differential characteristics can be found automatically which helps to increase the number of attacked rounds.

References

[BC04] Eli Biham and Rafi Chen. Near-collisions of sha-0. In *Advances in Cryptology–CRYPTO 2004*, pages 290–305. Springer, 2004.

[BCH+90] Bruno O Brachtl, Don Coppersmith, Myrna M Hyden, Stephen M Matyas Jr, Carl HW Meyer, Jonathan Oseas, Shaiy Pilpel, and Michael Schilling. Data authentication using modification detection codes based on a public one way encryption function, March 13 1990. US Patent 4,908,861.

[BDK01] Eli Biham, Orr Dunkelman, and Nathan Keller. The rectangle attackrectangling the serpent. *Advances in CryptologyEUROCRYPT 2001*, pages 340–357, 2001.

[Bih94] Eli Biham. New types of cryptanalytic attacks using related keys. *Journal of Cryptology*, 7(4):229–246, 1994.

[BKL+07] Andrey Bogdanov, Lars Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. Present: An ultra-lightweight block cipher. *Cryptographic Hardware and Embedded Systems-CHES 2007*, pages 450–466, 2007.

[BKN09] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolić. Distinguisher and related-key attack on the full aes-256. In *Advances in Cryptology-CRYPTO 2009*, pages 231–249. Springer, 2009.

[BLMN11] Alex Biryukov, Mario Lamberger, Florian Mendel, and Ivica Nikolic. Second-order differential collisions for reduced sha-256. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT*, volume 7073 of *LNCS*, pages 270 – 287. Springer, 2011.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. *Journal of CRYPTOLOGY*, 4(1):3–72, 1991.

[CDMP05] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-damgård revisited: How to construct a hash function. In *Advances in Cryptology–CRYPTO 2005*, pages 430–448. Springer, 2005.

[Cop94] Don Coppersmith. The data encryption standard (des) and its strength against attacks. *IBM journal of research and development*, 38(3):243–250, 1994.

[dBB94] Bert den Boer and Antoon Bosselaers. Collisions for the compression function of md5. In *Advances in CryptologyEurocrypt93*, pages 293–304. Springer, 1994.

[DCR06] Christophe De Canniere and Christian Rechberger. Finding sha-1 characteristics: General results and applications. *Advances in Cryptology–ASIACRYPT 2006*, pages 1–20, 2006.

[Des77] Des. Data encryption standard. In *In FIPS PUB 46, Federal Information Processing Standards Publication*, pages 46–2, 1977.

[DH77] Whitfield Diffie and Martin Edward Hellman. Exhaustive cryptanalysis of the nbs data encryption standard. *Computer*, 10(6):74–84, 1977.

[DKR97] Joan Daemen, Lars Knudsen, and Vincent Rijmen. The block cipher square. In *Fast Software Encryption*, pages 149–165. Springer, 1997.

[Dob98] Hans Dobbertin. Cryptanalysis of md4. *Journal of Cryptology*, 11(4):253–271, 1998.

[DSP88] JL De Siqueira and Jean-Francois Puget. Explanation-based generalisation of failures. In *Proceedings of ECAI*, volume 88, pages 339–344, 1988.

[FGL09] Ewan Fleischmann, Michael Gorski, and Stefan Lucks. Memoryless related-key boomerang attack on 39-round shacal-2. *Information Security Practice and Experience*, pages 310–323, 2009.

[FMS01] Scott Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm of rc4. In *Selected areas in cryptography*, pages 1–24. Springer, 2001.

[FO90] Philippe Flajolet and Andrew M Odlyzko. Random mapping statistics. In *Advances in cryptologyEUROCRYPT89*, pages 329–354. Springer, 1990.

[FS10] Alexander Felfernig and Monika Schubert. Fastdiag: A diagnosis algorithm for inconsistent constraint sets. In *21st International Workshop on the Principles of Diagnosis*, pages 31–38, 2010.

[HKK+03] Seokhie Hong, Jongsung Kim, Guil Kim, Jaechul Sung, Changhoon Lee, and Sangjin Lee. Impossible differential attack on 30-round shacal-2. In *Progress in Cryptology-INDOCRYPT 2003*, pages 97–106. Springer, 2003.

[HN00] Helena Handschuh and David Naccache. Shacal. In *Proceedings of First Open NESSIE Workshop*, 2000.

[Jou04] Antoine Joux. Multicollisions in iterated hash functions. application to cascaded constructions. In *Advances in Cryptology–CRYPTO 2004*, pages 306–316. Springer, 2004.

[Jun01] Ulrich Junker. Quickxplain: Conflict detection for arbitrary constraint propagation algorithms. In *IJCAI01 Workshop on Modelling and Solving problems with constraints*. Citeseer, 2001.

[Jun04] Ulrich Junker. Quickxplain: preferred explanations and relaxations for over-constrained problems. In *Proceedings of the National Conference on Artificial Intelligence*, pages 167–172. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2004.

[KKL+05] Jongsung Kim, Guil Kim, Sangjin Lee, Jongin Lim, and Junghwan Song. Related-key attacks on reduced rounds of shacal-2. *Progress in Cryptology-INDOCRYPT 2004*, pages 55–97, 2005.

[KKS01] John Kelsey, Tadayoshi Kohno, and Bruce Schneier. Amplified boomerang attacks against reduced-round mars and serpent. In *Fast Software Encryption*, pages 13–23. Springer, 2001.

[Knu93] Lars Ramkilde Knudsen. Cryptanalysis of loki 91. In *Advances in CryptologyAUSCRYPT'92*, pages 196–208. Springer, 1993.

[Knu95] Lars Knudsen. Truncated and higher order differentials. In *Fast Software Encryption*, pages 196–211. Springer, 1995.

[Knu98] Lars Knudsen. Deal - a 128-bit block cipher. In *NIST AES Proposal*, 1998.

[KYK10] Manoj Kumar, Pratibha Yadav, and Meena Kumari. Flaws in differential cryptanalysis of reduced round present. *Cryptology e-Print Archive Report*, 407, 2010.

[Lai94] Xuejia Lai. Higher order derivatives and differential cryptanalysis. In *Communications and Cryptography*, pages 227–233. Springer, 1994.

[Leu12a] Gaëtan Leurent. Analysis of differential attacks in arx constructions. In *Advances in Cryptology–ASIACRYPT 2012*, pages 226–243. Springer, 2012.

[Leu12b] Gaëtan Leurent. Arxtools: A toolkit for arx analysis. In *The Third SHA-3 Candidate Conference*, 2012.

[LIS12] Ji Li, Takanori Isobe, and Kyoji Shibutani. Converting meet-in-the-middle preimage attack into pseudo collision attack: Application to sha-2. In *Fast Software Encryption*, volume 7549 of *Lecture Notes in Computer Science*, pages 264–286. Springer Berlin Heidelberg, 2012.

[LK08] Jiqiang Lu and Jongsung Kim. Attacking 44 rounds of the shacal-2 block cipher using related-key rectangle cryptanalysis. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E91.A(9):2588–2596, 2008.

[LKKD06] Jiqiang Lu, Jongsung Kim, Nathan Keller, and Orr Dunkelman. Related-key rectangle attack on 42-round shacal-2. *Information Security*, pages 85–100, 2006.

[LM93] Xucjia Lai and JamesL. Massey. Hash functions based on block ciphers. In *Advances in Cryptology EUROCRYPT 92*, volume 658 of *Lecture Notes in Computer Science*, pages 55–70. Springer Berlin Heidelberg, 1993.

[LMM91] Xuejia Lai, James L Massey, and Sean Murphy. Markov ciphers and differential cryptanalysis. In *Advances in CryptologyEUROCRYPT91*, pages 17–38. Springer, 1991.

[LR12] Gaëtan Leurent and Arnab Roy. Boomerang attacks on hash function using auxiliary differentials. In *Topics in Cryptology–CT-RSA 2012*, pages 215–230. Springer, 2012.

[Luc00] Stefan Lucks. The saturation attack - a bait for twofish. Cryptology ePrint Archive, Report 2000/046, 2000.

[Mer79] Ralph Charles Merkle. Secrecy, authentication, and public key systems. 1979.

[MIO89] Shoji Miyaguchi, Masahiko Iwata, and Kazuo Ohta. New 128-bit hash function. In *Proc. 4th International Joint Workshop on Computer Communications, Tokyo, Japan*, pages 279–288, 1989.

[MMO85] Stephen M Matyas, Carl H Meyer, and Jonathan Oseas. Generating strong one-way functions with cryptographic algorithm. *IBM Technical Disclosure Bulletin*, 27(10A):5658–5659, 1985.

[MNS11] Florian Mendel, Tomislav Nad, and Martin Schläffer. Finding sha-2 characteristics: Searching through a minefield of contradictions. In *Advances in Cryptology–ASIACRYPT 2011*, pages 288–307. Springer, 2011.

[MRST09] Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S Thomsen. The rebound attack: Cryptanalysis of reduced whirlpool and grøstl. In *Fast Software Encryption*, pages 260–276. Springer, 2009.

[Mur09] Sean Murphy. The return of the boomerang. Technical report, Technical Report RHUL-MA-2009-20, Department of Mathematics, Royal Holloway, University of London, UK., 2009.

[MVDCP11] Nicky Mouha, Vesselin Velichkov, Christophe De Cannière, and Bart Preneel. The differential analysis of s-functions. In *Selected Areas in Cryptography*, pages 36–56. Springer, 2011.

[MY93] Mitsuru Matsui and Atsuhiro Yamagishi. A new method for known plaintext attack of feal cipher. In *Advances in CryptologyEurocrypt92*, pages 81–91. Springer, 1993.

[Nat08] National Institute of Standards and Technology. FIPS 180-3, Secure Hash Standard, Federal Information Processing Standard (FIPS), Publication 180-3. Technical report, Department of Commerce, August 2008.

[NIS01] NIST. *Advanced Encryption Standard (AES) (FIPS PUB 197).* National Institute of Standards and Technology, November 2001.

[PGV94] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash functions based on block ciphers: A synthetic approach. In *Advances in CryptologyCRYPTO93*, pages 368–378. Springer, 1994.

[Rei87] Raymond Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32(1):57–95, 1987.

[Rob65] John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM (JACM)*, 12(1):23–41, 1965.

[SKK+04] Yongsup Shin, Jongsung Kim, Guil Kim, Seokhie Hong, and Sangjin Lee. Differential-linear type attacks on reduced rounds of shacal-2. In *Information Security and Privacy*, pages 110–122. Springer, 2004.

[Wag99] David Wagner. The boomerang attack. In *Fast Software Encryption*, pages 156–170. Springer, 1999.

[Wan07] Gaoli Wang. Related-key rectangle attack on 43-round shacal-2. *Information Security Practice and Experience*, pages 33–42, 2007.

[WB01] Renate Wahrig-Burfein. *Schüler Bertelsmann - Wahrig Fremdwörterlexikon.* Bertelsmann Lexikon Verlag, 2001.

[Win84] Robert S Winternitz. Producing a one-way hash function from des. In *Advances in Cryptology*, pages 203–207. Springer, 1984.

[WLF+05] Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the hash functions md4 and ripemd. In *Advances in Cryptology–EUROCRYPT 2005*, pages 1–18. Springer, 2005.

[WY05] Xiaoyun Wang and Hongbo Yu. How to break md5 and other hash functions. In *Advances in Cryptology–EUROCRYPT 2005*, pages 19–35. Springer, 2005.

[WYY05a] Xiaoyun Wang, Yiqun Yin, and Hongbo Yu. Finding collisions in the full sha-1. In *Advances in Cryptology–CRYPTO 2005*, pages 17–36. Springer, 2005.

[WYY05b] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient collision search attacks on sha-0. In *Advances in Cryptology–CRYPTO 2005*, pages 1–16. Springer, 2005.

[Yuv79] Gideon Yuval. How to swindle rabin. *Cryptologia*, 3(3):187–191, 1979.

[ZG11] Bo Zhu and Guang Gong. Multidimensional meet-in-the-middle attack and its applications to katan32/48/64. Cryptology ePrint Archive, Report 2011/619, 2011.

A Characteristics

In this chapter we list different characteristics. As described in Section 6.3 and Section 8.1 we store only the words for A_i, E_i and W_i in our states. Therefore, we have to define the values $A_{(-4)-(-1)}$ and $E_{(-4)-(-1)}$ to have all input words for the first step. The following tables show only two neighboring boomerang characteristics because the initial values of other two can be assigned by an update on the boomerang state .

Remark on Table 22: The upper characteristic in Table 22 is very similar to the upper characteristic in Table 21. It differs only in differences of word ΔD_0 and ΔH_0 and has an additional step at the beginning and end each. The characteristics described in [LKKD06, Wan07, FGL09] identically except from the differences in word E_{-4}. For simplicity we set all conditions in this word on '?'.

Remark: on Table 23: The upper characteristic in Table 23 is equal to the upper characteristic in Table 22.

Table 21: SHACAL-2 - 33 Round Characteristics from [KKL+05]

i	A_i	E_i	W_i
-4	x------------------------------	x------------------------------	
-3	--x---x-----x-x--------x--x------	--x----------x--------------------	
-2	------------------------------	----------x----x---x---------	
-1	------------------------------	------------------------------	
0	x------------------------------	------------------------------	------------------------------
1	------------------------------	------x---x-------------x------	------------------------------
2	------------------------------	------------------------------	------------------------------
3	------------------------------	------------------------------	------------------------------
4	------------------------------	x------------------------------	------------------------------
5	------------------------------	------------------------------	------------------------------
6	------------------------------	------------------------------	------------------------------
7	------------------------------	------------------------------	------------------------------
8	------------------------------	------------------------------	x------------------------------
9	------------------------------	------------------------------	------------------------------
10	------------------------------	------------------------------	------------------------------
11	------------------------------	------------------------------	------------------------------
12	------------------------------	------------------------------	------------------------------
13	------------------------------	------------------------------	------------------------------
14	------------------------------	------------------------------	------------------------------
15	------------------------------	------------------------------	------------------------------
16	------------------------------	------------------------------	------------------------------
17	------------------------------	------------------------------	------------------------------
18	------------------------------	------------------------------	------------------------------
19	------------------------------	------------------------------	------------------------------
20	------------------------------	------------------------------	------------------------------
21	------------------------------	------------------------------	------------------------------
22	------------------------------	------------------------------	------------------------------
23	????????????????????????????????	????????????????????????????????	????????????????????????????????
24	????????????????????????????????	????????????????????????????????	????????????????????????????????
25	????????????????????????????????	????????????????????????????????	????????????????????????????????
26	????????????????????????????????	????????????????????????????????	????????????????????????????????
27	????????????????????????????????	????????????????????????????????	????????????????????????????????
28	????????????????????????????????	????????????????????????????????	????????????????????????????????
29	????????????????????????????????	????????????????????????????????	????????????????????????????????
30	????????????????????????????????	????????????????????????????????	????????????????????????????????
31	????????????????????????????????	????????????????????????????????	????????????????????????????????
32	????????????????????????????????	????????????????????????????????	????????????????????????????????

i	A_i	E_i	W_i
-4	????????????????????????????????	????????????????????????????????	
-3	????????????????????????????????	????????????????????????????????	
-2	????????????????????????????????	????????????????????????????????	
-1	????????????????????????????????	????????????????????????????????	
0	????????????????????????????????	????????????????????????????????	????????????????????????????????
1	????????????????????????????????	????????????????????????????????	????????????????????????????????
2	????????????????????????????????	????????????????????????????????	????????????????????????????????
3	????????????????????????????????	????????????????????????????????	????????????????????????????????
4	????????????????????????????????	????????????????????????????????	????????????????????????????????
5	????????????????????????????????	????????????????????????????????	????????????????????????????????
6	????????????????????????????????	????????????????????????????????	????????????????????????????????
7	????????????????????????????????	????????????????????????????????	????????????????????????????????
8	????????????????????????????????	????????????????????????????????	????????????????????????????????
9	????????????????????????????????	????????????????????????????????	????????????????????????????????
10	????????????????????????????????	????????????????????????????????	????????????????????????????????
11	????????????????????????????????	????????????????????????????????	????????????????????????????????
12	????????????????????????????????	????????????????????????????????	????????????????????????????????
13	????????????????????????????????	????????????????????????????????	????????????????????????????????
14	????????????????????????????????	????????????????????????????????	????????????????????????????????
15	????????????????????????????????	????????????????????????????????	????????????????????????????????
16	????????????????????????????????	????????????????????????????????	????????????????????????????????
17	????????????????????????????????	????????????????????????????????	????????????????????????????????
18	????????????????????????????????	????????????????????????????????	????????????????????????????????
19	------------------------------	------------------------------	????????????????????????????????
20	------------------------------	------------------------------	????????????????????????????????
21	--x---------x-------x---------	--x---x---x-x-------x--x------	????????????????????????????????
22	------------------------------	x------------------------------	????????????????????????????????
23	------------------------------	------------------------------	------------------------------
24	x------------------------------	x------------------------------	------------------------------
25	------------------------------	------------------------------	------------------------------
26	------------------------------	------------------------------	------------------------------
27	------------------------------	------------------------------	------------------------------
28	x------------------------------	------------------------------	------------------------------
29	--x---------x-------x---------	------------------------------	------------------------------
30	----x------?----------x------	------------------------------	------------------------------
31	------xx-----?-xx---------x---	------------------------------	------------------------------
32	--------x----------x----------	x------------------------------	------------------------------

Table 22: SHACAL-2 - 35 Round Characteristics from [LKKD06, Wan07, FGL09]

i	A_i	E_i	W_i
-4	----------	??????????????????????????????????	
-3	x---------	x---------	
-2	--x---x---x-x-------x--x------	--x---------x---------	
-1	----------	----------x---x--x---------	
0	----------	----------	x---------
1	x---------	----------	----------
2	----------	-----x---x----------x------	----------
3	----------	----------	----------
4	----------	----------	----------
5	----------	x---------	----------
6	----------	----------	----------
7	----------	----------	----------
8	----------	----------	----------
9	----------	----------	x---------
10	----------	----------	----------
11	----------	----------	----------
12	----------	----------	----------
13	----------	----------	----------
14	----------	----------	----------
15	----------	----------	----------
16	----------	----------	----------
17	----------	----------	----------
18	----------	----------	----------
19	----------	----------	----------
20	----------	----------	----------
21	----------	----------	----------
22	----------	----------	----------
23	----------	----------	----------
24	---x---x----------x----------	---x---x----------x----------	??????????????????????????????????
25	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
26	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
27	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
28	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
29	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
30	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
31	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
32	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
33	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
34	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????

i	A_i	E_i	W_i
-4	??????????????????????????????????	??????????????????????????????????	
-3	??????????????????????????????????	??????????????????????????????????	
-2	??????????????????????????????????	??????????????????????????????????	
-1	??????????????????????????????????	??????????????????????????????????	
0	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
1	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
2	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
3	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
4	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
5	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
6	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
7	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
8	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
9	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
10	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
11	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
12	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
13	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
14	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
15	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
16	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
17	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
18	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
19	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
20	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
21	----------	----------	??????????????????????????????????
22	x-x---x----x-x-------x--x------	x-x---------x----------	??????????????????????????????????
23	x---------	----------x----x--x---------	??????????????????????????????????
24	x---------	----------	??????????????????????????????????
25	x---------	----------	----------
26	----------	-----x---x----------x------	----------
27	----------	x---------	----------
28	----------	x---------	----------
29	----------	x---------	----------
30	----------	----------	----------
31	----------	----------	----------
32	----------	----------	----------
33	x---------	x---------	----------
34	--x---x---x-x-------x--x------	-----x---x----------x------	----------

Table 23: SHACAL-2 - 35 Round Characteristics from [LK08]

i	A_i	E_i	W_i
-4	----------	????????????????????????????????	
-3	x---------	x---------	
-2	--x---x---x-x-------x--x------	--x---------x---------	
-1	----------	----------x---x--x---------	
0	----------	----------	x---------
1	x---------	----------	----------
2	----------	-----x---x----------x------	----------
3	----------	----------	----------
4	----------	----------	----------
5	----------	x---------	----------
6	----------	----------	----------
7	----------	----------	----------
8	----------	----------	----------
9	----------	----------	x---------
10	----------	----------	----------
11	----------	----------	----------
12	----------	----------	----------
13	----------	----------	----------
14	----------	----------	----------
15	----------	----------	----------
16	----------	----------	----------
17	----------	----------	----------
18	----------	----------	----------
19	----------	----------	----------
20	----------	----------	----------
21	----------	----------	----------
22	----------	----------	----------
23	----------	----------	----------
24	---x---x----------x----------	---x---x----------x----------	????????????????????????????????
25	????????????????????????????????	????????????????????????????????	????????????????????????????????
26	????????????????????????????????	????????????????????????????????	????????????????????????????????
27	????????????????????????????????	????????????????????????????????	????????????????????????????????
28	????????????????????????????????	????????????????????????????????	????????????????????????????????
29	????????????????????????????????	????????????????????????????????	????????????????????????????????
30	????????????????????????????????	????????????????????????????????	????????????????????????????????
31	????????????????????????????????	????????????????????????????????	????????????????????????????????
32	????????????????????????????????	????????????????????????????????	????????????????????????????????
33	????????????????????????????????	????????????????????????????????	????????????????????????????????
34	????????????????????????????????	????????????????????????????????	????????????????????????????????

i	A_i	E_i	W_i
-4	????????????????????????????????	????????????????????????????????	
-3	????????????????????????????????	????????????????????????????????	
-2	????????????????????????????????	????????????????????????????????	
-1	????????????????????????????????	????????????????????????????????	
0	????????????????????????????????	????????????????????????????????	????????????????????????????????
1	????????????????????????????????	????????????????????????????????	????????????????????????????????
2	????????????????????????????????	????????????????????????????????	????????????????????????????????
3	????????????????????????????????	????????????????????????????????	????????????????????????????????
4	????????????????????????????????	????????????????????????????????	????????????????????????????????
5	????????????????????????????????	????????????????????????????????	????????????????????????????????
6	????????????????????????????????	????????????????????????????????	????????????????????????????????
7	????????????????????????????????	????????????????????????????????	????????????????????????????????
8	????????????????????????????????	????????????????????????????????	????????????????????????????????
9	????????????????????????????????	????????????????????????????????	????????????????????????????????
10	????????????????????????????????	????????????????????????????????	????????????????????????????????
11	????????????????????????????????	????????????????????????????????	????????????????????????????????
12	????????????????????????????????	????????????????????????????????	????????????????????????????????
13	????????????????????????????????	????????????????????????????????	????????????????????????????????
14	????????????????????????????????	????????????????????????????????	????????????????????????????????
15	????????????????????????????????	????????????????????????????????	????????????????????????????????
16	????????????????????????????????	????????????????????????????????	????????????????????????????????
17	????????????????????????????????	????????????????????????????????	????????????????????????????????
18	????????????????????????????????	????????????????????????????????	????????????????????????????????
19	????????????????????????????????	????????????????????????????????	????????????????????????????????
20	????????????????????????????????	????????????????????????????????	????????????????????????????????
21	x----------	x----------x----------	????????????????????????????????
22	--x---x---x-x-------x--x------	--x---------x---x----------	????????????????????????????????
23	----------	----------x----x--x----------	????????????????????????????????
24	----------	----------	----------
25	x---------	----------	----------
26	----------	-----x---x----------x------	----------
27	----------	----------	----------
28	----------	----------	----------
29	----------	x---------	----------
30	----------	----------	----------
31	----------	----------	----------
32	----------	----------	----------
33	x---------	x---------	----------
34	--x---x---x-x-------x--x------	-----x---x----------x------	----------

Table 24: SHACAL-2 - 35 Round Characteristics: rotated [FGL09]

i	A_i	E_i	W_i
-4	----------	??????????????????????????????	
-3	x---------	x---------	
-2	--x---x---x-x-------x--x------	--x---------x---------	
-1	----------	----------x---x--x---------	
0	----------	----------	x---------
1	x---------	----------	----------
2	----------	-----x---x----------x------	----------
3	----------	----------	----------
4	----------	----------	----------
5	----------	x---------	----------
6	----------	----------	----------
7	----------	----------	----------
8	----------	----------	----------
9	----------	----------	x---------
10	----------	----------	----------
11	----------	----------	----------
12	----------	----------	----------
13	----------	----------	----------
14	----------	----------	----------
15	----------	----------	----------
16	----------	----------	----------
17	----------	----------	----------
18	----------	----------	----------
19	----------	----------	----------
20	----------	----------	----------
21	----------	----------	----------
22	----------	----------	----------
23	----------	----------	----------
24	---x---x----------x----------	---x---x----------x----------	??????????????????????????????
25	??????????????????????????????	??????????????????????????????	??????????????????????????????
26	??????????????????????????????	??????????????????????????????	??????????????????????????????
27	??????????????????????????????	??????????????????????????????	??????????????????????????????
28	??????????????????????????????	??????????????????????????????	??????????????????????????????
29	??????????????????????????????	??????????????????????????????	??????????????????????????????
30	??????????????????????????????	??????????????????????????????	??????????????????????????????
31	??????????????????????????????	??????????????????????????????	??????????????????????????????
32	??????????????????????????????	??????????????????????????????	??????????????????????????????
33	??????????????????????????????	??????????????????????????????	??????????????????????????????
34	??????????????????????????????	??????????????????????????????	??????????????????????????????

i	A_i	E_i	W_i
-4	??????????????????????????????	??????????????????????????????	
-3	??????????????????????????????	??????????????????????????????	
-2	??????????????????????????????	??????????????????????????????	
-1	??????????????????????????????	??????????????????????????????	
0	??????????????????????????????	??????????????????????????????	----------
1	??????????????????????????????	??????????????????????????????	----------
2	??????????????????????????????	??????????????????????????????	----------
3	??????????????????????????????	??????????????????????????????	----------
4	??????????????????????????????	??????????????????????????????	----------
5	??????????????????????????????	??????????????????????????????	----------
6	??????????????????????????????	??????????????????????????????	----------
7	??????????????????????????????	??????????????????????????????	----------
8	??????????????????????????????	??????????????????????????????	----------
9	??????????????????????????????	??????????????????????????????	----------
10	??????????????????????????????	??????????????????????????????	----------
11	??????????????????????????????	??????????????????????????????	----------
12	??????????????????????????????	??????????????????????????????	----------
13	??????????????????????????????	??????????????????????????????	----------
14	??????????????????????????????	??????????????????????????????	----------
15	??????????????????????????????	??????????????????????????????	----------
16	??????????????????????????????	??????????????????????????????	----------
17	??????????????????????????????	??????????????????????????????	----------
18	??????????????????????????????-	??????????????????????????????-	----------
19	??????????????????????????????-	??????????????????????????????-	----------
20	??????????????????????????????-	??????????????????????????????-	----------
21	----------	----------	----------
22	--x---x-x-------x--x------x-x-	---------x----------x-x-	----------
23	----------x---	--------x---x--x----------	----------
24	----------x---	----------	----------
25	----------x---	----------	----------
26	----------	--x---x----------x----------	----------
27	----------	----------x---	----------
28	----------	----------x---	----------
29	----------	----------x---	----------
30	----------	----------	----------
31	----------	----------	----------
32	----------	----------	----------
33	----------x---	----------x---	----------
34	--x---x-x-------x--x-------x-	--x---x----------x----------	----------

Table 25: Characteristic from Table 24 repaired with QuickXPLain and $S1_{07}$

i	A_i	E_i	W_i
-4	----------	??????????????????????????????????	
-3	??????????????????????????????????	??????????????????????????????????	
-2	--x---x----x-x--------x--x------	??????????????????????????????????	
-1	??????????????????????????????????	----------x-----x---x---------	
0	??????????????????????????????????	----------	x---------
1	??????????????????????????????????	----------	??????????????????????????????????
2	??????????????????????????????????	------x----x----------x------	??????????????????????????-??????
3	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
4	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
5	??????????????????????????????????	??????????????????????????????-??	----------
6	??????????????????????????????????	??????????????????????????????????	----------
7	??????????????????????????????????	----------	----------
8	??????????????????????????????????	??????????????????????????????????	----------
9	??????????????????????????????????	??????????????????????????????????	x---------
10	??????????????????????????????????	----------	??????????????????????????????????
11	----------	----------	??????????????????????????????????
12	----------	??????????????????????????????????	??????????????????????????-????
13	----------	----------	??????????????????????????????????
14	??????????????????????????????????	----------	----------
15	??????????????????????????????????	----------	??????????????????????????????????
16	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
17	??????????????????????????????????	----------	??????????????????????????????????
18	----------	??????????????????????????????????	??????????????????????????????????
19	----------	??????????????????????????????????	----------
20	----------	??????????????????????????????????	??????????????????????????????????
21	??????????????????????????????????	??????????-??????????????????????	----------
22	----------	??????????????????????????????????	??????????????????????????????????
23	??????????????????????????????????	----------	??????????????????????????????????
24	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
25	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
26	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
27	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
28	??????????????????????????????????	??-????-?????????????-??????????	??????????????????????????????????
29	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
30	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
31	??????????????????????????????????	??????????????????????????????-???	??????????????????????????????????
32	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
33	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????
34	??????????????????????????????????	??????????????????????????????????	??????????????????????????????????

i	A_i	E_i	W_i
-4	??????????????????????????????????	??????????????????????????????????	
-3	??????????????????????????????????	??????????????????????????????????	
-2	??????????????????????????????????	??????????????????????????????????	
-1	??????????????????????????????????	??????????????????????????????????	
0	??????????????????????????????????	??????????????????????????????????	----------
1	??????????????????????????????????	??????????????????????????????????	----------
2	??????????????????????????????????	??????????????????????????????????	----------
3	??????????????????????????????????	??????????????????????????????????	----------
4	??????????????????????????????????	??????????????????????????????????	----------
5	??????????????????????????????????	??????????????????????????????????	----------
6	??????????????????????????????????	??????????????????????????????????	----------
7	??????????????????????????????????	??????????????????????????????????	----------
8	??????????????????????????????????	??????????????????????????????????	----------
9	??????????????????????????????????	??????????????????????????????????	----------
10	??????????????????????????????????	??????????????????????????????????	----------
11	??????????????????????????????????	??????????????????????????????????	----------
12	??????????????????????????????????	??????????????????????????????????	----------
13	??????????????????????????????????	??????????????????????????????????	----------
14	??????????????????????????????????	??????????????????????????????????	----------
15	??????????????????????????????????	??????????????????????????????????	----------
16	??????????????????????????????????	??????????????????????????????????	----------
17	??????????????????????????????????	??????????????????????????????????	----------
18	??????????????????????????????????	??????????????????????????????????	----------
19	??????????????????????????????????	??????????????????????????????????	----------
20	??????????????????????????????????	??????????????????????????????????	----------
21	----------	??????????????????????????????????	----------
22	????????????????????x????????x-	---------x-------------------x-x-	----------
23	????????????????????????????----	????????????????????????????----	----------
24	----------------------------x---	????????????????????????????----	----------
25	----------------------------x---	----------	----------
26	----------	--x---x----------x----------	----------
27	----------	----------------------------x---	----------
28	----------	----------------------------x---	----------
29	----------	----------------------------x---	----------
30	----------	----------	----------
31	----------	----------	----------
32	----------	----------	----------
33	----------------------------x---	?????????????????????????????x---	----------
34	--x---x-x-------x--x--------x-	--x---x----------x----------	----------

Table 26: Result of search with R_{17} on characteristics from Table 24

i	A_i	E_i	W_i
-4	----------	------n------n------D--n-------	
-3	----------n-----uu-u--------n--	----------n---uuu-----u---	
-2	--n---u---u-u--------u--u-------	-n-u--nnn-u---n--uu--uu----n--	
-1	-----u---u-n----------uu-u---n	----------u-----u---n---------	
0	--------uu-----u---n---u---n-nn-	----------	n----------
1	----------u--------u-n----------	----------	----------
2	-u----------u----------n------	------n----n----------u------	u-nn---n-u----n-n-----nn-----u--
3	-B-n---n----n-n-----u--n-n----un	----nn---u----------u----------	---n----------n-un-n-uuuu---u
4	n---u--n-nD--u-un---D---B-D----u	??????????????????????????????????x	?????????????xuuuuuuuu----------
5	-u-n--u--5D----n----u---uuuu----	??????????????????????????????-?x	----------
6	-u----------u----------	???????????????????????????????	----------
7	-u-----uu-u----------nu---------	----------	----------
8	--n-u-u-nn-u-n--u-n-B----------	uuu--n----------n--Bu--un-n--nu	----------
9	----------	--u--un---u---u---u-Bu-n--n-u-u-	n----------
10	----------n----------	----------	n--u-u--u----n-n-unnnun-nnn--nn-
11	----------	----------	--u-n-u--u----n---uuuuu-u--nnnnn
12	----------	-nuu-nnn-u-u-n--uu-u----------	-----u----------u------nnnn
13	----------	----------	----------unnnnnnn----------
14	-unnnnnnnnnnnnnnnnnn----------	----------	----------
15	----------	----------	----------n----------
16	----------	--n-u-u-n-nn--nnn--nn----------	----------
17	----------	----------	----uu----------B-nnnnnnnnnnnn-
18	----------	--unnnnnnnnnnnnnnnnn----------	----------
19	----------	----------	----------
20	----------	----------	----------
21	----------	----------	----------
22	----------	----------	----------n----------
23	----------	----------	----------
24	----------	----------	----------
25	----------	----------	----------
26	----------	----------	----------
27	----------	----------	----------
28	----------	----------	----------
29	----------	----------	----------
30	??????????????????????????????x----	??????????????????????????????x----	??????????????????????????????x----
31	????????????????????????????????????	??????????????????????????????-???	??????????????????????x----------
32	????????????????????????????????????	??????????????????????????????-???	????????????????????????????????????
33	??????????????????????????????-???	??????????????????????????????-???	????????????????????????????????????
34	????????????????????????????????????	????????????????????????????????????	????????????????????????????????????

i	A_i	E_i	W_i
-4	????????????????????????????????????	??????-??????-??????????5???????	
-3	??????????-?????--?-?????????-??	????????????A??-??-?---?????-???	
-2	??-???-????-?-????????-??-??????	?-?-??---?-?5?-??--???--?????-??	
-1	?????-???A?-???????????--?-???-	?-?-??---???A?--?--???-??A?---??	
0	????????-A?????-???-???-???-?5-?	?-?-??---?-?-?-??-????--?-???-??	----------
1	??????????-????????-?-??????????	??????-?????-?????-???-??-??????	----------
2	?-???????????-???????????-??????	??????-????-????????????-??????	----------
3	???-???-????-?-?????-??-?-????--	????---??A??????????-????-??????	----------
4	5???-??-?-???-?--?????????????-	????????????????????????????????	----------
5	?-?-??-??5???5?-????A?A?----????	????????????????????????????????	----------
6	?-???????A???A?????A?-A???????5?	??-??-???????????????????????-?	----------
7	?-?????--?-??A???A?5?--?????5???	??-??-???????????????????????-?	----------
8	??-?-?-?--?-?-??-?-?????????????	A--A?-???????5A??-???AA?--?-??--	----------
9	????????????A??????????????????	??-?A--???-??A-???-5?A?5??-?-?-?	----------
10	?????????????A??????-???????????	---??--???-???-??--????------?--	----------
11	????????????????????5????????????	?--??--???-???-??--?5-?-??-?-?-?	----------
12	????????????????????5????????????	?---?---?-?-?-???--?-????????????	----------
13	????????????????????5????????????	?---?---?-?-?-???--?-????????????	----------
14	?----------???????????	?---?-------?-???--?-????????????	----------
15	????????????????????A????????????	??-???-?-?--????????-????????????	----------
16	?????????????????????A????????????	??-?-?-?-5--??---??--????????????	----------
17	????????????????????????????????	??-?-?-?-5--??---??--????????????	----------
18	????????????????????????????????	??----------??5????5?5?	----------
19	????????????????????????????????x	??----------??????????x	----------
20	???????????????????????????????--	??----------???????5?5-	----------
21	----------	----------	----------
22	--u----n-u--------u--n---unn-nn-	---------n----------n-n-	----------
23	----------unnn---	--------u----Du---n----------	----------
24	----------n---	----------	----------
25	----------u---	----------	----------
26	----------	--u---n----------n----------	----------
27	----------	----------u---	----------
28	----------	----------n---	----------
29	----------	----------u---	----------
30	----------	----------	----------
31	----------	----------	----------
32	----------	----------	----------
33	----------u---	----------u---	----------
34	--x---x-x-------x--x-------x-	--x---x----------x----------	----------

Table 27: Characteristics from [FGL09] manually repaired

i	A_i	E_i	W_i
-4	----------	??????????????????????????????	
-3	x---------	x---------	
-2	--x---x---x-x-------x--x------	--x---------x---------	
-1	----------	----------x---x--x---------	
0	----------	----------	x---------
1	x---------	----------	----------
2	----------	------x---x----------x------	----------
3	----------	----------	----------
4	----------	----------	----------
5	----------	x---------	----------
6	----------	----------	----------
7	----------	----------	----------
8	----------	----------	----------
9	----------	----------	x---------
10	----------	----------	----------
11	----------	----------	----------
12	----------	----------	----------
13	----------	----------	----------
14	----------	----------	----------
15	----------	----------	----------
16	----------	----------	----------
17	----------	----------	----------
18	----------	----------	----------
19	----------	----------	----------
20	----------	----------	----------
21	----------	----------	----------
22	??????????????????????????????	??????????????????????????????	----------
23	??????????????????????????????	??????????????????????????????	----------
24	??????????????????????????????	??????????????????????????????	??????????????????????????????
25	??????????????????????????????	??????????????????????????????	??????????????????????????????
26	??????????????????????????????	??????????????????????????????	??????????????????????????????
27	??????????????????????????????	??????????????????????????????	??????????????????????????????
28	??????????????????????????????	??????????????????????????????	??????????????????????????????
29	??????????????????????????????	??????????????????????????????	??????????????????????????????
30	??????????????????????????????	??????????????????????????????	??????????????????????????????
31	??????????????????????????????	??????????????????????????????	??????????????????????????????
32	??????????????????????????????	??????????????????????????????	??????????????????????????????
33	??????????????????????????????	??????????????????????????????	??????????????????????????????
34	??????????????????????????????	??????????????????????????????	??????????????????????????????

i	A_i	E_i	W_i
-4	??????????????????????????????	??????????????????????????????	
-3	??????????????????????????????	??????????????????????????????	
-2	??????????????????????????????	??????????????????????????????	
-1	??????????????????????????????	??????????????????????????????	
0	??????????????????????????????	??????????????????????????????	----------
1	??????????????????????????????	??????????????????????????????	----------
2	??????????????????????????????	??????????????????????????????	----------
3	??????????????????????????????	??????????????????????????????	----------
4	??????????????????????????????	??????????????????????????????	----------
5	??????????????????????????????	??????????????????????????????	----------
6	??????????????????????????????	??????????????????????????????	----------
7	??????????????????????????????	??????????????????????????????	----------
8	??????????????????????????????	??????????????????????????????	----------
9	??????????????????????????????	??????????????????????????????	----------
10	??????????????????????????????	??????????????????????????????	----------
11	??????????????????????????????	??????????????????????????????	----------
12	??????????????????????????????	??????????????????????????????	----------
13	??????????????????????????????	??????????????????????????????	----------
14	??????????????????????????????	??????????????????????????????	----------
15	??????????????????????????????	??????????????????????????????	----------
16	??????????????????????????????	??????????????????????????????	----------
17	??????????????????????????????	??????????????????????????????	----------
18	??????????????????????????????	??????????????????????????????	----------
19	??????????????????????????????	??????????????????????????????	----------
20	??????????????????????????????	??????????????????????????????	----------
21	??????????????????????????????	??????????????????????????????	----------
22	??????????????????????????????	??????????????????????????????	----------
23	x?????????????????????????????	??????????????????????????????	----------
24	x----------	??????????????????????????????	----------
25	x----------	??????????????????????????????	----------
26	----------	??????????????????????????????	----------
27	----------	x?????????????????????????????	----------
28	----------	x?????????????????????????????	----------
29	----------	x---------	----------
30	----------	----------	----------
31	----------	----------	----------
32	----------	----------	----------
33	x----------	x---------	----------
34	??????????????????????????????	??????????????????????????????	----------

Table 28: Result of search with R_{19} on characteristics from Table 27

i	A_i	E_i	W_i
-4	----------	---nnn------u-n-------u---u--	
-3	x---------	x---------	
-2	--u---x---x-x-------x--x------	--n---------u----------	
-1	----------	----------x----x---x---------	
0	----------	----------	x---------
1	x---------	----------	----------
2	----------	-----x---x----------x------	----------
3	----------	----------	----------
4	----------	----------	----------
5	----------	x---------	----------
6	----------	----------	----------
7	----------	----------	----------
8	----------	----------	----------
9	----------	----------	x---------
10	----------	----------	----------
11	----------	----------	----------
12	----------	----------	----------
13	----------	----------	----------
14	----------	----------	----------
15	----------	----------	----------
16	----------	----------	----------
17	----------	----------	----------
18	----------	----------	----------
19	----------	----------	----------
20	----------	----------	----------
21	----------	----------	----------
22	----------	----------	----------
23	----------	----------	----------
24	---u---u----------u----------	--u-nnnn--------un----------	---u---u---------u----------
25	------n--u-un--B-nn-----Bu------	--------u---u----------u----u-u	n---------
26	???????xnnnD????xnnnnnnnnn------	????????????????????????????????x-	????????????????????????????x---
27	----------	A---------	nuuuuuuuuuu---unnnnu----------
28	----------	A---------	----------u----------u
29	----------	----------	n-----nnn-Dn---u---n----u-nnnn
30	----------	----------u----------u------	??????????????????????????????????x-
31	----------	----------	??????????????????????????????????x
32	----------	----------	????????????????????????????------
33	----------	----------	----------D----------
34	----------	----------	----------n----------n------

i	A_i	E_i	W_i
-4	----------	----------	
-3	----------	----------	
-2	----------	----------n-	
-1	----------	----------	
0	----------	----------	----------
1	----------	----------	----------
2	----------nu-	----------n-	----------
3	??????????????????????????????????x	????x--??x--????????xxxx--B????-	----------
4	??????????????????????????????????-	??????-???x-????????????x-?????x	----------
5	----------B-u-----u--B-----	---n-n-----u-B------u----u--	----------
6	---u----n--nnnn-----n-un---D	--u----u----B-u-uu----------	----------
7	un--nu------u-----n---n---B--	---------u----u-----uB---------	----------
8	--------D--u---------n---D--nu	---------u----------nn-n-u	----------
9	-u---------uu--u--nn--n--n--n--	----u------u------u---n--n---	----------
10	----u--n----------	-----n-n----n--u----------	----------
11	-n------u--nnn--u--n-------u---	-----n-u-----n--n-u-------u----	----------
12	---nn--u--uu---u-u----unn----u-u	-n--u----------u--u-n--n--u--n-	----------
13	--------u---------n---n------	----------u---n-n-----	----------
14	---n-n---nn---u--n----n-n-n---n-	----------u-----u----n-----	----------
15	--u-n------u---n----------n-uu-	-------u-n-uu----n---n---u---	----------
16	----------u--u-nnn--nn----n--u	----------u	----------
17	-u---u-------n------n---------	-----n----------	----------
18	u--n-uuuu---u--u--------un-n----	--n-u----------u----u-n-n-n---	----------
19	n------nnnnn-nnn----------u----	n-n---u---nu---nnnnnnn----n-----	----------
20	----------	----------	----------
21	--n---------un-------u---------	--u---------n-------n---------	----------
22	----------u------	----------	----------
23	u---------	----------	----------
24	u---------	----------	----------
25	n---------	----------	----------
26	----------	??????????????????????????x------	----------
27	----------	n---------	----------
28	----------	n---------	----------
29	----------	u---------	----------
30	----------	----------	----------
31	----------	----------	----------
32	----------	----------	----------
33	n---------	u---------	----------
34	--u-nuu----nn-------n--u------	------n---n----------u------	----------

B Used Searching Setups

$B1_{04}$:	-P1:	$\{A_{0,23-26},$	$E_{0,23-26},$	$W_{0,23-26}$	$\}$:	{'?', 'x'}
$B1_{07}$:	-P1:	$\{A_{23-29},$	$E_{23-29},$	W_{23-29}	$\}$:	{'?', 'x'}
$B2_{04}$:	-P1:	$\{A_{23-26},$	$E_{23-26},$	W_{23-26}	$\}$:	{'?', 'x'}
$B2_{07}$:	-P1:	$\{A_{23-29},$	$E_{23-29},$	W_{23-29}	$\}$:	{'?', 'x'}
$S1_{04}$:	-P1:	$\{A_{0,23},$	$E_{0,23},$	$W_{0,23}$	$\}$:	{'?', 'x'}
	-P2:	$\{A_{0,24},$	$E_{0,24},$	$W_{0,24}$	$\}$:	{'?', 'x'}
	-P3:	$\{A_{0,25},$	$E_{0,25},$	$W_{0,25}$	$\}$:	{'?', 'x'}
	-P4:	$\{A_{0,26},$	$E_{0,26},$	$W_{0,26}$	$\}$:	{'?', 'x'}
$M1_{07}$:	-P1:	$\{A_{0,23},$	$E_{0,23},$	$W_{0,23}$	$\}$:	{'?', 'x'}
	-P2:	$\{A_{0,24},$	$E_{0,24},$	$W_{0,24}$	$\}$:	{'?', 'x'}
	-P3:	$\{A_{0,25},$	$E_{0,25},$	$W_{0,25}$	$\}$:	{'?', 'x'}
	-P4:	$\{A_{0,26},$	$E_{0,26},$	$W_{0,26}$	$\}$:	{'?', 'x'}
	-P5:	$\{A_{0,27},$	$E_{0,27},$	$W_{0,27}$	$\}$:	{'?', 'x'}
	-P6:	$\{A_{0,28},$	$E_{0,28},$	$W_{0,28}$	$\}$:	{'?', 'x'}
	-P7:	$\{A_{0,29},$	$E_{0,29},$	$W_{0,29}$	$\}$:	{'?', 'x'}
$M1_{08}$:	-P1:	$\{A_{0,23},$	$E_{0,23},$	$W_{0,23}$	$\}$:	{'?', 'x'}
	-P2:	$\{A_{0,24},$	$E_{0,24},$	$W_{0,24}$	$\}$:	{'?', 'x'}
	-P3:	$\{A_{0,25},$	$E_{0,25},$	$W_{0,25}$	$\}$:	{'?', 'x'}
	-P4:	$\{A_{0,26},$	$E_{0,26},$	$W_{0,26}$	$\}$:	{'?', 'x'}
	-P5:	$\{A_{0,27},$	$E_{0,27},$	$W_{0,27}$	$\}$:	{'?', 'x'}
	-P6:	$\{A_{0,28},$	$E_{0,28},$	$W_{0,28}$	$\}$:	{'?', 'x'}
	-P7:	$\{A_{0,29},$	$E_{0,29},$	$W_{0,29}$	$\}$:	{'?', 'x'}
	-P8:	$\{A_{0,30},$	$E_{0,30},$	$W_{0,30}$	$\}$:	{'?', 'x'}
$M2_{09}$:	-P1:	$\{A_{0,23},$	$E_{0,23},$	$W_{0,23}$	$\}$:	{'?', 'x'}
	-P2:	$\{A_{0,24},$	$E_{0,24},$	$W_{0,24}$	$\}$:	{'?', 'x'}
	-P3:	$\{A_{0,25},$	$E_{0,25},$	$W_{0,25}$	$\}$:	{'?', 'x'}
	-P4:	$\{A_{0,26},$	$E_{0,26},$	$W_{0,26}$	$\}$:	{'?', 'x'}
	-P5:	$\{A_{0,27},$	$E_{0,27},$	$W_{0,27}$	$\}$:	{'?', 'x'}
	-P6:	$\{A_{0,28},$	$E_{0,28},$	$W_{0,28}$	$\}$:	{'?', 'x'}
	-P7:	$\{A_{0,29},$	$E_{0,29},$	$W_{0,29}$	$\}$:	{'?', 'x'}
	-P8:	$\{A_{0,30},$	$E_{0,30},$	$W_{0,30}$	$\}$:	{'?', 'x'}
	-P9:	$\{A_{1,24-28},$	$E_{1,24-28},$	$W_{1,24-28}$	$\}$:	{'?', 'x'}

$M2_{10}$: -P1: $\{A_{0,23}, E_{0,23}, W_{0,23}\}$: {'?', 'x'}
-P2: $\{A_{0,24}, E_{0,24}, W_{0,24}\}$: {'?', 'x'}
-P3: $\{A_{0,25}, E_{0,25}, W_{0,25}\}$: {'?', 'x'}
-P4: $\{A_{0,26}, E_{0,26}, W_{0,26}\}$: {'?', 'x'}
-P5: $\{A_{0,27}, E_{0,27}, W_{0,27}\}$: {'?', 'x'}
-P6: $\{A_{0,28}, E_{0,28}, W_{0,28}\}$: {'?', 'x'}
-P7: $\{A_{0,29}, E_{0,29}, W_{0,29}\}$: {'?', 'x'}
-P8: $\{A_{0,30}, E_{0,30}, W_{0,30}\}$: {'?', 'x'}
-P9: $\{A_{1,24-28}, E_{1,24-28}, W_{1,24-28}\}$: {'?', 'x'}
-P10: $\{A_{0,31}, E_{0,31}, W_{0,31}\}$: {'?', 'x'}

$M2_{11}$: -P1: $\{A_{0,23}, E_{0,23}, W_{0,23}\}$: {'?', 'x'}
-P2: $\{A_{0,24}, E_{0,24}, W_{0,24}\}$: {'?', 'x'}
-P3: $\{A_{0,25}, E_{0,25}, W_{0,25}\}$: {'?', 'x'}
-P4: $\{A_{0,26}, E_{0,26}, W_{0,26}\}$: {'?', 'x'}
-P5: $\{A_{0,27}, E_{0,27}, W_{0,27}\}$: {'?', 'x'}
-P6: $\{A_{0,28}, E_{0,28}, W_{0,28}\}$: {'?', 'x'}
-P7: $\{A_{0,29}, E_{0,29}, W_{0,29}\}$: {'?', 'x'}
-P8: $\{A_{0,30}, E_{0,30}, W_{0,30}\}$: {'?', 'x'}
-P9: $\{A_{1,24-28}, E_{1,24-28}, W_{1,24-28}\}$: {'?', 'x'}
-P10: $\{A_{0,31}, E_{0,31}, W_{0,31}\}$: {'?', 'x'}
-P11: $\{A_{21-23}, E_{21-23}, W_{21-23}\}$: {'?', 'x'}

$S2_{04}$: -P1: $\{A_{23}, E_{23}, W_{23}\}$: {'?', 'x'}
-P2: $\{A_{24}, E_{24}, W_{24}\}$: {'?', 'x'}
-P3: $\{A_{25}, E_{25}, W_{25}\}$: {'?', 'x'}
-P4: $\{A_{26}, E_{26}, W_{26}\}$: {'?', 'x'}

$S2_{07}$: -P1: $\{A_{23}, E_{23}, W_{23}\}$: {'?', 'x'}
-P2: $\{A_{24}, E_{24}, W_{24}\}$: {'?', 'x'}
-P3: $\{A_{25}, E_{25}, W_{25}\}$: {'?', 'x'}
-P4: $\{A_{26}, E_{26}, W_{26}\}$: {'?', 'x'}
-P5: $\{A_{27}, E_{27}, W_{27}\}$: {'?', 'x'}
-P6: $\{A_{28}, E_{28}, W_{28}\}$: {'?', 'x'}
-P7: $\{A_{29}, E_{29}, W_{29}\}$: {'?', 'x'}

R_{17} :

-P1:	$\{A_{0,(-4)-5},$	$E_{0,(-4)-3},$	$W_{0,(-4)-3}$	$\}$:	{'?', 'x'}
-P2:	$\{A_{0,24},$	$E_{0,24},$	$W_{0,24}$	$\}$:	{'?', 'x'}
-P3:	$\{A_{0,25},$	$E_{0,25},$	$W_{0,25}$	$\}$:	{'?', 'x'}
-P4:	$\{A_{0,26},$	$E_{0,26},$	$W_{0,26}$	$\}$:	{'?', 'x'}
-P5:	$\{A_{0,27},$	$E_{0,27},$	$W_{0,27}$	$\}$:	{'?', 'x'}
-P6:	$\{A_{0,28},$	$E_{0,28},$	$W_{0,28}$	$\}$:	{'?', 'x'}
-P7:	$\{A_{0,29},$	$E_{0,29},$	$W_{0,29}$	$\}$:	{'?', 'x'}
-P8:	$\{A_{0,23-25},$	$E_{0,23-25},$	$W_{0,23-25}$	$\}$:	{'-'}
-P9:	$\{A_{1,21-24},$	$E_{1,21-22},$	$W_{1,21-22}$	$\}$:	{'?', 'x'}
-P10:	$\{A_{1,25-26},$	$E_{1,23-24},$	$W_{1,23-24}$	$\}$:	{'?', 'x'}
-P11:	$\{A_{1,27-28},$	$E_{1,25-26},$	$W_{1,25-26}$	$\}$:	{'?', 'x'}
-P12:	$\{A_{0,20-23},$	$E_{0,22-23},$	$W_{0,22-23}$	$\}$:	{'?', 'x'}
-P13:	$\{A_{0,18-19},$	$E_{0,20-21},$	$W_{0,20-21}$	$\}$:	{'?', 'x'}
-P14:	$\{A_{0,16-17},$	$E_{0,18-19},$	$W_{0,18-19}$	$\}$:	{'?', 'x'}
-P15:	$\{A_{0,14-15},$	$E_{0,16-17},$	$W_{0,16-17}$	$\}$:	{'?', 'x'}
-P16:	$\{$	$E_{0,14-15},$	$W_{0,14-15}$	$\}$:	{'?', 'x'}
-P17:	$\{$	$E_{1,33},$		$\}$:	{'?', 'x'}

R_{19} :

-P1:	$\{A_{0,24},$	$E_{0,24-25},$	$W_{0,24}$	$\}$:	{'?', 'x'}
-P2:	$\{$	$E_{1,23-24},$		$\}$:	{'?', 'x'}
-P3:	$\{$	$E_{1,22},$		$\}$:	{'?', 'x'}
-P4:	$\{A_{1,22},$			$\}$:	{'?', 'x'}
-P5:	$\{A_{0,23-25},$	$E_{1,25},$	$W_{0,25}$	$\}$:	{'?', 'x'}
-P6:	$\{A_{1,21-23},$	$E_{1,21},$		$\}$:	{'?', 'x'}
-P7:	$\{A_{1,20},$	$E_{1,20},$	$W_{1,20}$	$\}$:	{'?', 'x'}
-P8:	$\{A_{1,19},$	$E_{1,19},$	$W_{1,19}$	$\}$:	{'?', 'x'}
-P9:	$\{A_{1,18},$	$E_{1,18},$	$W_{1,18}$	$\}$:	{'?', 'x'}
-P10:	$\{$	$E_{0,(-4)},$		$\}$:	{'?', 'x'}
-P11:	$\{A_{27-29},$	$E_{27-29},$		$\}$:	{'?', 'x'}
-P12:	$\{A_{30-34},$	$E_{30-34},$		$\}$:	{'?', 'x'}
-P13:	$\{A_{1,16-18},$	$E_{1,16-18},$		$\}$:	{'?', 'x'}
-P14:	$\{A_{1,13-15},$	$E_{1,13-15},$		$\}$:	{'?', 'x'}
-P15:	$\{A_{1,10-12},$	$E_{1,10-12},$		$\}$:	{'?', 'x'}
-P16:	$\{A_{1,7-9},$	$E_{1,7-9},$		$\}$:	{'?', 'x'}
-P17:	$\{A_{1,5-6},$	$E_{1,5-6},$		$\}$:	{'?', 'x'}
-P18:	$\{$		$W_{0,27-29}$	$\}$:	{'?', 'x'}
-P19:	$\{A_{1,(-4)-2},$	$E_{1,(-4)-2},$	$W_{0,33-44}$	$\}$:	{'?', 'x'}