Implementation and Analysis of a Key Recovery Attack on the Block Cipher SPECK32

Carina Fiedler

4th Year Project Report
Computer Science and Mathematics
School of Informatics
University of Edinburgh

2022

Abstract

This project focuses on the differential key recovery attack by Biryukov et al. [5]. We provide the first implementation of the attack and state several corrections to the analysis of its theoretical runtime complexity. Investigating the number of false positive key candidates in practice we observe some interesting patterns. Consequently, we provide some explanations for our observations via a precise algebraic representation and describe an algorithm adapted from Mouha et al. [14] to estimate the number of false positive key candidates for a given ciphertext pair.

Acknowledgements

I would like to thank my supervisor Vesselin Velichkov for proposing the project idea, providing encouragement, investing the time for in-depth discussions and allowing me to go beyond mere implementation and dig deeper into underlying concepts.

Special thanks to Laura Ritter for proofreading, motivational writing sessions in the kitchen and putting up with fluctuating degrees of craziness during the writing process.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

This project focuses on the differential key recovery attack by Biryukov et al. [5]. The description of the attack in the original paper is rather brief, omitting a lot of intricate details that would be required for the understanding of the mechanics of the attack by someone other than an experienced symmetric cryptographer. This dissertation provides the first (publicly available) implementation of this attack. After outlining some necessary background information in Chapter 1, Chapter 2 contains a concrete guide of how the attack can be implemented along with a more detailed and accessible explanation of how the attack works.

Contrary to the improved attack by Dinur [7], the attack has not been implemented in practice before, but contains predictions of the complexity of its different stages. The purpose of this project is to implement crucial parts of the attack for the smallest SPECK32 variant, experimentally test whether the theoretical complexities hold and investigate any discrepancies. A deeper understanding of the practical issues of the attack will be beneficial for future improvements to the key recovery step in differential attacks in general. Chapter 3 entails an extensive analysis of the observed runtime complexity. Further, we also state several corrections to the theoretical runtime analysis in the original paper, showing that the attack complexity is actually dominated by 2^{60} 11-round SPECK32 encryptions rather than the claimed 2^{55}.

We observed the number of false positive key candidates to behave differently than predicted and chose to investigate this phenomenon further. We provide some exemplary computational analysis, a rigorous algebraic characterization of the constraints on key candidates and the application of a graph theoretical approach to calculate an estimate of the number of false positive key candidates for a given ciphertext pair in chapter 4. Since the number of false positives has a direct impact on the runtime complexity of the attack, limiting the number of false positives is of similar importance as improving the probability of the differential trail. Our analysis provides some insight under which circumstances false positives occur. This can be the basis for further research to estimate the number of false positives more precisely and to identify optimal conditions to minimize the occurrence of false positives.

Symbol	Meaning
$+$	addition modular word size
$\oplus$	exclusive or (XOR)
$\wedge$	logical AND
$\lll$	left rotation with wrap-around
$\ggg$	right rotation with wrap-around
$\ll$	left rotation without wrap-around
#	number of
eq	$\text{eq}(\alpha,\beta,\gamma) = 1 \Leftrightarrow \alpha = \beta = \gamma$
w_h	hamming weight
K^i	i^{th} round key
Δ_L^i	XOR difference of left inputs to round i
Δ_R^i	XOR difference of right inputs to round i
x_i	i^{th} bit of word x
maj	takes the same value that the majority of its input has

Table 1.1: Notation overview.

1.2 Differential Cryptanalysis

1.2.1 Tracing Differences

In differential cryptanalysis we consider the exclusive or (XOR) difference of two texts at different stages during the encryption process.

Definition 1. *A differential is a pair of XOR differences (Δ plaintext, Δ ciphertext). A sequence of XOR differences of intermediate round inputs/outputs from the plaintext difference to the ciphertext difference is called a differential trail.*

For a truly random permutation, the probability that a particular differential occurs, i.e. that a particular input difference ΔP propagates to a particular output difference ΔC is $1/2^n$ where n is the word size [11]. But real ciphers are pseudo-random permutations and therefore it is possible to find a differential (ΔP, ΔC) such that the input difference ΔP propagates to the output difference ΔC with much higher probability than $1/2^n$.

ARX ciphers are made up of the simple building blocks: bitwise XOR, $\oplus$; bitwise addition modulo the word size, $+$; and bitwise left and right rotations, $\lll$ and $\ggg$, respectively.

Note that we can trace an XOR difference through an XOR-Operation and bitwise (wrap-around) rotations with probability 1 due to their linearity. Specifically, XOR-ing both texts with an unknown key does not have any effect on their difference, since XOR is a self-inversing operation. In rotations, incoming difference bits are merely mapped to different output bits in a traceable manner. The only non-linear operation in ARX ciphers is modular addition.

Example 1. *We are using hexadecimal notation for convenience. Consider a left input pair* $(x = 0,\ x^* = 3)$ *with input difference* $\Delta x = 3$ *and a right input pair* $(y = 7,\ y^* = 6)$ *with input difference* $\Delta y = 1$. *Calculating the pairwise modular addition* $z = x + y = 7$

and $z^* = x^* + y^* = 9$, we see that the output difference $z \oplus z^* = 0xE$. Now consider a different right input pair with the same input difference as before ($\hat{y} = 3$, $\hat{y}^* = 2$). Then $\hat{z} \oplus \hat{z}^* = 0x6 \neq z \oplus z^*$. So, the input differences of a modular addition do not determine its output difference.

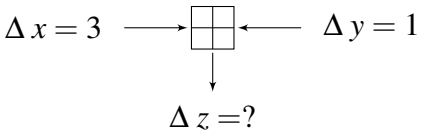


Figure 1.1: Example 1 - How do XOR differences propagate through modular addition? We consider two modular additions such that their inputs have XOR differences 3 and 1 and want to know the XOR difference of the two outputs of the additions.

Hence, the differential probability of modular addition with respect to the XOR operation is central to the understanding of differential properties of SPECK32.

1.2.2 Differential Probability of Modular Addition

Definition 2. *The differential probability with which the input XOR differences α, β propagate to the output difference γ, is defined as*

$$\begin{aligned} xdp^+(\alpha, \beta \to \gamma) &:= P_{x,y}[(x+y) \oplus ((x \oplus \alpha) + (y \oplus \beta)) = \gamma] \\ &:= \frac{\#\{x, y \in \{0,1\}^n : (x+y) \oplus ((x \oplus \alpha) + (y \oplus \beta)) = \gamma\}}{2^{2n}}. \end{aligned} \tag{1.1}$$

where # denotes the size of the set, and n denotes the word size.

An efficient logarithmic time algorithm to calculate the differential probability of modular addition has been presented by Lipmaa et.al. [12]:

Theorem 1 (Lipmaa)**.** *Denote left bitwise rotation without wraparound with $\ll$. Let*

$$E := eq(\alpha \ll 1, \beta \ll 1, \gamma \ll 1) := \big[1 \Leftrightarrow (\alpha \ll 1) = (\beta \ll 1) = (\gamma \ll 1)\big], \tag{1.2}$$

$$F := \alpha \oplus \beta \oplus \gamma \oplus (\beta \ll 1). \tag{1.3}$$

We say that $\delta = (\alpha, \beta \to \gamma)$ is possible if and only if

$$E \wedge F = 0. \tag{1.4}$$

Then

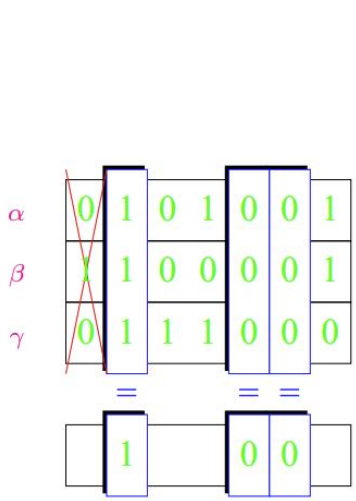
$$xdp^+(\delta) = \begin{cases} 0 & \text{if } \delta \text{ is not possible;} \\ 2^{-w_h(\neg eq(\alpha,\beta,\gamma) \wedge (2^{n-1}-1))} & \text{otherwise.} \end{cases} \tag{1.5}$$

where w_h is the hamming weight that is obtained as described below.

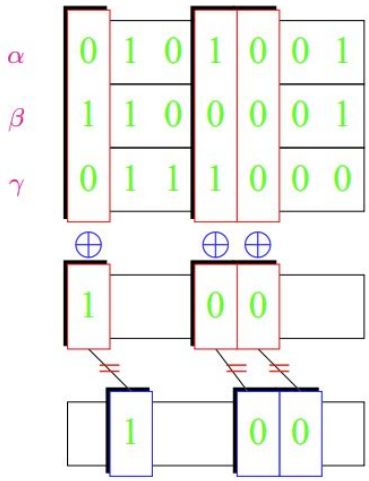
Condition 1.4 can be checked bitwise. We count indices starting from the least significant bit 0. Writing the differences α, β, γ bitwise underneath each other, we locate all columns that contain either only 1s or only 0s i.e. each column i such that $\text{eq}(\alpha_i, \beta_i, \gamma_i) = 1$ (see Figure 1.2a).

Then for the preceding column $j = i+1$, $\text{eq}(\alpha_{j-1}, \beta_{j-1}, \gamma_{j-1}) = 1$. Hence, $E_j = 1$. So, for the differential to be possible, we require that $F_j = 0$. Hence, for each preceding column j we need to check that the XOR difference of its elements equals the XOR difference of column i (which coincides with the value of β_i). This check is shown in Figure 1.2b. The differential is possible if and only if this is the case. Note that we do not need to perform the check in Figure 1.2b for any columns that are proceeded by columns containing both 0s and 1s since equation 1.4 is already satisfied in that case as $E_j = 0$.

Once we have established, that the differential is possible we, can determine its probability by counting the number of columns w_h that contain both 0s and 1s, neglecting the MSB column (see Figure 1.3). The probability of the differential is then 2^{-w_h}.



(a) Locate columns that contain either only 1s or only 0s.



(b) Compare the XOR differences of the pure 0s or 1s columns with the XOR differences of their preceding columns.

Figure 1.2: Condition of possible differential. Taken from Lipmaa et al. [13].

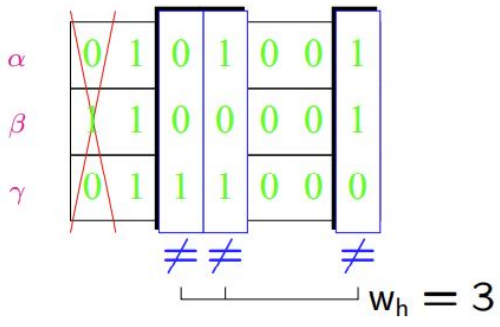


Figure 1.3: Calculate hamming weight w_h by counting the number of columns that contain both 0s and 1s. Taken from Lipmaa et al. [13].

1.2.3 Differential Trails and Key Recovery Attacks

The probability of a differential trail can then be calculated as the product of the differential probability with which the input differences propagate to the output differences of each round. Finding the differential with the highest probability is a hard problem, since the state space grows exponentially with the number of rounds. A common simplifying assumption made in cryptanalysis is to assume that the rounds are independent. This enables STP- and MILP-based automatic search algorithms [9], [16] that approximate the resulting probability.

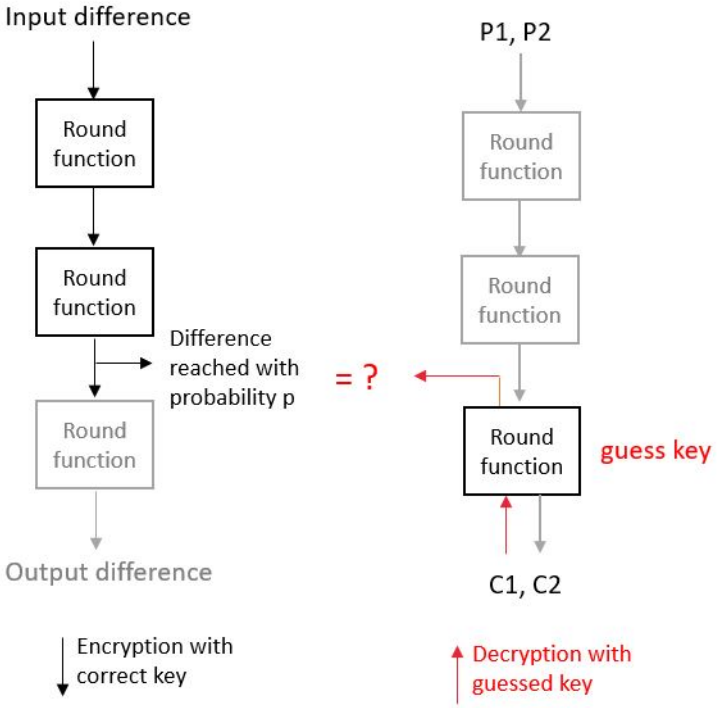


Figure 1.4: General scheme of key recovery attack. On the left: Differential trail with high probability. On the right: Key recovery attack by encrypting pairs, guessing last round key, partially decrypting and matching against the difference from the trail.

Once we have found a differential trail which occurs with considerably larger probability than uniform distribution, we can use this to extract key bits from the last round key. A detailed description of this key recovery attack applied to SPECK32 can be found in Chapter 2. The basic idea is that we encrypt a large number of pairs of plaintexts with the input difference of our differential trail. We then guess the last round key and partially decrypt the ciphertext. If we meet the differential in an earlier round of the cipher, this indicates that we have likely guessed the correct key. See Figure 1.4 for a high-level overview of this technique.

1.3 SPECK32

The emergence of embedded devices that require secure communication between each other pose new requirements on encryption schemes in terms of low computing power. The symmetric block cipher family SPECK has been proposed by the American National Security Agency in 2013 [3]. As a lightweight cipher it has been developed for high performance in constrained software environments. In fact, Beaulieu et al. [4] claimed that their performance efficiency measure, rank, indicated that SPECK had better overall performance than any existing block cipher at the time.

There exist different variants of block sizes to fit distinct needs of processors and devices, but we will focus on the smallest 32-bit block size variant with 16-bit words. To encrypt a 32-bit plaintext with SPECK32, we split it up into two halves and provide the 16 most-significant bits as left and 16 least-significant bits as right input to the first encryption round. The concatenation of the left and right output of the last round provides the ciphertext.

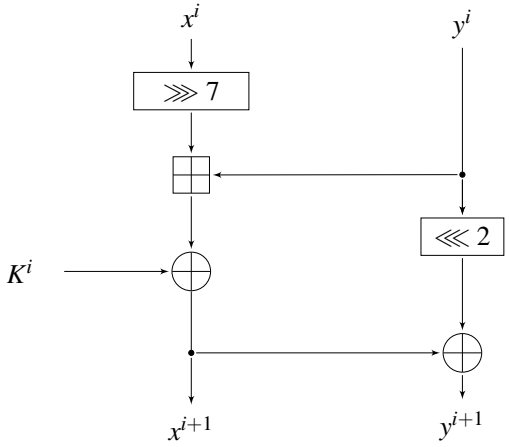


Figure 1.5: Round function of SPECK32

Internally, SPECK32's encryption algorithm consists of 22 rounds that are identically built and chained together. The left (right) output of the n^{th} round is used as the left (right) input of the $(n+1)^{th}$ round. Each encryption round is equipped with a 16-bit round key that has been derived from a master key. The i^{th} encryption round with left input x^i, right input y^i and round key K^i is defined in the following way [3] and visualized in Figure 1.5:

$$\text{Left output:} \qquad x^{i+1}(x^i, y^i, K^i) = \Big[(x^i \ggg 7) + y^i\Big] \oplus K^i \tag{1.6}$$

$$\text{Right output:} \qquad y^{i+1}(x^i, y^i, K^i) = \Bigg[\Big[(x^i \ggg 7) + y^i\Big] \oplus K^i\Bigg] \oplus (y^i \lll 2) \tag{1.7}$$

where $\oplus$ denotes exclusive or; $+$ represents bitwise addition modulo the word size 2^{16}; and $\lll$ and $\ggg$ denote left and right rotations with wraparound, respectively.

1.4 Overview of Attacks on SPECK32

While there has been extensive cryptanalysis on SPECK32 after its publication, no practically feasible attack on the full-round version is known so far and research interest has died down over the years. However, several attacks on round-reduced versions have been described in the literature. As the probability of a differential trail is negatively correlated with its length, reducing the number of rounds facilitates finding trails that occur with higher probability.

Since the publication paper of the SPECK family did not contain any cryptanalysis, the first differential key-recovery attacks were reported by Abed et al. in 2013 in [1]. While the paper is also introducing some other techniques, their differential attack can recover the secret key from 10 rounds of SPECK32 with a time complexity dominated by $2^{29.2}$ key guesses, which equates to the same number of 10-round SPECK32 encryptions.

Biryukov et al. [5] attack an additional round with a claimed theoretical time complexity dominated by 2^{55} 11-round SPECK32 encryptions compared to a brute-force time complexity of 2^{64}. An explanation of their attack technique is detailed in Chapter 2. Note that both attacks use distinct fixed differential trails and describe the attack specifically for that trail.

Dinur [7] later improves upon these by using a more unconventional key recovery technique involving a guess-and-determine algorithm and bitwise filters to discard wrong tuples. For the 32-block size variant Dinur's attack achieves a time complexity of 2^{46} for 11 rounds. Furthermore, it provides the best attack on 14 rounds of SPECK32 with a time complexity of 2^{63} using a 10-round differential. As Dinur writes in his conclusion, his attack framework is generic and can be used with an arbitrary improved differential trail. Hence, a lot of the subsequent research has been focused on developing techniques to improve and automate the search for better differential trails while using Dinur's 2-Round key-recovery attack [8],[16], resulting in a further improvement of the complexity of the 14-round attack to $2^{61.41}$ by Song et al. [16].

In 2019, Gohr [10] uses deep learning to recover the round keys based on machine-learned distinguishers, further reducing the time complexity to 2^{38} 11-round encryptions of SPECK32. However, this approach is not applied to a larger number of rounds. Hence, Dinur's key recovery method remains the most successful in terms of round coverage. However, Dinur's paper [7] already outlines some implementation results, and we therefore focus on the implementation and analysis of the attack by Biryukov et al. [5] which has not been implemented before.

Author	Complexity	Data (CP)
Biryukov et al.	2^{55}	2^{30}
Dinur	2^{46}	2^{14}
Gohr	2^{38}	$2^{14.5}$

Table 1.2: Overview of Key Recovery Attacks on 11 rounds of SPECK32. The data complexity is measured in the number of required chosen plaintexts (CP).

1.5 Contribution

This work provides the following contributions:

- First known implementation of the attack by Biryukov et al. [5].
- We measure the number of false positive key candidates to be drastically higher than predicted when guessing a more restricted amount of key bits. If this was the case for the full attack as well, this would directly influence its complexity.
- Extensive accessible description of the aforementioned attack including intricate details that have been omitted in the original paper.
- We empirically investigate the structure of false positive key candidates for a given ciphertext and observe that false positive key candidates have bit positions at which the value is fixed to the correct key bit. Further, there are constraints between different bits of the key.
- Detailed analysis of the key bit dependencies within false positive key candidates via a rigorous algebraic description.
- We describe how the graph theoretical approach by Mouha et al. [14] can be adapted to calculate the number of false positive key candidates for a chosesn ciphertext pair.
- Several corrections to the theoretical complexity analysis performed in the paper leading to an overall time complexity of 2^{60} instead of the claimed 2^{55}:
  - The optimal guess range in the second last round key is $K^9[15:7]$ instead of $K^9[15:5]$ resulting in 2^{26} instead of 2^{28} counter and carry guesses and 39 instead of 37 remaining bits in the brute-force stage.
  - The carry guess should be considered to be a counter in the sense that it increases the number of decryptions taking place in the key guessing phase, however it does not count towards the number of key candidates. Hence, the number of increments must be distributed over 2^{25} key candidates instead of 2^{28}.
  - To calculate the expected number of counters that are incremented 4 times, the probability of this happening for a single counter needs to be multiplied by the number of counters (2^{25}) instead of the number of increments.

Chapter 2

Key Recovery Attack on SPECK32

2.1 Main Idea

This chapter describes the implementation of the chosen-plaintext attack on a 11-round reduced version of SPECK32 from [5]. The differential key recovery attack uses the 8-round differential trail with input differences $\Delta_L^1 = 8054$, $\Delta_R^1 = A900$ and output differences $\Delta_L^9 = 8040$, $\Delta_R^9 = 8140$ occurring with probability 2^{-28} that has been presented in the paper.

round	Δ_L	Δ_R	$log_2 p$
0	-	-	-
1	8054	A900	-0
2	0	A402	-3
3	A402	3408	-3
4	50C0	80E0	-8
5	181	203	-4
6	C	800	-5
7	2000	0	-3
8	40	40	-1
9	8040	8140	-1
10	-	-	-
total			-28

Figure 2.1: Differential trail of input differences for SPECK32 from [5] with one round attached at the top and two rounds attached at the bottom. The last column details the logarithmic probability with which each difference occurs. The combined probability of the trail is the sum of all rows.

Notes on Notation: Round and round key indices start at LSB = 0. Δ_L^i, Δ_R^i denote the left and right input difference of round i, respectively - which are simultaneously the output differences of round i-1.

We attach one round at the top of the differential trail and two rounds at the bottom of the differential trail. This allows us to attack 11 rounds of encryption with an 8-round

differential, benefiting from the higher probability of the shorter differential. Note that contrary to the general schematic of the key recovery attack in section 1.2.3 this means that we need to guess two round keys instead of one and match against the difference of the differential trail after two rounds of decryption.

We encrypt a large number of plaintexts with a certain input difference with 11 rounds of SPECK32. We guess the last two round keys, partially decrypt the ciphertexts and observe whether the XOR difference at this interior stage of the cipher matches the prediction from the differential trail. Matches indicate correct key guesses. We then brute-force the rest of the key.

2.2 Plain- and Ciphertext Tuples Generation

We want to generate 2^{30} tuples such that the outputs of the first round satisfy the input difference $\Delta_L^1 = 8054\ \Delta_R = A900$ of the 8-round differential trail. Since we are unaware of the first round key, we cannot simply generate the actual pairs at the output of the first round and then derive its inputs (= the plaintext). However, we can trace the first round output difference back up to the modular addition by reversing the XOR-operation and rotation on the right-hand side as shown in Figure 2.2.

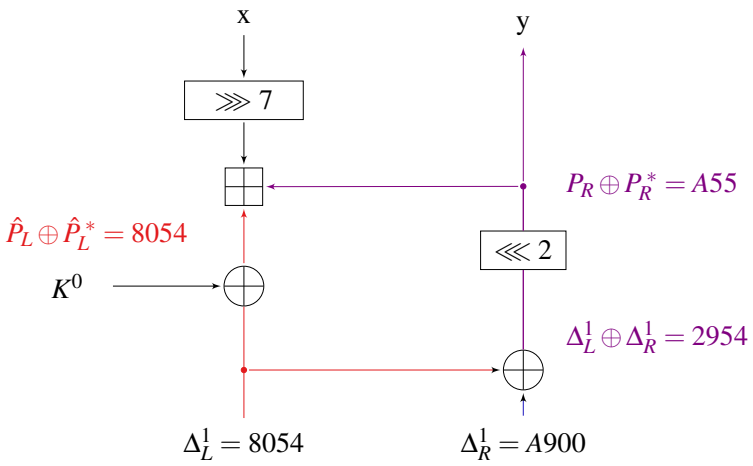


Figure 2.2: Tracing output difference of first round back up for plaintext generation.

- Then we can generate random values $\hat{P}_L, \hat{P}_L^*$ with XOR difference 8054 on the left-hand side - after the modular addition and before the XOR with the key.
- On the right-hand side, we generate random values P_R, P_R^* with XOR difference $A55$ - before the bitwise rotation.

Note that the latter correspond directly to the right input values, that is the right plaintext word. To compute the left plaintext half we first need to subtract the right input values from the generated values for the left-hand side modulo the word size 16, followed by reverting the rotation:

$$P_L = (\hat{P}_L - P_R) \lll 7, \quad P_L^* = (\hat{P}_L^* - P_R^*) \lll 7.$$

The concatenations $P = P_L||P_R$ and $P^* = P_L^*||P_R^*$ are then plaintexts with the desired output difference after 1 round of encryption. The pairs of generated plaintexts are then encrypted with 11 rounds of SPECK32 and stored together with their ciphertexts.

Algorithm 1 shows how the plain- and ciphertext tuple generation is implemented.

Algorithm 1 Generate Tuples

Output: List of filtered plain- and ciphertext tuples.

procedure GENERATE
  Initialize $tuples \leftarrow []$
  for i in 2^{30} **do**
    $P_R \leftarrow$ random value ▷ Generate random values with fixed difference.
    $P_R^* \leftarrow P_R \oplus 0xA55$
    $\hat{P}_L \leftarrow$ random value
    $\hat{P}_L^* \leftarrow \hat{P}_L \oplus 0x8054$
    $P_L \leftarrow (\hat{P}_L - P_R) \lll 7$ ▷ Modular subtraction of right-hand side.
    $P_L^* \leftarrow (\hat{P}_L^* - P_R^*) \lll 7$
    $P \leftarrow P_L||P_R$
    $P^* \leftarrow P_L^*||P_R^*$
    $C \leftarrow$ encrypt(P)
    $C^* \leftarrow$ encrypt(P^*)
    if filter(C, C^*) **then** ▷ Filter procedure is implemented in Algorithm 2.
      to tuples append (P, P^*, C, C^*)
    end if
  end for
  return tuples
end procedure

2.3 Filtering

Before we start the computationally expensive key guessing phase, we want to reduce our plain- and ciphertext pairs to the ones most likely to lead to a differential trail match after 2 rounds of decryption of the ciphertexts.

Starting from the desired Round-9 output difference Δ^9, we can make conclusions about some of the bits of the right Round-10 output difference Δ_R^{10}. As we have already seen in Section 2.2, we can revert to the right Round-10 output difference Δ_R^{10} without guessing any keys. Hence, if we can check some conditions on Δ_R^{10}, we can discard a substantial portion of the ciphertext tuples that cannot match the desired difference Δ^9.

Figure 2.3 shows how Δ^9 constricts Δ_R^{10}. The bits $\Delta_L^9[13:7]$ are all non-active. Due to the rotation they provide the LSB of the input to the modular addition in Round-10. The modular addition in Round-10 therefore preserves the value of the LSBs of $\Delta_R^9[6:0] =$100 0000 with probability 1. We can then XOR the LSBs 100 000 of the difference after the modular addition with $\Delta_R^9[14:4] =$ 0 0000 10 to obtain the required LSBs of $\Delta_R^{10} =$ 100 0010.

Hence, we match $(\Delta_L^{11} \oplus \Delta_R^{11}) \lll 2$ against **** **** *100 0010 and discard all plain- and ciphertext tuples that do not match the difference. This check is implemented in Algorithm 2.

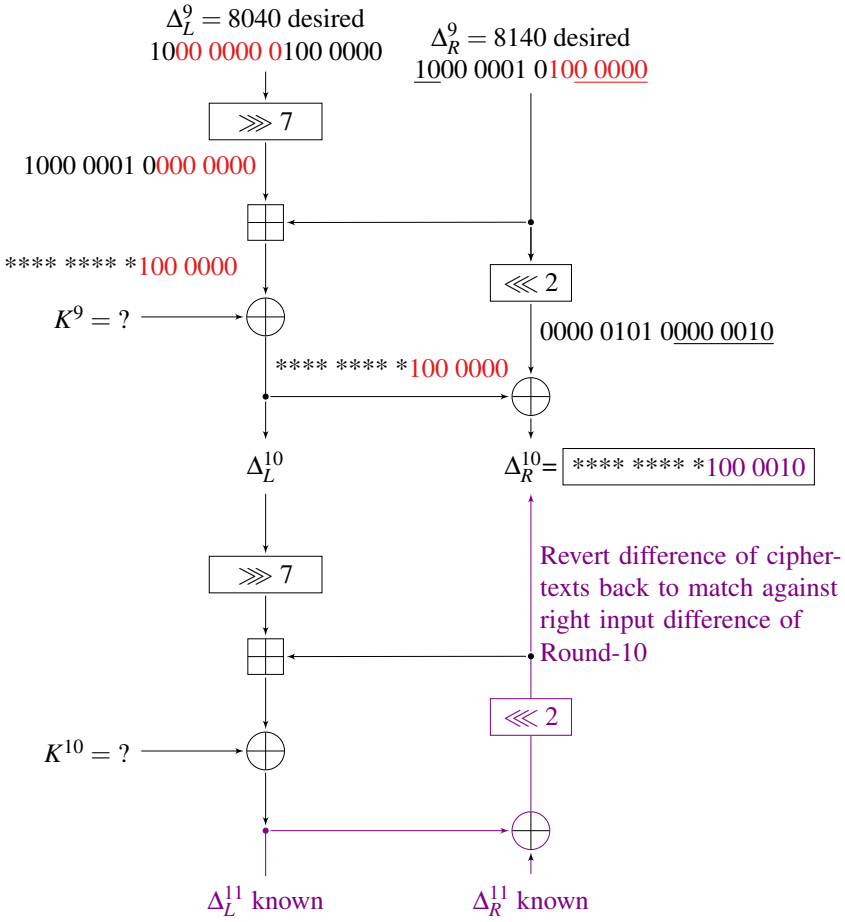


Figure 2.3: Filtering condition in Δ_R^{10} in violet. Red and underlined bits show how the desired Δ^9 difference leads to this requirement. Asterisks denote indeterminate bit values.

Algorithm 2 Filter tuples

Input: Two ciphertexts.
Output: Boolean indicating whether ciphertext pair passes filtering.

1: **procedure** FILTER(C, C^*)
2: $\Delta C \leftarrow C \oplus C^*$
3: $\Delta C_L \leftarrow \Delta C[31:16]$
4: $\Delta C_R \leftarrow \Delta C[15:0]$
5: $diff \leftarrow (\Delta C_L \oplus \Delta C_R) \ggg 2$
6: **return** $diff[6:0] == b100\ 0010$
7: **end procedure**

2.4 Round key Guessing

2.4.1 Procedure

We guess bits from the last two round keys and keep a separate counter for each of the guesses. We partially decrypt all of the filtered ciphertexts with the guessed keys and match against the expected differential Δ^9 at the input of Round-9. Correct matches indicate that the key guess may be correct and we increment its counter.

Since we have generated 2^{30} tuples and the probability of the differential trail to occur is 2^{-28}, we expect the correct key to have approximately 4 increments. Hence, we only keep key candidates with at least 4 increments for the brute-forcing phase.

Algorithm 3 shows the procedure for the full attack.

Algorithm 3 Find candidates

Input: List of filtered plain- and ciphertext tuples.
Output: List of key candidates.

```
1:  procedure FIND_CANDIDATES(tuples)
2:      Initialize candidates ← []
3:      Initialize counters[2^25] to 0s
4:      // Partially decrypt ciphertext pairs for different key guesses.
5:      for (P, P*, C, C*) in tuples do
6:          for key guess K^10 in 0 .. 2^16 do
7:              P^10 ← decrypt(C, K^10, 0)
8:              P*^10 ← decrypt(C*, K^10, 0)
9:              for key guess K^9[15 : 7] in 0 .. 2^9 do
10:                 for carry_guess in 0,1 do
11:                     P^9 ← decrypt(P^10, K^9[15 : 7]||0000000, carry_guess)
12:                     P*^9 ← decrypt(P*^10, K^9[15 : 7]||0000000, carry_guess)
13:                     // Check whether differential is met.
14:                     Δ^9 ← P^9 ⊕ P*^9
15:                     if Δ^9 == (0x8040, 0x8140) then
16:                         counter[K^9[15 : 7]||K^10] += 1
17:                     end if
18:                 end for
19:             end for
20:         end for
21:     end for
22:     for c in 2^25 do                    ▷ Keep key guesses with at least 4 increments.
23:         if counters[c] >= 4 then
24:             to candidates append c
25:         end if
26:     end for
27:     return candidates
28: end procedure
```

2.4.2 Guess Range

Due to the preliminary filtering step, we only need to match 25 bits of the input difference to Round-9. In particular, if $\Delta^9[31:30]$ and $\Delta^9[22:0]$ are met, $\Delta^9[29:23]$ is automatically met as well. This corresponds to the bits in red in Δ_L^9 in Figure 2.3, that propagate with probability 1 through the modular addition if the differential is met. If any bits in this range were flipped, this would lead to bit flips in Δ_R^9 as well. Hence, it is sufficient to check the described range.

To be able to do so, we need to guess the full last round key K^{10}, the 9 most significant bits of the penultimate round key $K^9[15-7]$ as well as the carry bit at position 6 to know how the neglected 7 least significant key bits influence the remainder of the modular subtraction.

The start index of this guessing range is off by 2 in the original paper. Hence, we have 2^{26} partial key (and carry bit) counters instead of the stated 2^{28}. In fact, it has no influence on the overall order of magnitude of the attack's time complexity whether we guess the remaining bits of K^9 in the key guessing or the brute-forcing stage. The differential bits $\Delta^9[29:23]$ are be precisely met if and only if a filtered pair also satisfies $\Delta_L^{10}[6:0] = 100\ 0000$, independent of any key bit guesses of $K^9[6:0]$. Hence, any guesses of K^9 at these positions would automatically be accepted, which is equivalent to brute-forcing them at a later stage. However, not including these bits in the key guessing stage, leads to a reduction of its time complexity without influencing the overall time complexity of the whole attack (see Section 3.1).

In practice, we cannot implement the full attack due to its high runtime complexity (see Chapter 3). Hence, we only guessed a subset of the above key bits and assumed that we know the rest of the correct key. We therefore did not implement the carry guess.

2.5 Brute-force Phase and Key Inversion

For these key candidates we brute-force the remaining $64-25=39$ bits of the last round keys K^9, K^8, K^7. Dinur [7] details how to reconstruct the master key from 4 consecutive round keys and hence the whole key schedule. Since the brute-force stage is computationally expensive, but conceptually trivial, we have not implemented this stage of the attack.

Chapter 3

Complexity Analysis

3.1 Theoretical Complexity Analysis

Note that the following calculations are estimates in order of magnitude using simplifying assumptions like independence of random variables that are standard in cryptanalysis.

Assuming a uniform distribution of binary words after 10 rounds of SPECK32, the filtering that matches against 7 bits reduces the number of tuples to $2^{30-7} = 2^{23}$. For each of these we try 2^{26} key and carry guesses (see Section 2.4.2). Hence, we need to perform $2^{26}(\text{counters}) \times 2^{23}(\text{tuples}) = 2^{49}$ 2-round decryptions for the different tuples and key guesses in total. We take 11-rounds of SPECK32 as the base unit for our analysis, so the key guessing phase has a complexity of $2^{49}(\text{2-round decryptions}) * \frac{2}{11}(\text{fraction of rounds}) \approx 2^{46.5}$.

Due to the filtering, we only need to match 25 bits of the input difference to Round-9 (see Section 2.4.2). Assuming that 2-round decryptions with an arbitrary key lead to uniformly distributed binary words (Wrong-Key Randomisation Hypothesis) we expect to see approximately $2^{49}(\text{decryptions})/\ 2^{25}$ (bits to match) $= 2^{24}$ total increments of any counter.

So far, we included the carry bit guess as a counter, since it does increase the number of performed decryptions and matches. However, we do not consider the carry guess as a factor when counting how many correct tuples a particular key guess has. Independently from the value of the carry guess, we increment the counter of the actual key guess if we observe a match. Hence, the carry guess increases the number of total increments, but does not influence the number of counters among which these increments are contributed. So, in the following, we consider 2^{25} counters, where we merge the counters for the two different carry guesses for a specific key into a single counter.

Assuming a uniform distribution of the increments among the counters, the probability of a counter to be incremented at least once is roughly $2^{24}(\text{total increments})/2^{25}(\text{counters}) = \frac{1}{2}$. Hence, we expect to see approximately $2^{25}(\text{counters}) \times (\frac{1}{2})^4 = 2^{21}$ counters that are incremented at least 4 times. These are the key candidates that are passed on to the brute-forcing phase.

The total number of key guesses in the exhaustive search of the remaining 39 key bits (see Section 2.5) is therefore approximately 2^{21}(candidates) $\times$ 2^{39}(remaining bits) = 2^{60} in contrast to the claimed 2^{55} in [5] Therefore, the overall time complexity is dominated by the 2^{60} 11-round encryptions for each key guess in the brute-force phase.

3.2 Empirical Complexity Analysis

3.2.1 Experiment Design and Limitations

While we can perform the generation and filtering of 2^{30} 11-round encryptions within a few minutes on a laptop (i7 processor, 2.6 GHz), the originally proposed key guessing phase with a time complexity of $2^{46.5}$ 11-round encryptions is far from being feasible in practice. Therefore, we guess only specific bits from the last two round keys and assume that we know the remaining round key bits. To model the original attack as close as possible, we investigate whether the position that we are guessing the bits in makes a difference - different positions within the key as well as guessing only bits in one of the last two keys.

Due to the efficiency requirements, the attack has been implemented in the C programming language. The complexity analysis of our measurements relies on the quality of the random number generator that is being used to produce keys and input values.

Taking the limitations around time complexity into account, we have used 50-100 separate sets of 2^{30} randomly generated tuples as a reasonable size for the majority of our experiments. The maximum number of bits that we guessed was 18 (9 bits in both last round keys), which ran for approximately 18 hours on the computing cluster for a single set.

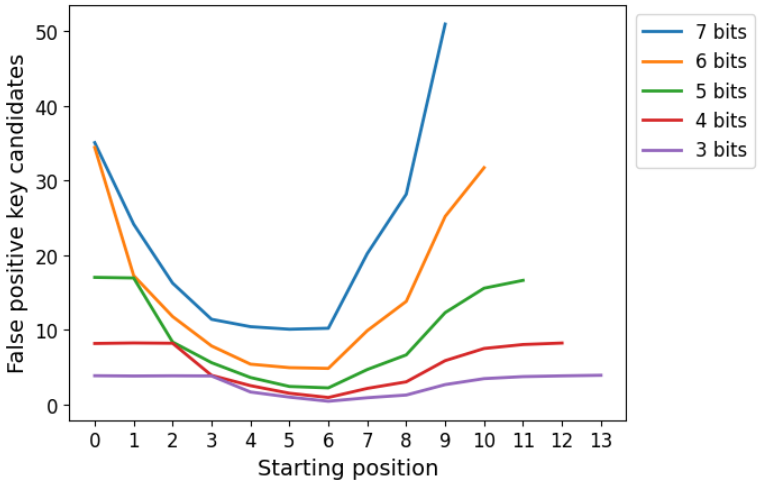
3.2.2 Preliminary Steps as Predicted

We consistently observe that the filtering step reduces to number of tuples to values close to 2^{23}. This verifies the uniform distribution assumption of the 7 LSB of the right Round-10 input difference Δ_R^{10}.

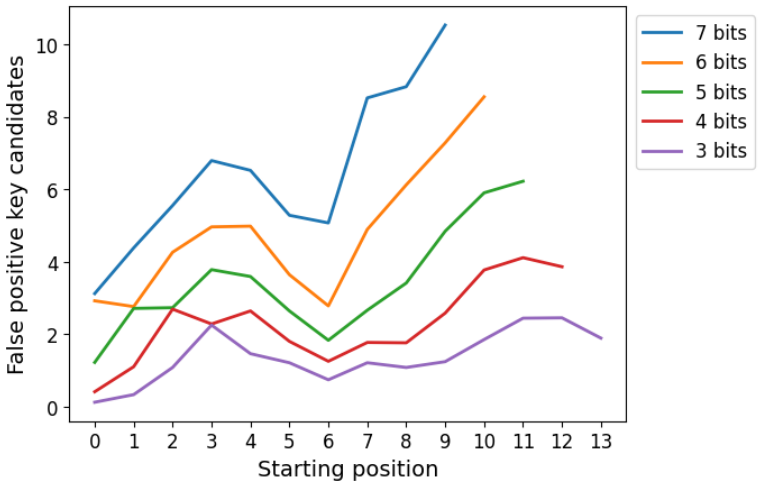
In section 3.1 we have outlined that we expect to see 4 increments of the correct key due to the probability of the differential trail that we are considering. Note that this is a lower bound, since multiple trails contribute to a differential. While we have picked a trail that occurs with a high probability, there are also other trails that take different values in the interior rounds, but agree with our trail on the input and output differences. Hence, the expected output difference should occur slightly more often that predicted by the expected value from our single trail. With an average of 6.24 counter increments of the correct key per set of 2^{30} tuples, our empirical results support this analysis. Over 100 separate sets, the correct key has an acceptance rate of 54% (i.e. has at least 4 increments and progresses through to the brute-force stage).

3.2.3 Higher Number of False Positive Key Candidates

We ran the generation of 2^{30} tuples, the filtering and the partial key guessing step 100 times for 3,4,5,6 and 7 consecutive key bit guesses for each possible starting position within the key. We guess only part of the key by taking the correct key as a template, defining the consecutive positions at which we guess the bits and then flipping those bits in all possible combinations while leaving the rest of the key set to the correct values. Figure 3.1 shows the measured average of false key guesses that had a counter with at least 4 increments (false positives).



(a) Key9 bit guesses



(b) Key10 bit guesses

Figure 3.1: Average number of false positive key candidates over 100 runs for different number of key guesses and starting positions

As expected the number of false positives is strictly larger when guessing a higher number of bits at the same position, since the key guesses for a smaller number of flexible bits are a subsets of the key guesses for a larger number of flexible bits. The number of false positives is correlated with the position that we are guessing the bits in. Since the non-linearity propagates from LSB to MSB in modular addition, it is not

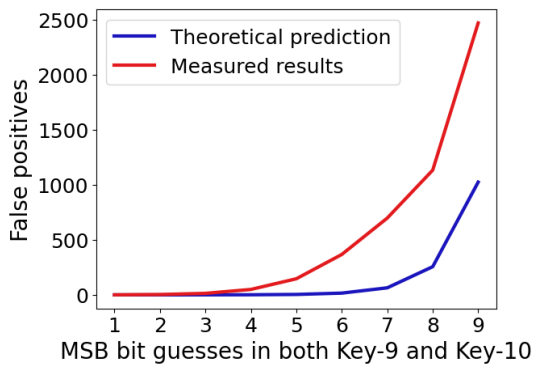
surprising that we get the highest number of false positives when we guess the most significant bits. Essentially, a bit flip in a high-positioned bit (i.e. closer to the MSB) might just produce some overflow borrow bit, that is discarded and has no effect while it is less likely for a lower bit flip not to disturb the difference.

It is interesting, that there is a notable local minimum at starting position 6 in both keys, since there is no straight-forward explanation of why this would be the case. Also the pattern of constant values for small starting positions and the following parabola-shaped levelling out to starting position 6 for bit guesses in Key-9 (Figure 3.1a) is worth further investigation.

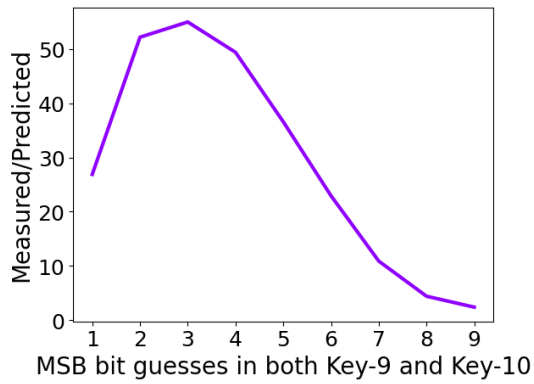
However, we are interested in modelling the full attack as close as possible and hence want to maximize the number of bits that we are guessing. We therefore focus our further efforts on guessing MSBs of both keys simultaneously, since the experiments with a small number of bit guesses indicate that this position produces the most false positives. Note that the observed false positives for a smaller number of bit guesses are a subset of the false positives observed for the full attack. Since we cannot guess as many bits as in the official attack, we will only be able to provide a lower bound for the absolute number of false positives.

For a small number of key bit guesses, the number of false keys that match the differential at an interior stage of the cipher is up to 55 times larger than the theoretic predictions suggest. While it misleadingly appears as if the measured number of false positives is increasing at a faster rate than the prediction in Figure 3.2a, in contrast to the prediction, the measurements do not necessarily indicate exponential growth. Figure 3.2b shows how the fraction of the number of measured false positive key candidates over the predicted number decreases with more bit guesses. While the measured result is still 2.4 times the expectation for 9 simultaneous bit guesses in both keys (18 bits in total), it is not possible to draw a conclusion about whether the number of false positives for the full attack will be higher than expected.

A further subtlety making this projection more difficult is the fact, that while so far we have guessed bits simultaneously in both keys (and assumed we know the rest of the key), for the remaining 7 bits, we would only be guessing bits from K^{10}. It is not clear whether this might have an effect on the further trend of the curve.



(a) Number of false positive key candidates for number simultaneous guesses in both keys



(b) Factor Measured/Predicted number of false positive key candidates

Chapter 4

Structure of False Positive Key Candidates

4.1 Motivation

We aim to provide some insight under which conditions false positive key candidates occur. Our experiments in Section 3 have indicated that the number of false positive key candidates might be significantly higher than expected, which would increase the predicted runtime of the full attack even further. Hence, efficient filtering methods to reduce false positive key candidates are an essential step to make the attack feasible on the 32-bit variant.

4.2 Experimental Approach

4.2.1 Number of False Positive Keys for a Particular Ciphertext Pair

To investigate this effect of higher false positives, we choose some ciphertext pairs from our dataset that are known to produce an increment for a false positive key candidate. We then iterate over all 2^{32} possible round keys K^9 and K^{10} and count how many keys satisfy the expected difference after two rounds of decryption. The answer is surprisingly many: all ciphertexts that produced false positive key increments did so for several million keys.

We only looked at a small number of ciphertext pairs and our method of selecting a pair that already produces false positives for a small number of bit guesses is prone to choose pairs that are likely to have a large number of false positives. However, Dinur [7] noted that randomness assumptions do not hold: We may expect each ciphertext to have on average one set of round keys (K^9, K^{10}) for which the desired difference is matched after two rounds of decryptions. But in reality, the keys satisfying this constraint are far from being uniformly distributed across the different possible ciphertexts. A small number of ciphertexts will have several keys satisfying this constraint, while the rest of the ciphertexts will have none. Hence, our selected ciphertexts might well be a prime example of a ciphertext that produces false positive key candidates.

It seems counter-intuitive that this many different keys can make the same ciphertext match the desired difference after two rounds of decryption. Therefore, we want to look further into the circumstances under which a key does or does not match this constraint for a given ciphertext pair.

4.2.2 XOR Difference of Correct Key and False Positive Keys

First, we want to know whether the false positive keys have any common patterns. We are especially interested in how they differ from the correct key. As above, we consider ciphertext pairs of some arbitrary key values that produce some false positives. We then iterate through all false positive keys for that pair and record the number of them that differ at bit position i from the correct key for all 32 key bits.

We make the following observations:

- The number of keys that differ in bit i are equal for many of the bit positions.
- There are bit positions at which none of the false positives differ from the correct key. In other words, these bit positions are fixed to the bit value of the correct key at this position for any key for which the desired difference after two rounds of decryption is matched. If we flip this bit, we will not be able to match this difference no matter what values we assign to any of the other key bits.
- The pattern of these bit positions varies with different ciphertext pairs, but there are bit positions that are fixed often (mostly in the LSBs of the last round key).
- The bits that are not fixed to a certain value still must have some constraints between each other, since the number of false positive keys is smaller than all possible combinations of bit guesses of the non-fixed bits.

For the fixed correct keys $K^9 = 0x63FB$, $K^{10} = 0xBC58$ Table 4.1 shows which bit positions of key candidates were fixed for a small selection of different ciphertext pairs (denoted by $\times$). All unmarked positions agree with the correct key for half of the false key candidates (rounded) and flip the bit for the other half. By ($\times$) we denote bit positions that vary from this number; either more or less than half of the key candidates agree with the correct bit at this position.

Due to the time complexity of these experiments we were only able to look at a small number of ciphertexts for few distinct correct keys. Nevertheless, we hypothesise that the occurence of fixed key bits is representative of a more general case of false positives. Since we treat bits independently in this analysis, we cannot capture any relationships between different key bits. To better understand the constraints different key bits pose on each other, we proceed to a more theoretical approach.

K9																
ID	Ciphertexts	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
0	(0xBE7A26AB, 0x3FC3AC1A)						×	×									
1	(0x6FD2EDCA, 0xF31D660D)								×								
2	(0x8C0559D6, 0x93BC9E0)																
3	(0xDB33820A, 0x50711440)								(×)								
4	(0x38A70B7A, BB6C93B9)							(×)									
5	(0xE4F35D9D, 0x6641D227)							×	(×)								
6	(0x19CACC27, 0x6799997C)						(×)	(×)									
7	(0xA67CD156, 0x23BE419C)								(×)								

K10																	
ID	Ciphertexts	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
0	(0xBE7A26AB, 0x3FC3AC1A)								×	×		×	×	×	×	×	
1	(0x6FD2EDCA, 0xF31D660D)				×	×	×			×	×			×	×	×	
2	(0x8C0559D6, 0x93BC9E0)						×		×			×	×	×	×	×	
3	(0xDB33820A, 0x50711440)					×		×	×		×				×	×	
4	(0x38A70B7A, BB6C93B9)							×	×	×	×			(×)	×		
5	(0xE4F35D9D, 0x6641D227)							×		×		×	×	×	×	×	
6	(0x19CACC27, 0x6799997C)		×	×	×	×	×	×			×			×	×		
7	(0xA67CD156, 0x23BE419C)						×		×	×	×					×	

Table 4.1: Fixed key bit positions in false positives keys for different ciphertext pairs under the same correct key. Positions where all false positive keys are fixed to the correct key bit are marked with $\times$. Positions where more or less than half of the false positive keys differ from the correct key bit are marked with $(\times)$.

4.3 Formal Approach

4.3.1 Preliminaries

Definition 3. *Analogous to the carry in addition modulo 2^n [12], we define the borrow in subtraction modulo 2^n for minuend x and subtrahend y recursively as $b_0 = 0$ and $b_{i+1} := (\neg x_i \wedge b_i) \oplus (\neg x_i \wedge y_i) \oplus (y_i \wedge b_i)\ \forall i \geq 0$. Equivalently, $b_{i+1} = 1 \Leftrightarrow y_i + b_i > x_i$ as can easily be verified via a truth table.*

Analogous to the property in modular addition, we then have $x - y\ (mod\ 2^n) = x \oplus y \oplus b$.

Definition 4. *The majority function maj(a,b,c) for $a,b,c \in \mathbb{F}_2$ equals 1 if the majority of its inputs is 1 and 0 otherwise.*

Lemma 1. *Let b be the borrow in subtraction modulo 2^n for minuend x and subtrahend y. Then $b_{i+1} = maj(\neg x_i, y_i, b_i)$ where the subscript i denotes the i^{th} bit.*

Proof.

$$\begin{aligned} \text{By definition 4}, (\neg x_i, y_i, b_i) = 1 &\Leftrightarrow \neg x_i + y_i + b_i > 1 \\ &\Leftrightarrow (1 - x_i) + y_i + b_i > 1 \\ &\Leftrightarrow y_i + b_i > x_i \end{aligned}$$

□

4.3.2 Algebraic Description of two Rounds of Decryption

To understand the dependencies between key bits and corresponding bits in the difference of messages, we describe the i^{th} bit in different steps of two rounds of decryption in relation to the left input value x and right input value y. Subscripts denote the bit position within the word and are evaluated mod 16. We denote the round keys $K^{10} = k$, $K^9 = m$ and the borrows in Round-10 and Round-9 b and d, respectively. For ease of use the appearing bits have been grouped by their type (i.e. first bits of of the left ciphertext, then bits of the right ciphertext etc.) and sorted in ascending order. In the derivation of expression (4.11) the bit x_{i-5} appears twice and hence cancels out.

The positions of the expressions (4.1) - (4.12) are illustrated in Figure 4.1. Equation (4.12) denotes the left output and equation (4.10) the right output after two rounds of decryption.

$$x_i \tag{4.1}$$

$$y_i \tag{4.2}$$

$$x_i \oplus k_i \tag{4.3}$$

$$x_i \oplus y_i \tag{4.4}$$

$$x_{i+2} \oplus y_{i+2} \tag{4.5}$$

$$x_i \oplus x_{i+2} \oplus y_{i+2} \oplus k_i \oplus b_i \quad \text{where } b_0 = 0; \quad b_{i+1} = \text{maj}(\neg(4.3), (4.5), b_i) \tag{4.6}$$

$$x_{i-7} \oplus x_{i-5} \oplus y_{i-5} \oplus k_{i-7} \oplus b_{i-7} \tag{4.7}$$

$$x_{i-7} \oplus x_{i-5} \oplus x_{i+2} \oplus y_{i-5} \oplus y_{i+2} \oplus k_{i-7} \oplus b_{i-7} \tag{4.8}$$

$$x_{i-7} \oplus x_{i-5} \oplus y_{i-5} \oplus k_{i-7} \oplus m_i \oplus b_{i-7} \tag{4.9}$$

$$x_{i-5} \oplus x_{i-3} \oplus x_{i+4} \oplus y_{i-3} \oplus y_{i+4} \oplus k_{i-5} \oplus b_{i-5} \tag{4.10}$$

$$x_{i-7} \oplus x_{i-3} \oplus x_{i+4} \oplus y_{i-5} \oplus y_{i-3} \oplus y_{i+4} \oplus k_{i-7} \oplus k_{i-5} \oplus m_i \oplus b_{i-7} \oplus b_{i-5} \oplus d_i$$
$$\text{where } d_0 = 0;\ d_{i+1} = \text{maj}(\neg(4.9), (4.10), d_i) \tag{4.11}$$

$$x_{i-14} \oplus x_{i-10} \oplus x_{i-3} \oplus y_{i-12} \oplus y_{i-10} \oplus y_{i-3} \oplus k_{i-14} \oplus k_{i-12} \oplus m_{i-7}$$
$$\oplus b_{i-14} \oplus b_{i-12} \oplus d_{i-7} \tag{4.12}$$

4.3.3 Constraints on False Positive Key Candidates

We can now formalize what it means for a key to be a key candidate for a given pair of ciphertexts $(c = x||y,\ c^* = x^*||y^*)$. A key is a candidate if the XOR difference of the output of two rounds of decryptions of the ciphertexts equals $\Delta^9 = (0x8040, 0x8140)$. Note that if we XOR an expression for two different ciphertexts being decrypted with the same key, all directly appearing bits of k and m in the expression cancel out, since they are constant for both calculations. However, the borrows b and d depend not only on the key, but also on the ciphertexts and can therefore differ between the two

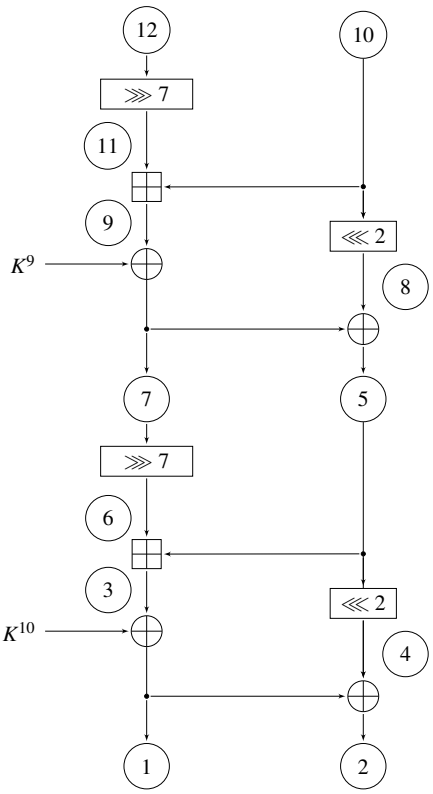


Figure 4.1: Positions of expressions 4.1-4.12 in the last two rounds.

calculations. Hence XOR-ing the expressions for the right output (expression 4.10) and the left output (expression 4.12), we have equations of the following two forms:

$$\Delta x_{i-5} \oplus \Delta x_{i-3} \oplus \Delta x_{i+4} \oplus \Delta y_{i-3} \oplus \Delta y_{i+4} \oplus \Delta b_{i-5} = \Delta^9_{Ri} \tag{4.13}$$

$$\Delta x_{i-14} \oplus \Delta x_{i-10} \oplus \Delta x_{i-3} \oplus \Delta y_{i-12} \oplus \Delta y_{i-10} \oplus \Delta y_{i-3} \oplus \Delta b_{i-14} \oplus \Delta b_{i-12} \oplus \Delta d_{i-7} = \Delta^9_{Li} \tag{4.14}$$

where $\Delta x = x \oplus x^*, \Delta y = y \oplus y^*, \Delta b = b \oplus b^*, \Delta d = d \oplus d^*$, and the subscript indicates the bit position within the word.

Since we consider a particular ciphertext pair, all bit values of x and y are constant and known. All equations of the form (4.13) and (4.14) therefore reduce to

$$\Delta b_{i-5} = 0/1 \tag{4.15}$$

$$\Delta b_{i-14} \oplus \Delta b_{i-12} \oplus \Delta d_{i-7} = 0/1 \tag{4.16}$$

depending on the ciphertext bits.

We can iteratively calculate Δb_{i-5} starting from $i = 5$ and observe whether the corresponding equation restricts the involved key bits that are appearing in the borrow expression. The complexity of the borrow expression potentially increases in each calculation step (unless terms cancel out). We primarily focus our analysis on equations of the form (4.15), since the rotation in the cipher causes even the 'simplest' expressions of the form (4.16) starting from $i = 7$ to contain the higher indices (and therefore

more complex) borrow expressions from the bottom modular subtraction. We characterize several situations in which the involved key bits get fixed to a certain value or remain flexible.

Equations that are arbitrarily satisfied (key bit remains flexible):

- Keys do not appear in borrow expression:
  - Special case $i = 5$: $b_0 = 0$ by definition, so there are no implicit key bits involved. Fully independent of any key bits, for a ciphertext pair to have any key candidates, the following needs to be satisfied:

$$\Delta x_0 \oplus \Delta x_2 \oplus \Delta x_9 \oplus \Delta y_2 \oplus \Delta y_9 = 0 \tag{4.17}$$

    This does not pose any restrictions on the key. The equation could be checked as an additional filtering of the ciphertexts before decryption.
  - Majority function predetermined for both ciphertexts: if the inputs $x_{i+2} \oplus y_{i+2}$ and b_i have equal values, the value of the majority function in 4.6 is already fixed independently from the input $x_i \oplus k_i$, i.e. from the key bit. If the key bit is not contained in either of the borrow expressions, the equation does not pose any restrictions on the key bit.
- Keys appear in borrow expression, but are automatically satisfied: For example the constraint is automatically satisfied if we require $\Delta b_i = 0$ and $b_i = b_i^*$ or if we require $\Delta b_i = 1$ and $b_i = \neg b_i^*$. If several key bits appear in normal and negated form in the borrow expressions, there are also other more complex equations that are automatically satisfied.

Note that the complement of this case are equations that are automatically not satisfied. This can only occur with a ciphertext pair that has no key candidates at all.

Equations that constrain key bits:

- Key bit appears only in one borrow expression: As described above it is possible that the majority function is predetermined and the key bit does not appear in the expression of the borrow bit. If this is the case for exactly one of the ciphertexts, equation 4.15 fixes the other borrow bit to a specific value. Depending on the expression of the other borrow bit this constrains the involved key bits. We have only observed the case in which the other borrow expression contains just one key bit. In that setting, the value of this key bit gets fixed to a specific value.
- Key bits appear in both borrow expressions: If the equation is not arbitrarily satisfied we have either one of two cases:
  - Fixing a single key bit: Δb only depends on the value of one of the appearing key bits. Then the equations fix that key bit to match the constraint. This is possible even if more than one key bit appears in the expression. For example if $b_i = \text{maj}(k_n, 1, k_l) = k_n \vee k_l$, $b_i^* = \text{maj}(k_n, 0, \neg k_l) = k_n \wedge \neg k_l$ and the differential requires $\Delta b_i = 0$. This forces $k_l = 1$ and does not pose any restriction on k_n.

– Relation between several keys without fixing to specific value: Δb truly depends on several key bits.
For example $b_i = \text{maj}(k_n, 1, k_l) = k_n \vee k_l$, $b_i^* = \text{maj}(\neg k_n, 1, \neg k_l) = \neg k_n \vee \neg k_l$ and the differential requires $\Delta b_i = 1$. This forces $k_n = k_l$.

4.3.4 Application to a Particular Ciphertext Pair

We applied this approach to the ciphertext pair (0xBE7A26AB, 0x3FC3AC1A) from subsection 4.2.2 and were able to verify all the fixed key bit positions in K^{10} that were observed in the computational approach. The equations of the form (4.15) either posed no restrictions on a key bit, fixed a key bit to a certain value or described a constraint for two key bits to have the same value. There were no more complex relationships between the key bits imposed by these equations. However, we know that equations of the form 4.16 can impose these types of constraints on K^{10}. For example, we observed a constraint of the type $k_n = k_l \vee k_o$. This explains the observation in the computational approach that the more flexible bits do not generate false positive key candidates in every possible combination.

As mentioned before, solving these equations manually is a tedious process. While we considered utilizing SAT or MILP solvers to aid the process, this would merely give us valid assignments without providing any insights into why a particular key constitutes a valid candidate. Even if we were able to enumerate the number of false positives for every possible ciphertext pair for a smaller word size, this would only allow very limited conclusions about the behaviour of false positive key candidates for larger word sizes. To be able to provide a better estimation of the number of false positives, we require a more compact method to calculate the number of fixed bits.

4.4 Graph Theoretical Approach

4.4.1 Fixing one Input in the Differential Probability Calculation

If we fix a particular ciphertext pair and ask for the number of false positive key candidates that lead to a match of the differential after two rounds of decryptions, we effectively fix the output differences Δ^9 to the values of the differential and Δ^{11} to the difference of the ciphertexts. Figure 4.2 shows that we can then derive the intermediate difference Δ^{10} and all input and output differences at the modular substractions in Round-9 and Round-10 since we can trace XOR differences through rotations and XOR operations. We label minuend, subtrahend and output differences $\alpha^9, \beta^9, \gamma^9$ and $\alpha^{10}, \beta^{10}, \gamma^{10}$ for Round-9 and Round-10 modular subtraction, respectively.

Calculating the number of false positive key candidates for this ciphertext pair is not too different from the general task of calculating the differential probability with which inputs difference Δ^{11} propagate through two decryption rounds of SPECK32 to the given output difference Δ^9. From a more detailed perspective this is the differential probability with which both $(\alpha^{10}, \beta^{10}) \rightarrow \gamma^{10}$ and $(\alpha^9, \beta^9) \rightarrow \gamma^9$ through modular subtraction. This is equivalent to the differential probability of the respective modular addition, since the differential probability of modular addition is symmetric in

its arguments [12]. Hence, we will use the terminology of addition/subtraction and carry/borrow expressions interchangeably in this context.

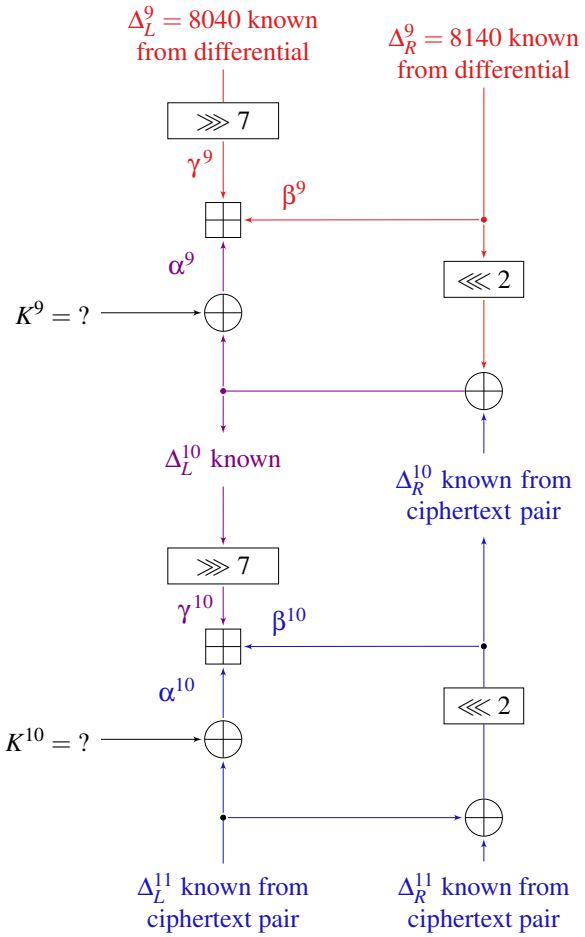


Figure 4.2: For a chosen ciphertext and under the assumption that the differenetial is met at Δ^9 all differences in the bottom two rounds are fixed (in particular Δ_L^{10}). The inputs to the modular additions can be derived from the fixed differences Δ^9 (known from the differential) and Δ^{11} (fixed by the ciphertexts).

However, in the bottom round we have some additional information: not only do we know the subtrahend difference β^{10}, but we can revert the actual subtrahend values from the ciphertexts. So, in contrast to the differential probability of modular addition with respect to XOR in Equation 1.1, we only iterate over the minuend values a, a^* while the subtrahend values b, $b^* = b \oplus \beta$ remain fixed. This differential probability that we want to calculate then becomes

$$\mathrm{xdp}^+(\alpha, \beta, b \to \gamma) := \frac{\#\{a \in \{0,1\}^n : (a+b) \oplus ((a \oplus \alpha) + (b \oplus \beta)) = \gamma\}}{2^n}. \tag{4.18}$$

The denominator corresponds to the number of minuend values for which the desired output difference γ is matched, i.e. the number of key guesses for K^{10} for which the desired Round-10 difference Δ^{10} is matched.

4.4.2 Biadjacency Matrix Algorithm for Number of Key Candidates

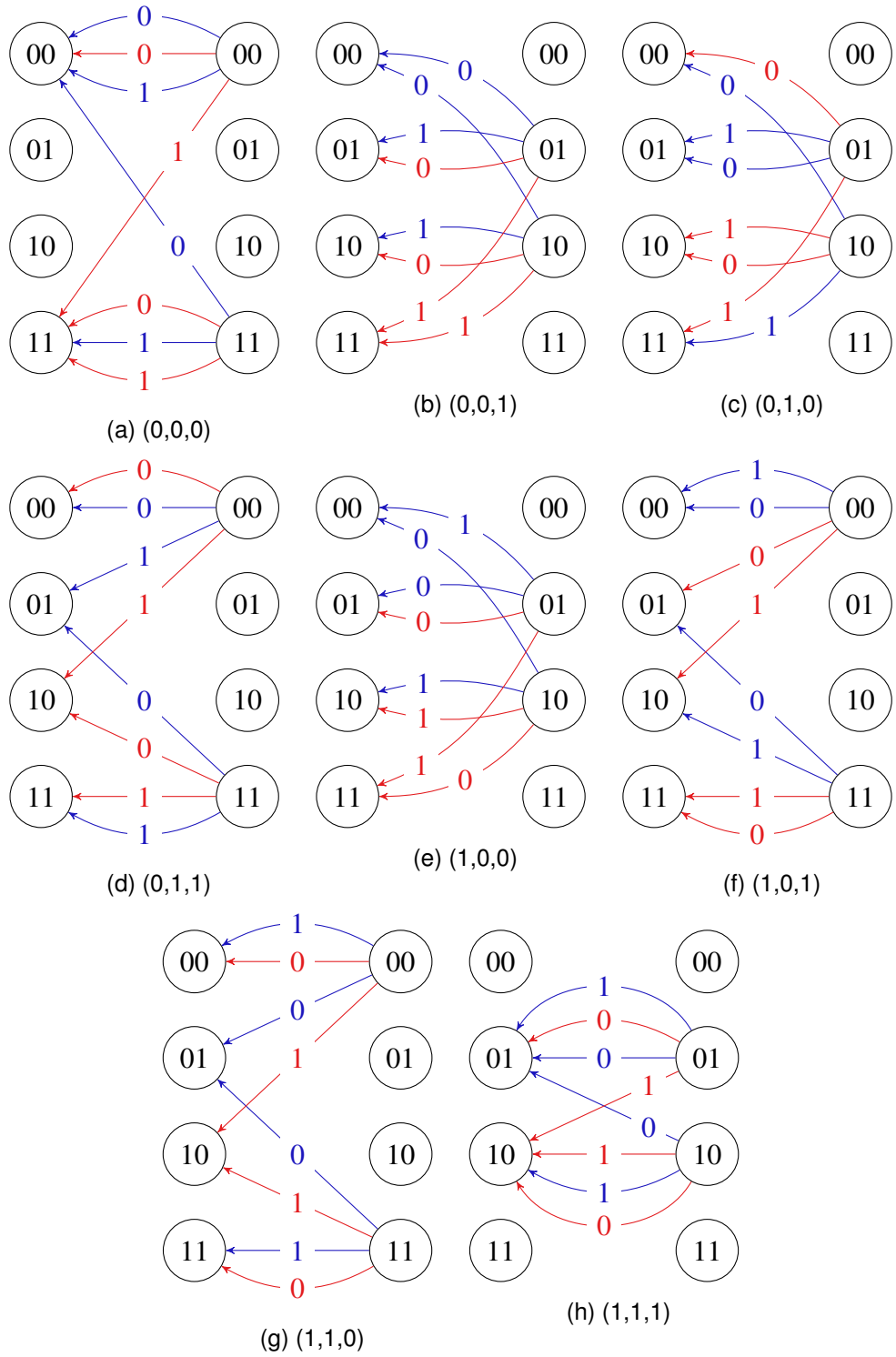


Figure 4.3: All possible subgraphs of xdp$^+$ adapted from [14]. Captions denote the XOR difference tuple $(\alpha_i, \beta_i, \gamma_i)$. The vertices are labelled with the incoming carry values $(c_i\, c_i^*)$ on the right and the outgoing carry values $(c_{i+1}\, c_{i+1}^*)$ on the left. The edges are labeled with the value of minuend a_i. The subgraphs for different values of subtrahend b_i are drawn in the same subfigure, but indicated in different colours: Blue edges belong to the subgraph for $b_i = 0$, red edges to the subgraph for $b_i = 1$.

We can utilize some counting techniques that have been proposed to calculate the differential probability in the general setting. Following the approach by Mouha et al.[14], we draw graphs that represent how the incoming carries (c_i, c_i^*) in a single bitwise modular additon propagate to the outgoing carries $(c_{i+1}\ c_{i+1}^*)$.

However, we have to adapt the graphs to reflect our situation of fixed input values b, $b^* = b \oplus \beta$ to the modular addition in the bottom round. For this purpose each of the 8 graphs is split up into two subgraphs depending on whether the subtrahend y_i is 0 (blue) or 1 (red). To demonstrate the symmetry and relation to the original graphs, we still visualize them as a single graph, but each of the 8 graphs in Figure 4.3 actually represents two separate subgraphs containing just the blue or the red edges, respectively. For each of the graphs for (α, β, γ), we read off the corresponding two biadjacency matrices

$$A_{(\alpha_i,\beta_i,\gamma_i,b_i=0)} \quad \text{and} \quad A_{(\alpha_i,\beta_i,\gamma_i,b_i=1)}.$$

$$A_{0000} = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_{0001} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2 \end{bmatrix} \quad A_{0010} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$A_{0011} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \quad A_{0100} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad A_{0101} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$A_{0110} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_{0111} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_{1010} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$A_{1011} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \quad A_{1110} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad A_{1111} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The remaining matrices are duplicates: $A_{1000} = A_{0010}$, $A_{1001} = A_{0011}$, $A_{1100} = A_{0110}$ and $A_{1101} = A_{0111}$.

The differential probability of input differences α, β propagating to output difference γ can be calculated via matrix multiplication of the biadjacency matrices of the subgraphs in Figure 4.3. How this process is applied in our case is detailed below.

Lemma 2. *In the start state of modular addition, both carries are 0, hence let* $S = [1,0,0,0]^T$ *and* $F = [1,1,1,1]$. *Let* (c,c^*) *be a fixed ciphertext pair and* k *be an undetermined round key. Let*

$$a = c[31:16] \oplus k, \quad b = (c[31:16] \oplus c[15:0]) \ggg 2$$
$$a^* = c^*[31:16] \oplus k, \quad b^* = (c^*[31:16] \oplus c^*[15:0]) \ggg 2$$

be the inputs to the modular subtraction in the last round. Define the input XOR differences $\alpha = a \oplus a^*$, $\beta = b \oplus b^*$. *Let the desired output difference*

$$\gamma = ((0x8140 \lll 2) \oplus \beta) \ggg 7.$$

For each bit position i, let $s[i] = \alpha_i \,||\, \beta_i \,||\, \gamma_i \,||\, b_i$ *be the concatenation of the corresponding bits in the differences. Then the differential probability with which the inputs* α, β, b *propagate to the desired output difference* γ *is*

$$xdp^+(\alpha,\beta,b \to \gamma) = 2^{-n} * FA_{s[n-1]} \ldots A_{s[1]} A_{s[0]} S \tag{4.19}$$

In particular, the number of key candidates for which the desired Δ^{10} *is matched, is*

$$\#\{k : \alpha,\beta,b \to \gamma\} = FA_{s[n-1]} \ldots A_{s[1]} A_{s[0]} S. \tag{4.20}$$

Algorithm to Estimate the Number of False Positive Key Candidates:

For a given ciphertext pair we can use the above algorithm (Lemma 2) to calculate the number of key guesses for K^{10} for which the desired Round-10 input Δ^{10} is met. This is a necessary condition to be a key candidate, but ultimately, we require that the Round-9 input Δ^9 is met as well, since this is where the actual differential trail ends.

However, we cannot apply the same approach to the modular subtraction of the penultimate round since we do not know the input value b to the modular subtraction as it is dependant on the last round key guess. In absence of a more precise estimate, we can approximate the number of key guesses for K^9 for which Δ^9 is met with the general formula for the differential probability of addition (Equation 1.5).

We obtain an estimate for the number of false positives by multiplying the resulting number of key guesses for K^{10} from Equation 4.20 with the number of key guesses for K^9 as the product of the 2^{16} possible key guesses with the differential probability from Equation 1.5.

4.4.3 Application to a Particular Ciphertext Pair

Performing this calculation for the ciphertext pair that we investigated in the previous sections, we obtain that for 512 key guesses for K^{10} the desired Round-10 output difference Δ^{10} is matched. This is consistent with our analysis in both the computational and rigorous approach. The constraints from the bottom addition (of type of Equation 4.15) fix precisely 7 bits of the key (for this particular key and ciphertext pair combination) and put no constraints on the remaining $16-7=9$ bits. Hence, $2^9 = 512$ key guesses for K^{10} satisfy the conditions imposed by the bottom addition.

For the particular ciphertext pair that we considered above, the algorithm in Theorem 1 (Lipmaa) calculates the differential probability of the differences in the top modular subtraction as 2^{-4}. Hence, from the 2^{16} possible K^9 round keys, we expect $2^{16-4} = 2^{12}$ to satisfy the constraints. Assuming independence of rounds, we obtain an estimate for the total number of false positive key candidates for the given ciphertext pair by multiplying the number of key guesses for K^{10} for which the desired Δ^{10} is matched with the number of key guesses for K^9 for which Δ^9 is matched. This results in an estimate of $2^9 * 2^{12} = 2^{21}$. With an actual measurement of $2,097,151 = 2^{21} - 1$ false positive key candidates for this ciphertext pair via brute-force enumeration in the computational approach this estimate is strikingly close.

4.4.4 Estimating the Number of False Positives more generally

While this is an efficient approach to estimate the number of false positive key candidates for a given ciphertext pair, it is not trivial to generalize it to an unspecified arbitrary ciphertext pair. We aimed to find criteria for nonzero probability or a generalized formula in the style of Lipmaa's Theorem 1 for the special case of calculating the differential probability in the bottom round with fixed input b. With such a criterion we could perform statistical analysis of the ciphertext space and estimate how many ciphertext pairs have nonzero proability i.e. have false positive key candidates.

Note that Lipmaa's criterion for nonzero differential probability still partially holds in this case. If the constraint in Equation 1.4 is violated, the differential is impossible, i.e. the difference Δ^{10} cannot be matched. However, the complementary implication does not hold anymore; meeting Lipmaa's criterion does not guarantee nonzero differential probability. This is due to the additional zeros that we introduced into the matrices when splitting them up into the two cases $b_i = 0$ and $b_i = 1$.

Computationally, we verified that for the product of two matrices it is still the case that the result has all-zero entries if and only if Lipmaa's criterion is violated. However, there are 192 combinations of three matrices such that their product has all-zero entries, while they satisfy Lipmaa's criterion. This suggests that we require an additional criterion to guarantee nonzero probability in this case. However, the corresponding difference strings of these matrices do not follow any easily recognizable pattern.

Regarding the actual probability formula in the nonzero case, we note that splitting the graphs results in nodes that have three incoming edges and subgraphs that have three nodes with at least one incoming edge. Therefore, the number of keys that are calculated via Formula 4.20 are not necessarily a power of 2, contrary to the general case described by Lipmaa (Theorem 1). Hence, it seems unlikely that a straightforward adaption of this formula is possible.

Chapter 5

Conclusion

5.1 Discussion

Next to the implementation of the key recovery attack by Biryukov et al. [5], we pointed out several inaccuracies in the complexity analysis of the original paper in Section 2.4.2 and Section 3.1.

In section 3.2.3 we observed a higher number of false positive key candidates than predicted when guessing only a small amount of key bits. Further, we noted that the number of false positives varies with the position at which we guess these key bits. The natural intuitive interpretation is that incorrect lower bit guesses disturb the differences since they lead to different incoming carries at fixed higher bits (that we do not guess). However, this hypothesis has been contradicted by our later findings that certain lower bit guesses are fixed to a specific value even if we guess the higher bits as well.

As we have seen in Table 4.1 these fixed bits occur predominantly in the LSBs of the last round key K^{10}. This insight suggests that it is likely that the downward trend in Figure 3.2b continues when we also guess the remaining LSBs of the last round key K^{10}. Hence, the overall number of false positive key candidates might be close to the original prediction.

However, we found a strong dependency of the number of false positive key candidates on the ciphertext pair and section 4.4.3 outlines an algorithm to calculate a better estimate of the number of false positive key candidates for a given ciphertext pair. It remains to be shown how this generalizes to the number of false positive key candidates in the attack.

To the best of our knowledge, this is the first in-depth analysis of the structure of false positive key candidates in SPECK32. Ultimately, our results challenge whether the Wrong-Key Randomisation Hypothesis, that assumes that 2 rounds of SPECK32 decryption with the wrong key result in an unbiased random binary word, is too simplified to model the number of false positive key candidates in a sensible way. In the context of linear cryptanalysis, the Wrong-Key Randomisation Hypothesis was later refined as a probability distribution by Bogdanov and Tischhauser [6] which led to counterintuitive insights about the optimal number of considered pairs in [2]. While

widely adopted in differential cryptanalysis, this hypothesis is underinvestigated for ARX ciphers and further modifications might be necessary.

Depending on the correct key, the chosen ciphertext pair and potentially further variables like the differential, a particular set of key candidates is more likely to receive increments than others. For example, we have observed on a small scale that the LSBs of the last round key K^{10} are fixed more often for different ciphertext pairs and correct keys. This suggests that key candidates that only differ in the MSBs from the correct key might have a higher chance of receiving increments. The increments are clearly not uniformly distributed among all key guesses, but concentrated on fewer candidates. This observation is compatible with Dinur's analysis in [7] which states that the distribution of solutions to the differential equations of addition is nonuniform and highly dependent on the ciphertext pair.

A deeper understanding of the increment distribution among the key guesses would allow for a better estimate of the time complexity and success rate of the attack. Since many published attacks require time and data complexities that are not feasible in practice, the choice of appropriate simplifying assumptions is crucial to their evaluation and comparison. While there has been some research into modeling the success rate of differential attacks [15], there is still a lack of understanding of the role that false positive key candidates play. Our analysis provides a starting point for further investigations under what circumstances and to what extend false positive key candidates occur.

5.2 Further Work

In particular, the results from this thesis lead to the following new research questions and tasks:

- Determine the level of precision of the false positive key candidate estimation algorithm in Section 4.4.3 on a wider range of key and ciphertext pairs.
- Establish a criterion for non-zero differential probability of modular addition with respect to the XOR operation with one fixed input in the style of Lipmaa.
- Investigate whether there exists a generalized formula for the differential probability of modular addition with respect to the XOR operation with one fixed input.
- With either or both of the above, perform statistical analysis of ciphertext pairs to estimate average number of false positive key candidates in the attack.
- The key bit dependencies outlined in the formal approach do not explain our initial observation of small number of false positives when only guessing bits around position 6. What are the causes of the observed patterns in Figure 3.1?
- The approach that we used to select the ciphertext pairs is prone to preference ciphertext pairs that have a high number of false positive key candidates. It remains an open question, whether ciphertext pairs with considerably lower but nonzero number of false positive key candidates exist.

- Investigate whether there is any correlation between the average number or the density of the distribution of false positives and the chosen differential. In particular, do the number or positions of active bits in the differential influence the number of false positive key candidates? If yes, how can we identify differentials with a good trade-off between a lower false positive rate and a high probability for the correct key?
- Investigate the extent to which ciphertext pairs with false positive key candidates overlap with the ciphertext pairs that increment the counter of the correct key. Is it possible to reduce the number of false positive key candidates without reducing the increments of the correct key (or at least reduce it at slower rate)?
- We considered the round keys independently from each other, while in reality they are closely related by the key schedule algorithm of the cipher. What implication does this have on the constraints on false positive key candidates? Can we use any known characteristics to discard some key guesses?

Bibliography

[1] Farzaneh Abed, Eik List, Stefan Lucks, and Jakob Wenzel. Cryptanalysis of the speck family of block ciphers. 2013. https://ia.cr/2013/568.

[2] Tomer Ashur, Tim Beyne, and Vincent Rijmen. Revisiting the wrong-key-randomization hypothesis. 2016. https://ia.cr/2016/990.

[3] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The simon and speck families of lightweight block ciphers. 2013. https://ia.cr/2013/404.

[4] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The simon and speck block ciphers on avr 8-bit microcontrollers. 2014. https://ia.cr/2014/947.

[5] Alex Biryukov, Arnab Roy, and Vesselin Velichkov. Differential analysis of block ciphers simon and speck. 2014. https://ia.cr/2014/922.

[6] Andrey Bogdanov and Elmar Tischhauser. On the wrong key randomisation and key equivalence hypotheses in matsui's algorithm 2. 2013. https://www.iacr.org/archive/fse2013/84240017/84240017.pdf.

[7] Itai Dinur. Improved differential cryptanalysis of round-reduced speck. 2014. https://ia.cr/2014/320.

[8] Ashutosh Dhar Dwivedi and Pawel Morawiecki. Differential cryptanalysis of round-reduced speck. 2018. https://ia.cr/2018/899.

[9] Kai Fu, Meiqin Wang, Yinghua Guo, Siwei Sun, and Lei Hu. Milp-based automatic search algorithms for differential and linear trails for speck. 2016. https://ia.cr/2016/407.

[10] Aron Gohr. Improving attacks on round-reduced speck32/64 using deep learning. 2019. https://ia.cr/2019/037.

[11] Howard Heys. A tutorial on linear and differential cryptanalysis. *Cryptologia*, 26, 06 2001. http://www.cs.bc.edu/~straubin/crypto2017/heys.pdf.

[12] Helger Lipmaa and Shiho Moriai. Efficient algorithms for computing differential properties of addition. 2001. https://ia.cr/2001/001.

[13] Helger Lipmaa and Shiho Moriai. Efficient algorithms for computing differential properties of addition. 2001. [Slides].

[14] Nicky Mouha, Vesselin Velichkov, Christophe De Cannière, and Bart Preneel. The differential analysis of s-functions. 2010. https://www.esat.kuleuven.be/cosic/publications/article-1473.pdf.

[15] Ali Aydin Selçuk. On probability of success in linear and differential cryptanalysis. *Journal of Cryptology*, 21:131–147, 2007. https://link.springer.com/content/pdf/10.1007/s00145-007-9013-7.pdf.

[16] Ling Song, Zhangjie Huang, and Qianqian Yang. Automatic differential analysis of arx block ciphers with application to speck and lea. 2016. https://ia.cr/2016/209.

Appendix A

Supplementary Code Manual

The supplementary material contains a minimal self-contained C implementation of the plain- and ciphertext tuple generation, filtering and key guessing stage with basic attack result output.

The code can be compiled with:

```
$ make attack
```

Then run the attack with:

```
$ ./attack <start pos K10> <no of bit guesses K10>
           <start pos K9>  <no of bit guesses K9> -g / <filename>
```

Parameter	Meaning
start pos K10/9	Specify the start position of the window in which to guess the key bits in the last and penultimate keys, respectively. Indices start at LSB = 0 and go up to MSB = 15.
no of bit guesses K10/9	Specify the number of bit guesses (size of the window) in the last and penultimate keys, respectively. Start pos + no of bit guesses needs to be in range 0-15.
-g	If this option is specified, a new set of random 2^{30} plain- and ciphertext tuples will be generated (seeded with the current time), stored in a file and used for the attack.
filename	Alternatively, a filename can be specified from which to read the pairs from.

Table A.1: Attack settings

Example usage:

```
$ ./attack 0 3 0 2 tuples_62476412
```

This guesses 2^5 key bits in the ranges K^{10}[2:0] and K^9[1:0] and uses the pregenerated pairs stored in the file 'tuples_62476412'.