Bachelor's Thesis Computing Science

A Linear Cryptanalysis of the DST40 Block Cipher

Quinn Ketelaars
s1039322

June 15, 2023

First supervisor/assessor:
Joan Daemen

Second supervisor:
Shahram Rasoolzadeh

Second assessor:
Bart Mennink

Radboud University

Abstract

We apply the linear cryptanalysis technique to a block cipher called DST40, which is used as the cryptographic function in an authentication protocol for RFID devices. DST40 is an unbalanced Feistel cipher with a short key length but a high number of rounds.

We find linear approximations for the cipher by looking at certain selections of bits in the plaintext and ciphertext, and how the vectors that represent these selections move through the cipher. We find that with certain choices for these vectors we can achieve as low as 10 active rounds out of 200 rounds total. This only works for keys that meet some requirement, which is a quarter of all possible keys.

However, even with a vulnerable key, this attack does not break the cipher, as an attack that makes use of this linear approximation requires more encryptions for obtaining challenge-response pairs than exhaustive key search would. We do perform an attack using this method on a reduced-round version of the cipher, and find that the attack works for up to 79 rounds of the cipher.

Contents

Chapter 1

Introduction

Radio Frequency IDentification (RFID) is a method of identification which uses electromagnetic fields to communicate [13]. This is done using radio transponders, which transmit or receive data such as IDs and challenge-response messages. A collection of RFID implementations called Passive Keyless Entry and Start (PKES) is used to handle remotely unlocking, locking and starting vehicles.

PKES systems in cars allow one to both unlock and start their car by simply having the key (referred to as key fob) in close proximity to the car and pressing a button on the car itself [14]. This removes the need to take out the key fob for unlocking the car. To ensure that the one trying to unlock or start the car possesses the key fob, the key fob needs to be authenticated. This authentication allows for an immobilizer as well, meaning that the engine will only start when the key fob is authenticated, preventing hot-wiring the car. Under the assumption that the owner has the key, the owner is identified when the key fob is authenticated. Numerous protocols have been used to achieve this, including but not limited to Microchip's KeeLoq [12] and both Digital Signature Transponder (DST) and UICE by Texas Instruments [7, 8]. These protocols use a Message Authentication Code (MAC) to authenticate the key fob. The car and the key fob contain the same cryptographic key, which they use to compute the output of a cipher, given a challenge. This output is the MAC. The car sends the challenge to the key fob, after which they both compute a response. The key fob returns the response and if it is the same as the response computed by the car, it unlocks or starts. The challenge should always be randomly chosen.

In these protocols the car transmits its messages in low frequency, which carries the signal only about 2 meters. This is how the car checks whether the key fob is in proximity. However, this makes the protocol vulnerable to a relay attack, where an attacker can amplify the signal coming from the car so that it still reaches the key when it is out of range, thus unlocking the car [5]. This allows an attacker to unlock and start a car on a drive way with the key fob inside the house, for example. The effectiveness of this attack, however, is decreased by the problem that after driving away with the car, the attacker cannot start the car again without replacing or reconfiguring the PKES system by physically accessing it.

The relay attack is not the only known vulnerability in these systems, though [4, 6, 15, 16]. Problems include a weak cipher vulnerable to algebraic attacks in the case of KeeLoq, keys that are too short and therefore do not provide enough security in the case of DST, and overall poor randomness in the cryptographic keys. Another problem is that some manufacturers keep their ciphers a trade secret, meaning that ciphers and their surrounding protocols are not reviewed before they are implemented and distributed.

This leads to more vulnerabilities in the final product than necessary, which are likely to be found by either researchers or real-world attackers [7].

In this thesis we will be looking at the Digital Signature Transponder from Texas Instruments, specifically the DST40 block cipher. It is used to encrypt given challenges which serve as the Message Authentication Code. This cipher was first reverse engineered and cracked using a brute force approach in 2005 [2]. Later, it was shown that a key could be recovered in seconds using two challenge-response pairs and a 5.4 TB lookup table containing all keys grouped by the response that the cipher produced given a certain challenge [16]. The weakness exploited in these two researches was the short key length of 40 bits. Clearly, it is possible to crack the key because of its length, but it still requires a lot of computation. Therefore, we aim to provide a cryptanalysis of DST40 and see if the cipher itself can be broken to reduce the effective key length. More specifically, we take our research question to be:

Can the DST40 cipher be broken by linear cryptanalysis, such that an advantage is gained over exhaustive key search?

Linear cryptanalysis is a technique to analyse a cipher by finding linear approximations to the effect of the cipher and using those to leak key bits. By constructing linear approximations that hold with a high probability, guesses can be made about the values of key bits. Linear cryptanalysis was introduced in 1992 by Matsui, using it to break the FEAL cipher [9]. Later, Matsui successfully used it to break DES, although the attack still requires too many known plaintexts to be practical [10]. In this thesis we will apply this technique to the DST40 cipher.

As for the relevance of DST40, it was shown that DST40 is still used in modern cars like the Tesla Model S, even though the newer DST80 was developed by Texas Instruments [16]. Since DST80 is based on the same design as DST40 (but with a key length of 80 bits) [15], a linear cryptanalysis of DST40 may also be relevant for DST80.

Chapter 2

Preliminaries

As the concept of symmetric key ciphers plays a large role in this analysis, it will be explained in this chapter.

Symmetric key cryptography refers to encrypted communication where encryption and decryption is done using the same secret key. Encryption and decryption are handled by ciphers, of which there are some different categories. Only block ciphers will be presented here.

The security of a cipher is measured in bits. n bits of security means that the cipher is distinguishable from a random permutation with a success probability of not greater than 2^{-n}. In practice, with n bits of security, 2^n encryptions are needed to find the secret key. Optimally, a cipher has as many bits of security as it has bits in the secret key such that the best possible attack is exhaustive key search, meaning an attacker has to try an encryption with all possible key values to find the correct one.

2.1 Block ciphers

Block ciphers take a block of data (plaintext) with a fixed size, as well as a key with a possibly different fixed size, and combine these in some way to produce an output block (ciphertext). If the cipher repeats the same process of combining blocks and keys, then a single one of such combinations is called a round. Block ciphers that use multiple rounds often change the key in a reversible way as well, which is called a key schedule. Block ciphers always have an inverse, which is used for decryption.

Most important for this thesis is the Feistel structure, because DST40 is built on this design. In one round, a generic Feistel structure divides the input block into two parts (usually of equal length). It takes one of those sides and combines it with the key of that round into some output that is added to the other side of the input block. The two sides are then swapped. This process repeats for some amount of rounds, after which the final output block combined is the ciphertext. Decryption is done by following the same steps, but backwards. So, it swaps the two halves of the encrypted data and then takes one half, combines it with the corresponding round key and adds that to the other half. A typical implementation of one round of a Feistel structure is shown in Figure 2.1. A popular block cipher based on a Feistel structure is DES, which has a 56 bit key and has 16 rounds.

A method one may employ to get information about the key that the cipher uses without knowing it is to make encryptions with carefully chosen differences in the plaintext and see what effect this has on the ciphertext. Some vulnerable block ciphers may give away information about their key in this way. This technique is known as differential

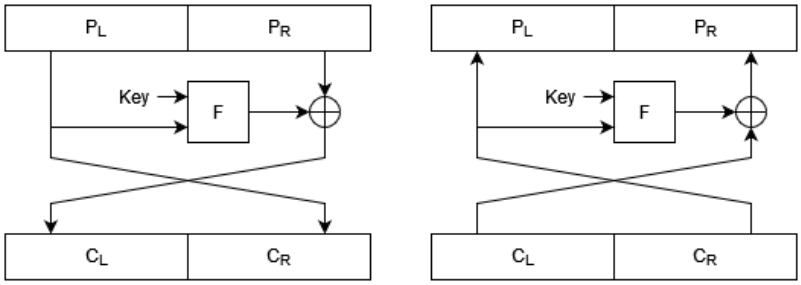

Figure 2.1: One Feistel round of encryption (left) and decryption (right)

cryptanalysis. Another popular method to analyze block ciphers is linear cryptanalysis, which this thesis is about.

2.2 Message authentication codes (MAC)

Message authentication codes may be used to verify that the person or device that is communicated with has knowledge of a secret key and to make sure that the message is not altered. This authentication protocol has a verifier on one side and a prover on the other.

In this case, where there is only one challenge-response pair, the verifier sends a random challenge to the prover and signs the challenge using a secret key. The prover then also signs the challenge using the same secret key and sends it back to the verifier. If the response matches their own signed challenge, the prover's message is authenticated. The security of this protocol depends on the security of the method that is used for signing the challenge. The function for signing the challenge is called the MAC-function.

The output of the MAC-function should always be unpredictable. So, if an attacker has a MAC-response which was computed using the secret key from the prover for a given challenge, then the attacker should not be able to derive the MAC-response for another challenge.

Chapter 3

Design of DST40

Although DST40 is a proprietary cipher that was never publicly released, it was reverse engineered by Bono et al. in 2005 [2]. The cipher is an unbalanced Feistel structure with a key schedule, and a 40-bit block and key length. In one round of this Feistel structure, the plaintext is divided into two parts. The left part is 38 bits long and the right part is 2 bits long. The 38-bit part is combined with the 40-bit round key by applying an F function. The 2-bit output of this F function is added to the 2-bit part of the plaintext using XOR. The two parts are then swapped, such that the 2-bit part becomes the left-most section in the new 38-bit part in the next round.

The plaintext is not divided into parts of equal length like in a regular Feistel structure, which is why it is called unbalanced. The round schematic is depicted in Figure 3.1, and the cipher runs for 200 rounds.

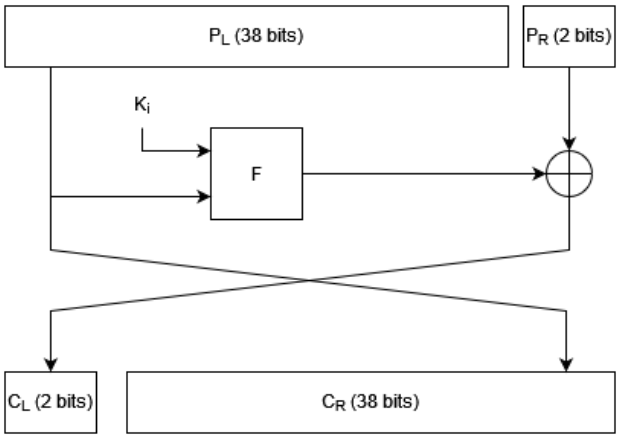

Figure 3.1: Schematic of one DST40 round

The key is updated every 3 rounds, starting on the second round. The key register is a Linear-Feedback Shift Register (LFSR). This LFSR consists of a register in which all bits shift one position to the right each time the register is updated. On a shift, the right-most bit will be added to some other bits using XOR, and will wrap around to become the leftmost bit. In this case, the bits used to XOR the bit wrapping around are k_0, k_2, k_{19}, k_{21}, as seen in Figure 3.2. Note that this is a maximum length LFSR, so any value in the LFSR will only repeat every $2^{40} - 1$ shifts (except for the all-0 state) [2].

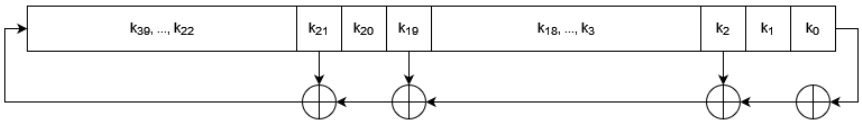

Figure 3.2: Schematic of the key register LFSR

3.1 The F function

The F function takes as input the secret key K_i (with i the round number) and the 38-bit part of the message and produces 2 bits of output. Although [2] suggests that in the implementation of DST40 the 2-bit part of the message is part of the input of F, the given tables show that they have no effect on the output. This makes sense, because the structure may not be invertible otherwise. Hence, for the sake of clarity, we keep them separated from the input here (as illustrated in Figure 3.1). The F function is specified in terms of 7 distinct functions:

- f_a and f_b take 2 key bits and 3 message (state) bits as input and return one bit.
- f_c and f_d take 3 key bits and 2 message bits as input and return one bit.
- f_e takes 3 key bits and one message bit as input and returns one bit. This is the function that was specified to take two message bits, but we only look at the one bit that has an effect on the output.
- g takes 4 bits of input from the f sub-functions and returns one bit.
- h takes 4 bits of input from the g sub-functions and returns 2 bits. The output of this function is the output of the F function.

The sub-functions are arranged as shown in Figure 3.3. The exact message and key bits that are given to the f sub-functions are described in the paper that covered its first analysis [2], in table form. We expand on this by providing the algebraic normal form (ANF) of the functions. Given that the functions take as input either $x_4, \ldots, x_0$ or $x_3, \ldots, x_0$, the ANF of each function looks like

$$\begin{aligned}
h &= \begin{pmatrix} x_0x_1 + x_0x_3 + x_1x_2 + x_2x_3 + x_2 + x_3 \\ x_0x_2 + x_0x_3 + x_1x_2 + x_1x_3 + x_1 + x_2 \end{pmatrix} \\
g &= x_0x_1 + x_0x_2 + x_0 + x_1x_3 + x_1 + x_2x_3 \\
f_a &= x_0x_1x_2 + x_0x_2x_3 + x_0x_3 + x_0x_4 + x_0 + x_1x_2x_4 + x_1x_3 + \\
&\quad x_1x_4 + x_2x_3x_4 + x_2x_3 + x_2x_4 + x_2 + x_3 \\
f_b &= x_0x_2 + x_0x_4 + x_0 + x_1x_2 + x_1x_4 + x_1 + x_2x_3 + x_2x_4 + \\
&\quad x_3x_4 + x_4 \\
f_c &= x_0x_1x_4 + x_0x_1 + x_0x_3 + x_1x_2x_4 + x_1x_2 + x_1x_3 + x_1 + \\
&\quad x_2x_3 + x_2 + x_3x_4 + x_3 \\
f_d &= x_0x_2 + x_0x_3 + x_0x_4 + x_0 + x_1x_2 + x_1x_3 + x_1x_4 + x_2 \\
f_e &= x_0x_2 + x_0x_3 + x_0 + x_1x_2 + x_1x_3 + x_3
\end{aligned} \tag{3.1}$$

As seen in Figure 3.3, each function appears multiple times. The f functions are numbered top to bottom as $f_1, \ldots, f_{16}$, where f_{15} and f_{16} are the defined by f_e, and the g functions

are numbered top to bottom as $g_1, \ldots, g_4$. Note that the definition of h is a vector since h has two output bits. The top row represents the right bit and the bottom row the left bit. The ANF representations of all functions are given in A.4.

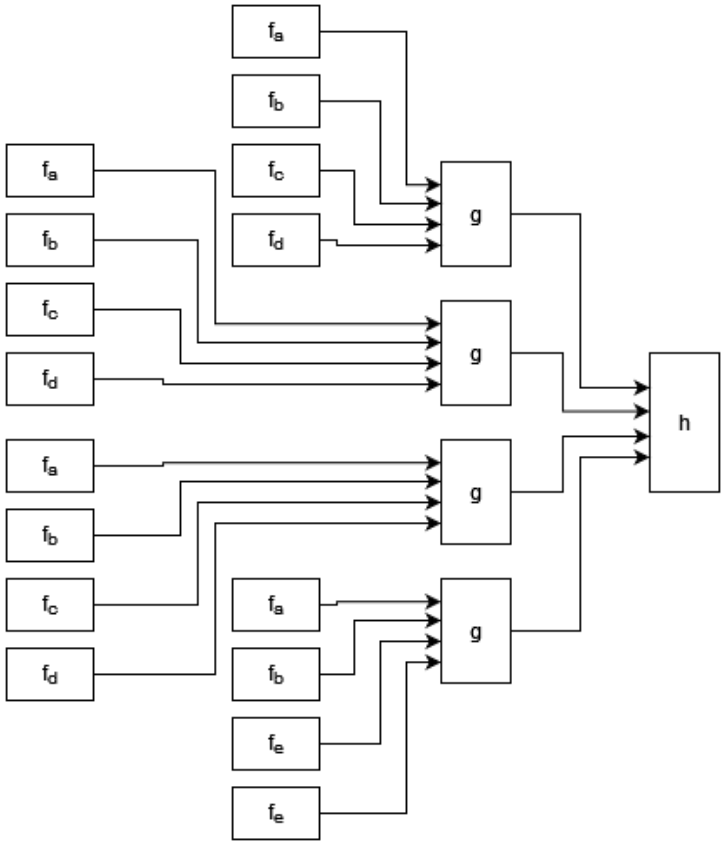

Figure 3.3: The internals of the F function

The F function's domain is 78 bits in size, consisting of 40 key bits and 38 message bits, and its image is 2 bits. This means that the image gives 2 bits of information about the input of the function, which is not meaningful.

Noteworthy is that the function's output is not uniform. If we generate a random key and message and compute the result, we get the results in Table 3.1. The same results

Output value	Occurrences (%)
00	25%
01	31%
10	19%
11	25%

Table 3.1: Number of occurrences of the output values of F

show up when the key is kept constant and the message is chosen randomly. However, this is not directly problematic for this cipher, because such a small difference fades over a span of 200 rounds.

Chapter 4

Linear Cryptanalysis

Linear cryptanalysis is a technique that involves trying to construct a linear approximation to a cipher. It was first successfully applied by M. Matsui to find attacks against FEAL and DES, with the ability to find a key with less encryptions than exhaustive key search [9, 10]. In practice it comes down to finding so called masks that form linear approximations that hold with a probability unequal to $\frac{1}{2}$. A linear approximation, in this context, is an addition of certain bit positions in the plaintext, ciphertext and key such that:

$$\bigoplus_{i\in\{1,\ldots,40\}} P_i \oplus \bigoplus_{j\in\{1,\ldots,40\}} C_i = \bigoplus_{k\in\{1,\ldots,40\}} K_i. \tag{4.1}$$

with P the plaintext, C the ciphertext and K the key, and the indices being bit positions. The chosen bit positions can be selected using masks. Both the mask and the bit string can be seen as vectors, and the effect of a mask as a matrix multiplication:

$$\begin{pmatrix} m_1 \\ \vdots \\ m_n \end{pmatrix}^T \cdot \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix} = \begin{pmatrix} m_1 & \ldots & m_n \end{pmatrix} \cdot \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix} \tag{4.2}$$

With a^T being the transposition of a. In practice, a 1 in the mask indicates that the bit at the same index in the data is "active". For example

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}^T \cdot \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} = 1 \cdot 0 + 0 \cdot 1 + 0 \cdot 1 + 1 \cdot 1 = 1$$

The first and last bit are selected to be added. Using the concept of masks, equation 4.1 can then be rewritten to

$$M_1^T \cdot P \oplus M_2^T \cdot C = M_3^T \cdot K \tag{4.3}$$

The goal is to find such masks for linear approximations that have a high correlation with propagation through the cipher rounds. In general, correlation is a measure of dependence between data with a coefficient $\rho \in [-1, 1]$, where 0 means that the data is independent, -1 means negative correlation and 1 means positive correlation. Here, the correlation coefficient is a measure for the amount of inputs and outputs for which the applied masks produce the same result. A correlation far from 0 corresponds to equation 4.1 holding with a probability far from $\frac{1}{2}$. In short, we need to find M_1, M_2 and M_3 such that equation 4.3 holds for either more than half or less than half of a set of many different pairs of P and C.

4.1 Correlation in the F function

We can describe the correlation of masks on the input and output of a function in a correlation matrix. If the input length of the function is n, and the output length m, then the matrix will be $2^m \times 2^n$ because there are 2^n input masks and 2^m output masks. The indices of the rows are then the numerical representations of the output masks, and the indices of the columns are the representations of the input masks (eg. for a 4-bit mask, index 3 represents the mask $(0, 0, 1, 1)$). All values in the matrix are then correlation coefficients, with 1 meaning that the masks will always give the same result when applied to the input and output of the function, -1 means that the results are always opposite and 0 means that the results are equal half of the time. A correlation matrix is a normalization of a Walsh transform [1], such that

$$C_f = 2^{-n} W_f \tag{4.4}$$

The Walsh transform is a $2^m \times 2^n$ matrix [1] that is defined at index (u, v) as

$$W_{u,v} = \sum_{x \in \mathbb{F}_2^n} (-1)^{u^T \cdot f(x) + v^T \cdot x} \tag{4.5}$$

with $u, v, x \in \mathbb{F}_2^n$. This is the vector space of binary vectors with length n, so the products in the equation are inner products.

Using this method, we aim to find linear approximations to the F function with high correlation. This will help to derive the correlations of linear approximations over a full round of DST40, which we can use in trails for the full cipher. We can calculate the Walsh matrices for each sub-function of the F function, and therefore also their correlation matrices. The correlation of a trail through this function is the product of correlations through the sub-functions. We move backwards from the last function, h:

$$C_h = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & -\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & -\frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \end{pmatrix} \tag{4.6}$$

The columns are the input masks 0 to 15 and the rows are the output masks 0 to 3. The top row in any such Walsh transformation or correlation matrix is trivial, because the row with index 0 represents the mask that selects nothing from the output, meaning that nothing is selected from the input either. Therefore, the only non-zero value in the top row is the 1 in the first column. In the rest of the matrix we can see a couple of non-zero correlations. Recall that h takes its input from four g functions, meaning that bits with a value of 1 in the input mask represent active output from the corresponding g function. From C_h it becomes clear that there is no input mask that selects from only one g function, because the non-zero correlations occur at the input masks $(0, 0, 1, 1), (0, 1, 0, 1), (0, 1, 1, 0), (1, 0, 0, 1), (1, 0, 1, 0), (1, 1, 0, 0)$. However, all output masks have only 4 out of these 6 possible input masks. Then for g:

$$C_g = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & -\frac{1}{2} & 0 & 0 & 0 \end{pmatrix} \tag{4.7}$$

For g it also holds that no input mask with a non-zero correlation selects from only one f function. But, the combinations $(0, 1, 1, 0)$ and $(1, 0, 0, 1)$ now do not occur. The full correlation matrices of f_a to f_e, which include masks over key bits, can be found in A.2.

The functions f_a to f_e depend on state bits and key bits. The key bits can be treated as constant which allows the correlation matrices of these functions to be simplified. The key is not altered by the Feistel structure after all, only by the key schedule. If masks for certain fixed key values are highly correlated, those key bits may be a passable guess for the linear approximation of the cipher. f_a and f_b depend on two key bits and three state bits, so we have four 2×8 correlation matrices. We look at the non-trivial rows (so the output mask is 1) for each key value and set out the values of the key bits against the values of the input mask in Table 4.1. Because all correlation values in the first column are non-zero, we can conclude that f_a and f_b are unbalanced. An unbalanced function has non-zero correlations for combinations of all-zero input masks and non-zero output masks. These values are interesting, because they are a "dead end" in a trail, since a mask stops propagating through the cipher when it becomes all-zero.

function	key bits	000	001	010	011	100	101	110	111
f_a	00	$-\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$-\frac{1}{4}$	$\frac{1}{4}$	$\frac{3}{4}$	$-\frac{1}{4}$	$\frac{1}{4}$
	01	$\frac{1}{4}$	$-\frac{1}{4}$	$-\frac{3}{4}$	$-\frac{1}{4}$	$-\frac{1}{4}$	$\frac{1}{4}$	$-\frac{1}{4}$	$\frac{1}{4}$
	10	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{3}{4}$	$-\frac{1}{4}$	$-\frac{1}{4}$	$-\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
	11	$-\frac{1}{4}$	$-\frac{1}{4}$	$-\frac{1}{4}$	$-\frac{1}{4}$	$\frac{1}{4}$	$-\frac{3}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
f_b	00	$\frac{1}{2}$	0	0	$\frac{1}{2}$	$-\frac{1}{2}$	0	0	$\frac{1}{2}$
	01	$-\frac{1}{2}$	0	0	$\frac{1}{2}$	$\frac{1}{2}$	0	0	$\frac{1}{2}$
	10	$-\frac{1}{2}$	0	0	$\frac{1}{2}$	$-\frac{1}{2}$	0	0	$-\frac{1}{2}$
	00	$\frac{1}{2}$	0	0	$\frac{1}{2}$	$\frac{1}{2}$	0	0	$-\frac{1}{2}$

Table 4.1: Correlation coefficients of input masks for f_a and f_b for fixed key bits

The functions f_c and f_d depend on three key bits and two state bits, so we get eight 2×4 matrices. f_e depends on only one state bit, so it has eight 2×2 matrices. We again use a table form to give the correlation coefficients in Table 4.2. The tables show that f_c and f_e are unbalanced as well, for certain key bits. Given these correlation values, the correlation of an approximation for the F function can be determined.

function	key bits	00	01	10	11
f_c	000	$\frac{1}{2}$	$-\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
	001	$-\frac{1}{2}$	$-\frac{1}{2}$	$-\frac{1}{2}$	$\frac{1}{2}$
	010	$-\frac{1}{2}$	$-\frac{1}{2}$	$\frac{1}{2}$	$-\frac{1}{2}$
	011	$\frac{1}{2}$	$-\frac{1}{2}$	$-\frac{1}{2}$	$-\frac{1}{2}$
	100	0	0	1	0
	101	0	0	-1	0
	110	0	1	0	0
	111	0	1	0	0
f_d	000	0	1	0	0
	001	0	0	-1	0
	010	0	0	1	0
	011	0	-1	0	0
	100	0	0	1	0
	101	0	-1	0	0
	110	0	1	0	0
	111	0	0	-1	0

function	key bits	0	1
f_e	000	0	1
	001	0	1
	010	1	0
	011	-1	0
	100	-1	0
	101	1	0
	110	0	-1
	111	0	-1

Table 4.2: Correlation coefficients of input masks for f_c, f_d and f_e for fixed key bits

4.2 Mask propagation through DST40

When looking at the propagation of masks through a cipher round, we start from the output of a cipher round and work our way up to the input. When a mask propagates through XOR, it does not change. Consider a mask m, and the equation

$$x \oplus y = z$$

Then a mask operating on z will also operate on x and y. Because of the distributivity of matrix multiplication, we can see that:

$$m^T \cdot z = m^T \cdot (x \oplus y) = m^T \cdot x \oplus m^T \cdot y$$

When a mask propagates through duplication, there will be two different masks on the other end. m_1 and m_2 are masks, and

$$x \mapsto (x_1, x_2), \quad x_1 = x_2$$

Then an addition of the masks exists before the duplication:

$$m_1^T \cdot x_1 \oplus m_2^T \cdot x_2 = m_1^T \cdot x \oplus m_2^T \cdot x = (m_1^T \oplus m_2^T) \cdot x = (m_1 \oplus m_2)^T \cdot x$$

Using these principles, we can set a mask at the output of a round of DST40 and derive what the input mask looks like. This allows us to compute the correlation of a linear approximation over a single round. Consider $A||C||B$ as the output mask, where $x||y$ is the concatenation. Suppose that at the XOR we have $x \oplus y = z$, and A is the mask over z. Then the masks for x and y are also A:

$$A^T \cdot z = A^T \cdot (x \oplus y) = A^T \cdot x \oplus A^T \cdot y$$

If we have $x \mapsto (x, x)$ at the duplication, the mask $C||B$ at the output of the round and the mask $E||D$ at the input of the F function. Then the mask over x before the duplication is $(C \oplus E)||(B \oplus D)$:

$$\begin{aligned}(C||B)^T \cdot x \oplus (E||D)^T \cdot x &= ((C||B)^T \oplus (E||D)^T) \cdot x \\ &= ((C||B) \oplus (E||D))^T \cdot x \\ &= ((C \oplus E)||(B \oplus D))^T \cdot x\end{aligned}$$

The input mask is then $(C \oplus E)||(B \oplus D)||A$ and the output mask is $A||C||B$, as visualised in Figure 4.1. We denote the function F with the input key K as F_K. A, C and E are 2 bits long and B and D are 36 bits long. We note the correlation over the F function, which is equal to that of the function F_K. We can conclude that the correlation of this linear approximation is given by

$$C_{F_K}((C \oplus E)||(B \oplus D)||A,\ A||C||B) = C_{F_K}(E||D,\ A) \tag{4.8}$$

This is the correlation matrix of the (n, m)-function F_K, with $n = 38$ and $m = 2$ in the case of DST40's function. This will be further defined in 4.1. The correlation of a trail of single round linear approximations over n rounds is then

$$C_{total} = \prod_{i=1}^{n} C_{F_{K_i}}(E_i||D_i, A_i) \tag{4.9}$$

with A_i, D_i and E_i masks for round i.

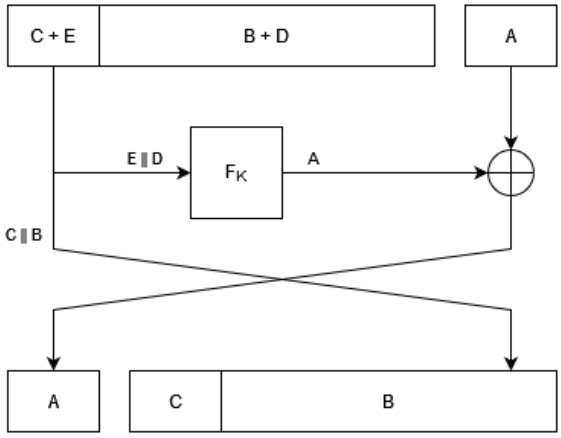

Figure 4.1: Mask propagation in DST40

4.3 Linear approximations

First, we look for good linear approximations over a single round. We write the masks as bit strings. There is one trivial approximation in which the output mask is 0, leading to $C_{F_K}(0,0) = 1$. For the other values of the output mask A from Figure 4.1 the approximation has a correlation of less than 1. Take the mask A to be 11, and take any of the four choices for the input mask, for example 1100. Then the correlation of this approximation is

$$C_h(1100, 11) = \frac{1}{2}$$

Then there are two g functions with a non-zero output mask: g_1 and g_2. For both of these we choose input from f_c and f_d, such that

$$C_{g_1}(0011, 1) = C_{g_2}(0011, 1) = -\frac{1}{2}$$

For these f_c and f_d functions (f_3, f_4, f_7, f_8) the correlations can be chosen to be 1 (or -1) for certain key bits. For example, choosing the key bits to be 110 and the input mask to be 01 for all of these functions leads to

$$C_{f_c}(01, 1) = C_{f_d}(01, 1) = 1$$

The correlation of the linear approximation over F_K is then

$$C_{F_K}(11110000, 11) = \frac{1}{2} \cdot (-\frac{1}{2} \cdot 1 \cdot 1) \cdot (-\frac{1}{2} \cdot 1 \cdot 1) = 2^{-3} \tag{4.10}$$

The mask $E||D = 11110000$ is derived from the definitions of f_3, f_4, f_7, f_8 with the input mask 01 for each of them. This is a linear approximation with a maximal correlation, since it is composed of the approximations with maximal correlations of each component function. Approximations with correlation 2^{-3} similarly exist for $A = 01$ and $A = 10$. This is a large value, but has an exponential increase with each round. For instance, with on average a quarter of rounds for which $C_{F_K}(0,0) = 1$ (since $A = 00$ about 25% of times on average), a linear approximation would have correlation

$$C_n = (2^{-3})^{150} \cdot 1^{50} = 2^{-450} \tag{4.11}$$

So, a different approach is needed.

4.4 Full cipher trails

With the knowledge of mask propagation and correlation in a single round, we can extend to the full cipher. The 200 rounds can be divided into active and inactive rounds. When the mask A from Figure 4.1 is 0, such that the mask $E||D$ is also 0 with $C_{F_K}(E||D, A) = C_{F_K}(0, 0) = 1$, the F function has no effect on the mask over the state, so the round is called inactive. When A is non-zero, the round is called active.

The goal is to find a mask at the start of the first round and one at the end of the last round that have high correlation. Since the cipher has 200 rounds, finding any masks with (relatively) high correlation requires approximations with high correlation for active rounds, as well as a lot of inactive rounds.

Since A is only 2 bits long, analyzing the three non-zero values is quite trivial. We again write the masks as bit strings. Take $A = 11$. Then we have $C_h(0011, 11) = \frac{1}{2}$. For the g and f functions we use the numbering from Chapter 3 and [2]. The active g functions are then g_3 and g_4, for which we choose different trails:

g_3 : $C_{g_3}(1100, 1) = -\frac{1}{2}$ leads to an active f_a (f_9) and f_b (f_{10}). For these we choose $C_{f_a}(000, 1) = \pm\frac{1}{4}$ and $C_{f_b}(000, 1) = \pm\frac{1}{2}$.

g_4 : $C_{g_4}(0011, 1)) = \frac{1}{2}$ leads to two active f_e's (f_{15} and f_{16}). For both we choose $C_{f_e}(0, 1) = \pm 1$.

We come back later to the negative values as they will play a role, but for now we keep the result positive. Note that this has an input mask equal to zero, while the output mask is non-zero, so this is a previously mentioned dead end. The correlation is

$$C_{F_K}(E||D, A) = C_{F_K}(0, 11) = \frac{1}{2}(-\frac{1}{2} \cdot -\frac{1}{4} \cdot \frac{1}{2})(\frac{1}{2} \cdot 1 \cdot 1) = 2^{-6} \tag{4.12}$$

With this, the absolute minimum number of active rounds can be achieved because, since $E||D$ is always all zero, the mask is never changed by the F function. This implies that B and C do not change in a single round, which also holds for A. The state is shifted by two bits each round, so if the masks are never changed in a single round, the mask is in the same position after every $40/2 = 20$ rounds of the cipher. By taking $A = 11$ and B and C all zero, there is only one active round every 20 rounds for 10 active rounds total. That one active round has correlation 2^{-6}, so over the complete 200 round cipher, there is a correlation such that

$$\begin{aligned} X &= 00000000000000000000000000000000000011 \\ Y &= 00000000000000000000000000000000000011 \\ C_{total}(X, Y) &= \prod_{i=1}^{200} C_{F_{K_i}}(E_i||D_i, A_i) = (2^{-6})^{10} \cdot 1^{190} = 2^{-60} \end{aligned} \tag{4.13}$$

Similarly, for the other possible mask values for A we find

$$\begin{aligned} C_{F_K}(E||D, A) = C_{F_K}(0, 01) &= -\frac{1}{2}(-\frac{1}{2} \cdot \frac{1}{4} \cdot \frac{1}{2})(\frac{1}{2} \cdot 1 \cdot 1) = 2^{-6} \\ C_{F_K}(E||D, A) = C_{F_K}(0, 10) &= \frac{1}{2}(-\frac{1}{2} \cdot -\frac{1}{4} \cdot \frac{1}{2})(\frac{1}{2} \cdot 1 \cdot 1) = 2^{-6} \end{aligned} \tag{4.14}$$

So, a round correlation of $\pm 2^{-6}$ can be achieved for all non-zero output masks, as well as a correlation of $\pm 2^{-60}$ for the full cipher approximation.

4.5 Key bit parity

Lastly, we express the sign of the correlation values for the f_a, f_b and f_e functions. As can be seen in the correlation tables, the sign is determined by the key bits. Hence, the correlation over the functions can be written as a function of the key bits. Let k_0, k_1, k_2 denote single key bits used in the function. Then

$$\begin{aligned} C_{f_a}(000,1) &= \frac{1}{4} \cdot (-1)^{k_0+k_1+1} \\ C_{f_b}(000,1) &= \frac{1}{2} \cdot (-1)^{k_0+k_1} \\ C_{f_e}(0,1) &= 1 \cdot (-1)^{k_0+k_2} \qquad k_2||k_1||k_0 \notin \{000,001,110,111\} \end{aligned} \tag{4.15}$$

By making the round correlations dependant on these functions, only the signs of the correlations depend on key bits. So, knowledge of the key bits reveals the sign of the correlation value. Conversely, knowledge of the sign of the correlation can give some information about the key bits. Since each function depends on 2 bits, only the parity can be derived. That means that it can be derived whether $k_0 \oplus k_1 = 0$ or $k_0 \oplus k_1 = 1$, for example. By looking up the exact function input, we can determine the key bit positions as well (except we use a 0-based index here) [2]. The total round correlation for each non-zero mask, as a function of key bits, is then

$$\begin{aligned} C_f(0,01) &= \frac{1}{2}(-\frac{1}{2} \cdot C_{f_1}(000,1) \cdot C_{f_2}(000,1))(\frac{1}{2} \cdot C_{f_{15}}(0,1) \cdot C_{f_{16}}(0,1)) \\ &= -2^{-6} \cdot (-1)^{k_{39}+k_{38}+k_{31}+k_{30}+k_{17}+k_{16}+k_1+k_0+1} \\ C_f(0,10) &= \frac{1}{2}(-\frac{1}{2} \cdot C_{f_5}(000,1) \cdot C_{f_6}(000,1))(\frac{1}{2} \cdot C_{f_{15}}(0,1) \cdot C_{f_{16}}(0,1)) \\ &= -2^{-6} \cdot (-1)^{k_{37}+k_{36}+k_{29}+k_{28}+k_{17}+k_{16}+k_1+k_0+1} \\ C_f(0,11) &= \frac{1}{2}(-\frac{1}{2} \cdot C_{f_9}(000,1) \cdot C_{f_{10}}(000,1))(\frac{1}{2} \cdot C_{f_{15}}(0,1) \cdot C_{f_{16}}(0,1)) \\ &= -2^{-6} \cdot (-1)^{k_{35}+k_{34}+k_{27}+k_{26}+k_{17}+k_{16}+k_1+k_0+1} \end{aligned} \tag{4.16}$$

Note that we make different choices of mask trails for different output masks, because that way more different key bits are covered. We only change the choice of one of the active g functions, so the correlation is still 2^{-6}. When the sum of the key bits is 0, -2^6 is multiplied by -1 due to the $+1$ term in the exponent, so then the correlation is positive.

Of course, these additions key bit positions can be seen as masks, which brings us back to the basic premise of linear cryptanalysis, as we now have all the values for the equation

$$M_1^T \cdot P \oplus M_2^T \cdot C = M_3^T \cdot K$$

along with three combinations of M_1, M_2 and M_3 such that the equation holds more than or less than 50% of the time for the full cipher based on the sign of the correlation.

4.6 Key restrictions

However, for the function f_e we noted that with key bits k_0, k_1, k_2 it holds that $C_{f_e}(0,1) = 1 \cdot (-1)^{k_0+k_2}$ with the restriction that $k_2||k_1||k_0 \notin \{000,001,110,111\}$. This is equivalent to $k_1 \neq k_2$. This means that the linear approximation given by the trail we constructed only has a correlation of $\pm 2^{-6}$ if the requirement that $k_{17} \neq k_9 \wedge k_{16} \neq k_8$ is met. Otherwise, if either $k_{17} = k_9$ or $k_{16} = k_8$, the correlation will be 0. This requirement is met by $2^{40} \cdot \frac{1}{2} \cdot \frac{1}{2} = 2^{38}$ keys, a quarter of the key space.

Chapter 5

A theoretical attack

Using the masks and correlations given in the previous chapter, a theoretical attack on the DST40 cipher can be described. The attack is theoretical because the correlation of the trail we found is only 2^{-60}, whereas the probability of guessing a key is 2^{-40}. Nonetheless, the linear approximation that was found in the previous chapter is extendable to an attack.

5.1 Attack description

In the previous chapter we found three sets of three masks that fit the equation

$$M_1^T \cdot P \oplus M_2^T \cdot C = M_3^T \cdot K$$

such that it holds more than half of the time with positive correlation and less than half of the time with negative correlation. The masks are fully written out in Table 5.1 as bit strings. We will assume that the key satisfies $k_{17} \neq k_9 \wedge k_{16} \neq k_8$.

M_1	00000000000000000000000000000000000000000001
M_2	00000000000000000000000000000000000000000001
M_3	11000000110000000000001100000000000000011
M_1	00000000000000000000000000000000000000000010
M_2	00000000000000000000000000000000000000000010
M_3	00110000001100000000001100000000000000011
M_1	00000000000000000000000000000000000000000011
M_2	00000000000000000000000000000000000000000011
M_3	00001100000011000000001100000000000000011

Table 5.1: Three sets of masks that fit equation 4.3 with a correlation of 2^{-60}

The idea of an attack using these masks is to generate many plaintext-ciphertext pairs and check for each pair whether $M_1^T \cdot P_i = M_2^T \cdot C_i$. If this happens in more than half of the cases, we guess that the sign of our mask correlation is positive, and if it happens in less than half the cases we guess that the sign is negative. This gives away $M_3^T \cdot K$, or the parity of the selected key bits. After this, we perform an exhaustive key search, but we can ignore all the keys that do not have the same parity on the selected bits that we found. So, the key space is effectively reduced. Because we have three sets of masks, with different selected key bits, we can reduce the effective key space further by repeating the process. This corresponds to Matsui's first algorithm for linear cryptanalysis on DES [10]. Each equation reduces the key space for the exhaustive key search by one bit, meaning

that with three equations the key space is reduced to 2^{37}. We present an algorithm for the attack in this case.

Algorithm 1 The linear approximation attack

$N \leftarrow$ the number of plaintext-ciphertext pairs
for $P_i \in \{P_1, \ldots, P_N\}, C_i \in \{C_1, \ldots, C_N\}$ **do**
    $P_i \leftarrow$ random 40-bit challenge
    $C_i \leftarrow \text{Enc}_K(P_i)$ ▷ Enc_K here is DST40 encryption
end for
$T \leftarrow 0$
$M_1, M_2, M_3 \leftarrow$ masks found using linear cryptanalysis
for $i \in \{1, \ldots, N\}$ **do**
    if $M_1^T \cdot P_i = M_2^T \cdot C_i$ **then**
        $T \leftarrow T + 1$
    end if
end for
if $T > N/2$ **then**
    $guess \leftarrow 1$ ▷ guess $M_3^T \cdot K = 1$
else
    $guess \leftarrow 0$ ▷ guess $M_3^T \cdot K = 0$
end if

The more plaintext-ciphertext pairs we have, the higher the chance of success. Matsui shows that the success rate of this algorithm is given by

$$\int_{-2\sqrt{N}|p-\frac{1}{2}|}^{\infty} \frac{1}{\sqrt{2\pi}} \exp \frac{x^2}{2}\, dx \tag{5.1}$$

[10], with p the probability that the linear equation holds. This $|p - \frac{1}{2}|$ corresponds to $|\frac{1}{2}C_{total}|$, which means that for a successful guess for the linear approximation 99.8% of times, we need $N = 2|p - \frac{1}{2}|^{-2} = 8|C_{total}|^{-2}$ plaintext-ciphertext pairs.

5.2 Reduced-round variants

Clearly, an amount of plaintext-ciphertext pairs in the order of $8 \cdot |2^{-60}|^{-2} = 2^{123}$ for a good success probability is not feasible, because there are only 2^{40} plaintext-ciphertext pairs with a block length of 40 bits, and it is a lot more expensive than exhaustive key search. However, we can apply the attack to a variant of DST40 with less rounds.

Suppose we have a DST40 cipher with only 20 rounds. We use the same sets of masks given in Table 5.1, since the number of rounds is only divided by 10. For a good success probability we take $N = 8|C_{total}|^{-2} = 2^{15}$ plaintext-ciphertext pairs and apply Algorithm 1 for each set of masks. A Python implementation is used for executing Algorithm 1 for each linear approximation, which is given in A.3.

The attack will be effective as long as $N < 2^{39}$, because with regular exhaustive key search the key will be found in 2^{39} encryptions on average. Rewriting results in $|C_{total}| > 2^{-18}$ which corresponds to 3 active rounds, since 1 in every 20 rounds has a correlation of 2^{-6}, and the other 19 have correlation 1. So, it works for at least $3 \cdot 20 = 60$ rounds, but the following 19 rounds will have a correlation of 1 as well. Therefore, the linear approximation attack works on this cipher if it has up to 79 rounds.

5.3 Key search

Using Algorithm 1 for each linear approximation we find three values for the key bit sums, which leaves an exhaustive key search to be done over 2^{37} keys. Preferably, this is implemented in a faster language than Python, like C++. The existing DST40 C++ implementation can easily be extended to run inside a for-loop. This loop can then be parallelized using OpenMP. This is done by simply adding the line:

```
#pragma omp parallel for
```

and compiling with g++ and the flag `-fopenmp`. This leads to a parallel implementation that can run at least 10 million 20-round DST40 encryptions per second on a laptop with an AMD Ryzen 5 3550H at 3.4 GHz. Hence, running all 2^{37} encryptions would take approximately 3.7 hours on a machine with these properties.

5.4 Dealing with truncated output

In Chapter 3 it was described that only the 24 right-most bits of the output register are used for the MAC response. This means that if an attacker is eavesdropping the communication, the output of the cipher will look like 24 bits, rather than 40. This is why two challenge response pairs are needed in the exhaustive key search phase after Algorithm 1. One pair will limit the key candidates to 2^{16} possibilities, and the second pair will narrow that down to one key.

This truncation, however, is no problem for the linear cryptanalysis method in this specific case. The ciphertext masks can be chosen such that they select none of the left-most 16 bits of the output, so they are not relevant in the attack before the exhautive key search phase.

Chapter 6

Related Work

This research takes the concept of linear cryptanalysis and applies it to the DST40 block cipher. This chapter discusses some related work on both of these topics. Note that so far, no papers on cryptanalysis of the DST40 cipher itself have been published.

6.1 DST40 analysis

DST40 was first reverse engineered in 2005 using a black-box approach [2]. The team used an Evaluation kit from the same manufacturer as the DST chips, Texas Instruments, to program the chips with chosen keys and messages. Using this method, they were able to completely reverse engineer the cipher. They then implemented the cipher themselves and were able to crack any 40-bit key by simply brute-forcing it using 16 FPGA's, which took less than an hour. Lastly, they simulated a DST device using their implementation and a cracked key.

In 2019 another group analyzed DST chips and optimized the attack to only take seconds [16]. They stored a lookup table with all combinations of plaintext and ciphertext pairs on a 6 TB hard drive. After setting up communication with the DST chip themselves using an Arduino, they implemented a proof of concept and demonstrated it on the Tesla Model S. A theoretical car-only attack was also proposed, but deemed infeasible.

6.2 Linear cryptanalysis

After the differential cryptanalysis method, linear cryptanalysis was first applied in 1992 to successfully break the FEAL cipher [9], and later to "break" DES as well [10] [11]. However, these papers try to find indices such that the equation 4.1 holds with a probability unequal to $\frac{1}{2}$, or $\epsilon = |p - \frac{1}{2}| > 0$, without describing the process in terms of masks and correlations. Just like DST40, DES is a block cipher based off of a Feistel structure, except DES is balanced and has a longer key length.

Some time later, a method using masks and correlations was developed for the design of block ciphers [3]. This is the method that is used in this thesis.

Chapter 7

Conclusions

DST40 serves as a MAC-function in a challenge-response protocol for Keyless Entry and Start systems in vehicles. It is a block cipher based on a Feistel structure with a block and key length of 40 bits. The two sides of the Feistel structure are unequal in length in DST40, with the left side being 38 bits long and the right side being 2 bits long. The key schedule is a Linear-Feedback Shift Register that updates every three rounds. The cipher runs for 200 rounds.

By applying the linear cryptanalysis technique we have found masks at the input and output of the F function, that form a linear approximation over one cipher round with a correlation of 2^{-3}. Moreover, we have found masks for which a dead end occurs in the trail with a correlation of 2^{-6}, meaning that the input mask of a round can be all-zero while the output mask is not. This leads to a linear approximation with a non-zero correlation, in which the mask is unaltered by the F function. With this, a trail can be constructed with only one active round in every twenty. For the full 200-round cipher this implies that there exist linear approximations with correlation 2^{-60}. However, this only works for a quarter of all possible keys, due to a restriction of some key bit values in the trail of the linear approximation.

We conclude that this does not allow for a feasible attack on the cipher, because more than 2^{120} challenge-response pairs are needed to perform an attack on the cipher using a linear approximation, which is far too much to be feasible. And even if that were feasible, only 2^{40} challenge-response pairs exist due to the 40-bit block length of the cipher.

Nonetheless, the results do show the possibility of an attack, so we demonstrate the linear approximation attack on a reduced-round variant. For a 20-round version of DST40, we have three linear approximations with a correlation of 2^{-6} each. We set a fixed key and generate 2^{15} challenge-response pairs. Using these linear approximations, we can derive the sum of three sets of key bits. The key space can then be limited to keys which satisfy the three sums of bit positions, effectively reducing the key space to 2^{37}. We show that the linear approximation attack works for the DST40 cipher with up to 79 rounds. Lastly, we show that brute-forcing the key in the reduced key space takes under 4 hours with very limited hardware.

Bibliography

[1] P. Guillot B. Dravie J. Parriaux. "Matrix representations of vectorial boolean functions and eigenanalysis". In: *HAL* (2016).

[2] SC. Bono. "Security Analysis of a Cryptographically-Enabled RFID Device". In: *14th USENIX Security Symposium* (2005).

[3] J. Daemen. "Cipher and Hash Function Design Strategies based on linear and differential cryptanalysis". PhD thesis. 1995.

[4] T. Eisenbarth. *Physical Cryptanalysis of KeeLoq Code Hopping Applications.*

[5] A. Francillon. *Relay Attacks on Passive Keyless Entry and Start Systems in Modern Cars.*

[6] S. Indesteege. *A Practical Attack on KeeLoq.*

[7] U. Kaiser. "Digital Signature Transponder". In: *RFID Security* (2008).

[8] U. Kaiser. "UICE: A High-Performance Cryptographic Module for SoC and RFID Applications". In: *Cryptology ePrint Archive, Paper 2007/258* (2007).

[9] A. Yamagishi M. Matsui. "A new method for known plaintext attack of FEAL cipher". In: *Advances in Cryptology* (1992).

[10] M. Matsui. "Linear cryptanalysis method for DES cipher". In: *Advances in Cryptology* (1993).

[11] M. Matsui. "The first experimental cryptanalysis of the data encryption standard". In: *Advances in Cryptology* (1994).

[12] Microchip. *An Introduction to KeeLoq Code Hopping.*

[13] *Radio-frequency identification.* https://en.wikipedia.org/wiki/Radio-frequency_identification.

[14] *Remote Keyless System.* https://en.wikipedia.org/wiki/Remote_keyless_system.

[15] L. Wouters. "Dismantling DST80-based Immobiliser Systems". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020 (2020).

[16] L. Wouters. "Passive Keyless Entry and Start Systems in Modern Supercars". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2019 (2019).

Appendix A

Appendix

A.1 DST40 code

The DST40 software implementation in C++. The functions are wrapped in a class which is prepared for C linkage. This is compiled to a shared library, which allows the C++ implementation of the DST40 class to be used in (for example) a Python script.

```
#include "f-boxes.cpp"

using namespace std;

typedef unsigned long long ull;

class DST40 {
    public:
        void round_key(bitset<SIZE>& key, int round_number);
        void feistel_round(bitset<SIZE>& x, bitset<SIZE>& key);
        ull dst40(ull x, ull key, int rounds);
};

void DST40::round_key(bitset<SIZE>& key, int round_number) {
    if (round_number % 3 == 2) {
        bool k39 = (key[0] ^ key[2] ^ key[19] ^ key[21]);
        key >>= 1;
        key.set(39, k39);
    }
}

void DST40::feistel_round(bitset<SIZE>& x, bitset<SIZE>& key) {
    bitset<SIZE> temp {f(x, key)};
    short r_bits = (x[1] << 1) | (x[0]);
    temp ^= r_bits;
    x >>= 2;
    x.set(38, temp[0]);
    x.set(39, temp[1]);
}

ull DST40::dst40(ull x_temp, ull k_temp, int rounds) {
    bitset<SIZE> x {x_temp};
    bitset<SIZE> key {k_temp};
    for (int i = 1; i <= rounds; i++) {
        feistel_round(x, key);
        round_key(key, i);
    }
    return x.to_ullong();
```

```
}

extern "C" {
    DST40* DST40_new() {
        return new DST40();
    }
    void DST40_run(DST40* dst40, ull& x, ull k, int rounds) {
        ull res = dst40 -> dst40(x, k, rounds);
        x = res;
    }
}
```

A.2 Correlation matrices

The f sub-functions' correlations (transposed for readability):

$$C_{f_a} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & -\frac{1}{4} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & \frac{1}{4} \\ 0 & 0 \\ 0 & \frac{1}{4} \\ 0 & \frac{1}{2} \\ 0 & 0 \\ 0 & 0 \\ 0 & \frac{1}{4} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & -\frac{1}{4} \\ 0 & 0 \\ 0 & 0 \\ 0 & \frac{1}{2} \\ 0 & -\frac{1}{4} \\ 0 & 0 \\ 0 & -\frac{1}{4} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & \frac{1}{4} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} C_{f_b} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & \frac{1}{2} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & -\frac{1}{2} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & \frac{1}{2} \\ 0 & \frac{1}{2} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} C_{f_c} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & \frac{1}{2} \\ 0 & 0 \\ 0 & 0 \\ 0 & -\frac{1}{4} \\ 0 & 0 \\ 0 & \frac{1}{4} \\ 0 & \frac{1}{4} \\ 0 & 0 \\ 0 & \frac{1}{4} \\ 0 & 0 \\ 0 & 0 \\ 0 & -\frac{1}{2} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & \frac{1}{4} \\ 0 & 0 \\ 0 & \frac{1}{4} \\ 0 & \frac{1}{4} \\ 0 & 0 \\ 0 & -\frac{1}{4} \\ 0 & 0 \end{pmatrix} C_{f_d} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & \frac{1}{2} \\ 0 & \frac{1}{2} \\ 0 & 0 \\ 0 & \frac{1}{2} \\ 0 & -\frac{1}{2} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} C_{f_e} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & \frac{1}{2} \\ 0 & -\frac{1}{2} \\ 0 & 0 \\ 0 & 0 \\ 0 & \frac{1}{2} \\ 0 & \frac{1}{2} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \tag{A.1}$$

A.3 Implementation of Algorithm 1

The Python 3 implementation of the attack described in Chapter 5. The DST40 encryptions are done using the shared library export from A.1.

```
from ctypes import cdll, c_ulonglong, c_int, byref
import random

SIZE = 40
LIB = cdll.LoadLibrary('./dst40.so')
KEY = 0x7f537a1329

class DST40(object):
    def __init__(self):
        self.obj = LIB.DST40_new()

    def dst40(self, x, key, rounds=200):
        result = c_ulonglong(x)
        LIB.DST40_run(self.obj, byref(result), c_ulonglong(key), c_int(rounds))
        return result.value

def mask(m, x):
    return bin(m & x).count('1') % 2

dst = DST40()
rounds = 20
n = 2**15

def lin(txt_mask, cpr_mask):
    t = 0
    for i in range(n):
        plaintext = random.getrandbits(SIZE)
        ciphertext = dst.dst40(plaintext, KEY, rounds)
        m1p = mask(txt_mask, plaintext)
        m2c = mask(cpr_mask, ciphertext)
        if m1p == m2c: t += 1

    guess = 0
    if (t > n/2): guess = 1
    return guess

txt_mask01 = 0x0000000001
txt_mask10 = 0x0000000002
txt_mask11 = 0x0000000003
cpr_mask01 = 0x0000000001
cpr_mask10 = 0x0000000002
cpr_mask11 = 0x0000000003
key_mask01 = 0xc0c0030003
key_mask10 = 0x3030030003
key_mask11 = 0x0c0c030003

guess01 = lin(txt_mask01, cpr_mask01)
guess10 = lin(txt_mask10, cpr_mask10)
guess11 = lin(txt_mask11, cpr_mask11)

print("actual 01 =", mask(key_mask01, KEY), "guess 01 =", guess01)
print("actual 10 =", mask(key_mask10, KEY), "guess 10 =", guess10)
print("actual 11 =", mask(key_mask11, KEY), "guess 11 =", guess11)
```

A.4 Feistel function definitions

The indices of x and k start at 1 from the least significant bit.

$F(X, K) = h(g_1, g_2, g_3, g_4)$

$$h = \begin{pmatrix} g_4g_3 + g_4g_1 + g_3g_2 + g_2g_1 + g_2 + g_1 \\ g_4g_2 + g_4g_1 + g_3g_2 + g_3g_1 + g_3 + g_2 \end{pmatrix}$$

$$\begin{aligned}
g_1 &= f_4f_3 + f_4f_2 + f_4 + f_3f_1 + f_3 + f_2f_1 \\
g_2 &= f_8f_7 + f_8f_6 + f_8 + f_7f_5 + f_7 + f_6f_5 \\
g_3 &= f_{12}f_{11} + f_{12}f_{10} + f_{12} + f_{11}f_9 + f_{11} + f_{10}f_9 \\
g_4 &= f_{16}f_{15} + f_{16}f_{14} + f_{16} + f_{15}f_{13} + f_{15} + f_{14}f_{13}
\end{aligned}$$

$$\begin{aligned}
f_1 &= x_{24}x_{32}x_{40} + x_{24}x_{40}k_{32} + x_{24}k_{32} + x_{24}k_{40} + x_{24} + x_{32}x_{40}k_{40} + x_{32}k_{32} + \\
&\quad x_{32}k_{40} + x_{40}k_{32}k_{40} + x_{40}k_{32} + x_{40}k_{40} + x_{40} + k_{32} \\
f_2 &= x_{23}x_{39} + x_{23}k_{39} + x_{23} + x_{31}x_{39} + x_{31}k_{39} + x_{31} + x_{39}k_{31} + x_{39}k_{39} + k_{31}k_{39} + k_{39} \\
f_3 &= x_8x_{16}k_{24} + x_8x_{16} + x_8k_{16} + x_{16}k_8k_{24} + x_{16}k_8 + x_{16}k_{16} + x_{16} + k_8k_{16} + k_8 + \\
&\quad k_{16}k_{24} + k_{16} \\
f_4 &= x_7k_7 + x_7k_{15} + x_7k_{23} + x_7 + x_{15}k_7 + x_{15}k_{15} + x_{15}k_{23} + k_7
\end{aligned}$$

$$\begin{aligned}
f_5 &= x_{22}x_{30}x_{38} + x_{22}x_{38}k_{30} + x_{22}k_{30} + x_{22}k_{38} + x_{22} + x_{30}x_{38}k_{38} + x_{30}k_{30} + \\
&\quad x_{30}k_{38} + x_{38}k_{30}k_{38} + x_{38}k_{30} + x_{38}k_{38} + x_{38} + k_{30} \\
f_6 &= x_{21}x_{37} + x_{21}k_{37} + x_{21} + x_{29}x_{37} + x_{29}k_{37} + x_{29} + x_{37}k_{29} + x_{37}k_{37} + k_{29}k_{37} + k_{37} \\
f_7 &= x_6x_{14}k_{22} + x_6x_{14} + x_6k_{14} + x_{14}k_6k_{22} + x_{14}k_6 + x_{14}k_{14} + x_{14} + k_6k_{14} + k_6 + \\
&\quad k_{14}k_{22} + k_{14} \\
f_8 &= x_5k_5 + x_5k_{13} + x_5k_{21} + x_5 + x_{13}k_5 + x_{13}k_{13} + x_{13}k_{21} + k_5
\end{aligned}$$

$$\begin{aligned}
f_9 &= x_{20}x_{28}x_{36} + x_{20}x_{36}k_{28} + x_{20}k_{28} + x_{20}k_{36} + x_{20} + x_{28}x_{36}k_{36} + x_{28}k_{28} + \\
&\quad x_{28}k_{36} + x_{36}k_{28}k_{36} + x_{36}k_{28} + x_{36}k_{36} + x_{36} + k_{28} \\
f_{10} &= x_{19}x_{35} + x_{19}k_{35} + x_{19} + x_{27}x_{35} + x_{27}k_{35} + x_{27} + x_{35}k_{27} + x_{35}k_{35} + k_{27}k_{35} + k_{35} \\
f_{11} &= x_4x_{12}k_{20} + x_4x_{12} + x_4k_{12} + x_{12}k_4k_{20} + x_{12}k_4 + x_{12}k_{12} + x_{12} + k_4k_{12} + k_4 + \\
&\quad k_{12}k_{20} + k_{12} \\
f_{12} &= x_3k_3 + x_3k_{11} + x_3k_{19} + x_3 + x_{11}k_3 + x_{11}k_{11} + x_{11}k_{19} + k_3
\end{aligned}$$

$$\begin{aligned}
f_{13} &= x_{18}x_{26}x_{34} + x_{18}x_{34}k_{26} + x_{18}k_{26} + x_{18}k_{34} + x_{18} + x_{26}x_{34}k_{34} + x_{26}k_{26} + \\
&\quad x_{26}k_{34} + x_{34}k_{26}k_{34} + x_{34}k_{26} + x_{34}k_{34} + x_{34} + k_{26} \\
f_{14} &= x_{17}x_{33} + x_{17}k_{33} + x_{17} + x_{25}x_{33} + x_{25}k_{33} + x_{25} + x_{33}k_{25} + x_{33}k_{33} + k_{25}k_{33} + k_{33} \\
f_{15} &= x_{10}k_{10} + x_{10}k_{18} + x_{10} + k_2k_{10} + k_2k_{18} + k_{18} \\
f_{16} &= x_9k_9 + x_9k_{17} + x_9 + k_1k_9 + k_1k_{17} + k_{17}
\end{aligned}$$