Analysis of NORX:
Investigating Differential and Rotational Properties

Jean-Philippe Aumasson[1], Philipp Jovanovic[2], and Samuel Neves[3]

[1] Kudelski Security, Switzerland
jeanphilippe.aumasson@gmail.com
[2] University of Passau, Germany
jovanovic@fim.uni-passau.de
[3] University of Coimbra, Portugal
sneves@dei.uc.pt

Abstract. This paper presents a thorough analysis of the AEAD scheme NORX, focussing on differential and rotational properties. We first introduce mathematical models that describe differential propagation with respect to the non-linear operation of NORX. Afterwards, we adapt a framework previously proposed for ARX designs allowing us to automatise the search for differentials and characteristics. We give upper bounds on the differential probability for a small number of steps of the NORX core permutation. For example, in a scenario where an attacker can only modify the nonce during initialisation, we show that characteristics have probabilities of less than 2^{-60} (32-bit) and 2^{-53} (64-bit) after only one round. Furthermore, we describe how we found the best characteristics for four rounds, which have probabilities of 2^{-584} (32-bit) and 2^{-836} (64-bit), respectively. Finally, we discuss some rotational properties of the core permutation which yield some first, rough bounds and can be used as a basis for future studies.

Keywords: NORX, AEAD, LRX, differential cryptanalysis, rotational cryptanalysis

1 Introduction

NORX [4] is a new scheme for authenticated encryption with associated data (AEAD) and was recently submitted to CAESAR [1]. NORX is based on well-known building blocks but refines those components to provide certain desirable features. Its layout is a modified version of the monkeyDuplex construction [9], which allows to process data in parallel. The duplex construction is an alteration of sponge functions [10], which were introduced alongside Keccak [12]. The core permutation F of NORX is derived from ChaCha [6] and BLAKE2 [5], which are parade examples for ARX primitives, i.e. cryptographic functions based solely on integer addition mod 2^n, bit rotations and XOR. However, the permutation F

is a so-called LRX[4] construction, because integer addition, which can be written as $a+b=(a\oplus b)+((a\wedge b)\ll 1)$ [21], is replaced by the approximation $(a\oplus b)\oplus((a\wedge b)\ll 1)$, a purely logic-based operation. The aim is to increase hardware friendliness and simplify cryptanalysis. Despite its famous predecessors, that have already resisted extensive analysis [3,19,25] and are deemed secure, this new permutation F still lacks in-depth analysis and its security level is yet unclear.

Differential cryptanalysis [13] is one of the most powerful and versatile attack techniques usable against symmetric primitives and belongs to the standard repertoire of every cryptanalyst. Therefore, it is not surprising that every new symmetric primitive is examined upon its resistance against differential attacks. Usually, it is much easier to establish bounds for strongly aligned ciphers, like AES [16], than for weakly aligned ones [8]. NORX rather belongs to the latter category and, despite some successful inroads into deriving bounds for weakly aligned ciphers [15,17], it is not obvious how to establish such bounds in the general case. Hence, in the first part of the paper, we investigate differential propagation in F and, based on that, introduce NODE [2], the NORX *Differential Search Engine*, a framework providing a way to search for differentials and characteristics in an automated way. Our approach is guided by the work of Mouha and Preneel [24], where a search framework was introduced for the ARX cipher Salsa20 [7]. Their framework constructs a description of the differential propagation behaviour of Salsa20, using well-known differential properties of integer addition [22]. The description is formulated in the CVC language, the standard input language of the constraint solver STP [18], which supports operations on bit vectors (like bitwise XOR, AND, modular addition, etc.) and therefore allows a straightforward modelling of the differential search problem. The resulting description has a simple shape, which facilitates cryptanalysis.

However, in order to use such a framework for NORX, some adjustments are necessary: The permutation F of NORX is not based on integer addition, and hence we can not rely upon already known results on the differential properties of the latter [22]. Therefore, we start with the mathematical modelling of differential propagation with respect to the non-linear operation $(a\oplus b)\oplus\big((a\wedge b)\ll 1\big)$ of NORX. All of our claims are supported by rigorous proofs. Then, we use these results to show how to adapt the search framework to the NORX permutation, which requires some more modifications, since the original framework [24] was developed for Salsa20, whereas F is based on ChaCha [6]. Finally, we present the results from our extensive empirical analysis of F^R.

The second part of this paper is dedicated to the rotational cryptanalysis [20] of the core permutation F^R. Rotational cryptanalysis is another important aspect for the security evaluation of ARX/LRX-based primitives. We present some basic rotational properties of F and based on that derive bounds for a few simple rotational attacks.

[4] This is not an official term. We introduce it to easily distinguish between ARX- and purely logic-based primitives. Terminology-wise it is not entirely correct, though, as integer addition can be obviously modelled by bitwise logical operations as well.

Outline. The paper is structured as follows. Section 2 introduces notation and recalls the basic layout of NORX, with a focus on its core permutation F^R, as it is the main target of our cryptanalysis efforts. Sections 3 and 4 present differential and rotational cryptanalysis of NORX and Section 5 concludes the paper.

2 Preliminaries

2.1 Notation

Hexadecimal numbers are denoted in `typewriter`, e.g. `c9` = 201. A *word* is either a 32-bit or 64-bit string, depending on the context. Parsing of data streams (as byte arrays) to word arrays is done in little-endian order. The concatenation of strings x and y is denoted by $x \parallel y$. The length of a bit string x is written as $|x|$, and its Hamming weight as $\mathsf{hw}(x)$. We use the standard notation $\neg$, $\wedge$, $\vee$ and $\oplus$ for bitwise NOT, AND, OR and XOR, $x \ll n$ and $x \gg n$ for left- and right-shift, and $x \lll n$ and $x \ggg n$ for left- and right-rotation of x by n bits.

2.2 Core Components of NORX

The NORX family of AEAD schemes is based on the *monkeyDuplex construction* [9,11] and parametrised by a *word size* $W \in \{32, 64\}$, a *round number* $1 \leq R \leq 63$, a *parallelism degree* $0 \leq D \leq 255$ and a *tag size* $|A| \leq 10W$. The meaning of the parameters is basically self-explanatory, for more details see [4].

The state S of NORX consists of sixteen words $s_0, \ldots, s_{15}$ each of size W bits, which are arranged in a 4×4 matrix. Thus, the state has a size of 512 bits for $W = 32$ and a size of 1024 bits for $W = 64$. Due to the duplex construction, the words of the state are divided into two types: $s_0, \ldots, s_9$ are called the *rate words* and $s_{10}, \ldots, s_{15}$ are called the *capacity words*[5]. The rate words are used for data processing, whereas the capacity words remain untouched and ensure the security of the scheme. S is initialised by loading a *nonce* n_0, n_1, a *key* $k_0, \ldots, k_3$ and *constants* $u_0, \ldots, u_9$ in the following way:

$$\begin{pmatrix} s_0 & s_1 & s_2 & s_3 \\ s_4 & s_5 & s_6 & s_7 \\ s_8 & s_9 & s_{10} & s_{11} \\ s_{12} & s_{13} & s_{14} & s_{15} \end{pmatrix} \longleftarrow \begin{pmatrix} u_0 & n_0 & n_1 & u_1 \\ k_0 & k_1 & k_2 & k_3 \\ u_2 & u_3 & u_4 & u_5 \\ u_6 & u_7 & u_8 & u_9 \end{pmatrix}$$

More information on the constants can be found in [4]. This initial state is transformed by F^{2R}, where F is the *round function*, interleaved with the injection of parameter and domain separation constants, before data processing starts, which uses F^R. Concrete instances of NORX, as given in [4], use $R \in \{4, 6\}$. The round function F of NORX is composed of a *column step*

$$\mathsf{G}(s_0, s_4, s_8, s_{12})\ \mathsf{G}(s_1, s_5, s_9, s_{13})\ \mathsf{G}(s_2, s_6, s_{10}, s_{14})\ \mathsf{G}(s_3, s_7, s_{11}, s_{15})$$

[5] These are also respectively known as the *outer* and *inner* part of the state [10,9].

followed by a *diagonal step*

$$\mathsf{G}(s_0, s_5, s_{10}, s_{15})\ \mathsf{G}(s_1, s_6, s_{11}, s_{12})\ \mathsf{G}(s_2, s_7, s_8, s_{13})\ \mathsf{G}(s_3, s_4, s_9, s_{14})$$

The function G transforms four words a, b, c, and d by doing

$$\begin{array}{ll} 1: & a \longleftarrow (a \oplus b) \oplus \big((a \wedge b) \ll 1\big) \\ 2: & d \longleftarrow (a \oplus d) \ggg r_0 \\ 3: & c \longleftarrow (c \oplus d) \oplus \big((c \wedge d) \ll 1\big) \\ 4: & b \longleftarrow (b \oplus c) \ggg r_1 \end{array} \qquad \begin{array}{ll} 5: & a \longleftarrow (a \oplus b) \oplus \big((a \wedge b) \ll 1\big) \\ 6: & d \longleftarrow (a \oplus d) \ggg r_2 \\ 7: & c \longleftarrow (c \oplus d) \oplus \big((c \wedge d) \ll 1\big) \\ 8: & b \longleftarrow (b \oplus c) \ggg r_3 \end{array}$$

where rotation offsets (r_0, r_1, r_2, r_3) have the values $(8, 11, 16, 31)$ for 32-bit and $(8, 19, 40, 63)$ for 64-bit.

Since our analysis focusses on the core permutation F^R, we do not go into the details of NORX's mode of operation. For more information on these topics, we refer to the official specification [4].

2.3 Weak States

The NORX specification [4] contains a discussion about the all-zero state, which is mapped to itself by F^R for any $R > 0$, and why it is no problem for the security of the scheme. However, due to the layout of F, there is another class of weak states. These are of the form

$$\begin{pmatrix} w & w & w & w \\ x & x & x & x \\ y & y & y & y \\ z & z & z & z \end{pmatrix}$$

with w, x, y, and z being arbitrary W-bit sized words. The column-pattern is preserved by F^R for an arbitrary value of $R > 0$. The ability to hit such a state purposely, is equivalent to the ability of reconstructing the key and therefore breaking the entire scheme. While there are quite many of these states, namely 2^{4W}, their number is still negligible compared to the total number of 2^{16W} states. Thus, the probability to hit such a state is 2^{-12W}, which translates to probabilities of 2^{-384} ($W = 32$) and 2^{-768} ($W = 64$). Additionally, this attack does not take into account the extra protection provided through the duplex construction, the asymmetric constants used during initialisation, or the domain separation constants which are integrated into the state before each application of F^R. All of the above features should impede the exploitation of these states.

3 Differential Cryptanalysis

This section is dedicated to the differential cryptanalysis of NORX. First, we introduce the required mathematical models to describe differential propagation in F^R of NORX. Then we describe how to construct the search framework and finally apply it to NORX and present our results.

3.1 Mathematical Models

Let n denote the word size, let x and y denote bit strings of size n and let α, β and γ denote differences of size n. We identify by α_i, β_i, γ_i, x_i and y_i the individual bits of α, β, γ, x and y, with $0 \leq i \leq n-1$.

Definition 1. *The non-linear operation* H *of* NORX *is the vector Boolean function defined by*

$$\mathsf{H} : \mathbb{F}_2^{2n} \longrightarrow \mathbb{F}_2^n,\ (x, y) \mapsto (x \oplus y) \oplus ((x \wedge y) \ll 1)$$

Definition 2. *Let* $f : \mathbb{F}_2^{2n} \longrightarrow \mathbb{F}_2^n$ *be a vector Boolean function and let* α, β *and* γ *be* n*-bit sized XOR-differences. We call* $(\alpha, \beta) \longrightarrow \gamma$ *a* (XOR-)differential *of* f *if there exist* n*-bit strings* x *and* y *such that the following equation holds:*

$$f(x \oplus \alpha, y \oplus \beta) = f(x, y) \oplus \gamma$$

Otherwise, if no such n*-bit strings* x *and* y *exist, we call* $(\alpha, \beta) \longrightarrow \gamma$ *an* impossible (XOR-)differential *of* f*.*

Plugging the non-linear operation H of NORX from Definition 1 into the formula of Definition 2, we see that an XOR-differential $(\alpha, \beta) \longrightarrow \gamma$ of H fulfils

$$\alpha \oplus \beta \oplus \gamma = ((x \wedge \beta) \oplus (y \wedge \alpha) \oplus (\alpha \wedge \beta)) \ll 1 \tag{1}$$

for n-bit strings x and y. Rewriting the above formula on bit level we get

$$\begin{aligned} 0 &= \alpha_0 \oplus \beta_0 \oplus \gamma_0 \\ 0 &= (\alpha_i \oplus \beta_i \oplus \gamma_i) \oplus (\alpha_{i-1} \wedge \beta_{i-1}) \oplus (x_{i-1} \wedge \beta_{i-1}) \oplus (y_{i-1} \wedge \alpha_{i-1}), \quad i > 0 \end{aligned}$$

Lemma 3 is an important step towards expressing differential propagation in NORX and is the analogue to Theorem 1 for integer addition from [22]. The lemma eliminates the dependence of Equation 1 on the bit strings x and y and therefore allows us to check in a constant amount of word operations if a given tuple (α, β, γ) of differences is an (impossible) XOR-differential of H.

Lemma 3. *For each XOR-differential* $(\alpha, \beta) \longrightarrow \gamma$ *of the non-linear operation* H *of* NORX *the following equation is satisfied:*

$$(\alpha \oplus \beta \oplus \gamma) \wedge (\neg((\alpha \vee \beta) \ll 1)) = 0 \tag{2}$$

Proof. See Appendix A.

Obviously, a tuple of differences (α, β, γ) not satisfying Lemma 3 is an impossible XOR-differential of H.

Definition 4. *Let* f *be a vector Boolean function and let* δ *be an XOR-differential in terms of Definition 2. The probability* xdp^f *of* δ *is defined as*

$$\mathsf{xdp}^f(\delta) = |\{x, y \in \mathbb{F}_2^n : f(x \oplus \alpha, y \oplus \beta) \oplus f(x, y) \oplus \gamma = 0\}| \cdot 2^{-2n}$$

The value $\mathsf{xdp}^f(\delta)$ *is also called the* XOR-differential probability *of* δ*. Moreover, for* $\mathsf{xdp}^f(\delta) = 2^{-w}$ *we call* w *the* XOR-(differential) weight *of* δ*.*

The differential probability of an impossible differential is always 0 by prerequisite, as $\{x, y \in \mathbb{F}_2^n : f(x \oplus \alpha, y \oplus \beta) \oplus f(x, y) \oplus \gamma = 0\}$ is then the empty set, see Definition 2. To compute the probability of a differential with respect to the non-linear operation H of NORX, we can use the following lemma.

Lemma 5. *Let δ be a XOR-differential with respect to the non-linear operation H of NORX. Its differential probability is then given by*

$$\mathsf{xdp}^{\mathsf{H}}(\delta) = 2^{-\mathsf{hw}((\alpha \vee \beta) \ll 1)}$$

Proof. See Appendix A.

Instead of looking at XOR-differences one could alternatively also analyse f-differentials, which is done in the following.

Definition 6. *Let $f : \mathbb{F}_2^{2n} \longrightarrow \mathbb{F}_2^n$ be a vector Boolean function and let α, β and γ be differences with respect to f. We call $(\alpha, \beta) \longrightarrow \gamma$ an* f-differential *of XOR if there exist n-bit strings x and y such that the following equation holds:*

$$f(x, \alpha) \oplus f(y, \beta) = f(x \oplus y, \gamma)$$

Otherwise, if no such n-bit strings x and y exist, we call $(\alpha, \beta) \longrightarrow \gamma$ an impossible f-differential *of XOR.*

Plugging the non-linear operation H of NORX into the formula of Definition 6 we obtain the following equation

$$\alpha \oplus \beta \oplus \gamma = ((x \wedge (\alpha \oplus \gamma)) \oplus (y \wedge (\beta \oplus \gamma))) \ll 1 \tag{3}$$

which can be expressed on bit level as

$$\begin{aligned} 0 &= \alpha_0 \oplus \beta_0 \oplus \gamma_0 \\ 0 &= (\alpha_i \oplus \beta_i \oplus \gamma_i) \oplus (x_{i-1} \wedge (\alpha_{i-1} \oplus \gamma_{i-1})) \oplus (y_{i-1} \wedge (\beta_{i-1} \oplus \gamma_{i-1})), \quad i > 0 \end{aligned}$$

Lemma 7. *Let H denote the non-linear operation of NORX. For each H-differential in terms of Definition 6 the following equation is satisfied:*

$$(\alpha \oplus \beta \oplus \gamma) \wedge (\neg(\gamma \ll 1) \oplus (\alpha \ll 1)) \wedge (\neg(\beta \ll 1) \oplus (\gamma \ll 1)) = 0 \tag{4}$$

Proof. See Appendix A.

Definition 8. *Let f be a vector Boolean function and δ be an f-differential in terms of Definition 6. The probability $\mathsf{fdp}^{\oplus}$ of δ is defined as*

$$\mathsf{fdp}^{\oplus}(\delta) = |\{x, y \in \mathbb{F}_2^n : f(x, \alpha) \oplus f(y, \beta) \oplus f(x \oplus y, \gamma) = 0\}| \cdot 2^{-2n}$$

We call $\mathsf{fdp}^{\oplus}(\delta)$ the f-differential probability *of δ. Moreover, for $\mathsf{fdp}^{\oplus}(\delta) = 2^{-w}$ we call w the* f-(differential) weight *of δ.*

Lemma 9. *Let H denote the non-linear operation of NORX and let δ be an H-differential in terms of Definition 6. Its probability is then given by*

$$\mathsf{Hdp}^{\oplus}(\delta) = 2^{-\mathsf{hw}(((\alpha \oplus \gamma) \vee (\beta \oplus \gamma)) \ll 1)}$$

Proof. See Appendix A.

While we exclusively consider XOR-differentials and -characteristics in the rest of the paper, f-differentials might be of interest for future investigations.

3.2 NODE – The NORX Differential Search Engine

Now that we have introduced the mathematical model, we describe in this part the framework NODE for the search of differential characteristics of a predefined weight. Our tool is freely available at [2] under a public domain-like license. We focus here on XOR-differentials, as introduced in Definition 2, i.e. differences are computed with respect to XOR and for the vector Boolean function we use the non-linear operation H of NORX. If we speak in the following of differentials we always refer to the above type. Below we show the general approach, and refer to Appendix B for the CVC code.

For modelling the differential propagation through a sequence of operations, we use a technique well known from algebraic cryptanalysis: For every output of an operation a new set of variables is introduced. These output variables are then modelled as a function of its input variables. Moreover, the former are used as input to the next operation. This is repeated until all required operations have been integrated into the problem description. Before we show how the differential propagation in F^R is modelled concretely, we introduce the required variables.

Let s denote the number of (column and diagonal) steps to be analysed and let $0 \leq i \leq 15$ and $0 \leq j \leq 2(s-1)$. For example, if we analyse F^2, we have $s = 4$. Let x_i, $y_{i,j}$ and z_i be W-bit sized variables, which model the input, internal and output XOR differences of a differential characteristic. Recall that $W \in \{32, 64\}$ denotes the word size of NORX. Moreover, let $w_{i,k}$, with $0 \leq k \leq s-1$, be W-bit sized helper variables which are used for differential weight computations or equivalently to determine the probability of a differential characteristic. We assume that the probability of a differential characteristic is the sum of weights of each non-linear operation H. Furthermore, let d denote a W-bit sized variable, which fixes the total weight of the characteristic we plan to search for. The description of the search problem is generated through the following steps:

1. Every time the function G applies the non-linear operation H we add two expressions to our description:
   (a) Append the equation $0 = (\alpha \oplus \beta \oplus \gamma) \wedge (\neg((\alpha \vee \beta) \ll 1))$ from Lemma 3, with α, β and γ each substituted by one of the variables x_i, $y_{i,j}$ or z_i. This ensures that only non-impossible characteristics are considered.
   (b) Add the expression $w_{i,k} = (\alpha \vee \beta) \ll 1$ from Lemma 5, with α and β substituted by the same variables x_i, $y_{i,j}$ or z_i as in step (a). This expression keeps track of the weight of the characteristic.
2. Every time the function G applies a rotation we apply the same rotation to the corresponding XOR difference, i.e. we add $\gamma = (\alpha \oplus \beta) \ggg r$ to the problem description, with α, β and γ substituted appropriately. Note that the rotation is a linear operation and thus does not change the differential probability.
3. Add an expression corresponding to the following equation:

$$d = \sum_{k=0}^{s-1} \sum_{i=0}^{15} \mathsf{hw}(w_{i,k}) \tag{5}$$

This equation ensures that indeed a characteristic of weight d is found. Depending on the technique how Hamming weights are computed, additional variables might be necessary. Refer to Appendix B for one possible implementation to compute Hamming weights in the CVC language.

4. Set the variable d to the target differential weight and append it to the problem description.
5. Exclude the trivial characteristic mapping an all-zero input difference to an all-zero output difference. To do so, it is sufficient to exclude the all-zero input difference. Therefore, append an expression equivalent to $\neg\big((x_0 = 0) \wedge ... \wedge (x_{15} = 0)\big)$ to the CVC description.

After the generation of the problem description is finished, it can be used to search for differential characteristics using STP. Alternatively, STP allows to convert the representation of the problem to SMT-LIB2 or CNF, enabling searches with other SMT or SAT solvers, like Boolector [14] or CryptoMiniSat [23].

3.3 Applications of NODE

In this part we describe the application of the search framework to the permutation F^R of NORX. Depending on the concrete attack model, there are different ways an attacker could inject differences into the NORX state. During initialisation an adversary is allowed to modify either the nonce words s_1 and s_2 (init_N) or nonce and key words $s_1, s_2, s_4, \ldots, s_7$ ($\mathsf{init}_{N,K}$). During data processing an attacker can inject differences into the words of the rate $s_0, \ldots, s_9$ (rate). Last but not least, we also investigate the case where an attacker can manipulate the whole state $s_0, \ldots, s_{15}$ (full). While an attacker is not able to influence the entire state at any point directly due to the duplex construction, the full scenario is nevertheless useful to estimate the general strength of F^R, because all of the other settings described above are special cases of the latter. Additionally, it could be useful for the chaining of characteristics: For example, an attacker could start with a search in the data processing part (i.e. under the rate setting) over a couple of steps, say F^{R_1}, and continue afterwards with a second search, starting from the full state for another couple of steps, say F^{R_2}, so that differentials from the second search connect to those from the first, resulting in differentials for $\mathsf{F}^{R_1+R_2}$. We will explore this Divide&Conquer strategy in more detail below.

For the rest of the paper, we denote a differential characteristic as a tuple of differences $(\delta_0, \ldots, \delta_n)$, where δ_0 is the *input difference* and δ_n is the *output difference*. The values δ_i for $0 < i < n$ are called *internal differences*. The weight of the probability that difference δ_i is transformed into difference δ_{i+1} by the r_i-fold iteration of F is denoted by w_i for $0 \leq i \leq n-1$. Recall, that we assume that the probability of the entire characteristic is equal to the multiplication of probabilities of the partial characteristics, and thus we have $w = \sum_{i=0}^{n-1} w_i$ for the total weight of the characteristic. The notation $\mathsf{F}^{R+0.5}$ describes that we do R full rounds followed by one more column step, e.g. $\mathsf{F}^{1.5}$ corresponds to one full round plus one additional column step.

Experimental Verification of the Search Framework. The goal of the experimental verification is to show that the framework indeed does what it is supposed to do, namely find differentials of a predetermined weight w in F^R. Therefore, we generated differentials for $\mathsf{F}^{1.5}$ (full) and verified them against a C reference implementation of $\mathsf{F}^{1.5}$. Under these prerequisites our framework found the first differentials at a weight of 12, for both $W = 32$ and $W = 64$, which thus should have a probability of about 2^{-12}. To get a better coverage of our verification test, we did not use only differentials of that particular weight, but generated random differentials of weights $w \in \{12, \ldots, 18\}$, which are listed in Appendix C.1 for both 32- and 64-bit. Then we applied them to the C implementation of $\mathsf{F}^{1.5}$ for 2^{w+16} pairs of randomly chosen input states having the input difference of the characteristic. In each case, we checked if the output difference had the predicted pattern. The number of pairs adhering the characteristic should be around 2^{16}. The results are illustrated in the first table of Appendix C.1 and show that the search framework indeed finds characteristics with the expected properties.

Lower Bounds for Differential Weights of F^R. We made an extensive analysis on the weight bounds of differential paths in F^R, where we investigated $1 \leq s \leq 4$ steps for our four different scenarios init_N, $\mathsf{init}_{N,K}$, rate and full. We tried to find the lowest weights where differentials appear for the first time. These cases are listed in Table 1 as entries without brackets. For example, in case of NORX32 under the setting full, there are no differentials in $\mathsf{F}^{1.5}$ with a weight smaller than 12. Entries in brackets are the maximal weights we were capable of examining without finding any differentials. Due to memory constraints, our methods failed for differential weights higher than those presented in Table 1. For example, our search routine did not find any characteristics of weight smaller than 40 (i.e. of probability higher than 2^{-40}) for the scenario $\mathsf{F}^{1.5}$, $\mathsf{init}_{N,K}$ and $W = 32$. The required amount of RAM, to execute this check, was approximately 49 GiB (using CryptoMiniSat with 16 threads) with a running time of 8 hours.

Table 1. Lower bounds for differential trail weights

	NORX32				NORX64			
	init_N	$\mathsf{init}_{N,K}$	rate	full	init_N	$\mathsf{init}_{N,K}$	rate	full
$\mathsf{F}^{0.5}$	6	2	2	0	6	2	2	0
$\mathsf{F}^{1.0}$	(60)	22	10	2	(53)	22	12	2
$\mathsf{F}^{1.5}$	(60)	(40)	(31)	12	(53)	(35)	(27)	12
$\mathsf{F}^{2.0}$	(61)	(45)	(34)	(27)	(51)	(37)	(30)	(23)

The security of NORX depends heavily on the security of the initialisation, which transforms the initial state by F^{2R}. As init_N is the most realistic attack scenario, we conducted a search over all possible 1- and 2-bit differences in the nonce words. Our search revealed that the best characteristics have weights of

67 (32-bit) and 76 (64-bit) under those prerequisites. Obviously, these weights are not too far away from the computationally verified values of 60 (32-bit) and 53 (64-bit) from Table 1, showing that the bounds for F (init_N) are quite tight.

Extrapolating the above results to F^8 (i.e. $R = 4$), we get lower weights of $61 + 3 \cdot 27 = 142$ (init_N) or $45 + 3 \cdot 27 = 126$ ($\mathsf{init}_{N,K}$) for NORX32 and $51 + 3 \cdot 23 = 132$ (init_N) or $37 + 3 \cdot 23 = 106$ ($\mathsf{init}_{N,K}$) for NORX64. However, these are only loose bounds and we expect the real ones to be considerably higher.

Search for Differential Characteristics in F^4. This part shows how we constructed differential characteristics in F^4 under the setting full for both versions of the permutation, i.e. 32- and 64-bit. Unsurprisingly, a direct approach to find such characteristics turned out to be infeasible, hence we decomposed the search into multiple parts and constructed the entire path step by step.

At first we made searches that only stretched over $R \leq 2$ rounds. After tens of thousands of iterations using many different search parameter combinations we found differentials having internal differences of Hamming weight 1 and 2 after one application of F. We also used a probability-1 differential in G, which is listed as the first entry in the table of Appendix C.2, as a starting place. We expanded all those characteristics for both word sizes, in forward and backward direction one column or diagonal step at a time, until their paths stretched the entire 4 rounds. The best differential paths we found this way have weights of 584 (32-bit) and 836 (64-bit), respectively. Both are depicted in Appendix C.3.

Iterative Differentials. We also performed extensive searches for iterative differentials in F for the setting full. Using our framework, we could show that there are no such differentials up to a weight of 29 (32-bit) and 27 (64-bit), before our methods failed due to computational constraints. Extrapolating these results to F^8 and F^{12}, i.e. the number of initialisation rounds for $R = 4$ and $R = 6$, we get lower weight bounds of 232 and 348, for 32-bit, or of 216 and 324 for 64-bit. The best iterative differentials we could find for F, have weights of 512 (32-bit) and 843 (64-bit) and are depicted in Appendix C.4. These weights are obviously much higher than our guaranteed lower bounds, and hence we expect that the latter are much better compared to the values we were able to verify computationally.

Differentials with Equal Columns. The class of weak states from Section 2.3 can be obviously transformed into XOR-differentials having four equal columns. The best differentials we could find for one round F have weight 44 for both 32-bit and 64-bit. They exploit an already well known probability-1 differential in G, see Appendix C.2. The 64-bit variant was also used in the construction of the characteristics with weight 836 in F^4 above. Concrete representations of these differentials can be found in Appendix C.5.

3.4 Further Applications

The techniques presented in this section are obviously not restricted to NORX only. In principle, every function based on integer addition, as shown for Salsa20 in [24], and/or bitwise logical operations, like OR, NAND, NOR and so on, can be analysed just as easily. For LRX ciphers, all one has to do is rewrite their non-linear operations in terms of bitwise logical AND, which then allows to reuse the results from above.

4 Rotational Cryptanalysis

Definition 10. *Let f be a vector Boolean function $f : \mathbb{F}_2^{2n} \longrightarrow \mathbb{F}_2^n$ and let x, y be n-bit strings. We call (x, y) a* rotational pair *with respect to f if the following equation holds:*

$$f(x, y) \ggg r = f(x \ggg r, y \ggg r)$$

Lemma 11. *Let H be the non-linear function of* NORX*, and let x, y be n-bit strings. The probability of (x, y) being a rotational pair is:*

$$\Pr(\mathsf{H}(x, y) \ggg r = \mathsf{H}(x \ggg r, y \ggg r)) = \frac{9}{16} \ (\approx 2^{-0.83})$$

Proof. See Appendix D.

Now we can use Lemma 11 and Theorem 1 from [20] (under the assumption that the latter holds for H, too) to compute the probability of $\Pr(\mathsf{F}^R(S) \ggg r = \mathsf{F}^R(S \ggg r))$ for a state S and a number of rounds R. It is given by:

$$\Pr(\mathsf{F}^R(S) \ggg r = \mathsf{F}^R(S \ggg r) = (9/16)^{4 \cdot 4 \cdot 2 \cdot R}$$

Table 2 summarizes the (rounded) weights (i.e. the negative logarithms of the probabilities) for different values of R, which are relevant for NORX.

Table 2. Weights for rotational distinguishers of F^R

R	4	6	8	12
w	106	159	212	318

As a consequence, the permutation F^R on a $16W$ state is indistinguishable from a random permutation for $R \geq 20$ if $W = 32$ and for $R \geq 39$ if $W = 64$ with probabilities of $\Pr \leq 2^{-531}$ and $\Pr \leq 2^{-1035}$ respectively.

Definition 12. *Let f be a vector Boolean function $f : \mathbb{F}_2^{2n} \longrightarrow \mathbb{F}_2^n$ and let x, y be n-bit strings. We call (x, y) a* rotational fixed point *with respect to f if the following equation holds:*

$$f(x, y) \ggg r = f(x, y)$$

Lemma 13. *Let f be a vector Boolean function $f : \mathbb{F}_2^{2n} \longrightarrow \mathbb{F}_2^n$, $(x, y) \mapsto f(x, y)$, which is a permutation on $\mathbb{F}_2^n$, if either x or y is fixed. The probability that (x, y) is a rotational fixed point is:*

$$\Pr(f(x, y) \ggg r = f(x, y)) = 2^{-(n-\gcd(r,n))}$$

Proof. See Appendix D.

A direct consequence of Lemma 13 is that for n even and $r = n/2$ the probability that (x, y) is a rotational fixed point is $2^{-n/2}$. The rotation $r = n/2$, which swaps the two halves of a bit string, is especially interesting for cryptanalysis as it results in the highest probability among all $0 < r < n$.

The non-linear function H of NORX obviously satisfies the requirement of being a permutation on $\mathbb{F}_2^n$, when one of its inputs is fixed. Therefore we get probabilities of 2^{-16} (32-bit, $r = 16$) and 2^{-32} (64-bit, $r = 32$), that (x, y) is a rotational fixed point of H.

5 Conclusion

In this paper, we provide an extensive analysis of the differential and rotational properties of NORX's core permutation F^R and derive some first bounds for attacks on the complete scheme. We introduce the mathematical models required to describe XOR- and H-differentials with respect to F^R. All mathematical claims are verified by rigorous proofs. Moreover, we present NODE, a framework, which allows to automatise the search for XOR-differentials and -characteristics. We show the results of our extensive experiments and can conclude that there is a large gap between those differential bounds that are computationally verifiable and the weights of the best differentials that we were able to find. In particular, when considering initialisation with F^8, the verifiable but extrapolated weight bounds have values of 126 ($\mathsf{NORX32}$) and 106 ($\mathsf{NORX64}$) for an attacker in the related key model. On the other hand, the best differentials for F^4 have weights of 584 (32-bit) and 836 (64-bit). Thus, initialisation with F^8 ($R = 4$) and F^{12} ($R = 6$) seems to have a high security margin against differential attacks.

For rotational cryptanalysis, we are able to derive lower weight bounds of 212 and 318 for distinguishers on F^8 and F^{12} using a mix of new and already known results. We stress that these distinguishers only hold for the bare permutation. They do not take into account the additional protection provided by the duplex construction of NORX or the asymmetric constants used during initialisation.

Acknowledgements. The authors would like to thank the anonymous reviewers for their comprehensive commentaries which helped to improve the quality of this paper.

References

1. CAESAR — Competition for Authenticated Encryption: Security, Applicability, and Robustness (2014), http://competitions.cr.yp.to/caesar.html

2. NODE — The NORX Differential Search Engine (2014), https://github.com/norx/NODE
3. Aumasson, J.P., Fischer, S., Khazaei, S., Meier, W., Rechberger, C.: New Features of Latin Dances: Analysis of Salsa, ChaCha and Rumba. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 470–488. Springer (2008)
4. Aumasson, J.P., Jovanovic, P., Neves, S.: NORX: Parallel and Scalable AEAD. In: Kutylowski, M., Vaidya, J. (eds.) ESORICS 2014. LNCS, vol. 8713, pp. 19–36. Springer (2014)
5. Aumasson, J.P., Neves, S., Wilcox-O'Hearn, Z., Winnerlein, C.: BLAKE2: Simpler, Smaller, Fast as MD5. In: Jacobson, M., Locasto, M., Mohassel, P., Safavi-Naini, R. (eds.) ACNS 2013. LNCS, vol. 7954, pp. 119–135. Springer (2013)
6. Bernstein, D.J.: ChaCha, a Variant of Salsa20. In: Workshop Record of SASC 2008: The State of the Art of Stream Ciphers (2008), http://cr.yp.to/chacha.html
7. Bernstein, D.J.: The Salsa20 Family of Stream Ciphers. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 84–97. Springer (2008)
8. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: On Alignment in Keccak. In: ECRYPT II Hash Workshop (May 2011)
9. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Permutation-based Encryption, Authentication and Authenticated Encryption, presented at DIAC 2012, 05–06 July 2012, Stockholm, Sweden
10. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Cryptographic Sponge Functions (January 2011), http://sponge.noekeon.org/CSF-0.1.pdf
11. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 320–337. Springer (2011)
12. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: The Keccak Reference (January 2011), http://keccak.noekeon.org/
13. Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. J. Cryptology 4(1), 3–72 (1991)
14. Brummayer, R., Biere, A.: Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays. In: Kowalewski, S., Philippou, A. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. LNCS, vol. 5505, pp. 174–177. Springer (2009), http://fmv.jku.at/boolector/
15. Daemen, J., Peeters, M., Assche, G.V., Rijmen, V.: Nessie Proposal: the Block Cipher Noekeon. Nessie submission (2000), http://gro.noekeon.org/
16. Daemen, J., Rijmen, V.: The Wide Trail Design Strategy. In: Honary, B. (ed.) Cryptography and Coding, LNCS, vol. 2260, pp. 222–238. Springer (2001)
17. Daemen, J., Van Assche, G.: Differential Propagation Analysis of Keccak. In: FSE 2012. LNCS, vol. 7549, pp. 422–441. Springer (2012)
18. Ganesh, V., Govostes, R., Phang, K.Y., Soos, M., Schwartz, E.: STP — A Simple Theorem Prover (2006–2013), http://stp.github.io/stp
19. Guo, J., Karpman, P., Nikolic, I., Wang, L., Wu, S.: Analysis of BLAKE2. In: Benaloh, J. (ed.) CT-RSA 2014. LNCS, vol. 8366, pp. 402–423. Springer (2014)
20. Khovratovich, D., Nikolić, I.: Rotational Cryptanalysis of ARX. In: Hong, S., Iwata, T. (eds.) FSE 2010. LNCS, vol. 6147, pp. 333–346. Springer (2010)
21. Knuth, D.E.: The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1, vol. 4A. Addison-Wesley, Upper Saddle River, New Jersey (2011), http://www-cs-faculty.stanford.edu/~uno/taocp.html
22. Lipmaa, H., Moriai, S.: Efficient Algorithms for Computing Differential Properties of Addition. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 336–350. Springer (2001)

23. Mate Soos: CryptoMinisat (2009–2014), http://www.msoos.org/cryptominisat2
24. Mouha, N., Preneel, B.: Towards Finding Optimal Differential Characteristics for ARX: Application to Salsa20. Cryptology ePrint Archive, Report 2013/328 (2013)
25. Shi, Z., Zhang, B., Feng, D., Wu, W.: Improved Key Recovery Attacks on Reduced Round Salsa20 and ChaCha. In: Kwon, T., Lee, M.K., Kwon, D. (eds.) ICISC 2012. LNCS, vol. 7839, pp. 337–351. Springer (2012)
26. Shoup, V.: A Computational Introduction to Number Theory and Algebra. Cambridge University Press, 2nd edn. (2009), http://shoup.net/ntb

A Addenda to Differential Cryptanalysis

Proof of Lemma 3. On bit level Equation 2 has the form

$$\begin{aligned} 0 &= \alpha_0 \oplus \beta_0 \oplus \gamma_0 \\ 0 &= (\alpha_i \oplus \beta_i \oplus \gamma_i) \wedge (\alpha_{i-1} \oplus 1) \wedge (\beta_{i-1} \oplus 1), \quad i > 0 \end{aligned}$$

Obviously, the least significant bits (i.e. $i = 0$) are identical for Equations 1 and 2. For $i > 0$ let $t = (\alpha_i \oplus \beta_i \oplus \gamma_i) \oplus (\alpha_{i-1} \wedge \beta_{i-1})$. If $t = 0$ then Equation 1 has always the solution $x_{i-1} = y_{i-1} = 0$. Otherwise, if $t = 1$, Equation 1 is only solvable if $\alpha_{i-1} = 1$ or $\beta_{i-1} = 1$, and these are exactly the cases captured in Equation 2.

Proof of Lemma 5. Without loss of generality we assume that $\alpha \neq 0$ or $\beta \neq 0$. Looking at Equation 1, we see that the term $(\alpha \oplus \beta \oplus \gamma)$ has no effect on the probability of the differential δ, since it does not depend on either x or y. It has therefore probability 1.

Analysing the bit level representation of Equation 1, we observe that the term $(x_{i-1} \wedge \alpha_{i-1}) \oplus (y_{i-1} \wedge \beta_{i-1}) \oplus (\alpha_{i-1} \wedge \beta_{i-1})$ is balanced (i.e., is 1 with probability 1/2) if $\alpha_{i-1} = 1$ or $\beta_{i-1} = 1$. Therefore, under the assumption of independence of α_i and β_i, the overall probability of δ can be computed by counting the number of 1s in the first $n - 1$ bits of $\alpha \vee \beta$ or, equivalently, of $(\alpha \vee \beta) \ll 1$, which proves the lemma.

Proof of Lemma 7. It is easy to see that the least significant bits (i.e. $i = 0$) of Equations 3 and 4 are the same. Therefore, we will consider them no longer. Looking at the bit level representation of Equation 3 (for $i > 0$) we consider two cases:

- $\alpha_i \oplus \beta_i \oplus \gamma_i = 0$: Here, Equation 3 has always the solution $x_{i-1} = y_{i-1} = 0$.
- $\alpha_i \oplus \beta_i \oplus \gamma_i = 1$: In this case, the bit level representation of Equation 3 is only solvable if either $\alpha_{i-1} \neq \gamma_{i-1}$ or $\beta_{i-1} \neq \gamma_{i-1}$. Furthermore, the bit level representation of Equation 4 is given by

  $$(\alpha_i \oplus \beta_i \oplus \gamma_i) \wedge (\alpha_{i-1} \oplus \gamma_{i-1} \oplus 1) \wedge (\beta_{i-1} \oplus \gamma_{i-1} \oplus 1) = 0, \qquad i > 0$$

  It is evident that the latter equation only holds if $(\alpha_i \oplus \beta_i \oplus \gamma_i) = 0$, $\alpha_{i-1} \neq \gamma_{i-1}$, or $\beta_{i-1} \neq \gamma_{i-1}$. As seen above, these are the very same conditions that define a H-differential.

Proof of Lemma 9. The claim can be proven analogously to Lemma 5. It follows from the fact that in the bit level representation of Equation 3 the expression

$$(x_{i-1} \wedge (\alpha_{i-1} \oplus \gamma_{i-1})) \oplus (y_{i-1} \wedge (\beta_{i-1} \oplus \gamma_{i-1}))$$

is balanced if $\alpha_{i-1} \oplus \gamma_{i-1} = 1$ or $\beta_{i-1} \oplus \gamma_{i-1} = 1$.

B CVC Code

Below we show exemplarily for NORX64 how to translate the differential search operations to the CVC language. Variables have the datatype `BITVECTOR(W)`, where $W = 64$ is the wordsize.

$0 = (\alpha \oplus \beta \oplus \gamma) \wedge (\neg((\alpha \vee \beta) \ll 1))$	`ASSERT(0 = BVXOR(BVXOR(`α`,`β`),`γ`) & (-(((`α` | `β`)≪1)[63:0])));`
$w = (\alpha \vee \beta) \ll 1$	`ASSERT(w = (((`α` | `β`)≪1)[63:0]));`
$\gamma = (\alpha \oplus \beta) \ggg 8$	`ASSERT(`γ` = (BVXOR(`α`,`β`)≫8)|((BVXOR(`α`,`β`) ≪56)[63:0]));`

Computation of $\mathsf{hw}(w)$ using helper variables $h_0, \ldots, h_5$, where $\mathsf{hw}(w) = h_5$:

`ASSERT(`m_1` = 0x5555555555555555);`	`ASSERT(`h_0` = BVPLUS(64,(`w` & `m_1`), (((`w`≫1)[63:0]) & `m_1`)));`
`ASSERT(`m_2` = 0x3333333333333333);`	`ASSERT(`h_1` = BVPLUS(64,(`h_0` & `m_2`), (((`h_0`≫2)[63:0]) & `m_2`)));`
`ASSERT(`m_4` = 0x0f0f0f0f0f0f0f0f);`	`ASSERT(`h_2` = BVPLUS(64,(`h_1` & `m_4`), (((`h_1`≫4)[63:0]) & `m_4`)));`
`ASSERT(`m_8` = 0x00ff00ff00ff00ff);`	`ASSERT(`h_3` = BVPLUS(64,(`h_2` & `m_8`), (((`h_2`≫8)[63:0]) & `m_8`)));`
`ASSERT(`m_{16}` = 0x0000ffff0000ffff);`	`ASSERT(`h_4` = BVPLUS(64,(`h_3` & `m_{16}`), (((`h_3`≫16)[63:0]) & `m_{16}`)));`
`ASSERT(`m_{32}` = 0x00000000ffffffff);`	`ASSERT(`h_5` = BVPLUS(64,(`h_4` & `m_{32}`), (((`h_4`≫32)[63:0]) & `m_{32}`)));`

C Selected Differentials

C.1 Experimental Verification of NODE

The first table shows the results from our verification of NODE, see Section 3.3. Notation is used as follows. w_e: expected weight, $\#S$: number of samples, v_e: expected value of input/output pairs adhering the differential, v_m: measured value of input/output pairs adhering the differential, w_m: measured weight. After that we list the differentials in 32- and 64-bit $\mathsf{F}^{1.5}$ that we used to perform the verification.

			NORX32			NORX64		
w_e	$\#S$	v_e	v_m	$v_m - v_e$	w_m	v_m	$v_m - v_e$	w_m
12	2^{28}	65536	65652	+116	11.997	65627	+91	11.997
13	2^{29}	65536	65788	+252	12.994	65584	+48	12.998
14	2^{30}	65536	65170	−366	14.008	65476	−60	14.001
15	2^{31}	65536	65441	−95	15.002	65515	−21	15.000
16	2^{32}	65536	65683	+147	15.996	65563	+27	15.999
17	2^{33}	65536	65296	−240	17.005	65608	+72	16.998
18	2^{34}	65536	65389	−147	18.003	65565	+29	17.999

δ_0	δ_1	w
00000000 00000400 80000080 80000000	00000000 00000000 00000000 80001000	12
00000000 80000400 80000080 00000000	00000000 00000000 00000000 21012100	
00000000 80000000 80808080 80000000	00000000 00000000 00000000 10808080	
00000000 80000000 80800000 80000080	00000000 00000000 00000000 10008080	
80000000 00000000 00000400 80000180	80001000 00000000 00000000 00000000	13
00000000 00000000 80000400 80000080	21012100 00000000 00000000 00000000	
80000000 00000000 80000000 80808080	10808080 00000000 00000000 00000000	
80000080 00000000 80000000 80800000	10008080 00000000 00000000 00000000	
80000080 80000000 00000000 00000400	00000000 80001000 00000000 00000000	14
80000180 00000000 00000000 80000400	00000000 21012100 00000000 00000000	
80808080 80000000 00000000 80000000	00000000 10808080 00000000 00000000	
80800000 80000080 00000000 80000000	00000000 10008080 00000000 00000000	
00000400 80000000 00000400 40100000	00100000 00000000 00000000 00000000	15
80000400 80000000 00000000 00100200	00200021 00000000 00000000 00000000	
80000000 80018000 00000400 00000000	80000010 00000000 00000000 00000000	
80000000 00800000 00040400 40000600	00000010 00000000 00000000 00000000	
00000400 80000080 80000000 00000000	00000000 00000000 80003000 00000000	16
80000400 80000080 00000000 00000000	00000000 00000000 63016100 00000000	
80000000 81808080 80000000 00000000	00000000 00000000 31808080 00000000	
80000000 80800000 80000080 00000000	00000000 00000000 30008080 00000000	
00000000 00000400 80000080 80000000	00000000 00000000 00000000 80001000	17
00000000 80000400 80000080 00000000	00000000 00000000 00000000 21012100	
00000000 80000000 80838780 80000000	00000000 00000000 00000000 10808080	
00000000 80000000 80800000 80000080	00000000 00000000 00000000 10008080	
00000400 00000000 80000000 C0000200	00100000 00000000 00000000 00606001	18
80000400 00000000 00000000 00000200	00200021 00000000 00000000 C24242C0	
80000000 00000000 80000000 00000000	80000010 00000000 00000000 61010160	
80000000 00000000 80000080 C0000000	00000010 00000000 00000000 60010160	

δ_0	δ_1	w
8000000000000000 0000000000000000 0000000000040000 8000000000000080	8000001000000000 0000000000000000 0000000000000000 0000000000000000	12
0000000000000000 0000000000000000 8000000000040000 8000000000000080	2100002001010000 0000000000000000 0000000000000000 0000000000000000	
8000000000000000 0000000000000000 8000000000000000 8000808000000080	1080000000808000 0000000000000000 0000000000000000 0000000000000000	
8000000000000080 0000000000000000 8000000000000000 0080808000000000	1000000000808000 0000000000000000 0000000000000000 0000000000000000	
4000001000000000 0000000000040000 8000000000000000 0000000000040000	0000000000000000 0000100000000000 0000000000000000 0000000000000000	13
0000001000020000 8000000000040000 8000000000000000 0000000000000000	0000000000000000 0000200000000021 0000000000000000 0000000000000000	
0000000000000000 8000000000000000 8000008000000000 0000000000040000	0000000000000000 8000000000000010 0000000000000000 0000000000000000	
4000000000020000 8000000000000000 0000800000000000 0000000004040000	0000000000000000 0000000000000010 0000000000000000 0000000000000000	
0000000000040000 8000000000000080 8000000000000000 0000000000000000	0000000000000000 0000000000000000 8000001000000000 0000000000000000	14
8000000000040000 8000000000000080 0000000000000000 0000000000000000	0000000000000000 0000000000000000 2100002001010000 0000000000000000	
8000000000000000 8003808000000080 8000000000000000 0000000000000000	0000000000000000 0000000000000000 1080000000808000 0000000000000000	
8000000000000000 0080800000000000 8000000000000080 0000000000000000	0000000000000000 0000000000000000 1000000000808000 0000000000000000	
0000000000000000 000000000C0000 8000000000000080 8000000000000000	0000000000000000 0000000000000000 0000000000000000 8000001000000000	15
0000000000000000 8000000000040000 8000000000000080 0000000000000000	0000000000000000 0000000000000000 0000000000000000 2300006001010000	
0000000000000000 8000000000000000 8000808000000080 8000000000000000	0000000000000000 0000000000000000 0000000000000000 1180000000808000	
0000000000000000 8000000000000000 0080800000000000 8000000000000080	0000000000000000 0000000000000000 0000000000000000 1000000000808000	
0000000000040000 4000001000080000 0000000000040000 8000000000000000	0000000000000000 0000000000000000 0000100000000000 0000000000000000	16
0000000000000000 0000001000020000 8000000000040000 8000000000000000	0000000000000000 0000000000000000 0000200000000021 0000000000000000	
0000000000040000 0000000000000000 8000000000000000 8000008000000000	0000000000000000 0000000000000000 8000000000000010 0000000000000000	
0000000004040000 C000000000E0000 8000000000000000 0000800000000000	0000000000000000 0000000000000000 0000000000000010 0000000000000000	
8000000000000080 8000000000000000 0000000000000000 0000000000040000	0000000000000000 8000007000000000 0000000000000000 0000000000000000	17
8000000000000080 0000000000000000 0000000000000000 8000000000040000	0000000000000000 E300006001010000 0000000000000000 0000000000000000	
8000808000000180 8000000000000000 0000000000000000 8000000000000000	0000000000000000 7180000000808000 0000000000000000 0000000000000000	
0080800000000000 8000000000000080 0000000000000000 8000000000000000	0000000000000000 7000000000808000 0000000000000000 0000000000000000	
0000000000040000 8000000000000000 0000000000040000 400000F000000000	0000100000000000 0000000000000000 0000000000000000 0000000000000000	18
8000000000040000 8000000000000000 0000000000000000 0000001000020000	0000200000000021 0000000000000000 0000000000000000 0000000000000000	
8000000000000000 8000008000000000 0000000000040000 0000000000000000	8000000000000010 0000000000000000 0000000000000000 0000000000000000	
8000000000000000 0000800000000000 000000000C040000 4000000000020000	0000000000000010 0000000000000000 0000000000000000 0000000000000000	

C.2 Probability-1 Differentials in G

Using NODE we could show that there are exactly 3 probability-1 differentials in both versions (32- and 64-bit) of G.

Differences	
δ_0	80000000 80000000 80000000 00000000
δ_1	00000000 00000001 80000000 00000000
δ_0	80000000 00000000 80000000 80000080
δ_1	80000000 00000000 00000000 00000000
δ_0	00000000 80000000 00000000 80000080
δ_1	80000000 00000001 80000000 00000000

Differences	
δ_0	8000000000000000 8000000000000000 8000000000000000 0000000000000000
δ_1	0000000000000000 0000000000000001 8000000000000000 0000000000000000
δ_0	8000000000000000 0000000000000000 8000000000000000 8000000000000080
δ_1	8000000000000000 0000000000000000 0000000000000000 0000000000000000
δ_0	0000000000000000 8000000000000000 0000000000000000 8000000000000080
δ_1	8000000000000000 0000000000000001 8000000000000000 0000000000000000

C.3 Best Differential Characteristics for F^4

The following two tables show the best differential characteristics in F^4 that we were capable to find with NODE. The values δ_0 and δ_4 are in- and output difference, respectively, and δ_1, δ_2, and δ_3 are internal differences. The differences are listed after a single application of F, respectively, and the values w_i, with $i \in \{0, \ldots, 3\}$, are the corresponding differential weights.

δ_0	w_0	δ_1	w_1
80140100 90024294 84246020 92800154	172	40100000 00000400 80000000 00000400	11
e4548300 52240214 e0202424 d0004054		00100200 80000400 80000000 00000000	
c4464046 00a08480 c1008108 90d43134		00000000 80000000 80008000 00000400	
e200c684 e2eac480 a4848881 06915342		40000200 80000000 00800000 00040400	
δ_2	w_2	δ_3	w_3
00000000 00000000 00000000 00000000	44	04042425 00100002 00020000 02100000	357
00000000 00000000 00000000 00000000		04200401 42024200 20042024 20042004	
00000000 80000000 00000000 00000000		10001002 80000200 25250504 10021010	
00000000 00000000 00000000 00000000		10020010 00001002 00000210 04252504	
δ_4			
c4001963 804da817 0c05b60e 12220503	total weight: 584		
9072b909 185b792a cc0d56cd 7e0ac646			
80116300 100c2800 8f003320 3b270222			
01056104 88000041 92002824 04210001			

δ_0	w_0	δ_1	w_1
00900824010288c5 4000443880011086 224012044220ac43 e004044484049520	349	8000000800050000 8000000000000000 4000000000000000 0000001000020080	27
4080882001010885 4600841880821086 a3c0721444632c43 c224440007849504		8000000800040000 8000000000000000 c000000000040000 8000001000020080	
81600850830b0484 840080c080868000 8004449040c14400 8102101840908a80		0000000000000000 8000008000000000 c000004000040000 4000808000020080	
6191548c08000581 0200004006038044 8104f01c8702c0e0 60605084938886e3		0000000000010080 0000800000000000 8000400004040000 8080800020000c0	
δ_2	w_2	δ_3	w_3
8000000000000000 0000000000000000 0000000000000000 0000000000000000	12	0000000000000000 0000000000000000 0000100000000000 0000202000000001	448
8000000000000000 0000000000000000 0000000000000000 0000000000000000		4200404002020040 0000000000000000 0000000000000000 0000200000000021	
8000000000000000 0000000000000000 0000000000000000 0000000000000000		8000000000000010 2100000001010020 0000000000000000 0000000000000000	
0000000000000000 0000000000000000 0000000000000000 0000000000000000		0000000000000000 0000000000000010 2000000001010020 0000000000000000	
δ_4			
321a4500060e4e2e 27404405026e500e 3806422387200a08 8c40f4a0884c0820	total weight: 836		
71540fb858cb9902 ee018cc282747980 c714164174ce3eb9 1a49a091101191e1			
786680d0e46406cb 14440844013274e6 03a843203f071b7c 09a840c00c0ccc78			
4000404a22120005 07220c4202016240 2aa4200a0a041a62 84a468682000601c			

C.4 Best Iterative Differentials for F

	Differences	w
δ_1	818c959b 00186049 eb5b7984 791c6da1	512
$=$	677b513d 80000400 00000227 5293655f	
δ_0	00809a2b bfa98bff c08b8e89 0000711c	
	800027c3 f984eb5b 6d81f915 b5aaa99d	

	Differences	w
δ_1	0000000100000000 0000000000000000 f77c78b200000d04 0000000000000000	843
$=$	be7fffeffe0f349f 0000000000000000 6c07fbd200000001 ff1ab5be4e7500be	
δ_0	0060c54927018000 0000000000000000 0000000000000000 b603fde900000000	
	b6035caf00000000 0000000000000000 0000000000000000 0000000000000000	

C.5 Best Differentials having Equal Columns of weight 44 in F

	Differences
δ_0	80000000 80000000 80000000 80000000
	80000000 80000000 80000000 80000000
	80000000 80000000 80000000 80000000
	00000000 00000000 00000000 00000000
δ_1	00102001 00102001 00102001 00102001
	42624221 42624221 42624221 42624221
	a1010110 a1010110 a1010110 a1010110
	20010110 20010110 20010110 20010110

	Differences
δ_0	8000000000000000 8000000000000000 8000000000000000 8000000000000000
	8000000000000000 8000000000000000 8000000000000000 8000000000000000
	8000000000000000 8000000000000000 8000000000000000 8000000000000000
	0000000000000000 0000000000000000 0000000000000000 0000000000000000
δ_1	0000102000000001 0000102000000001 0000102000000001 0000102000000001
	4200604002020021 4200604002020021 4200604002020021 4200604002020021
	a100000001010010 a100000001010010 a100000001010010 a100000001010010
	2000000001010010 2000000001010010 2000000001010010 2000000001010010

D Addenda to Rotational Cryptanalysis

Proof of Lemma 11. After evaluating and simplifying the equation $\mathsf{H}(x, y) \ggg r = \mathsf{H}(x \ggg r, y \ggg r)$ we get $((x \wedge y) \ll 1) \ggg r = ((x \ggg r) \wedge (y \ggg r)) \ll 1$. Translating this equation to bit vectors results in

$$\begin{aligned}&(x_{r-1} \wedge y_{r-1}, \ldots, x_0 \wedge y_0, 0, x_{n-2} \wedge y_{n-2}, \ldots, x_r \wedge y_r)\\ =\ &(x_{r-1} \wedge y_{r-1}, \ldots, x_0 \wedge y_0, x_{n-1} \wedge y_{n-1}, x_{n-2} \wedge y_{n-2}, \ldots, 0)\end{aligned}$$

The probability that those two vectors match is $(3/4)^2 = 9/16$, as $a \wedge b = 0$ with probability $3/4$ for bits a and b chosen uniformly at random.

Proof of Lemma 13. The first important observation is that the statement of this lemma is independent of the function f, as it only makes a claim on the image of f. Thus it is sufficient to prove the lemma for $z \ggg r = z$, where $z = f(x, y)$ and x or y was fixed.

We identify the indices of an n-bit string by the elements in $G := \mathbb{Z}/n\mathbb{Z}$. Let $\tau : G \longrightarrow G,\ i \bmod n \mapsto (i + 1) \bmod n$. Then τ obviously generates the cyclic group G, i.e. $\mathrm{ord}(\tau) = n$. Moreover, for an arbitrary $r \in \mathbb{Z}$ we have $\mathrm{ord}(\tau^r) = n/\gcd(r, n)$, see [26, §§6.2]. In other words, the subgroup $H := \langle \tau^r \rangle$ of G has order $n/\gcd(r, n)$. By Lagrange's theorem we have $\mathrm{ord}(G) = [G : H] \cdot \mathrm{ord}(H)$ and it follows for the group index $[G : H] = \gcd(r, n)$, which corresponds to the number of (left) cosets of H in G. These cosets contain the indices of a bit string which are mapped onto each other by a rotation $\ggg r$. This means that there are $2^{\gcd(r,n)}$ n-bit strings z which satisfy $z \ggg r = z$. Thus the probability, that an n-bit string z, chosen uniformly at random among all n-bit strings, satisfies $z \ggg r = z$ is $2^{-(n-\gcd(r,n))}$. This proves the lemma.