Cryptanalysis of Reduced NORX

Nasour Bagheri[1], Tao Huang[2], Keting Jia[3,4], Florian Mendel[5] and Yu Sasaki[2,6]

[1] SRTTU and IPM, Iran
[2] Nanyang Technological University, Singapore
[3] Department of Computer Science and Technology, Tsinghua University, China
[4] State Key Laboratory of Cryptology, P.O.Box5159, Beijing 100878, China
[5] Graz University of Technology, Austria
[6] NTT Secure Platform Laboratories, Japan
sasaki.yu@lab.ntt.co.jp

Abstract. NORX is a second round candidate of the ongoing CAESAR competition for authenticated encryption. It is a nonce based authenticated encryption scheme based on the sponge construction. Its two variants denoted by NORX32 and NORX64 provide a security level of 128 and 256 bits, respectively. In this paper, we present a state/key recovery attack for both variants with the number of rounds of the core permutation reduced to 2 (out of 4) rounds. The time and data complexities of the attack for NORX32 are 2^{119} and 2^{66} respectively, and for NORX64 are 2^{234} and 2^{132} respectively, while the memory complexity is negligible. Furthermore, we show a state recovery attack against NORX in the parallel mode using an internal differential attack for 2 rounds of the permutation. The data, time and memory complexities of the attack for NORX32 are $2^{7.3}$, $2^{124.3}$ and 2^{115} respectively and for NORX64 are $2^{6.2}$, $2^{232.8}$ and 2^{225} respectively. Finally, we present a practical distinguisher for the keystream of NORX64 based on two rounds of the permutation in the parallel mode using an internal differential-linear attack. To the best of our knowledge, our results are the best known results for NORX in nonce respecting manner.

Keywords: Authenticated encryption, CAESAR, NORX, Guess and determine, Internal differential attack, State recovery, Nonce respect.

1 Introduction

Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) [1] is a competition for designing authenticated encryption schemes. 57 algorithms were submitted to the first round of this competition. After over a year of analysis, the CAESAR committee announced 29 schemes as the second round candidates. NORX [2, 5] is one of them. It is a sponge based scheme which uses a permutation as its core, supports associated date and does not allow nonce to be repeated (known as nonce respecting). To provide efficiency in wide range of platforms, the only operations used in the core permutation of NORX are AND, XOR, rotation and shift. NORX consists of two

variants denoted by NORX32 and NORX64. NORX32 provides 128-bit security and NORX64 provides 256-bit security. An interesting feature of NORX, compared to some other sponge based candidates of CAESAR such as for instance Ascon [13] and Keyak [8], is its level of parallelism. More precisely, NORX's designers proposed a parallel mode that enables users to process several message blocks in parallel. This feature could be interesting in applications that need high throughput, e.g. video streaming.

Among the second round candidates of CAESAR, NORX is one of the fastest [16] and uses simple bitwise operations which makes it a good candidate for a wide range of platforms, assuming that it provides the desired security. On the other hand, no previous security analysis of NORX as a full AEAD, e.g. integrity and confidentiality, is known and the only known results [4, 11] more dedicated to the permutation of NORX rather than the application of the permutation in the mode.

Related work. In [4], Aumasson *et al.* analysed the differential property of the core permutation of NORX. They provided upper bounds on the differential probability for the reduced permutation. More precisely, by assuming that an attacker can only modify the nonce during initialisation, any single round differential characteristic has probabilities of less than 2^{-60} (for NORX32) and 2^{-53} (for NORX64). They extended their results to full (four) round permutation and showed that the best characteristics for four rounds have probabilities of 2^{-584} and 2^{-836} for NORX32 and NORX64, respectively.

In [11], Das *et al.* analysed the higher order differential properties of the core permutation of NORX. These results cover more rounds compared to the first order differential analysis provided in [4]. More precisely, they identified the higher order differential properties that allow practical distinguisher of the full round permutation of NORX64 and 3.5-round permutation of NORX32. The used approach is similar to zero-sum distinguishers [6], but it is probabilistic.

Although the results in [4, 11] can reach full rounds, it seems hard to exploit them to break integrity or confidentiality of NORX. In particular, the attacker's ability to control difference for the core permutation is significantly limited in the nonce respecting setting.

Our contribution. In this paper, we present several cryptanalysis against reduced-round NORX with respect to security notions claimed by the designers; recovering key or breaking confidentiality in the nonce respecting setting. We discuss two different types of attacks; guess and determine attack and internal differential attack. The attack results are summarised in Table 1.

Guess and determine attack is a widely used technique in analyzing stream ciphers and authenticated encryption schemes. The attack by Dinur and Jean [12] against authenticated encryption FIDES [10] is an example. The attacker first learns a part of the internal state values leaked from a plaintext-ciphertext pair. Then he partially guesses the hidden part of the state and recovers as many other state bits as possible. Since the NORX core permutation is invertible, recovering

Table 1. Summary of our attacks. "KR" represents "key recovery" and "KD" represents "keystream distinguisher".

Approach	Goal	Target	Rounds	Data	Time	Memory	Ref.
Guess and determine	KR	NORX64	2/4	2^{132}	2^{234}	negl.	Sect. 4
Guess and determine	KR	NORX32	2/4	2^{66}	2^{119}	negl.	Sect. 4
Internal difference	KR	NORX64	2/4	74	$2^{232.8}$	2^{225}	Sect. 5.2
Internal difference	KR	NORX32	2/4	158	$2^{124.3}$	2^{115}	Sect. 5.2
Internal differential-linear	KD	NORX64	2/4	90	negl.	negl.	Sect. 5.3

the internal state immediately allows to recover the secret key. We first describe a simple guess and determine attack that works up to 1.5 (out of 4) rounds of NORX. Then we show how to extend the attack to 2 rounds with the method of solving linear equations. Our attacks works for both NORX32 and NORX64.

While differential cryptanalysis [9] is generally difficult to apply in the nonce respecting setting, Jean *et al.* [14] have recently showed that difference between two parallel computations under the same nonce, could be exploited by the attacker and have applied it to fully parallelizable block cipher based scheme Silver [17]. The approach is called internal differential attack [18]. On the other hand, sponge based schemes generally have the serial structure, thus internal difference does not exist. However, it is still possible to introduce parallel computation to the sponge based schemes, and NORX is one of such designs. Hence, we mount an attack by exploiting the difference between two computations in the parallel mode of NORX. In the parallel mode, the same internal state is first duplicated, and the counter value, $0, 1, 2, \cdots$, is XORed to each state to make them distinct. Here, we focus on the very low Hamming wight difference caused by the counter values, which leads to high probability multiple differentials for 1 round. Using these differentials the internal state of the NORX with the permutation reduced to two rounds can be recovered. Moreover, we use the slow diffusion property of the NORX round function to present a practical distinguisher for the keystream of NORX64 with a permutation reduced to two rounds in parallel mode. This attack employs a deterministic truncated differential in forward direction for 1.5 round of the NORX64's permutation, followed by a probabilistic linear attack for a halve round of the permutation in backward direction.

Outline. The rest of the paper is organized as follows: in Sect. 2 we provide the required notations and also describe NORX as much as necessary for our analysis. In Sect. 3, we list several useful properties of the core permutation. In Sect. 4 we present a guess and determine attack. Our internal differential attack is described in Sect. 5. We present our distinguisher for NORX64 in Sect. 5.3. Finally, we provide closing remarks in Sect. 6.

2 Preliminaries

2.1 Notation

In this paper we mostly follow the notation used by the designer of NORX [5]. Depends on the context, a word is either a 32-bit or a 64-bit bit-string. The state of NORX is generally denoted by S. Each state includes 16 words and we denote the i-th word of S by s_i, for $0 \leq i \leq 15$. If we specify the state right after round j of the permutation, we denote it by S^j and denote its i-th word by s_i^j and the z-th bit of that word by $s_i^j[z]$. In the parallel mode of NORX, there are more than one lanes to process message blocks. In this case we denote the state of all lanes by $\bar{S}$ and we denote the state of the i-th lane by $\bar{S}_i$ and the j-th word of the i-th lane by $\bar{s}_{i,j}$. In general, we denote truncation of bit-string x from the i-th bit toward least significant bit (LSB) and up to the j-th bit toward most significant bit (MSB) by $x[j \sim i]$. The i-th bit of x is denoted by $x[i]$.

To denote bitwise AND, OR and XOR we use $\wedge$, $\vee$ and $\oplus$ respectively. By $x \ll n$, $x \gg n$, $x \lll n$ and $x \ggg n$ we denote left-shift, right-shift, left-rotate and right-rotate of bit-string x by n bits.

We use Δx to denote the difference in bit-string x and x', i.e. $\Delta x = x \oplus x'$.

By M_i and C_i we denote the i-th block of plaintext (message) and ciphertext. The nonce and the secret key are denoted by N and K respectively and their i-th words are denoted by n_i and k_i, respectively.

2.2 Specification of NORX

NORX [5] is a monkeyDuplex construction [7] based AEAD. It uses a $16w$-bit to $16w$-bit permutation, parameterized by a word size $w \in \{32, 64\}$. It has two variants denoted by NORX32 (where $w = 32$) and NORX64 (where $w = 64$). NORX is also parameterized by a parallelism degree $0 \leq p \leq 255$, number of rounds $0 \leq l \leq 63$ and a tag size $t \leq 4w$ and is denoted as NORXw-l-p-t. A high level representation of the NORX construction for $p = 1$ (serial mode of operation) and $p = 2$ (parallel mode of operation with two parallel lanes) are represented in Fig. 1 and Fig. 2 respectively. In these figures we have not considered the processes related to any auxiliary data, e.g. associated data.

The state S of NORXw consists of sixteen words $s_0, \ldots, s_{15}$, each of size w bits. The state's words $s_0, \ldots, s_{11}$ are called the rate words and the state's words $s_{12}, \ldots, s_{15}$ are called the capacity words. In each iteration of the permutation a block of message or associated data is XORed with the rate fraction of the state and a domain septation constant is XORed with the capacity fraction of the state (more details can be found in [5]). NORXw initiates the state by predefined constant $U = u_0 \| \ldots \| u_9$, the nonce $N = n_0 \| n_1$ and the key $K = k_0 \| \ldots \| k_3$. The matrix representation of S and the rule of assigning the words of constants, nonce and key in the initialization phase are as follows:

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

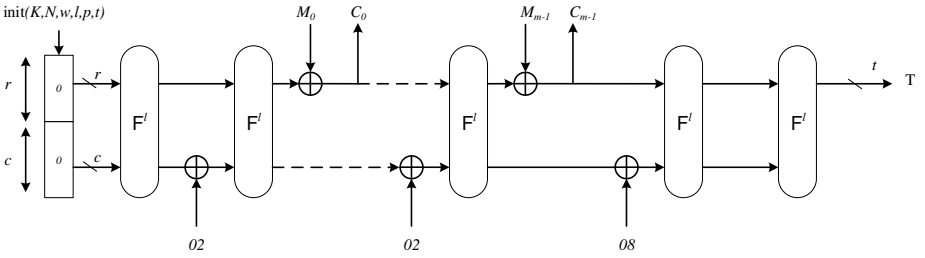

Fig. 1. The layout of NORX construction for $p = 1$ (fully serial) [5], where F^l denotes an l-round permutation of NORX.

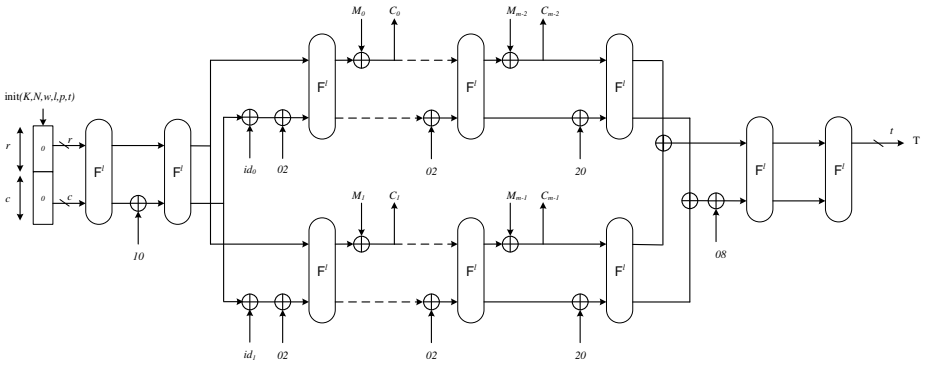

Fig. 2. The layout of NORX construction for $p = 2$ (include two parallel lanes) [5].

More details on the constants can be found in [2] but it does not affect our results. Each round of the permutation, called F, includes the application of a function called G to each column of state followed by applying it to each diagonal of state. Hence $\mathsf{F}(s_0, \ldots, s_{15})$ consists of column steps as follows:

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

followed by the following diagonal steps:

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

The function $\mathsf{G}(a, b, c, d)$ computes the following 8 operations:

$$\begin{array}{llll} 1.\ a = \mathsf{H}(a, b), & 2.\ d = (a \oplus d) \ggg r_0, & 3.\ c = \mathsf{H}(c, d), & 4.\ b = (b \oplus c) \ggg r_1, \\ 5.\ a = \mathsf{H}(a, b), & 6.\ d = (a \oplus d) \ggg r_2, & 7.\ c = \mathsf{H}(c, d), & 8.\ b = (b \oplus c) \ggg r_3, \end{array}$$

where $\mathsf{H}(x, y) = (x \oplus y) \oplus \big((x \wedge y) \ll 1\big)$. The rotation offsets (r_0, r_1, r_2, r_3) are $(8, 11, 16, 31)$ for NORX32 and $(8, 19, 40, 63)$ for NORX64.

It must be noted, to enhance the performance, compared to NORX V1 originally submitted to CAESAR [3], designers have excluded s_{10} and s_{11} from

capacity words and appended them to the rate words in NORX V2.0 tweaked for the second round [5]. This tweak has not changed the security claim. The security claims for both of integrity and confidentiality are 128 bits for NORX32 and 256 bits for NORX64 [5, Table 3.1].

Parallel mode of NORX. As depicted in Fig. 2 NORX supports parallel message processing, $p > 1$. In this case the state S is extended to a multi-state vector $\bar{S}$, where $\bar{S}_i$ indicates the input sate of the i-th lane. To ensure that the input state to each lane is a unique string, a counter i which indicates the lane number updates $\bar{s}_{i,13}$ to $\bar{s}_{i,13} \oplus i$. Among 5 recommended parameters by the designers [5], only 1 parameter, NORX64-4-4-256 supports the parallel mode.

3 Properties of Round Function

In this section, we show several properties of round function G that allow us to exploit relatively slow diffusion in backward direction, i.e. G^{-1}.

Computing G^{-1}. We begin with the observation that G^{-1} cannot be computed trivially. To compute G^{-1}, it is necessary to find x for a given pair of $\mathsf{H}(x, y)$ and y. We argue that this can be efficiently computed bit-by-bit from the LSB. Considering $\ll 1$, computation for the LSB is a simple XOR, thus $x[0]$ is computed by $\mathsf{H}(x, y)[0] \oplus y[0]$. After $x[0]$ is fixed, $x[1]$ can be computed similarly, and then the entire x is eventually computed. The cost of G^{-1} should be higher than G. For the sake of simplicity, we assume that the cost for G and G^{-1} are identical.

Computing G^{-1} with partially known state. For sponge-based AE schemes the rate words can be in general be recovered from a plaintext-ciphertext pair. For NORX, s_0 to s_{11} are known and s_{12} to s_{15} are secret. This motivates us to consider tracing known bit positions of $(a, b, c, d) \leftarrow \mathsf{G}^{-1}(a', b', c', d')$ when three words of a', b', c', d' are known.

- b can be computed from given b', c', d' by G^{-1}.
- c can be computed from given a', c', d' by G^{-1}.

These simple properties can be extended so that several bits of the unknown word is known. This corresponds to the situation that attackers guess several bits of s_{12} to s_{15} during the attack. We again begin with analyzing H^{-1} with partially known $x' = \mathsf{H}(x, y)$ and y.

Property 1. From all bits of x' and t consecutive bits of y, the corresponding t bits of x can be computed with a cost of 1-bit guess. Moreover, the guess is not necessary when t consecutive bits start from the LSB.

The property is depicted in Fig. 3. Attackers need to guess 1 bit of $(x \wedge y) \ll 1$ which is the lowest bit of consecutive t bits in y. Based on the guess, t consecutive bits of y can be recovered bit-by-bit. When t consecutive bits of x' are located in the LSB, t bits of x can be computed uniquely without guess.

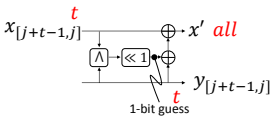

Fig. 3. Computing H^{-1} with partially known y.

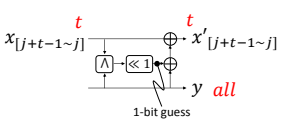

Fig. 4. Computing H^{-1} with partially known x'.

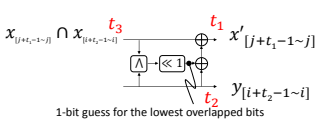

Fig. 5. Computing H^{-1} with partially known x' and y.

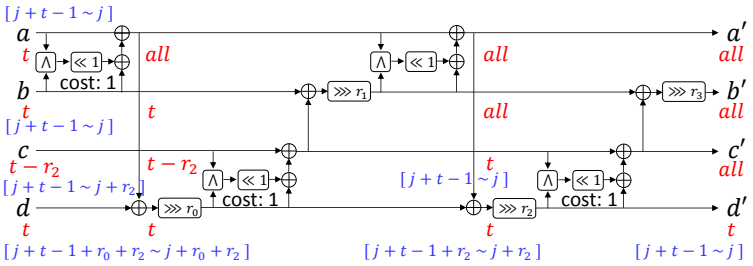

Fig. 6. Property 4: G^{-1} with partially known d'.

Property 2. From t consecutive bits of x' and all bits of y, the corresponding t bits of x can be computed with a cost of 1-bit guess. Moreover, the guess is not necessary when t consecutive bits start from the LSB.

Property 3. Suppose that t_1 consecutive bits of x' and t_2 consecutive bits of y are known. Then, the corresponding x in overlapped bit positions can be computed with a cost of 1-bit guess.

Property 2 and Property 3 are depicted in Fig. 4 and Fig. 5, respectively. The mechanism is the same as Property 1. In Property 3, attackers need to guess 1 bit of $(x \wedge y) \ll 1$, which is the lowest bit of overlapped bit positions. Then the other overlapped bits can be computed bit-by-bit.

Based on those properties of H^{-1}, we analyze the known bit positions of $(a, b, c, d) \leftarrow \mathsf{G}^{-1}(a', b', c', d')$ when three words are fully known and t bits of the last word are also known. In the property below, we assume $t > r_2$.

Property 4. Suppose that a', b', c' are fully known and t bits of d' are known. Then, t bits of a, t bits of b, $t - r_2$ bits of c, and t bits of d can be computed with a cost of 3-bit guess. Moreover, the guess is reduced to 1-bit when t consecutive bits of d' start from the LSB.

The property is depicted in Fig. 6. Numbers in red color represent the number of known bits and numbers in blue color $[n_1 \sim n_2]$ represent known bit positions. In Fig. 6, four H^{-1} functions are computed from the end to the beginning. The first H^{-1} is the case of Property 1. t bits of word c are computed with 1-bit

guess, and the guess can be omitted if the known bits start from the LSB. No difficulty exists in the second H^{-1}. The third H^{-1} is the case of Property 3. The number of known bits depends on r_2. Note that $d' \lll r_2$ is computed instead of $d' \ggg r_2$ during G^{-1}. The fourth H^{-1} is again the case of Property 1. The 1-bit guess can be omitted for the LSB case.

Property 5. Suppose that a', b', d' are fully known and t bits of c' are known. Then, $t - r_1$ bits of a, $t - r_1$ bits of b, t bits of c, and t bits of d can be computed with a cost of 4-bit guess. Moreover, the guess is reduced to 1-bit when t consecutive bits of c' start from the LSB.

Property 6. Suppose that a', c', d' are fully known and t bits of b' are known. Then, $t - r_1$ bits of a, t bits of b, all bits of c, and t bits of d can be computed with a cost of 2-bit guess.

Property 7. Suppose that b', c', d' are fully known and t bits of a' are known. Then, t bits of a, all bits of b, t bits of c, and $t - r_0$ bits of d can be computed with a cost of 3-bit guess. Moreover, the guess can be omitted when t consecutive bits of a' start from the LSB.

Extending Properties 4 to 7. The number of overlapped bits in Properties 4 to 7 can be generalized more. For example, in Property 4, $t - r_2$ bits of c can be replaced with t_3 bits of c, where t_3 takes one of the following 4 values depending on the relation of t, r_2 and the word size w.

$$
\begin{aligned}
&t_3 = 0, && \text{when } t - 1 < r_2 \text{ and } r_2 + t \leq w,\\
&t_3 = t - r_2, && \text{when } t - 1 \geq r_2 \text{ and } r_2 + t \leq w,\\
&t_3 = 2t - w, && \text{when } t - 1 \geq r_2 \text{ and } r_2 + t > w,\\
&t_3 = r_2 + t - w, && \text{when } t - 1 < r_2 \text{ and } r_2 + t > w.
\end{aligned}
$$

Note that the third case is the combination of $t - r_2$ consecutive bits and $r_2 + t - w$ consecutive bits instead of $2t - w$ consecutive known bits. Thus to preserve $2t - w$ bit vales, the cost is 2-bit guess per H^{-1}. Also note that even with the third case, preserving either of $t - r_2$ or $r_2 + t - w$ with the cost of 1-bit guess per H^{-1} is possible, and this is actually the case of Property 4 assuming $t > r_2$.

The similar extension can be applied to Properties 5 to 7. To avoid redundancy, we omit the details. Those extension may be useful for future analysis.

Recovering the input/output words of G. If 4 or more words among the 8 words $(a, b, c, d, a', b', c', d')$ of the input/output of G are known, it is possible to recover all of the other words for certain cases.

Property 8. Suppose that a, b, c, b', c' are fully known. Then, all the other words in the input/output of G can be recovered with a cost of single G^{-1} function.

Property 9. Suppose that a, a', b', c' are fully known. Then, all the other words in the input/output of G can be recovered with a cost of single G^{-1} function.

Property 10. Suppose that d, a', b', c' are fully known. Then, all the other words in the input/output of G can be recovered with a cost of single G^{-1} function.

4 Guess and Determine Attack

The encryption part of NORX leaks a large portion of the internal state, which may be exploited using a guess and determine attack. In this section, we will describe a simple guess and determine attack on 1.5 rounds NORX and then show how to extend it to 2 rounds with the method of solving linear equations.

4.1 Attack on 1.5 Rounds NORX

The 1.5 rounds NORX permutation involves four frames of internal states: the initial state, the state after the first column step, the state after the first diagonal step and the state after the second column step (the final state), which are denoted as S^0, $S^{0.5}$, S^1, and $S^{1.5}$ respectively. Thus, we refer to any bit in the computation with the notation $s_y^x[z]$, where x is the frame index, y is the word index ranged from 0 to 15, and z is the bit index ranged from 0 to $w-1$.

In the latest version of NORX [5], $s_0^0, \ldots, s_{11}^0$ and $s_0^{1.5}, \ldots, s_{11}^{1.5}$ can be easily obtained by an adversary in a known plaintext scenario. Assuming those words are known at the input and output of the permutation by querying messages with at least two blocks, then we can exploit the slow backward diffusion of round function G of NORX by guessing 3 of the 4 unknown words of the state $S^{1.5}$ and propagating the information backward to recover the full internal state. The procedure of the guess and determine attack is given below, and is also depicted in Fig. 7.

1. Guess the words $s_{12}^{1.5}, s_{13}^{1.5}, s_{14}^{1.5}$ in $S^{1.5}$. With the other known state words of $S^{1.5}$, recover $s_0^1, s_1^1, s_2^1, s_4^1, s_5^1, s_6^1, s_8^1, s_9^1, s_{10}^1, s_{12}^1, s_{13}^1, s_{14}^1$.
2. With the known values of s_4^1, s_9^1, s_{14}^1 and using Property 7, recover $s_4^{0.5}$. With the known values of s_2^1, s_8^1, s_{13}^1 and using Property 6, recover $s_8^{0.5}$.
3. Using Property 8, recover $s_{12}^0, s_0^{0.5}, s_{12}^{0.5}$ from $s_0^0, s_4^0, s_8^0, s_4^{0.5}, s_8^{0.5}$.
4. Using Property 9, recover s_{15}^1 from $s_0^{0.5}, s_0^1, s_5^1, s_{10}^1$.
5. Using Property 10, recover $s_{15}^{1.5}$ from $s_{15}^1, s_0^{1.5}, s_5^{1.5}, s_{10}^{1.5}$.

Thus, the full state of $S^{1.5}$ is recovered. We can compute backward to verify if the guessed words are correct. We want to note that this 1.5 rounds attack works for both NORX32 and NORX64 as all the operations are performed on the word level. We estimate that one guess and to determine the trail require roughly 2 NORX operations, the time complexity of the attack is 2^{97} for NORX32 and 2^{193} for NORX64.

4.2 Attack on 2 Rounds NORX

Now we will show how to extend the guess and determine attack to 2 rounds of NORX. The additional 0.5 round does not allow to establish the relation of S^0 and $S^{0.5}$ using the previous strategy, i.e. guessed words. Hence, we need to guess more bits and analyze the information propagation at bit level.

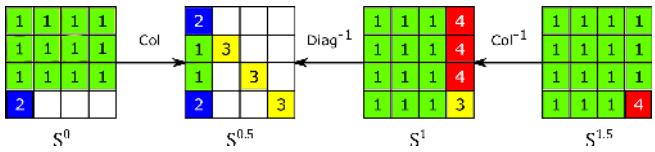

Fig. 7. Guess and determine attack on 1.5 rounds NORX. The order of known/recovered words is green(1)→blue(2)→yellow(3)→red(4).

Overview of the attack. Suppose that the five state frames of 2 rounds NORX are denoted by S^0, $S^{0.5}$, S^1, $S^{1.5}$, S^2 and the words $s_0^0, \ldots, s_{11}^0$, $s_0^2, \ldots, s_{11}^2$ are known. In addition to guess s_{12}^2, s_{13}^2, s_{14}^2, we further guess parts of the word s_{15}^2.

Next, we set all the unknown bits in S^0 as linear bits which related to themselves. With the forward computation of the NORX operations, any state bit can be classified into one of the three categories: *known bits*, *linear bits* and *non-linear bits*, such that a *known bit* can be computed with the known bits in S^0; a *linear bit* can be expressed as a linear combination of the unknown bits in S^0 XORed with some constant; and a *non-linear bit* can be expressed as non-linear combinations of the unknown bits in S^0 XORed with some constant. In this paper, We use $'u'$, $'k'$, $'l'$, $'n'$ to denote the unknown bits, known bits and linear bits, non-linear bits respectively.

Then, we do the same for the the bits in S^2 (assuming the guessed bits are known) and propagate backward.

If a bit in the internal state is linear/known in the forward computation while linear/known in the backward computation, we can establish a linear relation between the guessed/known bits in S^0 and the known bits in S^2. If there are enough linear equations, then we can solve the unknown bits in S^2, and thus recover the full internal state of S^2.

To establish more linear equations, we use a trick to increase the number of linear bits in the backward computation. In the H function, if a bit-relation $x_i' = x_i \oplus y_i \oplus (x_{i-1} \wedge y_{i-1})$ is computed, with $x_i' = 'k'$, $x_i = 'u'$, $y_i = 'l'$, $x_{i-1} = 'k'$ and $y_{i-1} = 'k'$. Then, if $x_i' = 1 \oplus (x_{i-1} \wedge y_{i-1})$, we have $x_i \wedge y_i = 0$, which is the non-linear term in computing x_{i+1}'. Note that we can control the value of any known bit by collecting more data and choose the needed ones. Hence, we can eliminate the non-linear part of the H function in many cases at the cost of increasing data complexity.

Since NORX64 and NORX32 have different word sizes, we will describe the attacks on NORX64 and NORX32 separately.

Attack on NORX64. For NORX64, we experimentally tested different choices of the guessed bits numbers and positions in s_{15}^2. To minimize the time complexity, the optimal choice we found is to guess $s_{15}^2[0, \ldots, 40]$.

We start the attack by building the linear system. Set the bits in $s_0^0, \ldots, s_{11}^0$, $s_0^2, \ldots, s_{14}^2$ and $s_{15}^2[0, \ldots, 40]$ as known bits. Set all the other unknown bits in S^0 and S^2 as linear bits. Then propagate the bit relations of S^0 forward and

bit relations of S^2 backward. In the backward propagation, we control certain values of known bits to increase the number of linear bits with the technique mentioned previously.

Two linear equations can be established on bits $s_{10}^0[0]$ and $s_{11}^0[0]$, as those bits are known in the forward direction while linear in the backward direction. The rest of the equations can be derived from the last column of $S^{0.5}$, see Table 2.

Table 2. Bit patterns of the last column of $S^{0.5}$. Red bits are used to establish linear equations.

Forward pattern
1111111111111111 1111111111111111 1111111111111111 1111111111111111
nnnnnnnnnnnnnnnn nnnnnnnnnnnnnnnn nnnnnnnnnnnnnnnn nnnnnnnnnnnnnnnn
nnnnnnnnnnnnnnnn nnnnnnnnnnnnnnnn nnnnnnnnnnnnnnnn nnnnnnnnnnnnnnnn
1111111111111111 1111111111111111 1111111111111111 1111111111111111
Backward pattern
nnnnnnnnnnnnnnnn nnnnnnnnnnnnnnnn nnnnnnn11111111k kkkkkkkkkkkkkkkk
nnnnnnnnnnnnnnnn nnnnnnnlkkkkkkkk kkkkkkkkkkkkk111 1111111111111111
nnnnnnnnnnnnnnnn nnnnnnnnnnnnnnnn nnnnnnnnnnnnnnnn nnnnnnnnnnnnnnnn
nnnnnnnnnnnnnnnn nnnnnnn11111111 1111111111111111 1111111111111111

The reason of choosing this column is that only the unknown bits in s_{15}^0 and s_{15}^2 are involved. At this point, the number of linear equations is still less than the unknown variables. To obtain more equations we make use of Property 2. By guessing $s_{11}^0[18]$ to be 0, the bits $s_{15}^0[27, \ldots, 43]$ can be recovered. This is derived from the known bits in s_3^0, s_7^0, s_{11}^0 and the partially recovered bits $s_3^{0.5}[0, \ldots, 16]$. The bit patterns are updated as Table 3.

Table 3. Updated bit patterns of the last column of $S^{0.5}$ for NORX64. Red bits are used to establish linear equations.

Forward pattern
1111111111111111 1111111111111111 1111111111111111k kkkkkkkkkkkkkk11
nnnnnnnnnnnnnnnn nnnnnnnnnn111111 1111111111nnnnnn nnnnnnnnnnnnnn1n
nnnnnnnnnnnnnnnn nnnnnnnnnn11111 11111111111nnnnn nnnnnnnnnnnnnnn1
1111111111111111 1111111111111111 1111111111111111 1111111111111111
Backward pattern
nnnnnnnnnnnnnnnn nnnnnnnnnnnnnnnn nnnnnnn11111111k kkkkkkkkkkkkkkkk
nnnnnnnnnnnnnnnn nnnnnnn1kkkkkkkk kkkkkkkkkkkkkk111 1111111111111111
nnnnnnnnnnnnnnnn nnnnnnnnnnnnnnnn nnnnnnnnnnnnnnnn nnnnnnnnnnnnnnn1
nnnnnnnnnnnnnnnn nnnnnnn11111111 1111111111111111 1111111111111111

There are 69 matched bits in the last column of $S^{0.5}$. Together with the two linear equations from bits $s_{10}^0[0]$ and $s_{11}^0[0]$, there are 71 linear equations can be established. On the other hand, there are 70 unknown bits involved in the linear system, after the 17 derived bits $s_{15}^0[27, \ldots, 43]$ are excluded. We verified the coefficient matrix of the linear system to confirm that it has rank 70. Hence, a unique solution can be derived from the linear system. Note that the linear bits

in the backward direction computation require certain value of 131 known bits. The attack can be summarized as follows.

1. Find an output of NORX encryption satisfied the conditions on the known bits. There are 131 conditions on the output bits and 1 condition on the input bits.
2. Guess the bits s_{12}^2, s_{13}^2, s_{14}^2, $s_{15}^2[0,\ldots,40]$. So totally 233 bits are guessed.
3. Determine the bits $s_{15}^2[41,\ldots,63]$ using the solutions of linear equations which have been derived.
4. Compute backward and verify the solution with the known bits in S^0.

We estimate that one guess and to determine trail require roughly 2 NORX operations. Then, the estimated time complexity of this attack is 2^{234} NORX operations. We expect to query for 2^{132} blocks to find a suitable input/output pair. So the data complexity is 2^{132} and memory complexity is negligible.

Attack on NORX32. Similar approach can be applied to attack reduced NORX32. In this case, we will first guess the bits s_{12}^2, s_{13}^2, s_{14}^2, $s_{15}^2[0,\ldots,21]$. By setting the value of bit $s_{11}^0[10]$ to 0, the bits $s_{15}^0[19,\ldots,24]$ can be recovered. The bit patterns of the last column of $S^{0.5}$ are given in Table 4. Here, we control 65 conditions on the output bits to increase the number of linear bits.

39 linear equations can be established from the last column of $S^{0.5}$. After excluded the derived bits, the number of unknown bits is 36. We computed the rank of the coefficient matrix which turned out to be 36. After further dropping the bits $s_7^2[1]$ and $s_{11}^2[0]$ from the system (to reduce the linear bits needed), a unique solution can be derived from the linear system with 37 equations.

The attack procedure is similar to NORX64, so we omit the details here. Since we guess a total number of 118 bits in the attack, the estimated time complexity is 2^{119}, and data complexity is 2^{66}.

Table 4. Updated bit patterns of the last column of $S^{0.5}$ for NORX32. 'l', 'n' and 'k' represent linear bit, non-linear bit and known bit, respectively. Red bits are used to establish linear equations.

Forward pattern
llllllll llllllll llllllll llkkkkll
nnnnnnnn nnnnnlll llnnnnnn nnnnnnln
nnnnnnnn nnnnnnll lllnnnnn nnnnnnnl
llllllll llllllll llllllll llllllll
Backward pattern
nnnnnnnn nnnnnnnn nnllllll llkkkkkk
nnnnnnnn nnlkkkkk kkkkklll llllllll
nnnnnnnn nnnnnnnn nnnnnnnn nnnnnnnl
nnnnnnnn nnllllll llllllll llllllll

5 Internal Differential Attack

5.1 Basic Key Recovery with Internal Differential Attack

This section explains the key recovery attack on the NORX reduced to 2 rounds with differential cryptanalysis in the nonce-respect setting.

Overall strategy. To exploit the difference under the same nonce, we exploit the parallel mode of NORX shown in Fig. 2, and focus on the difference between two lanes. As shown in Fig. 2, the state is duplicated and the lane number is XORed. For $p = 2$, lane numbers 0 and 1 make 1-bit difference in the state (LSB of s_{13}). This difference expands in the subsequent permutation F. Here, we build a high probability differential characteristic for the first 1 round.

Attackers only can observe the value of the rate words (s_0 to s_{11}) in each lane after 2 rounds. Hence, we perform some backward computation for the last 1 round with guessing capacity words (s_{12} to s_{15}). The number of secret bits in the state is $4w$, and the number of security bits claimed is also $4w$. Here, the difficulty is that the $4w$-bit secret value is completely different for the first lane and the second lane. Thus, we need to analyze $8w$-bit secret bits in a pair. This setting is quite different from conventional differential attack against block ciphers in which secret exists in the key and is common for both values in a pair.

To overcome this problem, we adopt the meet-in-the-middle approach. Namely, we guess up to $4w$ bits of the secret in the first lane, and recover several bits of the state after 1 backward round as the guess-and-determine attack in Sect. 4. The results are stored in a table with memory size up to $4w$ bits. Then, we do the same computation for the second lane, and compare the results of two lanes for picking up pairs satisfying the differential characteristic after 1 round. Suppose that the number of matched bits is $4w$. Then, among up to $2^{4w} \cdot 2^{4w} = 2^{8w}$ pairs, only up to $2^{8w}/2^{4w} = 2^{4w}$ pairs will remain as the candidates satisfying the differential characteristic. Finally, for each of the remaining candidate, we exhaustive guess the unguessed bits, and identify the correct state value.

We first discuss a simple attack against NORX64 in Sect. 5.1, and then show several optimization techniques and application to NORX32 in Sect. 5.2.

1-round differential characteristic. Lane number 0 and lane number 1 make 1-bit difference in the LSB of s_{13}. We trace the propagation of this difference.

Construction of differential characteristic is simple. The only non-linear component is the AND operation in H. We set 1-bit condition for each active bit to control its output difference. Because of the small number of rounds, we found that the probability of the characteristic is maximized by setting output difference of all active bits to 0. The obtained characteristic is shown in Table 5. To keep the probability of the characteristic high, we only specify the difference of 8 words in 2 diagonals, i.e. $(\Delta s_1, \Delta s_2, \Delta s_6, \Delta s_7, \Delta s_8, \Delta s_{11}, \Delta s_{12}, \Delta s_{13})$.

The characteristic in Table 5 includes 5 active bits in the first 0.5 round and 16 active bits in the next 0.5 round, in total 21 active bits. Therefore, the characteristic can be satisfied with probability at least 2^{-21}.

Table 5. 1-Round Differential Characteristic. '0' and '1' represent inactive bit and active bit, respectively. After 1 round, we only need the difference of 8 words. '?' represents that difference is not specified in that bit.

Initial difference			
0000000000000000	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000001	0000000000000000	0000000000000000
Difference after 0.5 round (column step)			
0000000000000000	0000002000000000	0000000000000000	0000000000000000
0000000000000000	4200004000020000	0000000000000000	0000000000000000
0000000000000000	2100000000010000	0000000000000000	0000000000000000
0000000000000000	2000000000010000	0000000000000000	0000000000000000
Difference after 1 round (diagonal step)			
????????????????	0000002000000400	0020000400000000	????????????????
????????????????	????????????????	4040000840000800	0800000a00000200
0420000100000100	????????????????	????????????????	2020000420000000
2020000400000000	0400000100000000	????????????????	????????????????

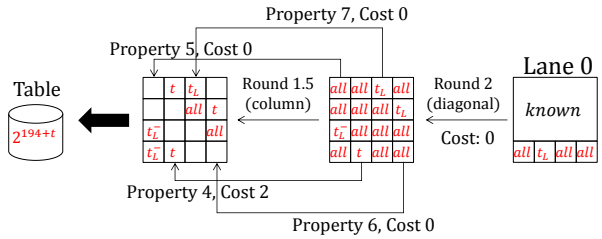

Fig. 8. Backward computation. *all*, t, t_L and t_L^- represent that all bits are known, t middle bits are known, t LSBs are known and $t-24$ LSBs are known, respectively.

1-round backward computation. For a plaintext-ciphertext pair for each lane, we recover the value of the rate words, i.e. $\bar{s}_{0,0}$ to $\bar{s}_{0,11}$ and $\bar{s}_{1,0}$ to $\bar{s}_{1,11}$. Then for the first lane, we exhaustively guess all bits of $(\bar{s}_{0,12}, \bar{s}_{0,14}, \bar{s}_{0,15})$ and t LSBs of $\bar{s}_{0,13}$, where the value of t will be determined later. For each guess, we can compute 1 round in backwards with the properties introduced in Sect. 3. The 1-round backward computation is depicted in Fig. 8.

For the first 0.5-round backward computation, 3 diagonals are fully known, which can be inverted easily. The remaining diagonal, $(\bar{s}_{0,2}, \bar{s}_{0,7}, \bar{s}_{0,8}, \bar{s}_{0,13}) \leftarrow \mathsf{G}^{-1}(\bar{s}'_{0,2}, \bar{s}'_{0,7}, \bar{s}'_{0,8}, \bar{s}'_{0,13})$, is the case of Property 4 with t LSBs in d'. Here, given that $r_2 = 40$ for NORX64, we replace " $\ggg r_2$" with " $\lll (64-r_2)$" to make the analysis simpler. The overlapped bit positions becomes $t-24$ bits starting from the LSB. In the end, we obtain t LSBs of $\bar{s}_{0,2}$ and $\bar{s}_{0,7}$, $t-(64-r_2) = t-24$ LSBs of $\bar{s}_{0,8}$, and t middle bits of $\bar{s}_{0,13}$. Because all partially known bits start from the LSB in H^{-1}, we do not need additional bit guess.

The next 0.5-round can be computed with Properties 4 to 7 as follows.

– The first column is the case of Property 5, in which we know $t-24$ LSBs of $\bar{s}'_{0,8}$. Different from Property 5, we only compute 2 words; $\bar{s}_{0,8}$ and $\bar{s}_{0,12}$. This

can avoid 1-bit guess, and thus $t-24$ bits of $\bar{s}_{0,8}$ and $\bar{s}_{0,12}$ can be computed without any guess.
- The second column is the case of Property 4 in which the consecutive t bits start from middle bits. Property 4 requires 3-bit guess to recover 4 input variables. Here, we only need to recover 2 input variables, a and d in Fig. 6, and this saves us to guess 1 bit. As a result, we need 2-bit guess to compute the corresponding t bits of $\bar{s}_{0,1}$ and $\bar{s}_{0,13}$.
- The third column is the case of Property 7. t bits of $\bar{s}_{0,2}$ and all bits of $\bar{s}_{0,6}$ can be computed without guess.
- The fourth column is the case of Property 6, which requires 2-bit guess to recover 4 input variables. Here, we only need to recover 2 input variables, b and c and this can be done without guess. Hence, t bits of $\bar{s}_{0,7}$ and all bits of $\bar{s}_{0,11}$ can be computed without guess.

In summary, we guess 192 bits of $\bar{s}_{0,12}, \bar{s}_{0,14}, \bar{s}_{0,15}$, t bits of $\bar{s}_{0,13}$ after round 2 and 2 bits in the middle, which leads to $2(t-24)+2t+(t+64)+(t+64) = 6t+80$ bits of $(\bar{s}_{0,1}, \bar{s}_{0,2}, \bar{s}_{0,6}, \bar{s}_{0,7}, \bar{s}_{0,8}, \bar{s}_{0,11}, \bar{s}_{0,12}, \bar{s}_{0,13})$ after round 1. Those are stored in a table with a memory of size 2^{194+t}.

After the computation of the first lane, we apply the same computation for the second lane, i.e. for state $\bar{S}_1$, with $194+t$-bit guess. For each result, we XOR the difference in two diagonals specified by the characteristic, and check the match with the table generated for the first lane. If the match is found, we guess the remaining $64-t$ bits of $\bar{s}_{0,13}$ for the first lane, and compute back to the initial value (IV) of NORX64. Only if the pair satisfies the characteristic and the guess is correct, the IV appears, which recovers the key simultaneously.

Attack procedure. Associated data is irrelevant. Hereafter, we set A to be empty. The attack procedure is described in Algorithm 1. Due to the probability of the characteristic, 2^{21} pairs are analyzed, which corresponds to 2^{21} iterations in Step 1. The 1-round backward computation is performed for each of 2^{21} pairs.

Complexity evaluation and choice of t. In Algorithm 1, for each of i in Step 1, Step 6 requires 2^{194+t} G computations and Step 9 requires 2^{194+t} G computations. After the match in Step 10, $2^{2(194+t)-(6t+80)} = 2^{308-4t}$ pairs will remain. Then, Step 12 requires $2^{308-4t+64-t} = 2^{372-5t}$ NORX64 operations. Those are iterated 2^{21} times for the iteration in Step 1. Hence, time complexity is less than $2^{21}(2^{194+t}+2^{194+t}+2^{372-5t})$ NORX64 operations. This is optimized when $t=30$, which leads to $2^{246.2}$ NORX64 operations.

Data complexity is only caused by Step 3. Two message blocks are queried in each iteration of Step 1. Thus, data complexity is 2^{22} message blocks.

Memory complexity is dominated by Step 6, which stores the result of 2^{194+t} computations. Thus, memory complexity is 2^{224} when $t=30$.

5.2 Optimized Key Recovery with Internal Differential Attack

Differential. The probability of the characteristic in Sect. 5.1 was 2^{-21}. This can be improved by using the differential instead of a single characteristic. Rig-

Algorithm 1: 2-round key recovery with internal-differential attack.

Input: characteristic with probability 2^{-21}, $194 + t$ guessed-bit positions
Output: K

for $i = 0, 1, \ldots, 2^{21} - 1$ **do**
  Randomly choose a nonce N^i and a 2-block message $M_0^i \| M_1^i$.
  Query $(N^i, M_0^i \| M_1^i)$ in the parallel mode to obtain $(C_0^i \| C_1^i)$.
  Compute $\bar{s}_{0,0}, \ldots, \bar{s}_{0,11} \leftarrow M_0^i \oplus C_0^i$ and $\bar{s}_{1,0}, \ldots, \bar{s}_{1,11} \leftarrow M_1^i \oplus C_1^i$.
  for $192 + t$ bits of $\bar{s}_{0,12}, \bar{s}_{0,13}, \bar{s}_{0,14}, \bar{s}_{0,15}$ and 2 bits in the middle **do**
    Obtain $6t + 80$ bits of $\bar{s}_0$ after round 1 and store them in a table T_0.
  end for
  for $192 + t$ bits of $\bar{s}_{1,12}, \bar{s}_{1,13}, \bar{s}_{1,14}, \bar{s}_{1,15}$ and 2 bits in the middle **do**
    Obtain $6t + 80$ bits of $\bar{s}_1$ after round 1 and xor the difference in Table 5.
    Check the match with T_0. If the match is found, go to the next step.
    for the remaining $64 - t$ bits of $\bar{s}_{0,13}$ **do**
      Compute back to the initial value of NORX64.
      if the result satisfies the form of
      $(u_0, n_0, n_1, u_1, k_0, k_1, k_2, k_3, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9)$ **then**
        return $K = (k_0, k_1, k_2, k_3)$.
      end if
    end for
  end for
end for

orously evaluating the probability of the differential is hard. However, thanks to the high probability of the characteristic, we can evaluate it experimentally.

We chose 2^{24} pairs at uniformly random, and 6937 pairs could satisfy the output difference with respect to the $6t+80$ bits computed during the backward computation. Thus, the probability of the differential is $6937/2^{24} \approx 2^{-11.23}$, which improves the complexity of Algorithm 1 by roughly 10 bits.

Multiple lanes. NORX64-4-4-256 supports 4 parallel lanes. When a 4-block message is processed, lane numbers XORed to the duplicated states is 0, 1, 2 and 3. Thus, we can make the pair with $\Delta\bar{S}_{13} = 0x01$ between lane 0 and lane 1 and between lane 2 and lane 3. Besides, we can also consider the internal difference between lane 0 and lane 2 and between lane 1 and lane 3, which makes $\Delta\bar{S}_{13} = 0x02$. The best characteristic for this difference is obtained by rotating Table 5 by 1 bit. We also experimentally verified the probability of the differential, which is $7532/2^{24} \approx 2^{-11.12}$, slightly better than the case with $\Delta\bar{S}_{13} = 0x01$. The average probability for two cases is $2^{-11.18}$.

In summary, we can make 4 pairs per 4-block message query, which halves the data complexity evaluation in Sect. 5.1.

Multiple differentials. While the characteristic in Table 5, 2^{-21}, is optimized, we found that there are 2^5 characteristics having the same probability.

Recall the complexity evaluation in Sect. 5.1. Let N_{pair}, T_{mitm}, and T_{veri} be number of pairs to analyze, time complexity for the meet-in-the-middle match per pair, and time complexity for verifying matched candidates, respectively. Roughly, the total complexity is given by $N_{pair}(T_{mitm} + T_{veri})$. Now, suppose that there are D characteristics with the same probability and we aim to find a pair satisfying any of D choices. This reduces the number of pairs to N_{pair}/D, while the number of valid candidates after the match becomes D times, thus the cost for verification becomes $D*T_{veri}$. The essence of this technique is that the 1-round backward computation with a cost of T_{mitm} is independent of D choice of characteristics, thus one computation can be reused for testing D characteristics.

To derive such characteristics, we focus on the differential propagation in the last H function (during round 0.5 to round 1). There are 4 active bits during the last H function in $(\Delta s'_1, \Delta s'_6, \Delta s'_{11}, \Delta s'_{12}) \leftarrow \mathsf{G}(\Delta s_1, \Delta s_6, \Delta s_{11}, \Delta s_{12})$. The characteristic in Table 5 was derived by setting the output difference of those 4 bits to 0, which causes the probability drop by 2^{-4}. However, the differences from those bits only linearly related to the output of the entire characteristic, thus any combination of 0 or 1 for the output difference from those 4 bits generates 2^4 distinct characteristics with exactly the same probability.

The same occurs in the other diagonal $\mathsf{G}(\Delta s_2, \Delta s_7, \Delta s_8, \Delta s_{13})$. 4 bits in positions 8, 32, 53 and 58 are active in the last H, while only bit position 8 will be later used for the meet-in-the-middle match. As a result, 2 distinct characteristics can be considered in bit position 8. Along with 2^4 choices for the other diagonal, we have $D = 2^5$ characteristics with the same probability.

The discussion above is about the multiple characteristics. It can easily be extended to multiple differentials. We experimentally tested the probability of 2^5 differentials, and confirmed that all of them has almost the same probability.

Strictly speaking, matching multiple differentials during the meet-in-the-middle match in Step 10 of Algorithm 1 is a so-called 3-list problem, which requires more cost than the 2-list case. We observe that 2^5 differentials only differ in 10 bits, that is, all of them have the same difference in the other $6t + 70$ bits. Thus, we first apply the filter in $6t - 70$ bits, then check the details for 10 bits. Matching $6t - 70$ bits reduces the number of candidates sufficiently small, thus using multiple differentials gives negligible impacts to the complexity.

Optimized complexity for NORX64. To satisfy one of the 2^5 differentials with probability $2^{-11.18}$, $2^{6.18}$ pairs need to be analyzed. The number of iterations of Step 1 in Algorithm 1 becomes $i = 2^{6.18}$. By exploiting four lanes, $2^{6.18}$ message-block queries are sufficient to construct $2^{6.18}$ pairs. Thus the data complexity is $2^{6.18}$ message blocks.

The usage of the multiple differentials slightly changes the balance between T_{mitm} and T_{veri}, i.e. higher T_{mitm} and lower T_{veri} offers the best balanced complexity. We found that $t = 31$ instead of $t = 30$ yields the best complexity, which is $\frac{2^{11.18}}{2^5}\left(2 * 2^{194+t} + 2^5 * 2^{372-5t}\right) = 2^{232.8}$. The memory complexity increases due to the increase of t, which is $2^{194+t} = 2^{225}$.

Table 6. 1.5-round internal differential-linear distinguisher. '0' and '1' represent unaffected bits and bits that maybe affected by the internal difference, respectively. After 1.5 round 8 bits of the rate will never be affected by the internal difference in S_{13}.

Initial difference			
0000000000000000	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000000	0000000000000000	0000000000000000
0000000000000000	0000000000000001	0000000000000000	0000000000000000
Unaffected bits after 0.5 round, i.e. $S^{0.5}$			
0000000000000000	000000e000000000	0000000000000000	0000000000000000
0000000000000000	ce0000c000060001	0000000000000000	0000000000000000
0000000000000000	e700000000030000	0000000000000000	0000000000000000
0000000000000000	e000000000010000	0000000000000000	0000000000000000
Unaffected bits after 1 round, i.e. S^{1}			
bfc07bffc03ef807	000003e00000fc00	00e0007c00000000	e0003fe000000000
fe003bc003de00ff	fffcffff1ffff9ff	c7c003ffc000f80f	ff80007e00038e01
ffe000030001c700	df00000001ef007f	fffe7fffcfffbcff	e3e001ffe0000007
e1e000fc00000003	7c0000010000c000	e000000000e0003f	ff803efe07bf9e7b
Unaffected bits after 1.5 round, i.e. $S^{1.5}$			
ffffffffffffffff	ffffffffffffffff	ffffffffffffffff	ffffffff0ffff87
ffffffffffffffff	ffffffffffffffff	ffffffffffffffff	ffffffffffffffff
ffffffffffffffff	ffffffffffffffff	ffffffffffffffff	ffffffffffffffff
ffffffffffffffff	ffffffffffffffff	ffffffffffffffff	fffcffff9fffffff

Application to NORX32. Because NORX32 does not formally support the parallel mode, we only explain it briefly. The attack on NORX32 is harder than the one for NORX64 with respect to the following two points.

- The probability of the multiple differential becomes relatively smaller to the word size (32 instead of 64).
- The rotation number $r_2 = 16$ is exactly a half of the word size, which generates less number of overlapped bits during the backward computation ($r_2 = 40$ in NORX64).

Those make the advantage of the attack smaller than the case of NORX64. The attack strategy is the same as NORX64 and all the optimization techniques can also be applied. We experimentally verified that the probability of the differential is $2^{-12.25}$ and there are 2^5 differentials with the same probability. We choose $t = 17$ for the number of partially guessed bits. In the end, the best attack complexity is $2^{7.25}$, $2^{124.25}$ and 2^{115} in data, time and memory, respectively.

5.3 Distinguisher with Internal Differential-Linear Attack

In this section, we present an internal differential-linear attack on round-reduced NORX. In more detail, we show an efficient distinguisher for NORX64 with $p \geq 2$ for up to 2 rounds. As shown in Table 6, the internal difference in the LSB of s_{13}, i.e. $s_{13}[0]$, does not affect 8 bits in the rate part of the NORX64 state after 1.5 rounds, i.e. $s_3^{1.5}[3, 4, 5, 6, 24, 25, 26, 27]$. This property leads to a trivial distinguisher for the keystream of NORX64 reduced to 1.5 rounds in the parallel mode.

Unfortunately, by adding 0.5 rounds all bits of the rate are affected by the internal difference and no significant bias can be observed with practical complexity (2^{40} experiments). However, using Property 4 described in Sect. 3, all the bits at the output of the first $\mathsf{H}(a, b)$ in each G function in the last half-round can be derived from the rate part of the output. This significantly improves the attack leading to an efficient distinguisher for 2 rounds of NORX64 in the parallel mode. We can observe significant biases in several of the 256 bits. For instance bit 175 has a bias of -0.15. If an adversary aims to distinguish the keystream related to NORX64-4-4-256 based on this bit, it proceeds as follows:

1. Query $\frac{1}{0.15^2}$ 2-block messages for NORX64-4-4-256 and receive the corresponding ciphertexts.
2. Partially decrypt each ciphertext up to the output of the first $\mathsf{H}(a, b)$ in the last half-round.
3. Verify the matching for each ciphertext pair and output the total amount of the matching in bit 175 for all ciphertext pairs, N.
4. Output NORX64 if $N \geq \frac{1}{2\times 0.15^2}$.

The data complexity of this attack is about 90 message blocks, while the success probability is 97.7% [15].

6 Conclusion

In this paper, we present the first cryptanalysis of NORX in the nonce-respecting setting. Our attack exploits the slow diffusion of NORX's round function especially in backward direction. We investigate several attacks against NORX and all of them cover two rounds of the permutation. On the other hand, while the presented guess and determine attack covers two rounds of NORX's variants yet it may be possible to be extended to more rounds by employing advanced equation solving techniques. We do not think it can be extended to the full 4-round NORX. However, the results of this paper can be considered as a starting point for future analysis in this direction.

Additionally, we presented a practical distinguisher for 2-round NORX64 encryption in parallel mode that could not be applied to NORX32 or serial-NORX64. This observation may be considered as a lower diffusion in NORX64 compared to the NORX32, for the same number of rounds. In addition, this attack along with the given internal differential attack could be considered as evidences that parallel mode of NORX has lower security bound, compared to serial mode.

Acknowledgments. The authors would like to thank the organizers of ASK 2015 that initiated this work. Keting Jia is supported by the National Natural Science Foundation of China (No. 61133013 and 61402256) and 973 Program (No.2013CB834205). Part of this work was done while Florian Mendel was visiting NTU and has been supported in part by the Austrian Science Fund (project P26494-N15).

References

1. CAESAR: Competition for Authenticated Encryption: Security, Applicability,and Robustness (2013), http://competitions.cr.yp.to/caesar.html
2. Aumasson, J., 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)
3. Aumasson, J., Jovanovic, P., Neves, S.: NORX V1 (2014), http://competitions.cr.yp.to/round1/norxv1.pdf
4. Aumasson, J., Jovanovic, P., Neves, S.: Analysis of NORX: investigating differential and rotational properties. In: Aranha, D.F., Menezes, A. (eds.) LATINCRYPT 2014. LNCS, vol. 8895, pp. 306–324. Springer (2015)
5. Aumasson, J., Jovanovic, P., Neves, S.: NORX V2.0 (2015), http://competitions.cr.yp.to/round2/norxv20.pdf
6. Aumasson, J., Meier, W.: Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi. NIST mailing list (2009), http://aumasson.jp/data/papers/AM09.pdf
7. 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)
8. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V., Keer, R.V.: CAESAR submission: Keyak v2 (2015), http://competitions.cr.yp.to/round2/keyakv2.pdf
9. Biham, E., Shamir, A.: Differential Cryptanalysis of the Full 16-Round DES. In: Brickell, E.F. (ed.) CRYPTO '92. LNCS, vol. 740, pp. 487–496. Springer (1992)
10. Bilgin, B., Bogdanov, A., Knezevic, M., Mendel, F., Wang, Q.: Fides: Lightweight authenticated cipher with side-channel resistance for constrained hardware. In: Bertoni, G., Coron, J. (eds.) CHES 2013. LNCS, vol. 8086, pp. 142–158. Springer (2013)
11. Das, S., Maitra, S., Meier, W.: Higher order differential analysis of NORX. IACR Cryptology ePrint Archive 2015, 186 (2015), http://eprint.iacr.org/2015/186
12. Dinur, I., Jean, J.: Cryptanalysis of FIDES. In: Cid, C., Rechberger, C. (eds.) FSE 2014. LNCS, vol. 8540, pp. 224–240. Springer (2014)
13. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.1 Submission to the CAESAR Competition (2014), http://competitions.cr.yp.to/round2/asconv11.pdf
14. Jean, J., Sasaki, Y., Wang, L.: Analysis of the CAESAR candidate Silver. In: Dunkelman, O., Keliher, L. (eds.) SAC 2015. LNCS, Springer (2015), to appear.
15. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT '93. LNCS, vol. 765, pp. 386–397. Springer (1993)
16. Nikolic, I.: CAESAR candidates speed comparison (2014), http://www1.spms.ntu.edu.sg/~syllab/speed/
17. Penazzi, D., Montes, M.: Silver v.1. Submitted to the CAESAR competition (2014)
18. Peyrin, T.: Improved differential attacks for ECHO and Grøstl. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 370–392. Springer (2010)