

# Cryptanalysis of NORX v2.0

Colin Chaigneau<sup>1</sup>, Thomas Fuhr<sup>2</sup>, Henri Gilbert<sup>1,2</sup>, Jérémy Jean<sup>2</sup> and  
Jean-René Reinhard<sup>2</sup>

<sup>1</sup> University of Versailles Saint-Quentin-en-Yvelines (UVSQ), Versailles, France

[Colin.Chaigneau@uvsq.fr](mailto:Colin.Chaigneau@uvsq.fr)

<sup>2</sup> Agence nationale de la sécurité des systèmes d'information (ANSSI) Crypto Lab,  
51, boulevard de La Tour-Maubourg 75700 Paris 07 SP, France

[<Firstname.Lastname>@ssi.gouv.fr](mailto:<Firstname.Lastname>@ssi.gouv.fr)

**Abstract.** NORX is an authenticated encryption scheme with associated data being publicly scrutinized as part of the ongoing CAESAR competition, where 14 other primitives are also competing. It is based on the sponge construction and relies on a simple permutation that allows efficient and versatile implementations. Thanks to research on the security of the sponge construction, the design of NORX, whose permutation is inspired from the permutations used in BLAKE and ChaCha, has evolved throughout three main versions (v1.0, v2.0 and v3.0).

In this paper, we investigate the security of the full NORX v2.0 primitive that has been accepted as third-round candidate in the CAESAR competition. We show that some non-conservative design decisions probably motivated by implementation efficiency considerations result in at least one strong structural distinguisher of the underlying sponge permutation that can be turned into an attack on the full primitive. This attack yields a ciphertext-only forgery with time and data complexity  $2^{66}$  (resp.  $2^{130}$ ) for the variant of NORX v2.0 using 128-bit (resp. 256-bit) keys and breaks the designers' claim of a 128-bit, resp. 256-bit security. Furthermore, we show that this forgery attack can be extended to a key-recovery attack on the full NORX v2.0 with the same time and data complexities. We have implemented and experimentally verified the correctness of the attacks on a toy version of NORX. We emphasize that the scheme has recently been tweaked to NORX v3.0 at the beginning of the third round of the CAESAR competition: the main change introduces some key-dependent internal operations, which make NORX v3.0 immune to our attacks. However, the structural distinguisher of the permutation persists.

**Keywords:** CAESAR Competition · NORX · Cryptanalysis · Forgery Attack · Symmetry

## 1 Introduction

The purpose of authenticated encryption (AE) is to encrypt and authenticate a plaintext message in a combined way. A slight extension of this functionality named authenticated encryption with associated data (AEAD) allows to authenticate at the same time some extra unencrypted data named associated data. An example of a broadly deployed AEAD algorithm is the GCM mode of operation of AES [MV04]. The aim of the ongoing international competition CAESAR, that has been launched in 2014, is to select a portfolio of AE(AD) algorithms that offer stronger security guarantees or whose performance profiles are better suited to some execution environments than AES-GCM.

NORX is a family of AEAD algorithms designed by Aumasson, Jovanovic and Neves, and is one of the 15 CAESAR candidates that were selected in August 2016 for the third round of the competition. The overall structure of the NORX algorithm adopts the so-called monkeyDuplex construction, which is derived from the sponge construction and iterates a

keyless permutation  $P$  of a large state [BDPV12]. The design of the permutation  $P$  used by NORX is partly inspired from those of the stream cipher ChaCha [Ber08], the SHA-3 finalist BLAKE [AHMP10] and its more efficient variant BLAKE2 [ANWW13]. This permutation operates over states that can be represented as  $4 \times 4$  matrix of words whose size  $w$  is either 32 or 64 bits. It follows a design close to so-called ARX primitives, as it uses only “R” operations (circular rotations and shifts), “X” (exclusive or) operations, and modified “A” operations (modular additions, modified in that carries only propagate to one position to the left). The key length  $k$ , the default tag length  $t$ , and the claimed security level of NORX are all equal to  $4w$ , in other words either all equal to 128 bits or to 256 bits depending on the value of  $w$ .

Three main versions of NORX have been published so far: NORX v1.0 (March 2014), the initial submission to the CAESAR competition; NORX v2.0 (August 2015), the version that was evaluated and selected for the third round; NORX v3.0 (September 2016), a version published shortly after the beginning of the third round that will serve as a basis for the third-round evaluation. In all versions, the NORX family consists of two main sub-families of algorithms associated with the word sizes  $w = 32$  and  $w = 64$ .

Lightweight variants of NORX, called NORX-8 and NORX-16, have also been proposed by the same designers [AJN15c]. They follow the same generic strategy as NORX v2.0, with word sizes  $w = 8$  and  $w = 16$ .

**Related Work.** There exists a handful of papers that study the security of NORX, which we briefly recall here. First, the designers of NORX provided their own analysis of the permutation  $P$  in the specifications and [AJN15a]. They conclude that no high-probability differential exists in the primitive, that word-level rotational cryptanalysis does not threaten the scheme, and that no structural distinguisher of the permutation can be used in an attack on the mode. Later in [DMM15], Das et al. describe statistical variants of zero-sum distinguishers that allow to distinguish 3.5 rounds of the permutation of NORX-32 and the full-round permutation of NORX-64 from random permutations. At FSE 2016, Bagheri et al. show in [BHJ+16] that the slow diffusion of  $G^{-1}$  can be leveraged into a state/key recovery for a reduced versions of NORX v2.0 where the underlying permutation applies half the rounds (two out of four). More recently, Dwivedi et al. [DKM+16] analyze the state-recovery resistance of several CAESAR candidates, including NORX, with respect to SAT solvers. About NORX, they conclude that state recovery is only possible on NORX-32 when the underlying permutation does not apply more than 1.5 round. Finally, throughout this paper, we also refer to [JLM14] where Jovanovic et al. give a security proof of the NORX mode.

**Our Contributions.** Our main result is an attack on NORX v2.0 that shows that the security level of the NORX v2.0 algorithms is at most  $2w + 2$  bits, i.e. about 66 or 130 bits depending whether  $w = 32$  or  $w = 64$ , instead of the  $4w$  bits claimed by the designers. The attack can be viewed in two ways:

1. as an *existential forgery attack* with success probability  $2^{-2w-2}$ , for instance  $2^{-66}$  if  $w = 32$  bits, that requires to get the authenticated encryption of one single short chosen plaintext or,
2. as an existential forgery attack with success probability greater than 50% that requires the knowledge of  $2^{2w+2}$  ciphertexts with their associated tags and the same number of forgery attempts.<sup>1</sup>

Both variants of the attack break the claim of the designers stating that NORX v2.0 offers a  $4w$ -bit level of security. We additionally observe that once a forgery attempt

<sup>1</sup>Enforcing a limitation of the amount of data handled with a single key does not thwart our attack as changing the key does not drop the marginal success probability of a single forgery attempt.

succeeds, a key-recovery attack can be easily mounted as the secret key is only injected during the initialization phase. Namely, a successful forgery can reveal the full internal state at the expense of an extra offline computation of about  $2^{2w}$  operations. Then, same as was done in [DJ15] on the FIDES authenticated encryption scheme [BBK<sup>+</sup>13], the full sponge can be inverted, which reveals the initial state that contains the secret key. This can be achieved if we assume either chosen-plaintexts attacks, or that a ciphertext-only adversary interacts with a decryption oracle and gets the plaintext corresponding to any successful forgery. We have implemented and experimentally verified the correctness of the attacks on a toy version of NORX v2.0, where the word size  $w$  is reduced to 8 bits.

The attack leverages an interaction between the two following non-conservative properties of NORX v2.0.

- The *capacity* of the NORX sponge is low: only  $4w$  bits, one fourth of the state size, i.e. 128 bits if  $w = 32$  bits and 256 bits if  $w = 64$  bits. This more aggressive choice than the  $6w$ -bit capacity that was selected for NORX v1.0 was motivated by performance considerations, as it allowed to increase the rate of the sponge construction by a factor 1.25. It was also supported by the security bounds derived from the security proofs of [JLM14] (substantiated in the security goals section of the algorithm specification [AJN15b]), up to the fact that the underlying permutation  $P$  does not behave like an ideal permutation.
- The permutation  $P$  used in the NORX sponge has strong structural properties that substantially deviate from the expected behavior of an ideal permutation. Our attack leverages the structural property that  $P$  commutes with a circular rotation of the columns of the internal state  $4 \times 4$  matrix. This property has some connection with the weaker structural property of  $P$  already observed by the designers in [AJN15a] that the set of states whose four columns are equal is invariant under  $P$ .

The former attack can be viewed as a kind of rotational cryptanalysis at the state level rather than at the word level as considered in [AJN15a] on NORX or more generally in [KN10]. It also has some connection with the *invariant permutation attacks* sub-class of invariant subspace attacks introduced in [LMR15], since in both cases a permutation of the state words that commutes with a cryptographic state permutation is leveraged, one difference being that an invariant permutation property is used here in a keyless and constant-less context.

While the two non-conservative properties leveraged by the attack (the low capacity and the existence of a commuting rotation) still hold for NORX v3.0, one of the “tweaks” introduced in NORX v3.0 appears to thwart the former attack, namely the involvement of the key in the finalization of the tag computation, using an Even-Mansour-like construction. This finalization was also selected during the conception of another monkeyDuplex-based CAESAR candidate, namely ASCON [DEMS16].

Finally, we also investigate the security claims of NORX v2.0 and NORX v3.0. We show that the generic success probability of a forgery attack has to be related to the cumulative length of forgery attempts (instead of the total number of forgery attempts), as it is also the case for other AEAD schemes such as GCM. Moreover, we show that even if the total length of decryption queries is strongly limited, the authenticity bound of the proof does not guarantee the security level of  $2^{4w}$  claimed for NORX.

**Organization.** The rest of this paper is organized as follows. Section 2 gives detailed descriptions of NORX variants discussed in the paper. In Section 3, we describe our attack on NORX v2.0. In Section 4, we study the applicability of our attack to other variants of NORX. Finally, in Section 5, we discuss the results of our attack and compare them to the security claims made by the designers of NORX and to the bounds derived from the security proofs.

## 2 Specifications of NORX

We provide in this section a description of the NORX family of *Authenticated Encryption with Associated Data* (AEAD) algorithms, through the description of NORX v2.0. We start by detailing in Subsection 2.1 the keyed-sponge mode and its core permutation. Then, in Subsection 2.2, we describe the security goals claimed by the designers. Finally, we outline in Subsection 2.3 the main differences between NORX v2.0 and the other members of the NORX family.

**Notations.** In the sequel, we use  $x||y$  to denote the concatenation of two bit-strings  $x$  and  $y$ , and  $|x|$  to represent the bit-length of the bit string  $x$ .

### 2.1 Description of NORX v2.0

We now describe NORX v2.0, which is the version our attack targets. It relies on  $w$ -bit words operations, with  $w \in \{32, 64\}$ . We note NORX- $w$  when we consider NORX with a given  $w$  value.

**Mode of Operation.** NORX is based on the monkeyDuplex sponge construction [BDPVA11] and relies on a  $16w$ -bit permutation  $P$  that we describe later.

The monkeyDuplex sponge construction operates on an internal state  $S$ , which in the case of NORX v2.0 is divided into two distinct parts of respective bit-sizes  $r = 12w$  and  $c = 4w$  for a total size of  $16w$  bits. We represent the  $16w$ -bit internal state  $S$  of the construction as a  $4 \times 4$  matrix of  $w$ -bit words as follows

$$S = \begin{bmatrix} s_0 & s_1 & s_2 & s_3 \\ s_4 & s_5 & s_6 & s_7 \\ s_8 & s_9 & s_{10} & s_{11} \\ s_{12} & s_{13} & s_{14} & s_{15} \end{bmatrix}.$$

The value  $r$  is called the *rate* of the sponge, and denotes the amount of data that can be processed by each call to permutation  $P$ . The rate part  $S^r$  of the state consists of its first 12 words. The value  $c$  is called the *capacity* and informally represents the security level expected from the construction. The capacity part  $S^c$  of the state consists of its last four words. The internal state  $S$  can then be written as  $S = S^r||S^c$ .

The encryption algorithm  $\text{Enc}$  takes as inputs a  $k$ -bit key  $K$ , an  $n$ -bit nonce  $N$ , a plaintext  $M$  and associated data in the form of a header  $A$  and a trailer  $Z$ . The header, plaintext and trailer are three optional strings. The encryption algorithm computes a  $t$ -bit authentication tag  $T$ , and a ciphertext  $C$  of same bit-length as the plaintext. Similarly, the decryption algorithm  $\text{Dec}$  takes as inputs  $(K, N, A, C, Z, T)$  and returns either  $\perp$  or  $M$  depending on the validity of the authentication tag  $T$ .

Encryption and decryption algorithms begin by an initialization phase that sets the internal state to  $S_{init}$ : it consists in storing the  $4w$ -bit key  $K \stackrel{\text{def}}{=} k_0||k_1||k_2||k_3$ , the  $2w$ -bit nonce  $N \stackrel{\text{def}}{=} n_0||n_1$  and some initialization constants  $(u_i)$  in the internal state, as follows:

$$S_{init} = \begin{bmatrix} n_0 & n_1 & u_2 & u_3 \\ k_0 & k_1 & k_2 & k_3 \\ u_8 & u_9 & u_{10} & u_{11} \\ u_{12} & u_{13} & u_{14} & u_{15} \end{bmatrix}.$$

After this step, some parameters of the cipher are XORed to  $s_{12}$ ,  $s_{13}$ ,  $s_{14}$  and  $s_{15}$ . Finally,  $P$  is applied to the full state.

The processing of the header, plaintext and trailer are similar. We assume that header, plaintext and trailer are split in blocks of bit-length  $12w$ . To achieve this, any non-empty

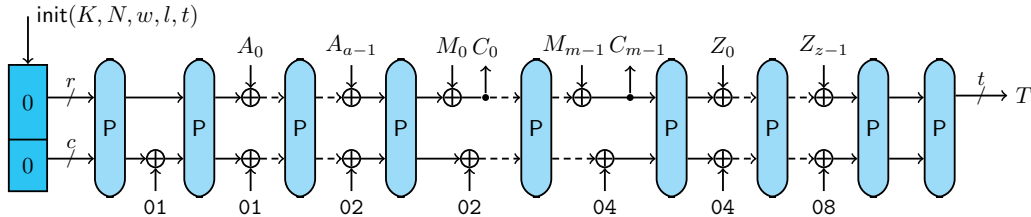
field  $A$ ,  $M$  or  $Z$  is padded using the so-called multi-rate padding function  $\text{pad}$ , which works as follows:

$$\text{pad}(X) = 10^{f(X,w)}1,$$

where  $f(X, w)$  is the smallest nonnegative integer such that  $12w$  divides the total bit-length of  $X||\text{pad}(X)$ . Header, plaintext and trailer blocks are then processed iteratively. The whole mode of operation is depicted in Figure 1. Each block  $B$  is handled as follows.

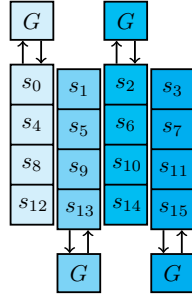
1. A domain separation constant is first XORed to the last word of the internal state, namely  $s_{15}$ . Its value depends on the type of data being processed:  $0x01$  for the header,  $0x02$  for the plaintext, and  $0x04$  for the trailer.
2. The permutation  $P$  updates the internal state  $S$ ; that is:  $S \leftarrow P(S)$ .
3. The header, plaintext, or trailer block  $B$  is XORed in the rate part of the state; that is:  $S^r \leftarrow S^r \oplus B$ .
4. If  $B$  is a plaintext block  $M_i$ , the rate part (after XOR with  $B$ ) is used as ciphertext block  $C_i$ . Note that if  $M_i$  is the last plaintext block, the part of  $C_i$  obtained from the padding is not returned as part of the ciphertext.

The last step is the tag generation. To compute the tag, first domain separation constant  $0x08$  is XORed to  $s_{15}$ . Then,  $P$  is applied twice to  $S$ . The  $t$ -bit tag  $T$  (where  $t = 4w$ ) is extracted as the 4-tuple of state words  $(s_0, s_1, s_2, s_3)$ .

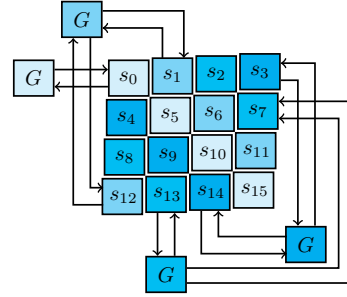


**Figure 1:** NORX v2.0 mode: the padded bit-strings of  $12w$ -bit blocks  $A = A_0||\dots||A_{a-1}$ ,  $M = M_0||\dots||M_{m-1}$  and  $Z = Z_0||\dots||Z_{z-1}$  are processed by the monkeyDuplex sponge construction.

**The permutation  $P$ .** The permutation  $P$  consists of  $l$  consecutive applications of a round function  $F$ , i.e.  $P = F^l$ . The function  $F$  in turns consists of two layers of a smaller permutation denoted  $G$ , which acts on  $4w$  bits. The permutation  $G$  is first computed in parallel on the four columns of  $S$ , then on its four diagonals, as depicted in Figure 2 and Figure 3. The pseudo-codes for both functions  $F$  and  $G$  are given in Algorithm 1 and Algorithm 2, respectively.



**Figure 2:** Function  $G$  applies on state columns.



**Figure 3:** Function  $G$  applies on state diagonals.

---

**Algorithm 1** – Compute  $F(S)$

---

**Input:** Internal state  $(s_0, \dots, s_{15})$

**Output:** Updated  $(s_0, \dots, s_{15})$

- 1:  $(s_0, s_4, s_8, s_{12}) \leftarrow G(s_0, s_4, s_8, s_{12})$
  - 2:  $(s_1, s_5, s_9, s_{13}) \leftarrow G(s_1, s_5, s_9, s_{13})$
  - 3:  $(s_2, s_6, s_{10}, s_{14}) \leftarrow G(s_2, s_6, s_{10}, s_{14})$
  - 4:  $(s_3, s_7, s_{11}, s_{15}) \leftarrow G(s_3, s_7, s_{11}, s_{15})$
  - 5:  $(s_0, s_5, s_{10}, s_{15}) \leftarrow G(s_0, s_5, s_{10}, s_{15})$
  - 6:  $(s_1, s_6, s_{11}, s_{12}) \leftarrow G(s_1, s_6, s_{11}, s_{12})$
  - 7:  $(s_2, s_7, s_8, s_{13}) \leftarrow G(s_2, s_7, s_8, s_{13})$
  - 8:  $(s_3, s_4, s_9, s_{14}) \leftarrow G(s_3, s_4, s_9, s_{14})$
  - 9: **return**  $S$
- 

---

**Algorithm 2** – Compute  $G(a, b, c, d)$

---

**Input:**  $4w$ -bit tuple  $(a, b, c, d)$

**Output:** Updated  $(a, b, c, d)$

- 1:  $a \leftarrow H(a, b)$
  - 2:  $d \leftarrow (a \oplus d) \ggg r_0$
  - 3:  $c \leftarrow H(c, d)$
  - 4:  $b \leftarrow (c \oplus b) \ggg r_1$
  - 5:  $a \leftarrow H(a, b)$
  - 6:  $d \leftarrow (a \oplus d) \ggg r_2$
  - 7:  $c \leftarrow H(c, d)$
  - 8:  $b \leftarrow (c \oplus b) \ggg r_3$
  - 9: **return**  $(a, b, c, d)$
- 

Internally, the  $G$  function uses linear rotations of words and a non-linear operation, denoted by  $H$ , that mimics the modular addition modulo  $2^w$  of bit-strings  $x$  and  $y$ :

$$H(x, y) = (x \oplus y) \oplus ((x \wedge y) \ll 1).$$

The rotation constants  $r_0, r_1, r_2$  and  $r_3$  used in  $G$  depend on the word size (see Table 1).

**Table 1:** Rotation constants in the permutation  $G$ .

Instance	$r_0$	$r_1$	$r_2$	$r_3$
NORX-32	8	11	16	31
NORX-64	8	19	40	63

## 2.2 Security Claims

First of all, the designers of NORX claim no security in the event where nonces are reused: a key/nonce pair should be used only once for encryption. Similarly, there is no guarantee of security under the release of unverified plaintext [ABL<sup>+</sup>14]. Namely, if during the decryption of a ciphertext, any information on the plaintext leaks before the tag has been successfully verified, the security can no longer be ensured.

In other cases, the designers of NORX claim security levels for both confidentiality and authenticity that are equivalent to an exhaustive search of the key, which corresponds to a level of security of 128 bits for NORX-32 and 256 bits for NORX-64.

Additionally, any forgery attack in which the adversary has  $x$  forgery attempts should succeed with probability close to  $x \cdot 2^{-t}$ .

The designers also impose limitations on the amount of data that can be processed with one key. In particular, the security claims are valid as long as the usage of a key  $K$  induces fewer than  $2^{2w}$  calls to the underlying permutation<sup>2</sup>  $P$ .

## 2.3 NORX Variants

We outline here the differences between NORX v2.0 and the other members of the NORX family, either the successive entries to the CAESAR competition, or the lightweight variants. We also mention a parallel alternative to the serial mode of operation presented in Subsection 2.1.

**NORX v1.0.** NORX v1.0 (also named NORX v1 in some submission documents) is the initial version of NORX submitted to the CAESAR competition in March 2014. The main difference between NORX v1.0 and NORX v2.0 relates to the capacity size, which has been reduced from  $6w$  bits to  $4w$  bits. This change yields an increased rate with a direct impact on the efficiency of the cipher, and has been justified by security proofs, e.g. [JLM14]. The security claims are left unchanged between the two versions.

**NORX v3.0.** NORX v3.0 is the latest version of NORX submitted to the CAESAR competition in September 2016. Several changes have been brought to NORX between versions 2.0 and 3.0. In previous versions, a potential state-recovery attack would enable the adversary to forge valid tags by computing the encryption forwards, or even to recover the key by deducing the internal state after initialization by computing backwards. In v3.0, this is no longer possible as the key  $K$  is XORed to the capacity part of the state after the initialization step, and after each of both applications of  $P$  during the generation of the authentication tag. Consequently, the tag is extracted as the capacity part  $S^r$  of the state after the last key addition.

Another modification is that NORX v3.0 uses  $4w$ -bit nonces instead of  $2w$ -bit nonces for previous versions. Again, the security claims are the same as in NORX v2.0.

**NORX-8 and NORX-16.** These two primitives target lightweight applications and are variants of the NORX v2.0 design, with smaller word sizes, namely  $w = 8$  and  $w = 16$ , respectively. To achieve decent security levels, their capacities cannot be limited to  $4w$  words (which would be 32 and 64 bits, respectively). Instead, their respective capacities are increased to 88 bits and 128 bits, respectively, and their capacity parts are defined as  $(s_5, \dots, s_{15})$  and  $(s_8, \dots, s_{15})$ , respectively.

The respective key lengths for NORX-8 and NORX-16 are 80 and 96 bits, and the tag length is again the same as the key length, which define the security levels claimed for these two primitives.

In the case of NORX-8, the tag length exceeds the rate of the sponge construction. Consequently, the tag cannot be extracted at once. Instead, the 40 bits of the rate part are extracted as the first half of the tag, then an extra constant 0x08 is XORed to  $s_{15}$ ,  $P$  updates the internal state, and the second half of the tag is extracted as the rate part of the state.

The amount of data processed with a given key is limited to respectively  $2^{24}$  and  $2^{32}$  messages.

<sup>2</sup>Note that the NORX specifications (v2.0 and v3.0) are unclear whether the data limitation refers either to a number of initializations or to a number of calls to the core permutation. We chose the latter as it captures both cases.



**Parallel Mode of Operation.** The NORX variants submitted to the CAESAR competition offer a parallel mode of operation, which enables to process in parallel  $p > 1$  blocks of plaintext simultaneously. Basically, the state of the mode of operation is diversified into  $p$  branches, the plaintext blocks are dispatched over the branches for processing, the branches are combined, and the trailer and tag are handled as in the serial mode.

### 3 Cryptanalysis of NORX v2.0

We give in this section the details of a ciphertext-only forgery attack on NORX v2.0 that exists due to a combination of aggressive choices made by the designers. The attack indeed relies on strong non-random properties of the underlying permutation  $P = F^l$  used in a keyed-sponge mode, as well as a relatively small sponge capacity. Additionally, we show that the forgery attack yields a plaintext-recovery attack and a key-recovery attack with the same complexities. We begin in Subsection 3.1 by giving non-random properties of  $F$  that extend to  $P$ , describe a simplified version of the forgery attack in Subsection 3.2 and then the full attack in Subsection 3.3. We discuss requirements for the adversarial model in Subsection 3.4 and give extensions of the attack in Subsection 3.5.

#### 3.1 Non-Random Properties of $F$

In the specification document of NORX [AJN15b] and in another analysis paper [AJN15a], the designers acknowledge the use of a permutation that presents non-random properties. They argue that distinguishers on the permutation do not affect the overall monkeyDuplex construction since domain separation constants are used at the mode level. Security proofs have been written for the NORX mode, e.g. [JLM14], which assumes an ideal permutation and sets aside its structural weaknesses.

In the sequel, we recall a strong distinguisher on  $F$  and later show how to leverage it to attack the full primitive. We note that our attack does not invalidate the security proofs of the mode, which rely on the assumption that the permutation is ideal and does not present any distinguisher like the one we describe.

**Previous Work.** First, in [AJN15b], the designers use the constraint solver STP to confidently assume that the permutations used in all NORX variants present only a single fixed-point, namely the all-zero state:  $\{x, F(x) = x\} = \{0\}$ . Later in [AJN15a], the same authors introduce a class of  $2^{4w}$  *weak states* of the form

$$\begin{bmatrix} a & a & a & a \\ b & b & b & b \\ c & c & c & c \\ d & d & d & d \end{bmatrix}, \quad (a, b, c, d) \in \text{GF}(2^w),$$

where all the four columns of the state are equal. Due to the column/diagonal applications of  $G$  in the permutation  $F$  (see Section 2), it is easy to see that the set of these weak states is stable by  $F$ : starting from a weak state, applying  $F$  any number of times leads to a weak state. In particular, the set of weak states is stable by  $P = F^l$ .

**A Stronger Distinguisher.** We note here that there exists a larger class of  $2^{8w}$  states behaving in a similar way, where the two left columns equal the two right ones; namely, states of the form:

$$\begin{bmatrix} a & e & a & e \\ b & f & b & f \\ c & g & c & g \\ d & h & d & h \end{bmatrix}, \quad (a, b, c, d, e, f, g, h) \in \text{GF}(2^w).$$



Again, this larger class is stable by F and P.

Additionally, we note that one can slightly generalize the notion by considering “rotated” variants of one state. More formally, we denote by  $S^{\lll i}$  the state  $S$  where the columns are left-rotated by  $i$  positions. Given  $x_i \in \text{GF}(2^w)$ ,  $0 \leq i < 16$ , consider the state

$$S = \begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{bmatrix},$$

and the three states obtained by rotating the columns of  $S$  by one, two and three positions:

$$S^{\lll 1} = \begin{bmatrix} x_1 & x_2 & x_3 & x_0 \\ x_5 & x_6 & x_7 & x_4 \\ x_9 & x_{10} & x_{11} & x_8 \\ x_{13} & x_{14} & x_{15} & x_{12} \end{bmatrix}, S^{\lll 2} = \begin{bmatrix} x_2 & x_3 & x_0 & x_1 \\ x_6 & x_7 & x_4 & x_5 \\ x_{10} & x_{11} & x_8 & x_9 \\ x_{14} & x_{15} & x_{12} & x_{13} \end{bmatrix}, S^{\lll 3} = \begin{bmatrix} x_3 & x_0 & x_1 & x_2 \\ x_7 & x_4 & x_5 & x_6 \\ x_{11} & x_8 & x_9 & x_{10} \\ x_{15} & x_{12} & x_{13} & x_{14} \end{bmatrix}.$$

Our main observation is that F and the column rotations commute, that is:

$$\forall i \in \{1, 2, 3\}, \quad F(S^{\lll i}) = F(S)^{\lll i}.$$

We define by *symmetric* a state  $S$  that is invariant by rotation by two positions:  $S = S^{\lll 2}$ . Similarly, we say that the capacity part of an internal state is symmetric if this internal state restricted to that part is invariant by rotation by two positions.

In the following section, we show how the small proportion of the internal state allocated to the capacity in both NORX-32 v2.0 and NORX-64 v2.0 allows to use this structural distinguisher to mount a ciphertext-only forgery attack on these two primitives.

### 3.2 Ciphertext-Only Forgery of NORX v2.0 Without Padding

Recall that the security of NORX- $w$  relies on a capacity of  $4w$  bits, and its key and tag sizes are of the same size  $4w$  bits.

We now consider a modified version of NORX, in which the plaintext (and therefore ciphertext) lengths are always a multiple of the block size  $12w$ . Therefore, no padding needs to be added to the plaintext before encryption. This modification enables us to describe a simplified version of our attack, which can be adapted to the full NORX v2.0 as shown in Section 3.3.

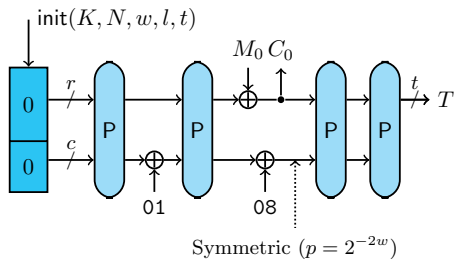
The following describes a ciphertext-only forgery attack against NORX v2.0 without padding, that requires  $q$  valid ciphertext/tag pairs  $(C, T)$ , performs  $q$  forgery attempts, and has success probability

$$1 - \left(1 - \frac{1}{2^{2w}}\right)^q.$$

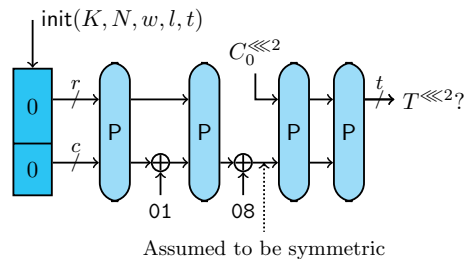
In particular, the forgery attacks succeeds with probability  $1 - 1/e \approx 63\%$  for  $q = 2^{2w}$ , and with probability about  $q \cdot 2^{-2w}$  for smaller values of  $q$ . We require that there is no trailer, that the plaintexts and ciphertexts lengths are multiples of the block size, and that the cipher does not apply any padding. Without loss of generality, we assume there is no header and that the plaintext and ciphertext length is exactly one block. If it is not the case, the attack applies directly by applying ciphertext modifications only on the last block.

Assume that an attacker has  $q$  known tuples  $(N^i, C^i, T^i)$  in its possession, resulting from the NORX- $w$  encryption of unknown messages  $M^i$ , under known pairwise distinct nonces  $N_i$  and unknown key  $K$ :

$$(N^i, C^i, T^i) = \text{Enc}(K, N^i, M^i).$$



**Figure 4:** Forgery first step: assume the capacity is symmetric (probability  $2^{-2w}$ ).



**Figure 5:** Forgery second step: attempt forgery with rotated ciphertext and tag.

Given such a tuple,  $(N, C, T)$ , the attacker attempts to produce a forgery by considering the message  $(N, C^{\lll 2}, T^{\lll 2})$ . The ciphertext and tag parts of the message are rotated variants of the initial ciphertext and tag. In the event that the capacity part of the state is symmetric before the last two calls to  $P$  for the generation of the tag (see Figure 4), the states  $S_*$  and  $S'_*$  at the same point of the computation are rotated versions of each other, and due to the fact that  $P$  and the rotation commute, this is also satisfied by the tags. More formally, we have the internal state  $S'_*$  as

$$\begin{aligned} S'_* &= C_0^{\lll 2} \parallel S_*^c, \\ &= C_0^{\lll 2} \parallel (S_*^c)^{\lll 2}, \\ &= (C_0 \parallel S_*^c)^{\lll 2}. \end{aligned}$$

and evaluate the two last applications of  $P$ , which gives

$$\begin{aligned} P^2(S'_*) &= P^2\left((C_0 \parallel S_*^c)^{\lll 2}\right), \\ &= (P^2(C_0 \parallel S_*^c))^{\lll 2}, \end{aligned}$$

and then yield the equality on the authentication tags

$$T'_* = T_*^{\lll 2}.$$

The probability for a tuple to yield an internal state such that its capacity is symmetric before the last two calls to  $P$  for the generation of the tag (see Figure 4) is  $2^{-2w}$ .

All in all, as an attacker has a probability of  $2^{-2w}$  to forge a valid message due to the symmetries in  $P$ , he only needs about  $2^{2w}$  known ciphertext/tag pairs to launch the attack and break the authenticity of NORX-w.

### 3.3 Forgery Attack Against NORX v2.0

We now adapt the attack to take into account the padding systematically applied by NORX to any non-empty plaintext.

The difficulty introduced by the padding is that the attacker has no longer access to the whole rate part of the state  $S_*$ : the part corresponding to the padding is not included in the ciphertext. In order to minimize this unknown component, we consider only messages of size  $12w - 2$  bits, which lead to the minimal padding length of two bits.

In order to forge a message using the commuting rotation property of  $P$ , the attacker has to produce a ciphertext  $C'$  such that the state  $S'_*$  is the rotated version of state  $S_*$ . In addition to the constraint that the capacity part of the state remains unchanged, new

constraints are introduced by the padding, stemming from the matching between

$$(S_*^r)^r = \begin{bmatrix} c'_0 & c'_1 & c'_2 & c'_3 \\ c'_4 & c'_5 & c'_6 & c'_7 \\ c'_8 & c'_9 & c'_{10} & c'_{11} \parallel v \end{bmatrix} \quad \text{and} \quad (S_*^r)^{\lll 2} = \begin{bmatrix} c_2 & c_3 & c_0 & c_1 \\ c_6 & c_7 & c_4 & c_5 \\ c_{10} & c_{11} \parallel v & c_8 & c_9 \end{bmatrix},$$

with  $v$  the unknown part of  $S_*^r$ . Note that the 2-bit padding  $v$  only depends on  $C$  and  $C'$  through their length, and is thus repeated in both  $S_*$  and  $S_*^r$ . Denoting by  $\underline{x}$  the last two bits of  $x$ , the padding constraints are satisfied if we set the bits of  $C'$  to the corresponding known bits of  $C$ , and additionally

$$\underline{c'_9} = v \quad \text{and} \quad \underline{c_9} = v.$$

Setting  $\underline{c_9} = \underline{c'_9}$ , the constraints boil down to  $\underline{c_9} = v$  which holds with probability  $2^{-2}$ .

Overall, taking the padding into account results in a decrease of the advantage of the attacker, that can be limited to a factor  $2^{-2}$  for the most favorable message length. This attack can trivially be extended to any padding length  $p \leq 2w$  with complexity  $2^{2w+p}$  instead of  $2^{2w+2}$ .

### 3.4 Adversarial Model Discussion

Our attack is efficient on the padded version of NORX only if the length of the padding appended to the plaintext leading to the ciphertext the adversary tries to modify is minimal. Formally, if we keep the minimal padding length of two bits, this can lead to the following two scenarios:

- In a chosen-plaintext setting, the adversary can select plaintexts of length equal to  $12w - 2 \pmod{12w}$ . The success probability of each forgery attempt is then  $2^{-2w-2}$ .
- In a ciphertext-only setting, the attack still works as the adversary does not need to know the value of the corresponding plaintext. However, it requires that ciphertexts whose last block has a specific length are available. Under the hypothesis that the length of the message follows a uniform distribution modulo  $12w$ , the adversary can try to modify only those ciphertexts, which introduces a factor  $12w$  in the data complexity.

We note that this constraint relies on the general description of NORX at the bit level, whereas the functional requirements of the CAESAR competition acts on byte strings. Consequently, to launch the attack in that case, ciphertexts of  $L$  bytes are required, with  $L = -1 \pmod{12w/8}$  and the advantage of the attacker becomes  $q \cdot 2^{-2w-8}$ . If this requirement on  $L$  does not hold, the data complexity would increase by a factor  $12w/8$ , assuming again that the ciphertext byte-lengths modulo  $12w/8$  are uniformly distributed.

### 3.5 Key-Recovery Attack Against NORX v2.0

**Recovering the Key.** We now discuss whether it is possible to recover the encryption key from a successful forgery attempt. Once the adversary achieves such a forgery, he knows that with overwhelming probability, the capacity part of the state at the end of the encryption step is symmetric. Therefore, only  $2^{2w}$  values are possible for the capacity part of the state at that point. As the adversary knows the value of the rate part, he can recover the full state by an exhaustive search over these  $2^{2w}$  values. Trying all  $2^{2w}$  possible symmetric values at the input of  $\mathbf{F}^8$  allows to filter (on average) one internal state.

Let us suppose that the adversary additionally knows at that point the value of the plaintext returned by the decryption algorithm on his successful forgery. He can then

compute backwards up to the initialization of the state and filter the correct guess on the  $4w$ -bit constants, which subsequently reveals the  $4w$ -bit secret key.

We have successfully verified the forgery and key-recovery attacks on a toy version of NORX v2.0, by taking the word size  $w = 8$  and adopting the rotation constants of NORX-8. The pseudo-code of the attack can be found in `app:sec:pseudocode`.

**Adversarial Models.** In a ciphertext-only setting, the adversary does not get the value of the plaintext after the decryption and cannot perform the last step of the key-recovery attack. It is however possible in chosen-plaintext or chosen-ciphertext settings. If the adversary can query a decryption oracle, he gets the value of the plaintext he needs to compute backwards and recovers the key.

If the adversary can query an encryption oracle, he can encrypt arbitrary one-block plaintexts and try to forge valid ciphertexts by modifying the answers of the oracle. He can then perform the key-recovery attack on the initial plaintext-ciphertext pair.

## 4 Application to Other Variants of NORX

In this section, we study the application of our attack to other versions or variants of NORX. Namely, we show the following properties that we explain below.

1. NORX-8 is not harmed at all by our attack.
2. The parameters chosen in NORX v1.0 and NORX-16 makes our attack just as efficient as generic attacks. A consequence is that increasing the key and tag sizes for these versions would not increase their security. In particular, a surprising behavior is that if one increases the key and tag lengths of NORX-16 to 128 bits, then the security drops to  $2^{66}$ .
3. NORX v3.0 has a small class of keys on which our attack is as efficient as a generic key-recovery attack.

**NORX v1.0.** We recall that the main difference between NORX v1.0 and NORX v2.0 is that in NORX v1.0, the rate part of the state consists of words  $(s_0, \dots, s_9)$  and the capacity part of the state consists of words  $(s_{10}, \dots, s_{15})$ .

Let us consider an adversary who tries to launch our attack against NORX v1.0. Let us suppose that the bit-length of the last block is exactly  $8w$ . He can only apply the rotation on the first two rows of the state after the output of the last ciphertext block, which are filled with the last eight ciphertext words. On the last row, the same symmetry condition as in NORX v2.0 has to hold, which occurs with probability  $2^{-2w}$ .

The adversary then has to ensure that the third row of the state during its forgery attempts can be derived by a column-wise rotation of the third row of the state during the generation of the ciphertext he tries to modify. The third row of the state during the encryption equals  $(s_8, s_9, s_{10}, s_{11})$ , where  $s_8$  and  $s_9$  have just been updated by XORing the padding.

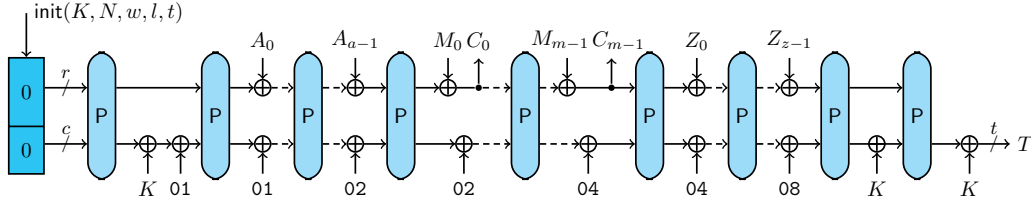
Then, during the verification of the forgery attempt, the third row contains the same value  $(s_8, s_9, s_{10}, s_{11})$ , The symmetry relations he tries to obtain are as follows:

$$s_8 = s_{10}, \quad s_9 = s_{11},$$

which hold with probability  $2^{-2w}$ .

The overall success probability of the adversary is thus  $2^{-4w}$ , which is exactly the success probability of a generic forgery attempt as the tag length is  $t = 4w$ .

**NORX v3.0.** During the tag generation phase, the only difference between NORX v2.0 and NORX v3.0 consists in XORing the key  $K$  after each application of  $P$ , as depicted on Figure 6.



**Figure 6:** NORX v3.0 serial mode.

As a consequence, the rotation property between the states during the real encryption and the forgery attempt can only be preserved before the last application of  $P$  if the key  $K = k_0 || k_1 || k_2 || k_3$  is itself symmetric; that is, if  $k_0 = k_2$  and  $k_1 = k_3$ . In that case, our attack still works.

These relations can be seen as defining a class of  $2^{2w}$  weak keys on NORX v3.0. However, the resulting attack enables an adversary to generate forgeries with data complexity  $2^{2w}$ , which is equivalent to the size of the weak key set. Furthermore, the forgery attack cannot be trivially turned into a key-recovery attack against NORX v3.0. Our attack therefore has a very limited impact on the security of NORX v3.0.

**NORX-8.** Recall that NORX-8 is very similar to NORX v2.0, but that the authentication tag cannot be fully extracted at once from the rate part of the state. Instead, after the extraction of the first 40 tag bits, a diversification constant is injected in the state,  $P$  is computed and the last 40 tag bits are extracted from the rate part of the state.

Even if the adversary achieves the rotation property after the last ciphertext block, this property is broken after the addition of the diversification constant, and no predictable property holds for the second half of the tag. In that case, only the first 32 bits of the tag (which are extracted from the first row of the state) can be predicted, leading to a forgery with probability  $2^{32-80} = 2^{-48}$ .

Furthermore, the rotation property itself only holds with probability  $2^{-48}$ , due to symmetry conditions on the last three rows of the state, which contain the capacity part. The overall success probability of our attack is therefore  $2^{-96}$ , making it less efficient than a generic attack.

**NORX-16.** In NORX-16, the capacity part of the state covers the last two rows, i.e.  $(s_8, \dots, s_{15})$ . Therefore, the rotation property holds with probability  $2^{-4w} = 2^{-64}$ . NORX-16 uses 96-bit keys and produces 96-bit tags, which are extracted as  $(s_0, \dots, s_6)$  after the last application of  $P$ . If the rotation property holds, the adversary knows the target values of  $(s_0, \dots, s_3)$  (by rotation of the valid tag), but he still needs to guess  $s_4$  and  $s_5$ . Taking account of the 2-bit loss due to the padding, the overall success probability of each forgery attempt is  $2^{-64-2 \times 16-2} = 2^{-98}$ , which is just below the generic bound for a forgery attempt.

This shows that increasing the key and tag sizes of NORX-16 would not increase its security, as our attack would still be valid. More surprisingly, using 128-bit tags would enable the adversary to always forge successfully once the rotation property is verified, leading to an attack with success probability  $2^{-66}$  for each forgery attempt.

## 5 Discussion About NORX Security Claims

**NORX v2.0 Security Claims** In [AJN15b, Section 3], the NORX designers claim that no forgery attack with  $q$  attempts should succeed with probability significantly greater than  $q \cdot 2^{-4w}$ . Our attack succeeds with probability about  $q \cdot 2^{-2w-2}$ , which violates this claim.

The designers also claim that no key-recovery attack should cost fewer than  $2^{4w}$  operations. Our attack costs  $2^{2w+2}$  operations on average. One could argue that the limitation of the amount of data treated with a given key limits the success probability of our attack. Nevertheless, contrary to attacks based on the birthday paradox, the marginal success probability of a single forgery attempt using our attack does not drop once the key is changed. Consequently, our attack enables the adversary to find the value of one of the keys used with time and data complexity of about  $2^{66}$  operations (for  $w = 32$ ), regardless of the change frequency.

**Forgeries Against NORX v3.0 for Long Messages.** For both NORX v2.0 and NORX v3.0, the security claim saying that any forgery attack with  $q$  attempts should have a success probability of about  $q \cdot 2^{-4w}$  does not totally hold.

For any long ciphertext  $C$  that contains, say,  $2^m + 1$  blocks of  $12w$  bits, one can modify only the first block of the ciphertext, keep the same tag value and obtain a forgery with probability about  $2^{m-4w}$ . Indeed, before each application of  $P$  during the decryption phase, the internal state during the forgery attempt collides with the internal state during the decryption of the initial message with probability  $2^{-c} = 2^{-4w}$ . Once a collision occurs, it holds for all the subsequent steps of the decryption process, as the two decrypted ciphertexts have common suffixes. The overall collision probability is therefore approximately  $2^m \times 2^{-4w}$ , and such a collision leads to equal tag values, making the forgery attempt successful. We note that this technique shares some ideas with the long-message internal collision attack on iterated MACs discussed in [PvO95, Section 3].

For NORX v2.0 and NORX v3.0, this property still holds when the nonce is modified in the forgery attempt. For NORX v2.0, as no key is involved after the initialization phase, one consequence of this property is that a given ciphertext of  $2^m$  blocks has the same tag value under two different keys and nonces with probability  $2^{m-4w}$ .

The impact (at least on NORX v3.0) of this remark has to be mitigated by the fact that similar properties can apply to other AEAD schemes such as AES-GCM [MV04]. It is also covered by the security proof, which leads to bounds involving the total length of encryption and decryption queries, and not only the number of forgery attempts.

**NORX Security Proof.** In [AJN15b], the designers partly derive their security analysis from security proofs of the keyed-sponge mode of operation which can be found in [JLM14]. Namely, the distinguishing advantage of any chosen-plaintext adversary against NORX is upper bounded by:

$$\Pr[\text{Privacy}] \leq \frac{3(q_p + \sigma_\varepsilon)^2}{2^{b+1}} + \left(\frac{8eq_p\sigma_\varepsilon}{2^b}\right)^{1/2} + \frac{rq_p}{2^c} + \frac{q_p + \sigma_\varepsilon}{2^k}.$$

Similarly, the upper bound for the success probability of any forgery attempt is given by:

$$\begin{aligned} \Pr[\text{Forgery}] \leq & \frac{(q_p + \sigma_\varepsilon + \sigma_{\mathcal{D}})^2}{2^b} + \left(\frac{8eq_p\sigma_\varepsilon}{2^b}\right)^{1/2} + \frac{rq_p}{2^c} \\ & + \frac{q_p + \sigma_\varepsilon + \sigma_{\mathcal{D}}}{2^k} + \frac{(q_p + \sigma_\varepsilon + \sigma_{\mathcal{D}})\sigma_{\mathcal{D}}}{2^c} + \frac{q_{\mathcal{D}}}{2^t}. \end{aligned}$$

In these formulae,  $b$  is the state size,  $c$  is the capacity,  $r$  is the rate,  $q_p$  is the number of calls to the internal permutation,  $q_{\mathcal{D}}$  is the number of forgery attempts, and  $\sigma_\varepsilon$  and  $\sigma_{\mathcal{D}}$

are the number of total computations of the internal permutations during encryption and decryption queries, respectively.

Our attack succeeds with probability  $q_{\mathcal{D}}/2^{2w+2}$ , which is significantly larger than this bound for a small number of queries (as we would have  $\sigma_{\mathcal{E}} = \sigma_{\mathcal{D}} = 4q_{\mathcal{D}}$  as we only need to make one-block encryption and decryption queries).

We emphasize that our attack does not contradict the proof of the NORX mode of operation, as it relies on the use of an ideal internal permutation instead of  $P$ . However, it reveals that the proof does not apply to the instantiation of the mode chosen by the designers, as the selected NORX permutation presents (at least) one strong structural distinguisher.

**Security Level of NORX-8.** In [AJN15c], the authors do not provide an explicit link between the above security bounds and the claimed security level of NORX-8 and NORX-16. In particular, they only state that no more than  $2^{24}$  (resp.  $2^{32}$ ) initialization phases should be performed with the same key, but they do not give any limit to the total length of messages encrypted with a key. We can notice that if one encrypts constant 0 blocks, NORX can be viewed as a stream cipher, and therefore the Babbage-Golić [Bab95, Gol97] Time-Data tradeoff applies. In particular, NORX-8 has an internal state of only 128 bits. Therefore, if one can encrypt  $2^m \gg 2^{48}$  message blocks under the same key with NORX-8, the security level drops below 80 bits since a state-recovery attack of time and memory complexity at most  $2^{128-m}$  can be mounted, that can in turn easily be converted into a key-recovery attack using backward computations.

**Interpretation of the NORX Proof.** Finally, we would like to raise the following problem. In the bound derived from the proof of the NORX mode of operation, the term  $q_p \sigma_{\mathcal{D}}/2^c$  appears. In the case of NORX-32 for both v2.0 and v3.0, the capacity equals  $c = 128$ . Note that  $\sigma_{\mathcal{D}}$  can roughly be considered as the total length of decryption queries, and is only limited to  $2^{64}$  in the specifications. In real-life applications,  $\sigma_{\mathcal{D}}$  could possibly reach between  $2^{40}$  and  $2^{48}$ .

In that case,  $q_p$  has to be smaller than  $2^{80}$  to  $2^{88}$  if one wants to conclude any meaningful information from the bound. Note however that  $q_p$  represents the number of calls to the internal permutation made by the adversary. In our view, as  $P$  is an unkeyed permutation, these calls do not involve any secret and can therefore be interpreted as offline computations. The security of NORX as derived from the security proof then drops between 80 and 88 bits.

However, this remark is very unlikely to lead to an attack on NORX v3.0 that would match this bound, for two reasons. First, when looking at the details of the proof, this term captures the event that a direct call to  $P$  by the adversary collides with an application of  $P$  during the verification of a decryption query. As the adversary does not get much information from decryption queries, it is unlikely that he can detect such an event. Second, the mode of operation of NORX v3.0 (with key additions after initialization and during the tag computation) is close to the sandwich sponge construction by Naito [Nai16]. In the same paper, this construction is proved to be indistinguishable from a PRF up to a bound without such a term proportional to online-times-offline complexity; whereas a similar term still appears in the best known bounds for the usual sponge construction.

## 6 Conclusion

In this paper, we demonstrated a ciphertext-only forgery attack against the AEAD scheme NORX v2.0 that was selected for the third round of the CAESAR competition. It requires  $2^{66}$  (resp.  $2^{130}$ ) known plaintexts and  $2^{66}$  (resp.  $2^{130}$ ) forgery attempts for the 128-bit (resp. 256-bit) key, 128-bit (resp. 256-bit) tag variant of NORX. This attack can be turned into a key



recovery if the adversary also has access to unencrypted data, i.e. in the chosen-plaintext or chosen-ciphertext settings.

We also studied its applicability to other versions and variants of NORX, and found that it competes with generic attacks against NORX v1 and NORX-16. Unlike a similar scheme with an ideal permutation, these algorithms cannot be securely used with an increased key and tag length.

Our results emphasize that security proofs of modes of operations need to be handled carefully. First, strong structural distinguishers on an internal primitive that is modeled as ideal should not be allowed. Second, one has to be very careful when deriving the level of security offered by an algorithm from the bound given by a security proof. Finally, the impact on the exact security of the cipher when an unwanted event occurs needs to be minimized, as it is the case in NORX v3.0.

## Acknowledgement

This work has been partially supported by the French Agence Nationale de la Recherche through the BRUTUS project under Contract ANR-14-CE28-0015.

## References

- [ABL<sup>+</sup>14] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to securely release unverified plaintext in authenticated encryption. In Sarkar and Iwata [SI14], pages 105–125.
- [AHMP10] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C.-W. Phan. SHA-3 proposal BLAKE. Submission to NIST (Round 3), 2010.
- [AJN15a] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. Analysis of NORX: Investigating differential and rotational properties. In Diego F. Aranha and Alfred Menezes, editors, *LATINCRYPT 2014*, volume 8895 of *LNCS*, pages 306–324. Springer, Heidelberg, September 2015.
- [AJN15b] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. NORX v2.0. Submission to the CAESAR Competition, 2015.
- [AJN15c] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. NORX8 and NORX16: Authenticated encryption for low-end systems. *Cryptology ePrint Archive*, Report 2015/1154, 2015.
- [ANWW13] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O’Hearn, and Christian Winnerlein. BLAKE2: Simpler, smaller, fast as MD5. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 13*, volume 7954 of *LNCS*, pages 119–135. Springer, Heidelberg, June 2013.
- [Bab95] Steve Babbage. Improved “exhaustive search” attacks on stream ciphers. In *European Convention on Security and Detection*, no. 408 in *IEE Conference Publication*, pages 161–166. IET, 1995.
- [BBK<sup>+</sup>13] Begül Bilgin, Andrey Bogdanov, Miroslav Knežević, Florian Mendel, and Qingju Wang. Fides: Lightweight authenticated cipher with side-channel resistance for constrained hardware. In Guido Bertoni and Jean-Sébastien Coron, editors, *CHES 2013*, volume 8086 of *LNCS*, pages 142–158. Springer, Heidelberg, August 2013.

- [BDPV12] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In Ali Miri and Serge Vaudenay, editors, *SAC 2011*, volume 7118 of *LNCS*, pages 320–337. Springer, Heidelberg, August 2012.
- [BDPVA11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: single-pass authenticated encryption and other applications. In *International Workshop on Selected Areas in Cryptography*, pages 320–337. Springer, 2011.
- [Ber08] Daniel J. Bernstein. ChaCha, a variant of Salsa20, 2008.
- [BHJ<sup>+</sup>16] Nasour Bagheri, Tao Huang, Keting Jia, Florian Mendel, and Yu Sasaki. Cryptanalysis of reduced NORX. In Thomas Peyrin, editor, *FSE 2016*, volume 9783 of *LNCS*, pages 554–574. Springer, Heidelberg, March 2016.
- [DEMS16] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2. Submission to the CAESAR Competition, 2016.
- [DJ15] Itai Dinur and Jérémy Jean. Cryptanalysis of FIDES. In Carlos Cid and Christian Rechberger, editors, *FSE 2014*, volume 8540 of *LNCS*, pages 224–240. Springer, Heidelberg, March 2015.
- [DKM<sup>+</sup>16] Ashutosh Dhar Dwivedi, Miloš Klouček, Pawel Morawiecki, Ivica Nikolić, Josef Pieprzyk, and Sebastian Wójtowicz. SAT-based cryptanalysis of authenticated ciphers from the CAESAR competition. *Cryptology ePrint Archive*, Report 2016/1053, 2016.
- [DMM15] Sourav Das, Subhamoy Maitra, and Willi Meier. Higher order differential analysis of NORX. *Cryptology ePrint Archive*, Report 2015/186, 2015.
- [Gol97] Jovan Dj. Golić. Cryptanalysis of alleged A5 stream cipher. In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 239–255. Springer, 1997.
- [JLM14] Philipp Jovanovic, Atul Luykx, and Bart Mennink. Beyond  $2^{c/2}$  security in sponge-based authenticated encryption modes. In Sarkar and Iwata [SI14], pages 85–104.
- [KN10] Dmitry Khovratovich and Ivica Nikolić. Rotational cryptanalysis of ARX. In Seokhie Hong and Tetsu Iwata, editors, *FSE 2010*, volume 6147 of *LNCS*, pages 333–346. Springer, Heidelberg, February 2010.
- [LMR15] Gregor Leander, Brice Minaud, and Sondre Rønjom. A generic approach to invariant subspace attacks: Cryptanalysis of robin, iSCREAM and Zorro. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 254–283. Springer, Heidelberg, April 2015.
- [MV04] David McGrew and John Viega. The galois/counter mode of operation (GCM). *Submission to NIST.*, 2004.
- [Nai16] Yusuke Naito. Sandwich construction for keyed sponges: Independence between capacity and online queries. In Sara Foresti and Giuseppe Persiano, editors, *Cryptology and Network Security - 15th International Conference, CANS 2016, Milan, Italy, November 14-16, 2016, Proceedings*, volume 10052 of *Lecture Notes in Computer Science*, pages 245–261, 2016.

- 
- [PvO95] Bart Preneel and Paul C. van Oorschot. MDx-MAC and building fast MACs from hash functions. In Don Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 1–14. Springer, Heidelberg, August 1995.
- [SI14] Palash Sarkar and Tetsu Iwata, editors. *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*. Springer, Heidelberg, December 2014.

## A Ciphertext-only Forgery and Key-Recovery Attack

The pseudo-code for the forgery and key-recovery attacks are given in the following [Algorithm 3](#). We have implemented the attack on a toy example of NORX v2.0 derived from the source code provided by the designers as part of the CAESAR competition. We in particular emphasize that due to the CAESAR requirements, all the inputs are byte strings, hence the padding cannot be restricted to less than one byte.

---

### Algorithm 3 – Forgery and Key-Recovery Attack on NORX v2.0

---

**Input:**  $2^{2w}$  ciphertext/tag pairs  $(C_i, T_i)$ ,  $2w$ -bit nonce  $N = n_0 || n_1$

**Output:** Secret key  $K$

```

1: for each ciphertext  $C_i = (c_0, \dots, c_{10})$  and tag  $T_i = (t_0, \dots, t_3)$  do
2:    $C'_i \leftarrow (c_2, c_3, c_0, c_1, c_6, c_7, c_4, c_5, c_{10}, c_9, c_8)$ 
3:    $T'_i \leftarrow (t_2, t_3, t_0, t_1)$ 
4:    $M' \leftarrow \text{Dec}(N, C'_i, T'_i)$ 
5:   if  $M' \neq \perp$  then
6:     for all words  $a, b$  do
7:        $S \leftarrow (c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_9, a, b, a, b)$ 
8:        $S \leftarrow S \oplus M' || 0^c$ 
9:        $s_{15} \leftarrow s_{15} \oplus 08$ 
10:       $S \leftarrow P^{-1}(S)$ 
11:       $s_{15} \leftarrow s_{15} \oplus 02$ 
12:       $S \leftarrow P^{-1}(S)$ 
13:      if  $(s_0, s_1, s_2, s_3) = (n_0, n_1, u_2, u_3)$  then
14:        return  $K = (s_4, s_5, s_6, s_7)$ 
15:      end if
16:    end for
17:  end if
18: end for

```

---