NORX8 and NORX16: Authenticated Encryption for Low-End Systems

Jean-Philippe Aumasson\textsuperscript{1}, Philipp Jovanovic\textsuperscript{2} and Samuel Neves\textsuperscript{3}

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

Abstract—This paper presents NORX8 and NORX16, the 8-bit and 16-bit versions of the authenticated cipher NORX, one of the CAESAR candidates. These new versions are better suited for low-end systems—such as “internet of things” devices—than the original 32-bit and 64-bit versions: whereas 32-bit NORX requires 64 bytes of RAM or cache memory, NORX8 and NORX16 require just 16 and 32 bytes, respectively. Both of the low-end variants were designed to retain the security properties of the initial NORX and be fast on small CPUs.

Keywords—authenticated encryption, lightweight, CAESAR

I. INTRODUCTION

NORX\textsuperscript{1} is a family of authenticated ciphers: given a secret key, a nonce, a message, and (optionally) associated data, a NORX cipher returns an encrypted message and an authentication tag. NORX was submitted to the CAESAR competition\textsuperscript{2} in 2014, and has been analysed with respect to its core algorithm as well as to its mode of operation\textsuperscript{3},\textsuperscript{4}, and is not known to have cryptographic weaknesses.

The submission to CAESAR included algorithms based either on 32-bit or on 64-bit word arithmetic, denoted as NORX\textsubscript{32} and NORX\textsubscript{64}, respectively. This paper describes two new variants of NORX designed for low-end systems: NORX\textsubscript{8} and NORX\textsubscript{16}. These are based on 8-bit and 16-bit words, and require respectively 1/4 and 1/2 the memory as the previously smallest NORX instance. We designed NORX\textsubscript{8} and NORX\textsubscript{16} to offer a low-memory, secure, and fast enough authenticated cipher for resource-constrained systems.

II. SPECIFICATION

This section is a succinct specification of the new NORX instances and although it is basically self-contained, we refer to the CAESAR submission document\textsuperscript{1} for detailed documentation and design rationale.

A. Generalities

In this work, we introduce two new classes of the NORX family:

1) NORX\textsubscript{8}, with word size $w = 8$, tag size $t \leq 80$, and number of rounds $1 \leq l \leq 63$.
2) NORX\textsubscript{16}, with word size $w = 16$, tag size $t \leq 96$, and number of rounds $1 \leq l \leq 63$.

Instances from these new classes are denoted by NORX\textsubscript{w-l-p-t}. We assume that the parallelism degree $p$ is set to 1 (fully serial versions)—although parallel versions could be constructed, we do not expect relevant use cases.

1) Encryption Interface: NORX8 and NORX16 encryption take as input keys $K$ of $k = 80$ and $k = 96$ bits, respectively, a nonce $N$ of $n = 32$ bits, and a datagram $A \parallel M \parallel Z$ where, $A$ is a header, $M$ a message, and $Z$ a trailer. $|A|$, $|M|$, and $|Z|$ are allowed to be 0. NORX encryption produces a ciphertext $C$, with $|C| = |M|$, and an authentication tag $T$.

2) Decryption Interface: NORX decryption is similar to encryption: Besides $K$ and $N$, it takes as input a datagram $A \parallel C \parallel Z$, where $A$ and $Z$ denote header and trailer, and $C$ the encrypted payload, with $|A|$, $|C|$, and $|Z|$ may be 0. The last input is an authentication tag $T$. Decryption either returns failure, upon failed verification of the tag, or produces a plaintext $M$ of the same size as $C$ if the tag verification succeeds.

B. Layout Overview

Like the original NORX versions, the new variants are based on the \textit{monkeyDuplex} construction\textsuperscript{5},\textsuperscript{6}. An overview of the layout is given in Figure\textsuperscript{1}.

The round function $F$ is a permutation of $b = r + c$ bits, where $b$ is called the width, $r$ the rate (or block length), and $c$ the capacity. We call $F$ a \textit{round} and $F^l$ denotes its $l$-fold iteration. The internal state $S$ of
NORX is viewed as a concatenation of 16 words, i.e. 
\( S = s_0 \parallel \cdots \parallel s_{15} \), which are conceptually arranged in a 
4 \times 4 matrix. The so-called rate words are used for 
data block injection which are \( s_0, \ldots, s_4 \) for NORX8 
and \( s_0, \ldots, s_7 \) for NORX16. The capacity words on the 
other hand are unchanged during data processing and 
ensure the security of the scheme. These are \( s_5, \ldots, s_{15} \) 
and \( s_8, \ldots, s_{15} \) for NORX8 and NORX16, respectively. 
Proposals for concrete parameters are given in Table I. 
We consider NORX8 and NORX16 with \( l = 4 \) to be the 
default instances.

| Table I | Proposed parameter combinations of the lightweight 
NORX variants. |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( w )</td>
<td>( l )</td>
</tr>
<tr>
<td>8</td>
<td>4 or 6</td>
</tr>
<tr>
<td>16</td>
<td>4 or 6</td>
</tr>
</tbody>
</table>

C. The Round Function \( F \)

\( F \) processes a state \( S \) by first transforming its four 
columns with

\[
\begin{align*}
G(s_0, s_4, s_8, s_{12}) & = G(s_1, s_5, s_9, s_{13}) \\
G(s_2, s_6, s_{10}, s_{14}) & = G(s_3, s_7, s_{11}, s_{15})
\end{align*}
\]

and then transforming its four diagonals with

\[
\begin{align*}
G(s_0, s_5, s_{10}, s_{15}) & = G(s_1, s_6, s_{11}, s_{12}) \\
G(s_2, s_7, s_8, s_{13}) & = G(s_3, s_4, s_9, s_{14})
\end{align*}
\]

Those two operations are called column step and diagonal step, as in BLAKE2 \[2\] and NORX \[11, 8\].

The permutation \( G \) transforms four words \( a, b, c, d \) by computing (top-down, left-to-right):

1. \( a \leftarrow (a \oplus b) \oplus ((a \wedge b) \ll 1) \)
2. \( d \leftarrow (a \oplus d) \gg r_0 \)
3. \( c \leftarrow (c \oplus d) \oplus ((c \wedge d) \ll 1) \)
4. \( b \leftarrow (b \oplus c) \gg r_1 \)

5. \( a \leftarrow (a \oplus b) \oplus ((a \wedge b) \ll 1) \)
6. \( d \leftarrow (a \oplus d) \gg r_0 \)
7. \( c \leftarrow (c \oplus d) \oplus ((c \wedge d) \ll 1) \)
8. \( b \leftarrow (b \oplus c) \gg r_1 \)

The rotation offsets \( (r_0, r_1, r_2, r_3) \) are \( (1, 3, 5, 7) \) for 
NORX8, and \( (8, 11, 12, 15) \) for NORX16. They were 
chosen such that similar performance and security 
goals are achieved as in the case of NORX32 and 
NORX64 \[1, 8\]. In particular, full diffusion is provided 
after F^2 in both cases.

D. Encryption and Tag Generation

NORX encryption can be divided into three main 
phases: initialisation, message processing, and tag generation. 
Processing of a datagram \( A \parallel M \parallel Z \) is done in 
up to three steps: header processing, payload processing, 
and trailer processing. The number of steps depends on 
whether \( A, M, \) or \( Z \) are empty or not. NORX skips 
processing phases of empty message parts. For example, 
in the simplest case when \( |A| = |Z| = 0, |M| > 0 \), 
message processing is done in one step, since only the 
payload \( M \) needs to be encrypted and authenticated.

Below, we first describe the padding and domain 
separation rules, then each of the aforementioned phases.

1) Padding: NORX uses the multi-rate padding \[6\], 
deﬁned by \( \text{pad}_r : X \mapsto X \parallel 0^{9r} \) with bitstrings \( X \) 
and \( 10^q1 \), and \( q = (−|X| − 2) \mod r \). This extends \( X \) 
to a multiple of the rate \( r \) and guarantees that the last 
block of \( \text{pad}_r(X) \) differs from the all-zero block \( 0^r \).

2) Domain Separation: NORX performs domain 
separation by XORing a domain separation constant to the 
least signiﬁcant byte of \( s_{15} \) each time before the state is 
transformed by the permutation \( F^i \). Distinct constants are 
used for the different phases of message processing and 
tag generation. Table II gives the speciﬁcation of those 
constants and Figure 1 illustrates their integration into 
the state of NORX.

<table>
<thead>
<tr>
<th>Table II</th>
<th>Domain separation constants.</th>
</tr>
</thead>
<tbody>
<tr>
<td>header</td>
<td>payload</td>
</tr>
<tr>
<td>01</td>
<td>02</td>
</tr>
</tbody>
</table>

3) Initialisation: This phase processes a secret key \( K \), 
a nonce \( N \) and the parameters \( w, l, p, \) and \( t \). For \( w = 8 \), 
we have \( K = k_0 \parallel \cdots \parallel k_9 \) and \( N = n_0 \parallel \cdots \parallel n_3 \) of 
 sizes 80 and 32 bits, respectively. For \( w = 16 \), we have 
\( K = k_0 \parallel \cdots \parallel k_{15} \) and \( N = n_0 \parallel n_1 \) of sizes 96 and 32
bits, respectively. First, the state \( S = s_0 \parallel \cdots \parallel s_{15} \) is initialised. For NORX8 this is done by

\[
\begin{pmatrix}
  s_0 & s_1 & s_2 & s_3 \\
  s_4 & s_5 & s_6 & s_7 \\
  s_8 & s_9 & s_{10} & s_{11} \\
 s_{12} & s_{13} & s_{14} & s_{15}
\end{pmatrix}
\leftarrow
\begin{pmatrix}
  n_0 & n_1 & n_2 & n_3 \\
  k_0 & k_1 & k_2 & k_3 \\
  u_8 & u_9 & u_{10} & u_{11} \\
 u_{12} & u_{13} & u_{14} & u_{15}
\end{pmatrix}
\]

For NORX16 state initialisation is

\[
\begin{pmatrix}
  s_0 & s_1 & s_2 & s_3 \\
  s_4 & s_5 & s_6 & s_7 \\
  s_8 & s_9 & s_{10} & s_{11} \\
 s_{12} & s_{13} & s_{14} & s_{15}
\end{pmatrix}
\leftarrow
\begin{pmatrix}
  n_0 & n_1 & n_2 & n_3 \\
  k_0 & k_1 & k_2 & k_1 \\
  u_8 & u_9 & u_{10} & u_{11} \\
 u_{12} & u_{13} & u_{14} & u_{15}
\end{pmatrix}
\]

The constants can be computed in both cases through

\[
(u_0,\ldots,u_{15}) \leftarrow \text{F}^2(0,\ldots,15)
\]

using the respective variants of \( \text{F}^2 \). Note, however that only a particular subset of the generated constants is used: \( u_{14} \) and \( u_{15} \) for NORX8 and \( u_8,\ldots,u_{15} \) for NORX16. Afterwards, the parameters \( w, l, p, \) and \( t \) are integrated into the state \( S \) by XORing them to \( s_{12}, s_{13}, s_{14}, \) and \( s_{15} \), respectively. Finally, \( S \) is updated with \( \text{F}^l \).

4) Message Processing: Message processing is the main phase of NORX encryption or decryption.

a) Header Processing: If \( |A| = 0 \), this step is skipped, otherwise let \( \text{pad}_r(A) = A_0 \parallel \cdots \parallel A_{a-1} \) denote the padded header data, with \( r \)-bit sized header blocks \( A_i = a_{i,0} \parallel \cdots \parallel a_{i,r/w-1} \) and \( 0 \leq i \leq a-1 \). Then \( A_i \) is processed by:

\[
s_{15} \leftarrow s_{15} \oplus 01 \\
S \leftarrow \text{F}^l(S) \\
s_j \leftarrow s_j \oplus a_{i,j}, \text{ for } 0 \leq j \leq r/w-1
\]

b) Payload Processing: If \( |M| = 0 \), this step is skipped. Otherwise, payload data is padded using the multi-rate padding and then encrypted. Let \( \text{pad}_r(M) = M_0 \parallel \cdots \parallel M_{m-1} \). To encrypt \( M_i = m_{i,0} \parallel \cdots \parallel m_{i,r/w-1} \) and get a new ciphertext block \( C_i = c_{i,0} \parallel \cdots \parallel c_{i,r/w-1} \) the following steps are executed

\[
s_{15} \leftarrow s_{15} \oplus 02 \\
S \leftarrow \text{F}^l(S) \\
s_j \leftarrow s_j \oplus m_{i,j}, \text{ for } 0 \leq j \leq r/w-1 \\
c_{i,j} \leftarrow s_j
\]

for \( 0 \leq i < m-1 \). For \( i = m-1 \), the procedure is almost the same, but only a truncated ciphertext block is created such that \( C \) has the same length as (unpadded) \( M \). In other words, padding bits are never written to \( C \).

c) Trailer Processing: Trailer data \( Z \) is processed in a similar way as header data. The only difference is that the domain separation constant \( 04 \) is used instead of \( 02 \), see Table II

5) Tag Generation: Computation of the authentication tag \( T \) is handled slightly different for NORX8 and NORX16. Both variants first execute the following steps:

\[
s_{15} \leftarrow s_{15} \oplus 08 \\
S \leftarrow \text{F}^l(S) \\
T \leftarrow T \parallel s_0 \parallel \cdots \parallel s_{r/w-1}
\]

While the tag can be extracted at once for NORX16, this is not possible for NORX8 in most cases as the rate only has a size of 40 bits. Thus, NORX8 performs the following operations for each block required after the first:

\[
s_{15} \leftarrow s_{15} \oplus 08 \\
S \leftarrow \text{F}^l(S) \\
T \leftarrow T \parallel s_0 \parallel \cdots \parallel s_{r/w-1}
\]

In summary, 3l rounds are necessary to extract an 80-bit tag for NORX8.

E. Decryption and Tag Verification

NORX decryption mode is similar to the encryption mode. The only two differences are described below.

1) Message Processing: Processing header \( A \) and trailer \( Z \) of \( A \parallel C \parallel Z \) is done in the same way as for encryption. Decryption of the encrypted payload \( C \) is achieved as follows:

\[
s_{15} \leftarrow s_{15} \oplus 02 \\
S \leftarrow \text{F}^l(S) \\
m_{i,j} \leftarrow s_j \oplus c_{i,j}, \text{ for } 0 \leq j \leq r/w-1 \\
s_j \leftarrow c_{i,j}
\]

Like in encryption, as many bits are extracted and written to \( M \) as unpadded encrypted payload bits.

2) Tag Verification: This step is executed after tag generation. Let \( T \) and \( T' \) denote the received and the generated tag. If \( T = T' \), tag verification succeeds; otherwise it fails, the decrypted payload is discarded and an error is returned.
III. HARDWARE REQUIREMENTS

In this section, we present preliminary estimates of the required gate-equivalents (GE) when realising NORX8 and NORX16 in hardware. We assume that the ciphers are implemented for a technology that needs 7 GE per D-flip-flop, 3 GE per XOR and 2 GE per AND. To store the states of NORX8 and NORX16 a total of 128 and 256 D-flip-flops are necessary amounting to $7 \cdot 128 = 896$ GE and $7 \cdot 256 = 1792$ GE, respectively. Implementing $G$ requires 12 XORs, 4 ANDs, and some bit shifts for $\ll 1$ and cyclic rotations $r$. Bit shifts can be ignored for GE estimations since they are realised through re-wiring. The $8$- and 16-bit $G$ functions therefore need $(3 \cdot 12 + 2 \cdot 4) \cdot 8 = 44 \cdot 8 = 352$ GE and $44 \cdot 16 = 704$ GE, respectively. The difference between the column and diagonal steps is also just a re-wiring of state elements and therefore requires no additional GE. Absorption of $r$-bit data blocks is realised through bitwise XOR. Thus, an additional number of $3 \cdot 40 = 120$ GE (NORX8) and $3 \cdot 128 = 384$ GE (NORX16) are necessary. In summary, the lower bounds for hardware implementations can be estimated by $896 + 352 + 120 = 1368$ GE for NORX8 and $1792 + 704 + 384 = 2880$ GE for NORX16.

IV. SECURITY GOALS

NORX8 and NORX16 follow the same security paradigms like their bigger siblings NORX32 and NORX64, i.e. it is assumed that adversaries are nonce-respecting and that nothing but an error is returned on a tag verification failure. The security of the schemes is limited by key and tag sizes of $k = t = 80$ bits (NORX8) and of $k = t = 96$ bits (NORX16) for our proposed instances, see Table [1]. We set the usage exponent $e$ to 24 (NORX8) and 32 (NORX16) which limits the number of initialisations to $2^e$ before a given key has to be changed. According to the results for keyed sponge constructions presented in [9], NORX8 and NORX16 are expected to indeed achieve the generic security bounds of 80 and 96 bits, respectively.

V. PRELIMINARY CRYPTANALYSIS

We conducted a preliminary analysis of certain properties of NORX with $w \in \{8, 16\}$, and used similar techniques as presented in [13] for $w \in \{32, 64\}$. The rotation offsets $(1, 3, 5, 7)$, for $w = 8$, and $(8, 11, 12, 15)$, for $w = 16$, of the round function were chosen such that $F^2$ provides full diffusion, as already mentioned in Section II-C. Additionally, the above rotation offsets ensure that the function $G$ has no fixed points, which are values that satisfy $G(a, b, c, d) = (a, b, c, d)$, except for the trivial one $G(0, 0, 0, 0) = (0, 0, 0, 0)$. As a consequence, $F^2$ also has no fixed points except for the all-zero point. Our SMT/SAT-solver-based differential cryptanalysis of $F^2$ showed that the probabilities of differential characteristics are upper-bounded by $2^{−29}$ ($w = 8$) and $2^{−37}$ ($w = 16$). Moreover, the differential probabilities for $F$ during initialisation are upper-bounded by $2^{−31}$ ($w = 8$) and $2^{−53}$ ($w = 16$) for the case where only the nonce words can be modified. In this scenario, 7 rounds of initialisation can be roughly upper-bounded by $2^{−31+3(−29)} = 2^{−118}$ (NORX8) and $2^{−53+3(−37)} = 2^{−164}$ (NORX16), respectively.

VI. CONCLUSION

In this work, we presented NORX8 and NORX16, the two latest members of the NORX family of authenticated encryption schemes targeted at low-end systems. The reference source code for both new variants of NORX will be released on the official NORX website [10]. The NORX family of authenticated encryption algorithms is free for everyone to use and we have neither filed nor have we planned to file a patent application for the algorithm.

REFERENCES