KNOT: Algorithm Specifications and Supporting Document

KNOT is a family of bit-slice lightweight authenticated encryption algorithms and hash functions, which is well-suited for both hardware and software environments. Chinese knot is an ancient art of weaving. In Chinese, " knot" has the meaning of connection, reunion and harmony, which is the origin of our submission name " KNOT".

1 Notation

The following table summarizes the notation used throughout this document.

Notation	Meaning
$y \parallel x$	Concatenation of two bitstrings x and y
0^l	All-zero bitstring of length l
$\|x\|$	Length in bits of the bitstring x
$x \oplus y$	XOR of bitstrings x and y
$x_{m-1} \parallel \cdots \parallel x_1 \parallel x_0$	x_0 is the least significant bit (or block), x_{m-1} is the most significant bit(or block).
$\lfloor x \rfloor_l$	Truncation of bitstring x to its first (least significant) l bits
$\lceil x \rceil_l$	Truncation of bitstring x to its last (most significant) l bits
$\{0,1\}^k$	The set of bit strings of length k
$\{0,1\}^*$	The set of bit strings of all lengths
S	A b-bit state of the Sponge/Duplex construction
$S_r,\ S_c$	The r-bit rate and c-bit capacity part of a state S
nr (or nr_0, nr_f, nr_h)	The number of rounds for an underlying permutation
p_b	A round transformation with a width of b bits
$p_b[nr]$	A permutation consisting of nr-round p_b
KNOT-AEAD(k,b,r)	A KNOT AE member with k-bit key, b-bit state and r-bit rate
KNOT-Hash(n,b,r,r')	A KNOT hash member with n-bit hash output, b-bit state, r-bit absorbing rate and r'-bit squeezing rate

2 The KNOT Permutations

The underlying permutations of each KNOT member iteratively apply an SP-network round transformation. KNOT uses 3 different round transformations, which are defined

by the width b (b=256, 384 or 512). Each of the rounds consists of the following 3 steps: $AddRoundConstant_b$, $SubColumn_b$, $ShiftRow_b$. Let p_b denote a round transformation, the following is a pseudo C code for p_b :

{ $AddRoundConstant_b(STATE, RC)$
$SubColumn_b(STATE)$
$ShiftRow_b(STATE)$
}
where RC denotes a round constant.

We use nr (or nr_0, nr_f, nr_h) to denote the number of rounds for an underlying permutation. The concrete values of nr (or nr_0, nr_h) for each KNOT member are given afterwards.

2.1 The State

A b-bit state is pictured as a $4 \times \frac{b}{4}$ rectangular array of bits. Let $W = w_{b-1} \parallel \cdots \parallel w_1 \parallel w_0$ denote a state, the first $\frac{b}{4}$ bits $w_{\frac{b}{4}-1} \parallel \cdots \parallel w_1 \parallel w_0$ are arranged in row 0, the next $\frac{b}{4}$ bits $w_{\frac{b}{2}-1} \parallel \cdots \parallel w_{\frac{b}{4}+1} \parallel w_{\frac{b}{4}}$ are arranged in row 1, and so on, as illustrated in Fig. 1. In the following, for convenience of description, a cipher state is described in a two-dimensional way, as illustrated in Fig. 2.

$$\begin{bmatrix} w_{\frac{b}{4}-1} & \cdots & w_1 & w_0 \\ w_{\frac{b}{2}-1} & \cdots & w_{\frac{b}{4}+1} & w_{\frac{b}{4}} \\ w_{\frac{3b}{4}-1} & \cdots & w_{\frac{b}{2}+1} & w_{\frac{b}{2}} \\ w_{b-1} & \cdots & w_{\frac{3b}{4}+1} & w_{\frac{3b}{4}} \end{bmatrix}$$

Fig. 1. A b-bit State

$$\begin{bmatrix} a_{0,\frac{b}{4}-1} & \cdots & a_{0,1} & a_{0,0} \\ a_{1,\frac{b}{4}-1} & \cdots & a_{1,1} & a_{1,0} \\ a_{2,\frac{b}{4}-1} & \cdots & a_{2,1} & a_{2,0} \\ a_{3,\frac{b}{4}-1} & \cdots & a_{3,1} & a_{3,0} \end{bmatrix}$$

Fig. 2. Two-dimensional Way

2.2 The $AddRoundConstant_b$ Transformation

A simple bitwise XOR of a d-bit round constant to the first d bits of the intermediate state, with $d = 6$, 7 or 8.

KNOT uses 3 different binary linear feedback shift registers (LFSR) to generate 3 different sets of constants. For each LFSR, the initial value is defined as RC[0] := $0x1$. Each set of round constants {RC[i], $0 \leq i \leq 2^d - 1$}) are generated by a d-bit maximal-length LFSR:

1. $d = 6$. At each round, the 6 bits $(rc_5, rc_4, rc_3, rc_2, rc_1, rc_0)$ are left shifted over 1 bit, with the new value to rc_0 being computed as $rc_5 \oplus rc_4$.
2. $d = 7$. At each round, the 7 bits $(rc_6, rc_5, rc_4, rc_3, rc_2, rc_1, rc_0)$ are left shifted over 1 bit, with the new value to rc_0 being computed as $rc_6 \oplus rc_5$.
3. $d = 8$. At each round, the 8 bits $(rc_7, rc_6, rc_5, rc_4, rc_3, rc_2, rc_1, rc_0)$ are left shifted over 1 bit, with the new value to rc_0 being computed as $rc_7 \oplus rc_5 \oplus rc_4 \oplus rc_3$.

For convenience, we use $CONST_d$ to denote the set of constants generated by the d-bit LFSR. For a KNOT member, the choice of d depends on the total number of rounds.

2.3 The $SubColumn_b$ Transformation

Parallel application of S-boxes to the 4 bits in the same column. The operation of $SubColumn_b$ is illustrated in Fig. 3. The input of an S-box is $Col(j) = a_{3,j} \parallel a_{2,j} \parallel a_{1,j} \parallel a_{0,j}$ for $0 \leq j \leq \frac{b}{4} - 1$, and the output is $S(Col(j)) = b_{3,j} \parallel b_{2,j} \parallel b_{1,j} \parallel b_{0,j}$.

The S-box used in KNOT is a 4-bit to 4-bit S-box $S : F_2^4 \to F_2^4$. The action of this S-box in hexadecimal notation is given by the following table.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	4	0	A	7	B	E	1	D	9	F	6	8	5	2	C	3

$$\begin{pmatrix} a_{0,\frac{b}{4}-1} \\ a_{1,\frac{b}{4}-1} \\ a_{2,\frac{b}{4}-1} \\ a_{3,\frac{b}{4}-1} \end{pmatrix} \cdots \begin{pmatrix} a_{0,1} \\ a_{1,1} \\ a_{2,1} \\ a_{3,1} \end{pmatrix} \begin{pmatrix} a_{0,0} \\ a_{1,0} \\ a_{2,0} \\ a_{3,0} \end{pmatrix}$$
$$\downarrow S \quad \cdots \quad \downarrow S \quad \downarrow S$$
$$\begin{pmatrix} b_{0,\frac{b}{4}-1} \\ b_{1,\frac{b}{4}-1} \\ b_{2,\frac{b}{4}-1} \\ b_{3,\frac{b}{4}-1} \end{pmatrix} \cdots \begin{pmatrix} b_{0,1} \\ b_{1,1} \\ b_{2,1} \\ b_{3,1} \end{pmatrix} \begin{pmatrix} b_{0,0} \\ b_{1,0} \\ b_{2,0} \\ b_{3,0} \end{pmatrix}$$

Fig. 3. $SubColumn_b$ **Operates on the Columns of the State**

2.4 The $ShiftRow_b$ Transformation

A left rotation to each row over different offsets. Row 0 is not rotated, row 1 is left rotated over c_1 bit, row 2 is left rotated over c_2 bits, row 3 is left rotated over c_3 bits. The parameters (c_1, c_2, c_3) only depend on b, Table 1 gives the concrete values of (c_1, c_2, c_3) for the 3 different state width b.

Let $\lll x$ denote left rotation over x bits, the operation $ShiftRow_b$ is illustrated in Fig.4.

$$\begin{aligned}
\left(a_{0,\frac{b}{4}-1} \ \cdots \ a_{0,1} \ \ a_{0,0}\right) &\xrightarrow{\lll 0} \left(a_{0,\frac{b}{4}-1} \ \cdots \ a_{0,1} \ \ a_{0,0}\right) \\
\left(a_{1,\frac{b}{4}-1} \ \cdots \ a_{1,1} \ \ a_{1,0}\right) &\xrightarrow{\lll c_1} \left(a_{1,\frac{b}{4}-c_1-1} \ \cdots \ a_{1,\frac{b}{4}-c_1+1} \ \ a_{1,\frac{b}{4}-c_1}\right) \\
\left(a_{2,\frac{b}{4}-1} \ \cdots \ a_{2,1} \ \ a_{2,0}\right) &\xrightarrow{\lll c_2} \left(a_{2,\frac{b}{4}-c_2-1} \ \cdots \ a_{2,\frac{b}{4}-c_2+1} \ \ a_{2,\frac{b}{4}-c_2}\right) \\
\left(a_{3,\frac{b}{4}-1} \ \cdots \ a_{3,1} \ \ a_{3,0}\right) &\xrightarrow{\lll c_3} \left(a_{3,\frac{b}{4}-c_3-1} \ \cdots \ a_{3,\frac{b}{4}-c_3+1} \ \ a_{3,\frac{b}{4}-c_3}\right)
\end{aligned}$$

Fig. 4. $ShiftRow_b$ **Operates on the Rows of the State**

3 A Bit-slice Description of the KNOT Permutations

In the following, we present an equivalent description of $SubColumn_b$ and $ShiftRow_b$ transformations. Based on them, one can easily write a code for a software implementation of KNOT-AEAD and KNOT-Hash, i.e., a bit-slice implementation. Our software implementation of each KNOT member is just based on these results.

Table 1. ShitRow offsets for the 3 state width

b	c_1	c_2	c_3
256	1	8	25
384	1	8	55
512	1	16	25

3.1 The $SubColumn_b$ Transformation

As shown in Fig. 2, a b-bit state is described as a $4 \times \frac{b}{4}$ array. Let $A_{b,i} = a_{i,\frac{b}{4}-1}||\cdots||a_{i,1}||a_{i,0}$ denote the i-th row, $i = 0, 1, 2, 3$. $A_{b,i}$ can be regarded as a $\frac{b}{4}$-bit word.

Let $A_{b,0}, A_{b,1}, A_{b,2}, A_{b,3}$ be 4 inputs of $SubColumn_b$, $B_{b,0}, B_{b,1}, B_{b,2}, B_{b,3}$ be the 4 outputs, where $A_{b,i}$ and $B_{b,i}$ denote the i-th row of the state. Let $T_{b,i}$ denote $\frac{b}{4}$-bit temporary variables, $i = 1, 2, 3, 5, 6, 8, 9, 11$. The $SubColumn_b$ transformation can be computed in the following 12 steps:

1. $T_{b,1} = \sim A_{b,0}$; 2. $T_{b,2} = A_{b,1} \,\&\, T_{b,1}$; 3. $T_{b,3} = A_{b,2} \oplus T_{b,2}$;
4. $B_{b,3} = A_{b,3} \oplus T_{b,3}$; 5. $T_{b,5} = A_{b,1} | A_{b,2}$; 6. $T_{b,6} = A_{b,3} \oplus T_{b,1}$;
7. $B_{b,2} = T_{b,5} \oplus T_{b,6}$; 8. $T_{b,8} = A_{b,1} \oplus A_{b,3}$; 9. $T_{b,9} = T_{b,3} \,\&\, T_{b,6}$;
10. $B_{b,0} = T_{b,8} \oplus T_{b,9}$; 11. $T_{b,11} = B_{b,2} \,\&\, T_{b,8}$; 12. $B_{b,1} = T_{b,3} \oplus T_{b,11}$;

where "$\sim$" denotes NOT, "&" bitwise AND, "$|$" bitwise OR, "$\oplus$" bitwise XOR.

3.2 The $ShiftRow_b$ Transformation

Let $B_{b,0}, B_{b,1}, B_{b,2}, B_{b,3}$ be 4 inputs of $ShiftRow_b$ transformation, $D_{b,0}, D_{b,1}, D_{b,2}, D_{b,3}$ be the 4 outputs. Then:

$D_{b,0} = B_{b,0}$; $D_{b,1} = B_{b,1} \lll c_1$; $D_{b,2} = B_{b,2} \lll c_2$; $D_{b,3} = B_{b,3} \lll c_3$.

where "$B \lll x$" denotes a left rotation over x bits within a $\frac{b}{4}$-bit word B; c_i (i=1,2,3) are the rotation offsets, which are specified in Table 1.

4 KNOT AEAD

An authenticated encryption with associated data (AEAD) algorithm is a function with four inputs and one output. The four inputs are a variable-length plaintext, variable-length associated data, a fixed-length nonce, and a fixed-length key. The output is a variable-length ciphertext.

From a security point of view, an AEAD algorithm should ensure both the confidentiality of the plaintexts (under adaptive chosen-plaintext attacks) and the integrity of the ciphertexts (under adaptive forgery attempts).

4.1 Parameter Sets of KNOT AEAD

The mode of operation of KNOT is based on Duplex mode MonkeyDuplex, which is proposed in [19] and utilized in Ascon [24] and Ketje [13]. Let S denote a b-bit state, S_r and S_c denote the rate and capacity parts of S. For the initialization, the number of rounds is nr_0; for the

Table 2. Parameters for the 4 members of KNOT-AEAD(k, b, r) Family

Name	Bit Size				Constants	Rounds		
	k	b	r	c		nr_0	nr	nr_f
KNOT-AEAD(128, 256, 64)	128	256	64	192	$CONST_6$	52	28	32
KNOT-AEAD(128, 384, 192)	128	384	192	192	$CONST_7$	76	28	32
KNOT-AEAD(192, 384, 96)	192	384	96	288	$CONST_7$	76	40	44
KNOT-AEAD(256, 512, 128)	256	512	128	384	$CONST_7$	100	52	56

where KNOT-AEAD(128, 256, 64) is the primary AEAD member.

processing of the associated data and plaintext blocks, the number of rounds is nr; for the finalization, the number of rounds is nr_f. The concrete values of nr_0, nr and nr_f for each KNOT AEAD member are given in Table 2.

The KNOT AEAD family has 4 members. For each member, the key length, the nonce length and the tag length are all equal to k bits. Let KNOT-AEAD(k, b, r) denote a KNOT-AEAD member with k-bit key, b-bit state and r-bit rate. Table 2 presents the parameter sets of the 4 AEAD members. The followings are some additional remarks:

1. The first KNOT-AEAD (i.e., KNOT-AEAD(128, 256, 64)) is the primary AEAD member. It needs the lowest hardware area and software memory, due to its 256-bit state size.
2. The first KNOT-AEAD has a security strength of 125 bits, the fourth KNOT-AEAD has a security strength of 253 bits. The security of the third KNOT-AEAD is in-between the first and the fourth one (see Section 7.1 for the security bounds).
3. The second KNOT-AEAD has almost the same security level with the first one, but it uses a higher bitrate due to its bigger permutation width, hence it has a higher throughput than the first AEAD.

For a b-bit KNOT permutation $p_b[n_r]$, consider its bit-slice description (see Section 3), let $A_{b,0}, A_{b,1}, A_{b,2}, A_{b,3}$ denote the 4 subblocks of the b-bit input of $p_b[n_r]$, then:

1. For KNOT-AEAD(128, 384, 192), $S_r = A_{b,1} \parallel A_{b,0}$, $S_c = A_{b,3} \parallel A_{b,2}$.
2. For the other 3 KNOT-AEAD members, $S_r = A_{b,0}$, $S_c = A_{b,3} \parallel A_{b,2} \parallel A_{b,1}$.

Similarly, the tag (or hash ouptput) extraction from a b-bit state also starts from $A_{b,0}$.

4.2 Padding

The padding function $pad_r(X)$ returns a bit string obtained by appending a single 1 and the smallest number of 0s to the bit string X such that the length of the padded bit string is a multiple of r bits.

The associated data AD is firstly padded by applying the $pad_r(X)$ function, then the padded associated data is divided into u blocks of r bits: $AD_{u-1} \parallel \cdots \parallel AD_0$. If the length of the associated data is zero, then no padding is applied and no associated data is processed.

$$pad_r(AD) = \begin{cases} 0^{r-1-(|AD| \bmod r)} \parallel 1 \parallel AD = AD_{u-1} \parallel \cdots \parallel AD_0 & \text{if } |AD| > 0, \\ \varnothing & \text{if } |AD| = 0. \end{cases} \quad (1)$$

The same padding process is applied to divide the plaintext P into v blocks of r bits: $P_{v-1} \parallel \cdots \parallel P_0$. Similarly, if the length of the plaintext is zero, no padding is applied and no plaintext is processed.

$$pad_r(P) = \begin{cases} 0^{r-1-(|P| \bmod r)} \parallel 1 \parallel P = P_{v-1} \parallel \cdots \parallel P_0 & \text{if } |P| > 0, \\ \varnothing & \text{if } |P| = 0. \end{cases} \quad (2)$$

An Example - When *AD* and *P* are represented as Byte Arrays When AD and P are represented as byte arrays, we present an example for clarity. The following is a 12-byte associated data:

$$ADarray[12] = \{0X01, 0X02, 0X03, 0X04, 0X05, 0X06, 0X07, 0X08, 0X09, 0X0A, 0X0B, 0X0C\}.$$

where $0X01$ is the 0th byte, and $0X0C$ is the 11th byte. Let $r = 64$, then the padded associated data has 16 bytes (i.e., 2 blocks of 64 bits each):

$$pad_r(ADarray[12]) = \{0X01, 0X02, 0X03, 0X04, 0X05, 0X06, 0X07, 0X08,\\ 0X09, 0X0A, 0X0B, 0X0C, 0X01, 0X00, 0X00, 0X00\}.$$

After the dividing, we have:

$$AD_0 = 0X08 \parallel 0X07 \parallel 0X06 \parallel 0X05 \parallel 0X04 \parallel 0X03 \parallel 0X02 \parallel 0X01,$$
$$AD_1 = 0X00 \parallel 0X00 \parallel 0X00 \parallel 0X01 \parallel 0X0C \parallel 0X0B \parallel 0X0A \parallel 0X09.$$

4.3 Initialization

The authenticated encryption process is initialized by loading the key K and the nonce N (both k bits). The b-bit state is initialized as:

$$S = \begin{cases} (0^{128} \parallel K \parallel N) \oplus (1 \parallel 0^{383}) & \text{for KNOT-AEAD(128,384,192),} \\ K \parallel N & \text{for the other 3 AEAD members.} \end{cases} \tag{3}$$

Then nr_0 rounds of the round transformation $p_b[nr_0]$ are applied to the initial state:

$$S \leftarrow p_b[nr_0](S) \tag{4}$$

4.4 Processing Associated Data

Each padded associated data block A_i $(i = 0, \cdots, u-1)$ is processed as follows. The block A_i is XORed to the first r bits of the internal state S, then the state S is updated by the nr-round permutation $p_b[nr]$:

$$S \leftarrow p_b[nr](S_c \parallel (S_r \oplus A_i)), \quad 0 \le i \le u-1. \tag{5}$$

After the last associated data block has been processed or if $|A| = 0$, a single-bit domain separation constant is XORed to the internal state S:

$$S \leftarrow S \oplus (1 \parallel 0^{b-1}) \tag{6}$$

4.5 Encryption

Each padded plaintext block P_i $(i = 0, \cdots, v-1)$ is processed as follows. The ciphertext block C_i is equal to the XOR of the plaintext block P_i with the first r bits of the internal state S. For each block except the last one, the state S is updated by the nr-round permutation $p_b[nr]$:

$$C_i \leftarrow S_r \oplus P_i \tag{7}$$

$$S \leftarrow \begin{cases} p_b[nr](S_c \parallel C_i), & \text{if } 0 \le i < v-1, \\ S_c \parallel C_i, & \text{if } i = v-1. \end{cases} \tag{8}$$

For the last ciphertext C_{v-1} with a length of $0 \le l < r$ bits, $l = |P| \mod r$:

$$C_{v-1} \leftarrow \lfloor C_{v-1} \rfloor_l \tag{9}$$

Then, the ciphertext $C = C_{v-1} \parallel \cdots \parallel C_0$.

4.6 Decryption

For each ciphertext block except the last one, the plaintext block P_i is computed by XORing the ciphertext block C_i with the first r bits S_r of the internal state. Then, S_r is replaced by C_i. For each ciphertext block except the last one, the internal state is updated by nr rounds of the permutation $p_b[nr]$:

$$P_i \leftarrow S_r \oplus C_i \tag{10}$$

$$S \leftarrow p_b[nr](S_c \parallel C_i) \quad , 1 \leq i < v-1. \tag{11}$$

For the last ciphertext block C_{v-1} with a length of $0 \leq l < r$ bits:

$$P_{v-1} \leftarrow \lfloor S_r \rfloor_l \oplus C_{v-1} \tag{12}$$

$$S \leftarrow S_c \parallel (\lceil S_r \rceil_{r-l} \oplus (0^{r-1-l} \parallel 1)) \parallel C_{v-1} \tag{13}$$

4.7 Finalization

In the finalization, the state is firstly updated by nr_f rounds of the permutation $p_b[nr_f]$. Then the tag consists the first k bits of the state.

$$S \leftarrow p_b[nr_f](S) \tag{14}$$

$$T \leftarrow \lfloor S \rfloor_k \tag{15}$$

The encryption process returns the concatenation of the ciphertext blocks and the tag as its output. To avoid any ambiguity, note that we have used two variables C and T to denote the ciphertext and the tag respectively. However, the encryption process returns only one output by appending the tag T to the ciphertext C:

$$\text{Output: } T \parallel C$$

We emphasize that the decryption process shall not return the plaintext if the verification fails, that is to say, the decryption process returns the plaintext only if the calculated tag value matches the received tag value.

4.8 Procedures for KNOT-AEAD

Algorithm 1 specifies the procedures of authenticated encryption and decryption-verification for KNOT-AEAD.

5 KNOT Hash

A hash function is a function with one input and one output. The input is a variable-length message, and the output is a fixed-length hash value. It should be computationally infeasible to find a collision or a (second) preimage for a hash function. A hash function should also be resistant against length extension attacks.

Guo *et al.* extended the Sponge construction [26] by allowing a different squeezing bit rate r' to offer a tradeoff between efficiency and preimage security. Increasing r' will directly reduce the time spent in the squeezing process, but might reduce the preimage security. On the other hand, decreasing r' will increase the time spent in the squeezing process, but might improve the preimage security. The work from Andreeva *et al.* also independently proposed a similar extension [3]. In our design of each KNOT-Hash member, a different squeezing bit rate $r' > r$ is used to improve the squeezing efficiency at the cost of a reduction for preimage security.

Algorithm 1 Procedures of Authenticated Encryption and Decryption-Verification for KNOT-AEAD(k, b, r)

Authenticated Encryption	Decryption-Verification
INPUT: key $K \in \{0,1\}^k$, nonce $N \in \{0,1\}^k$, associated data $AD \in \{0,1\}^*$, plaintext $P \in \{0,1\}^*$. **OUTPUT:** $T \parallel C$, (with ciphertext $C \in \{0,1\}^*$ and tag $T \in \{0,1\}^k$).	**INPUT:** key $K \in \{0,1\}^k$, nonce $N \in \{0,1\}^k$, associated data $AD \in \{0,1\}^*$, $T \parallel C$, (with ciphertext $C \in \{0,1\}^*$ and tag $T \in \{0,1\}^k$). **OUTPUT:** plaintext $P \in \{0,1\}^*$ or $\perp$.
Padding: $AD_{u-1} \parallel \cdots \parallel AD_0 \leftarrow pad_r(AD)$ $P_{v-1} \parallel \cdots \parallel P_0 \leftarrow pad_r(P)$	**Padding**: $AD_{u-1} \parallel \cdots \parallel AD_0 \leftarrow pad_r(AD)$
Initialization: **if** $b = 384$ and $r = 192$, **then** $S = (K \parallel N) \oplus (1 \parallel 0^{383})$ **else** $S = K \parallel N$ $S \leftarrow p_b[nr_0](S)$	**Initialization**: **if** $b = 384$ and $r = 192$, **then** $S = (K \parallel N) \oplus (1 \parallel 0^{383})$ **else** $S = K \parallel N$ $S \leftarrow p_b[nr_0](S)$
Processing Associated Data: $c = b - r$ for $i = 0, \cdots, u-1$, do $S \leftarrow p_b[nr](S_c \parallel (S_r \oplus A_i))$ $S \leftarrow S \oplus (1 \parallel 0^{b-1})$	**Processing Associated Data**: $c = b - r$ for $i = 0, \cdots, u-1$, do $S \leftarrow p_b[nr](S_c \parallel (S_r \oplus A_i))$ $S \leftarrow S \oplus (1 \parallel 0^{b-1})$
Encryption: for $i = 0, \cdots, v-2$, do $\{C_i \leftarrow S_r \oplus P_i$ $S \leftarrow p_b[nr](S_c \parallel C_i)$ $\}$ $S_r \leftarrow S_r \oplus P_{v-1}$ $l = \lvert P \rvert \mod r$ $C_{v-1} \leftarrow \lfloor S_r \rfloor_l$	**Decryption**: for $i = 0, \cdots, v-2$, do $\{P_i \leftarrow S_r \oplus C_i$ $S \leftarrow p_b[nr](S_c \parallel C_i)$ $\}$ $l = \lvert C \rvert \mod r$ $P_{v-1} \leftarrow \lfloor S_r \rfloor_l \oplus C_{v-1}$ $S_r \leftarrow (\lceil S_r \rceil_{r-l} \oplus (0^{r-1-l} \parallel 1)) \parallel C_{v-1}$
Finalization: $S \leftarrow p_b[nr_f](S)$ $T \leftarrow \lfloor S \rfloor_k$ **return** $T \parallel C_{v-1} \parallel \cdots \parallel C_0$	**Finalization**: $S \leftarrow p_b[nr_f](S)$ $T' \leftarrow \lfloor S \rfloor_k$ **if** $T = T'$, then **return** $P_{v-1} \parallel \cdots \parallel P_0$ **else return** $\perp$

Table 3. Parameters for the 4 members of KNOT-Hash(n, b, r, r') Family

Name	Bit Size					Constants	Rounds
	n	b	c	r	r'		nr_h
KNOT-Hash(256,256,32,128)	256	256	224	32	128	$CONST_7$	68
KNOT-Hash(256,384,128,128)	256	384	256	128	128	$CONST_7$	80
KNOT-Hash(384,384,48,192)	384	384	336	48	192	$CONST_7$	104
KNOT-Hash(512,512,64,256)	512	512	448	64	256	$CONST_8$	140

where KNOT-Hash(256, 256, 32, 128) is the primary hash member.

5.1 Parameter Sets of KNOT Hash

The KNOT hash family has 4 members. Let KNOT-Hash(n, b, r, r') denote a KNOT hash member with n-bit hash output, b-bit state, r-bit absorbing rate and r'-bit squeezing rate. Table 3 presents the parameter sets of the 4 members of the KNOT hash family, KNOT-Hash(256, 256, 32, 128) is the primary hash member.

Like the cases in the KNOT-AEAD members, similarly, the first KNOT-Hash needs the lowest hardware area and software memory, and has a security strength of 112 bits. The fourth KNOT-Hash has a security level which is around 224 bits, the security of the third KNOT-Hash is in-between the first and the fourth one. The second KNOT-Hash has almost the same security level with the first one, but has a higher throughput.

5.2 Padding

The input message M (including the empty string) is padded by applying the $pad_r(X)$ function, then the padded message is divided into v blocks of r bits: $M_{v-1} \parallel \cdots \parallel M_0$.

$$pad_r(M) = 0^{r-1-(|M| \bmod r)} \parallel 1 \parallel M = M_{v-1} \parallel \cdots \parallel M_0. \tag{16}$$

5.3 Initialization

The b-bit state is initialized as:

$$S = \begin{cases} 1 \parallel 0^{383} & \text{for KNOT-Hash(256, 384, 128, 128),} \\ 0^b & \text{for the other 3 hash members.} \end{cases} \tag{17}$$

5.4 Absorbing

Each padded message block M_i $(i = 0, \cdots, v-1)$ is processed as follows. The block M_i is XORed to the first r bits of the internal state S, then the state S is updated by the nr_h rounds of the round transformation $p_b[nr_h]$:

$$S \leftarrow p_b[nr_h](S_c \parallel (S_r \oplus M_i)), \quad 0 \le i \le v-1. \tag{18}$$

5.5 Squeezing

After the last message block has been processed, the n-bit output is extracted from the $S_{r'}$ part of the state at a time, until a total of $\frac{n}{r'}$ extractions are completed.

for $i = 0$ to $\frac{n}{r'} - 2$, do
$\{H_i \leftarrow S_{r'}$
$S \leftarrow p_b[nr_h](S)$
$\}$
$H_{\frac{n}{r'}-1} \leftarrow S_{r'}$
$H \leftarrow H_{\frac{n}{r'}-1} \parallel \cdots \parallel H_0$

5.6 Procedures for KNOT-Hash

Algorithm 2 Procedures for KNOT-Hash(n, b, r, r')

INPUT:
message $M \in \{0,1\}^*$.
OUTPUT:
hash value $H \in \{0,1\}^n$.

Padding:
$M_{v-1} \parallel \cdots \parallel M_0 \leftarrow pad_r(M)$
Initialization:
if $b = 384$ and $r = 192$, **then**
$S \leftarrow 1 \parallel 0^{383}$
else $S \leftarrow 0^b$
Absorbing:
$c = b - r$
for $i = 0, \cdots, v-1$, do
$S \leftarrow p_b[nr_h](S_c \parallel (S_r \oplus M_i))$
Squeezing:
for $i = 0, \cdots, \frac{n}{r'} - 2$, do
$\{H_i \leftarrow S_{r'}$
$S \leftarrow p_b[nr_h](S)\}$
$H_{\frac{n}{r'}-1} \leftarrow S_{r'}$
return $H \leftarrow H_{\frac{n}{r'}-1} \parallel \cdots \parallel H_0$

Algorithm 2 specifies the procedures for KNOT-Hash.

6 Pairs of KNOT-AEAD and KNOT-Hash Members

We group KNOT-AEAD and KNOT-Hash members into pairs according to their parameters. As listed in Table 4, there are four KNOT-Pairs, where KNOT-Pair I is the primary pair. For each KNOT-Pair, the corresponding KNOT-AEAD and KNOT-Hash use identical round transformations p_b (ignoring the constant addition) in the underlying permutations, which leads to a unified hardware implementation and a reduced ROM utilization in implementing the software.

Table 4. List of Pairs of KNOT-AEAD and KNOT-Hash Members

	AEAD Member	Hash Member
KNOT-Pair I	**KNOT-AEAD(128, 256, 64)**	**KNOT-Hash(256, 256, 32, 128)**
KNOT-Pair II	KNOT-AEAD(128, 384, 192)	KNOT-Hash(256, 384, 128, 128)
KNOT-Pair III	KNOT-AEAD(192, 384, 96)	KNOT-Hash(384, 384, 48, 192)
KNOT-Pair IV	KNOT-AEAD(256, 512, 128)	KNOT-Hash(512, 512, 64, 256)

where KNOT-Pair I is the primary pair.

7 Security Model

7.1 Security of AEAD Modes

Let Π be an authenticated encryption scheme, P a list of idealized permutations (which Π may depend on), and \$ an ideal version of $\mathcal{E}_K$. Define the advantage of an adversary $\mathcal{A}$ in breaking the privacy of Π as follows:

$$Adv_{\Pi}^{\text{priv}}(\mathcal{A}) = |Pr_{p,K}(\mathcal{A}^{p^{\pm},\mathcal{E}_K} = 1) - Pr_{p,\$}(\mathcal{A}^{p^{\pm},\$} = 1)|$$

where the probabilities are taken over the random choices of $p, \$, K$, and $\mathcal{A}$, if any. The fact that the adversary has access to both the forward and inverse permutations in p is denoted by $p^{\pm}$. The adversary $\mathcal{A}$ is nonce-respecting, which means that it never makes two queries to $\mathcal{E}_K$ or \$ with the same nonce. Let $Adv_{\Pi}^{priv}(q_p, q_{\mathcal{E}}, \lambda_{\mathcal{E}})$ denote the maximum advantage taken over all adversaries that query $p^{\pm}$ at most q_p times, and that make at most $q_{\mathcal{E}}$ queries of total length (over all queries) at most $\lambda_{\mathcal{E}}$ blocks to $\mathcal{E}_K$ or \$. This privacy notion is also known as the indistinguishability under chosen plaintext attack (IND-CPA) security of an (authenticated) encryption scheme.

As above, let P denote the list of underlying idealized permutations of Π. Define the advantage of an adversary $\mathcal{A}$ in breaking the integrity of Π as follows:

$$Adv_{\Pi}^{\text{auth}}(\mathcal{A}) = Pr_{p,K}(\mathcal{A}^{p^{\pm},\mathcal{E}_K,\mathcal{D}_K} \text{ forges})$$

where the probability is taken over the random choices of $p, \$, K$, and $\mathcal{A}$, if any. If $\mathcal{D}_K$ ever returns a message other than $\perp$ on input of $(N; H, C, T; A)$ where (C, A) has never been output by $\mathcal{E}_K$ on input of a query $(N; H, M; T)$ for some M, then we say that adversary $\mathcal{A}$ forges. Adversary $\mathcal{A}$ is nonce-respecting, which means that it never makes two queries to $\mathcal{E}_K$ with the same nonce. Nevertheless, $\mathcal{A}$ is allowed to repeat nonces in decryption queries. Let $Adv_{\Pi}^{\text{auth}}(q_p, q_{\mathcal{E}}, \lambda_{\mathcal{E}}, q_{\mathcal{D}}, \lambda_{\mathcal{D}})$ denote the maximum advantage taken over all adversaries that query $p^{\pm}$ at most q_p times, and that make at most $q_{\mathcal{E}}$ queries of total length (over all queries) at most $\lambda_{\mathcal{E}}$ blocks to $\mathcal{E}_K$, and at most $q_{\mathcal{D}}$ queries of total length at most $\lambda_{\mathcal{D}}$ blocks to $\mathcal{D}_K$.

The constant ρ is defined by r and c as follows:

$$\rho = \begin{cases} 3.4 \times 2^{\frac{c-r}{2}} & \text{if} \quad \frac{c}{5} < r \leq c - 2log_2 c, \\ \dfrac{1.4r}{log_2 r + r - c - 2} & \text{if} \quad c \leq r \leq c + elog_2 c - e\beta \end{cases} \tag{19}$$

where $\beta = log_2 e + log_2 log_2 e$.

Theorem 1 ([35]). *Let $\Pi = (\mathcal{E}, \mathcal{D})$ be KNOT-AEAD based on an ideal underlying primitive p. Then,*

$$Adv_{\Pi}^{priv}(q_p, q_{\mathcal{E}}, \lambda_{\mathcal{E}}) \leq \frac{3(q_p + \sigma_{\mathcal{E}})^2}{2^{b+1}} + \frac{\sigma_{\mathcal{E}}}{\min\{2^{b/2}, 2^c\}} + \frac{2\rho q_p}{2^c} + \frac{q_p + \sigma_{\mathcal{E}}}{2^k}$$

where $\sigma_{\mathcal{E}}$ denotes the number of primitive evaluations via the encryption queries.

Theorem 2 ([35]). *Let $\Pi = (\mathcal{E}, \mathcal{D})$ be KNOT-AEAD based on an ideal underlying primitive p. Then,*

$$Adv_{\Pi}^{auth}(q_p, q_{\mathcal{E}}, \lambda_{\mathcal{E}}, q_{\mathcal{D}}, \lambda_{\mathcal{D}}) \leq \frac{(q_p + \sigma_{\mathcal{E}} + \sigma_{\mathcal{D}})^2}{2^b} + \frac{\sigma_{\mathcal{E}}}{\min\{2^{b/2}, 2^c\}} + \frac{2\rho q_p}{2^c} + \frac{q_p + \sigma_{\mathcal{E}} + \sigma_{\mathcal{D}}}{2^k} + \frac{(q_p + \sigma_{\mathcal{E}} + \sigma_{\mathcal{D}})\sigma_{\mathcal{D}}}{2^c} + \frac{q_{\mathcal{D}}}{2^\tau}$$

where $\sigma_{\mathcal{E}}$ ($\sigma_{\mathcal{D}}$) denotes the number of primitive evaluations via the encryption (decryption) queries.

7.2 Security of Hash Modes

For an extended Sponge hash function with a different squeezing bitrate r', the following theorem presents its security against generic attacks.

Theorem 3 ([10, 26]). *For a Sponge hash function with hash size n, absorbing bitrate r, capacity c and squeezing bitrate r', the best known generic attacks require the following amount of computations when the underlying permutation is modeled as a random permutation:*

1. ***Preimage attacks:*** $\min\{2^{\min\{n,b\}}, \max\{2^{\min\{n,b\}-r'}, 2^{c/2}\}\}$
2. ***Second-preimage attacks:*** $\min\{2^n, 2^{c/2}\}$
3. ***Collision attack:*** $\min\{2^{n/2}, 2^{c/2}\}$

8 Expected Security Strength of Each KNOT-AEAD and KNOT-Hash Members

8.1 KNOT-AEAD

The KNOT-AEAD algorithms maintain security as long as the nonce is unique (not repeated under the same key), Table 5 presents the security goals for the 4 KNOT-AEAD members.

The AEAD algorithm lose the plaintext confidentiality if the same nonce N is used to encrypt two different plaintexts under the same key K. In this case, the attacker can

Table 5. Claimed Security Strength and Data Limit for the 4 KNOT-AEAD Members

	Plaintext Confidentiality	Ciphertext Integrity	Data Limit
KNOT-AEAD(128, 256, 64)	125	125	2^{64}
KNOT-AEAD(128, 384, 192)	128	128	2^{64}
KNOT-AEAD(192, 384, 96)	189	189	2^{96}
KNOT-AEAD(256, 512, 128)	253	253	2^{128}

Note: The security strength is indicated by the logarithm base 2 of the attack cost, and the unit is the underlying KNOT permutation $P_b[nr]$ used by the KNOT-AEAD member.

detect the first differing block of the two plaintexts and calculate the XOR difference of the corresponding plaintext block.

Plaintexts can not be returned by the decryption-verification process if the ciphertext is invalid. Release of unverified decrypted ciphertexts also has an impact on the plaintext confidentiality as it allows an attacker to collect data which may be useful at a later attack.

Note that nonce violation and release of unverified decrypted ciphertext have no influences on integrity.

For each KNOT-AEAD member, Table 5 also presents the upper limit M on the data complexity, i.e., the number of processed plaintext and associated data blocks is limited to M blocks under one key. For example, KNOT-AEAD(128, 256, 64) has a message length limit of 2^{67} bytes, and KNOT-AEAD(192, 384, 96) has a limit of 3×2^{66} bytes.

The security strength and the message length limit for each KNOT-AEAD member are mainly determined by Theorem 1 and Theorem 2 in Section 7.1. According to Theorem 1, a Duplex-based AEAD algorithm can offer confidentiality as long as the total complexity $q_p + \sigma_{\mathcal{E}}$ does not exceed $\min\{2^{b/2}, 2^k\}$ and the total number of primitive queries q_p does not exceed $2^c/\rho$, where ρ denotes a constant. According to Theorem 2, a Duplex-based AEAD algorithm can offer integrity as long as the total complexity $q_p + \sigma_{\mathcal{E}} + \sigma_{\mathcal{D}}$ does not exceed $2^c/\sigma_{\mathcal{D}}$, where $\sigma_{\mathcal{E}}$ ($\sigma_{\mathcal{D}}$) denotes the number of primitive evaluations via the encryption (decryption) queries.

For each KNOT-AEAD member, the underlying permutations used in the processing of associated data/plaintexts and the finalization are not ideal, but we emphasize that the number of rounds is sufficient for the claimed security strength. Non-random properties for these permutations are known, but it do not violate the claimed security.

8.2 KNOT-Hash

Table 6 presents the security of each KNOT-Hash member against the 3 generic attacks, which is based on Theorem 3 in Section 7.2. The message length limit is simply in accordance with the corresponding KNOT-AEAD member. The AEAD and the corresponding hash (with close security strength and the same round transformation) can be put together. Note that the message length limit on each KNOT-Hash member does not violate the claimed security listed in Table 6.

Length extension means that an attacker tries to predict the value of $H(X \parallel M)$ for some string X, given the hash value $H(M)$ for an unknown input M [10]. For a Sponge function, length extension is successful if one can recover the inner state at the end of the squeezing of M, which comes down to state recovery. If a hash function is secure against state recovery attacks, then it can resist length extension attacks.

Table 6. Claimed Security Strength and Data Limit for the 4 KNOT-Hash Members

Name	Security (bit)			Data limit
	pre.	2nd pre.	col.	
KNOT-Hash(256, 256, 32, 128)	128	112	112	2^{64}
KNOT-Hash(256, 384, 128, 128)	128	128	128	2^{64}
KNOT-Hash(384, 384, 48, 192)	192	168	168	2^{96}
KNOT-Hash(512, 512, 64, 256)	256	224	224	2^{128}

Table 7. Difference Distribution Table of the KNOT S-box

ΔI \ ΔO	0	1 2 4 8	3 5 6 9 A C	7 B D E	F
0	16	· · · ·	· · · · · ·	· · · ·	·
1	·	· · 2 ·	· 2 2 · · 2	2 · 2 2	2
2	·	2 · · ·	2 · · 2 2 ·	4 · · 2	2
4	·	· · · ·	· · · · 4 2	· 4 2 2	2
8	·	· · · ·	· · · · · 4	· · 4 4	4
3	·	2 · · ·	2 · 4 2 2 ·	· · · 2	2
5	·	· · 2 ·	· 2 2 · 4 ·	2 4 · ·	·
6	·	2 · · ·	2 4 · 2 2 2	· · 2 ·	·
9	·	4 4 · ·	· · · 4 · ·	· 4 · ·	·
A	·	· 2 2 4	2 · · · · ·	2 · 2 ·	2
C	·	4 4 2 ·	· 2 2 · · ·	2 · · ·	·
7	·	2 · 4 ·	2 · · 2 2 2	· · 2 ·	·
B	·	· 2 · 4	2 2 2 · · 2	· · · 2	·
D	·	· · 2 ·	· 2 2 4 · ·	2 4 · ·	·
E	·	· 2 · 4	2 2 2 · · ·	· · 2 ·	2
F	·	· 2 2 4	2 · · · · 2	2 · · 2	·

where ΔI is the input difference, ΔO is the output difference.

9 Security Analysis

In this section, we present the results of our security analysis of the KNOT permutations. The round constant addition is ignored, which has no impact on the results. For a fixed permutation width b and a fixed number of rounds, we regard the permutations parameterized by different round constants as the same permutation.

9.1 Properties of the KNOT S-box

Let x_0, x_1, x_2, x_3 and y_0, y_1, y_2, y_3 respectively denote the input and output of the KNOT S-box, where x_0 is the least significant bit. The following is the algebraic normal form (ANF) of the S-box:

$$
\begin{aligned}
y_0 &= x_0x_1 + x_2 + x_0x_2 + x_3 + x_1x_3 + x_0x_1x_3 + x_2x_3 \\
y_1 &= x_1 + x_2 + x_0x_3 + x_2x_3 + x_1x_2x_3 \\
y_2 &= 1 + x_0 + x_1 + x_2 + x_1x_2 + x_3 \\
y_3 &= x_1 + x_0x_1 + x_2 + x_3
\end{aligned}
$$

The algebraic degree of the KNOT S-box is 2. In Table 7 and Table 8, we present the difference distribution table and linear distribution table of the KNOT S-box respectively.

9.2 Differential Cryptanalysis

Differential [12] and linear [31] cryptanalysis are among the most powerful techniques available for block ciphers. A block cipher is a set of 2^k permutations operating on b-bit vectors, where k is the key length and b the block length. To distinguish a b-bit KNOT permutation from a random permutation using differential cryptanalysis (DC), there must be a predictable difference propagation with a probability significantly larger than 2^{1-b}. A difference propagation is composed of a set of differential trails, where its probability is the

Table 8. Linear Distribution Table of the KNOT S-box

ΓI \ ΓO	0	1 2 4 8	3 5 6 9 A C	7 B D E	F
0	8	· · · ·	· · · · · ·	· · · ·	·
1	·	· 2 · ·	2 4 2 · 2 4	2 2 · 2	2
2	·	· · · ·	· · · 4 4 ·	· · 4 4	·
4	·	2 · · ·	2 2 4 2 · 4	2 2 2 ·	2
8	·	· · · ·	· · · · 4 ·	· 4 · 4	4
3	·	4 2 · ·	2 · 2 · 2 4	2 2 · 2	2
5	·	2 2 · ·	4 2 2 2 2 ·	· 4 2 2	·
6	·	2 4 · ·	2 2 · 2 · 4	2 2 2 ·	2
9	·	· 2 4 ·	2 · 2 · 2 ·	2 2 4 2	2
A	·	4 · · ·	4 4 · · · ·	4 · · ·	·
C	·	2 · · 4	2 2 4 2 · ·	2 2 2 ·	2
7	·	2 2 · ·	· 2 2 2 2 ·	4 · 2 2	4
B	·	· 2 4 ·	2 · 2 4 2 ·	2 2 · 2	2
D	·	2 2 4 4	· 2 2 2 2 ·	· · 2 2	·
E	·	2 4 · 4	2 2 · 2 · ·	2 2 2 ·	2
F	·	2 2 4 4	· 2 2 2 2 ·	· · 2 2	·

where ΓI is the input selection pattern, ΓO is the output selection pattern.

sum of the probabilities of all differential trails that have the specified input difference and output difference [22]. The weight of a differential trail (or a difference propagation) is the negative of the binary logarithm of its probability.

M.Matsui has presented a search algorithm for the best differential/linear trail of DES in [32], which uses branch-and-bound methods. Based on the improved search algorithm [6], we have written a program to search for the best differential trails for each of the 3 KNOT permutations. Let $W^D_{b,i}$ denote the weight of the best i-round differential trail ($i = 1, 2, 3, \dots$) for the b-bit KNOT permutation. The followings are our experimental results:

1. $W^D_{256,i} = W^D_{384,i} = W^D_{512,i}$ for $1 \le i \le 11$.
2. $W^D_{256,i} = W^D_{256,i-3} + 16$ for $12 \le i \le 49$; $W^D_{384,i} = W^D_{384,i-3} + 16$ for $12 \le i \le 73$; $W^D_{512,i} = W^D_{512,i-3} + 16$ for $12 \le i \le 97$.

Our search results from 1 round to 14 rounds are presented in Table 9. Using the above results, one can easily calculate the weight of the best differential trail for a given number of rounds of a KNOT permutation. Especially, we have:

1. For the 256-bit KNOT permutation, the probability of the best 49-round differential trail is 2^{-258};
2. For the 384-bit KNOT permutation, the probability of the best 73-round differential trail is 2^{-386};
3. For the 512-bit KNOT permutation, the probability of the best 97-round differential trail is 2^{-514}.

For RECTANGLE, we have investigated the clustering of differential trails. The probability of the best 15-round differential trail is 2^{-66}. Based on the branch-and-bound algorithm, we have searched for all the differential trails of 15-round RECTANGLE with probability between 2^{-66} and 2^{-76} (up to a rotation equivalence) and examined all the difference propagations made up of the investigated trails. The followings are the experimental results [41]:

Table 9. Probabilities of the Best Differential Trails of the 3 KNOT permutations

♯ R	Prob.	♯ R	Prob.	♯ R	Prob.
1	2^{-2}	6	2^{-18}	11	2^{-55}
2	2^{-4}	7	2^{-25}	12	2^{-60}
3	2^{-7}	8	2^{-32}	13	2^{-66}
4	2^{-10}	9	2^{-40}	14	2^{-71}
5	2^{-14}	10	2^{-49}	⋮	⋮

where ♯ R denotes the number of rounds, and Prob. denotes the probability of the best differential trail.

1. There are 32 best difference propagations with probability $1300 \times 2^{-76} \approx 2^{-65.66}$ each. Each is composed of 7 differential trails.
2. Among all the difference propagations, the maximum number of trails of a difference propagation is 131, i.e., a difference propagation is composed of at most 131 different differential trails.

From the above results, it can be seen that the clustering of differential trails of RECTANGLE is very limited, which can not be used to construct an effective difference propagation with more than 14 rounds.

For the KNOT permutations, due to their bigger widths, it seems very time-consuming (or unpractical) to investigate the clustering of the differential trails. However, due to the great similarity between the KNOT permutations and RECTANGLE, it can be deduced that the clustering of differential trails is also limited for the KNOT permutations.

9.3 Linear Cryptanalysis

Assume a linear trail hold with probability p , define the bias ϵ as $(p - \frac{1}{2})$, the correlation contribution Cor as 2ϵ. To distinguish a b-bit KNOT permutation from a random permutation using linear cryptanalysis (LC), there must be a predictable linear propagation with an amplitude (i.e., $|Cor|$) significantly larger than $2^{-\frac{n}{2}}$. A linear propagation is composed of a set of linear trails, where its amplitude is the sum of the correlation contributions of all linear trails that have the specified input and output selection patterns [22]. The correlation contributions of the linear trails are signed and their sign depends on the value of the round keys. The weight of a linear trail/propagation is the negative of the binary logarithm of its amplitude.

Since the strong round key dependence of interference makes locating the input and output selection patterns for which high correlations occur practically infeasible [22], we have to use the following theorem for an estimation.

Theorem 4 ([22]). *The square of a correlation (or correlation contribution) is called correlation potential. The average correlation potential between an input and an output selection pattern is the sum of the correlation potentials of all linear trails between the input and output selection patterns:*

$$E(C_t^2) = \sum_i (C_i)^2$$

where C_t is the overall correlation, and C_i the correlation coefficient of a linear trail.

Table 10. Correlation Amplitude of the Best Linear Trails of the 3 KNOT Permutations

♯ R	$\|Cor\|$	♯ R	$\|Cor\|$	♯ R	$\|Cor\|$
1	2^{-1}	6	2^{-10}	11	2^{-26}
2	2^{-2}	7	2^{-13}	12	2^{-29}
3	2^{-4}	8	2^{-17}	13	2^{-32}
4	2^{-6}	9	2^{-20}	14	2^{-35}
5	2^{-8}	10	2^{-23}	⋮	⋮

We have searched for the best linear trails for each of the 3 KNOT permutations. Let $W^L_{b,i}$ denote the weight of the best i-round linear trail ($i = 1, 2, 3, \dots$) for the b-bit KNOT permutation. The followings are our experimental results:

1. $W^L_{256,i} = W^L_{384,i} = W^L_{512,i}$ for $1 \leq i \leq 8$.
2. $W^L_{256,i} = W^L_{256,i-1} + 3$ for $9 \leq i \leq 40$; $W^L_{384,i} = W^L_{384,i-1} + 3$ for $9 \leq i \leq 40$; $W^L_{512,i} = W^L_{512,i-1} + 3$ for $9 \leq i \leq 40$.

Our search results from 1 round to 14 rounds are presented in Table 10. Using the above results, one can easily calculate the weight of the best linear trail for a given number of rounds of a KNOT permutation. Especially, we have:

1. For the 256-bit KNOT permutation, the probability of the best 40-round linear trail is 2^{-113}, and it can be deduced that the probability of the best 49-round linear trail is 2^{-140};
2. For the 384-bit KNOT permutation, the probability of the best 40-round linear trail is 2^{-113}, and it can be deduced that the probability of the best 73-round linear trail is 2^{-212};
3. For the 512-bit KNOT permutation, the probability of the best 40-round linear trail is 2^{-113}, it can be deduced that the probability of the best 97-round linear trail is 2^{-284}.

For RECTANGLE, we have investigated the clustering of linear trails. The correlation potential of the best 15-round linear trail is 2^{-74}. Also based on the branch-and-bound algorithm, we have searched for all the linear trails of 15-round RECTANGLE with correlation potential between 2^{-74} and 2^{-80} (up to a rotation equivalence) and examined all the linear propagations made up of the investigated trails. The followings are the experimental results [41]:

1. There are 128 best linear propagations with an average correlation potential $1860 \times 2^{-80} \approx 2^{-69.14}$ each, which is lower than 2^{-64}. Each is composed of 891 linear trails.
2. Among all the linear propagations, the maximum number of trails of a linear propagation is 891. Actually, the best linear propagations have the maximum number of trails.

From the above results, it can be seen that the clustering of linear trails of RECTANGLE is limited, which can not be used to construct an effective linear propagation with more than 14 rounds.

For the KNOT permutations, it also seems very time-consuming (or unpractical) to investigate the clustering of the linear trails. Due to the great similarity between the KNOT permutations and RECTANGLE, it can also be deduced that clustering of linear trails is limited for the KNOT permutations.

9.4 Integral and Division Cryptanalysis

Integral cryptanalysis (or square attack) [20, 27] considers the propagation of sums of a set of carefully chosen data. Division property [36] is a new extension of integral cryptanalysis. In this subsection, we adopt the MILP-based search strategy presented in [38] to analyze the underlying permutations of KNOT.

We have found 17-, 17- and 19-round integral distinguishers for the KNOT permutations with state size b =256, 384 and 512 bits respectively. All of the three distinguishers take a constant value in the single bit at the bottom left corner of the state and take all possibilities in the other $b-1$ bits. This set of input values will result in balancedness at the bottom left corner bit of the state for all the three distinguishers. The program ran for several days, and could not return useful information for 18, 18 and 20 rounds with state size 256, 394 and 512 bits respectively. However, according to the known results of division cryptanalysis against other ciphers, we believe that the longest division-based integral distinguishers for the 3 state sizes are around 17, 17, and 19 rounds (at most 1-3 rounds are added) respectively.

9.5 Impossible Differential Cryptanalysis

Impossible differential cryptanalysis [9] exploits differential trails with probability 0. Impossible differential distinguishers are usually constructed by meet-in-the-middle approach, that is to say, one differential trail with probability one along the forward direction and one differential trail with probability one along the backward direction, whose conditions cannot be met in the middle.

For a b-bit permutation, let $numfd_b$ denote the minimal number of rounds for full dependency. According to the state of art of impossible cryptanalysis, the longest impossible differential distinguisher for a b-bit permutation is around $2numfd_b$ rounds. For the 256-, 384- and 512-bit KNOT permutation, $numfd_b$ is 8, 9 and 10 respectively. Take the KNOT S-box into account, we estimate that the longest impossible differential distinguisher is around 17, 19 and 21 rounds for the 256-, 384- and 512-bit KNOT permutation respectively.

9.6 Algebraic Attacks

The KNOT S-box does not exhibit any special algebraic structure. Furthermore, it seems that successful applications of algebraic attacks [18] on block ciphers/permutations can only reach a very limited number of rounds [34, 29]. Therefore, we do not expect that algebraic attacks form a danger for any KNOT member.

9.7 Summary

Table 11 summarizes the maximal lengths of distinguishers that have been found or estimated for various cryptanalytic methods. According to the results in this section, we can estimate the security of KNOT-AE and KNOT-Hash against various attacks. In Section 10.6, we will justify the choices of number of rounds for the KNOT permutations based on our security evaluations against known attacks.

10 Design Rationale

In this section, we justify the choices we took during the design of KNOT-AEAD and KNOT-Hash.

Table 11. Distinguisher Length of KNOT Permutations for Various Attacks

	Distinguisher Length for $b = 256$	Distinguisher Length for $b = 384$	Distinguisher length for $b = 512$
differential cryptanalysis	48	72	96
linear cryptanalysis	44^*	66^*	87^*
division cryptanalysis	17	17	19
imp. differential cryptanalysis	17^*	19^*	21^*

Note: the length of linear distinguisher is deduced from our known experimental results; the length of impossible differential distinguisher is estimated based on the state-of-art results on other ciphers.

10.1 Sponge and Duplex Constructions

The mode of KNOT-AEAD is based on the Monkey-Duplex construction [19, 13], and the mode of KNOT-Hash is based on the extended Sponge construction [10, 26]. The two constructions are provably secure against generic attacks and extended in many publications, here are a small part [13, 24, 35, 10, 26]. The Sponge construction is used in the design of the SHA-3 winner Keccak [11] and several lightweight hash functions, e.g. Photon and Spongent [26, 15]. A significant fraction of the CAESAR competition submissions use Sponge/Duplex-based modes for AEAD schemes, including Ascon, Ketje, Keyak, NORX, PRIMATEs [24, 13, 14, 4, 2] and so on. Besides AEAD and hash functions, the two constructions can also be used for constructing MACs, stream ciphers, Pseudo-random generators. In addition, the fundamental cryptographic primitive underlying the two constructions is a fixed-length permutation, which has the advantages that it does not have a key schedule and that its inverse does not need to be implemented.

Compared with classical Merkle-Damgard hash constructions, the Sponge-based design generally has a smaller footprint, mainly due to the smaller state size. The lightweight hash functions PHOTON and SPONGENT are both based on the Sponge construction. Similarly, several lightweight AEADs are built on the Duplex construction, including Ascon, Ketje and Beetle [16].

Based on the two mode constructions and RECTANGLE-like permutations, in this submission, we demonstrate that we can build lightweight AEAD and hash primitives that can be well-suited for both hardware and software environments.

10.2 Bit-Slice Technique and Lightweight Primitives

The bit-slice technique was introduced by Biham in 1997 for speeding up the software speed of DES [8], and was used in the design of the Serpent block cipher [1]. In a bit-slice implementation, one software logical instruction corresponds to simultaneous execution of m hardware logical gates, where m is the length of a subblock. Ascon[24], JH [39], Keccak(SHA-3) [11], Noekeon [21] and Trivium [23] are 5 other primitives that can benefit from the bit-slice technique for their software performance. It is worth noticing that these primitives not only perform well in hardware but also in software. Furthermore, a bit-slice implementation is safe against implementation attacks such as cache and timing attacks compared with a table-based implementation [33]. When it comes to a dedicated lightweight AEAD/Hash with bit-slice style, there is plenty of room for improvement on tradeoff between security, performance and resource requirements.

The KNOT permutations are extensions of the lightweight block cipher RECTANGLE. Consider a b-bit SP-network permutation, the S-layer consists of $\frac{b}{4}$ 4-bit S-boxes in parallel,

Table 12. Choice of the Rotation Offsets of ShiftRow$_b$ transformation

Permutation width b	Number of candidates	Minimal number of rounds for full dependency
256	11	8
384	1	9
256	24	10

thus the subblock length is $\frac{b}{4}$ for a bit-slice implementation. Let a b-bit state be arranged as a $4 \times \frac{b}{4}$ array. First, apply the same S-box to each column independently. Then, the P-layer should make each column dependent on some other columns, aiming to provide good diffusion. In such a situation, $\frac{b}{4}$-bit rotations are probably the best choice: they are simple wirings in hardware implementation; they can achieve the goal of mixing up different columns; they can be easily implemented in software using bit-slice technique. So far, we got the framework of KNOT permutations.

10.3 The ShiftRow Transformation

Let c_i $(i = 0, 1, 2, 3)$ denote the left rotation offset of the i-th row. For each permutation width b, the choice criteria of c_i are as follows:

1. The four offsets are different;
2. $c_0 = 0$, $c_1 = 1$, $c_2 \equiv 0 \pmod 8$, $c_3 \equiv 1$ or $7 \pmod 8$, and $c_1 < c_2 < c_3$;
3. Full dependency after a minimal number of rounds.

Table 12 gives our experimental results. For each of the candidates satisfying the above criteria, after the minimal number of rounds for full dependency, each of the b input bits influences each of the b output bits. For each b, we choose one candidate as the rotation offsets of the ShiftRow$_b$ transformation.

10.4 Design Criteria of the S-box

Let S denote a 4×4 S-box. Let $\triangle I, \triangle O \in F_2^4$, define $ND_S(\triangle I, \triangle O)$ as:

$$ND_S(\triangle I, \triangle O) = \sharp\{x \in F_2^4 | S(x) \oplus S(x \oplus \triangle I) = \triangle O\}.$$

Let $\Gamma I, \Gamma O \in F_2^4$, define the imbalance $Imb_S(\Gamma I, \Gamma O)$ as:

$$Imb_S(\Gamma I, \Gamma O) = |\sharp\{x \in F_2^4 | \Gamma I \bullet x = \Gamma O \bullet S(x)\} - 8|.$$

where $\bullet$ denotes the inner product on F_2^4. The design criteria of the S-box of KNOT are as follows:

1. Bijective, i.e., $S(x) \neq S(x')$ for any $x \neq x'$.
2. For any non-zero input difference $\triangle I \in F_2^4$ and any non-zero output difference $\triangle O \in F_2^4$, we require:
$$ND_S(\triangle I, \triangle O) \leq 4.$$
3. Let $\triangle I \in F_2^4$ be a non-zero input difference and $\triangle O \in F_2^4$ a non-zero output difference. Let $wt(x)$ denote the Hamming weight of x. Define $SetD1_S$ as:
$SetD1_S = \{(\triangle I, \triangle O) \in F_2^4 \times F_2^4 | wt(\triangle I) = wt(\triangle O) = 1 \ and \ ND_S(\triangle I, \triangle O) \neq 0\}$.

Let $CarD1_S$ denote the cardinality of $SetD1_S$, we require $CarD1_S = 2$.

4. For any non-zero input selection pattern $\Gamma I \in F_2^4$ and any non-zero output selection pattern $\Gamma O \in F_2^4$, we require:
$$Imb_S(\Gamma I, \Gamma O) \leq 4.$$
5. Let $\Gamma I \in F_2^4$ be a non-zero input selection pattern and $\Gamma O \in F_2^4$ a non-zero output selection pattern, define $SetL1_S$ as:
$SetL1_S = \{(\Gamma I, \Gamma O) \in F_2^4 \times F_2^4 | wt(\Gamma I) = wt(\Gamma O) = 1 \text{ and } Imb_S(\Gamma I, \Gamma O) \neq 0\}$.
Let $CarL1_S$ denote the cardinality of $SetL1_S$, we require $CarL1_S = 2$.
6. No fixed point, i.e., $S(x) \neq x$ for any $x \in F_2^4$.

10.5 Selection of the S-box of KNOT

In the following, an S-box means a 4×4 S-box.

Definition 1 ([28]). *Two S-boxes S and S' are called **permutation-then-XOR equivalent** if there exist 4×4 permutation matrices P_0, P_1 and constants $a, b \in F_2^4$ such that $S'(x) = P_1(S(P_0(x) + a)) + b$. The equivalence is called **PE equivalence** for short.*

If an S-box satisfies criteria 1-5 (see Section 9.4), then any of its PE equivalent S-boxes also satisfies criteria 1-5. In [41], we have shown that there are only 4 PE classes fulfilling criteria 1-5. We list a representative for each PE class in Table 13. In each row of Table 13, the first integer represents the image of 0, the second the image of 1, and so on.

Up to adding constants before and after an S-box, which does not change any of the criteria 1-5 and furthermore does not change the probability of the best differential/linear trail for a specific number of rounds, there are $4\times4!\times4! = 2304$ S-boxes that can be generated from the 4 representatives in Table 13. After discarding a part of the S-box candidates which can result in a differential (or linear) trail with a single active S-box in each round, there remained only 528 S-boxes, see [41] for details.

Next, we create a further filtering by considering the security of the underlying permutation against differential and linear cryptanalysis. Fix the ShiftRow_{256} transformation, for each choice of the remained 528 S-boxes, by checking the probability of the best differential trail and the correlation coefficient of the best linear trail up to 20 rounds, we have chosen 96 S-boxes with good performance in this filtering. By further checking the best differential and linear trails with $b = 384$ and 512, we finally choose one S-box with the best performance from the 96 S-boxes. Finally, by adding constants before and after the S-box, we can get 256 different S-boxes. Among the 256 S-boxes, we choose one with no fixed point, low area requirement and good software performance as the S-box for KNOT.

10.6 The Number of Rounds nr_0, nr, nr_f and nr_h

During the initialization, the key and nonce are firstly loaded to the b-bit state; then, after the $p_b[nr_0]$ transformation, it shall destroy all structures an attacker can apply in choosing

Table 13. Representatives for all the 4 PE classes fulfilling criteria 1-5

PE_0	6,0,8,15,12,3,7,13,11,14,1,4,5,9,10,2
PE_1	3,2,8,13,15,5,6,10,9,14,4,7,0,12,11,1
PE_2	6,8,15,4,12,7,9,3,11,1,0,14,5,10,2,13
PE_3	8,1,6,12,5,15,10,3,7,11,13,2,0,14,9,4

$K \parallel N$ [13]. Based on our security analysis of the KNOT permutations, we selected $nr_0 =$ 52, 76, 76 and 100 for the 4 KNOT-AEAD members respectively.

During the processing of associated data and plaintexts, an attacker can not have access to the inner state. Based on our security analysis of the KNOT permutations, differential cryptanalysis is the most effective approach. Moreover, taking the security strength of each KNOT-AEAD into account, it requires that, for the permutation $p_b[nr]$, there is no effective differential propagation with probability above 2^{-k}. As a result, we selected $nr =$ 28, 28, 40 and 52 for the 4 KNOT-AEAD members respectively.

As for forgery attacks, the success probability of correctly predicting the output of the final permutation $p_b[nr_f]$ from its input difference should not be better than a random guess. Hence, it requires that there is no effective differential propagation with probability above 2^{-k} for $p_b[nr_f]$. Adding an extra redundancy, we selected $nr_f =$ 32, 32, 44 and 56 for the 4 KNOT-AEAD members respectively.

Rebound attack [37] is one of the most effective cryptanalysis against hash functions. It can be applicable to both AES-based and permutation-based hash functions. Based on the state of the art of rebound attacks and our differential cryptanalysis of the KNOT permutations, we estimate that the highest attacked rounds using rebound attacks with 3 inbound phases is at most 60, 69, 93 and 123 for the 4 KNOT-Hash members respectively. Adding an extra redundancy of about 12%, we selected $nr_h =$ 68, 80, 104 and 140 for the 4 KNOT-Hash members respectively.

11 Performance in Various Environments

11.1 Hardware Evaluation of KNOT

The hardware implementation area of an AEAD algorithm greatly depends on design parameters, such as the utilized protocol, the width of the interface and etc. These design parameters normally are chosen for specific application scenarios and reflect the applicability and the flexibility of an AEAD algorithm. In order to study the intrinsic hardware performance of KNOT, we focus on the core datapath of an KNOT. Together with various compact controllers, the core datapath of an KNOT is capable to accomplish all functional modes, including AE mode and hash mode.

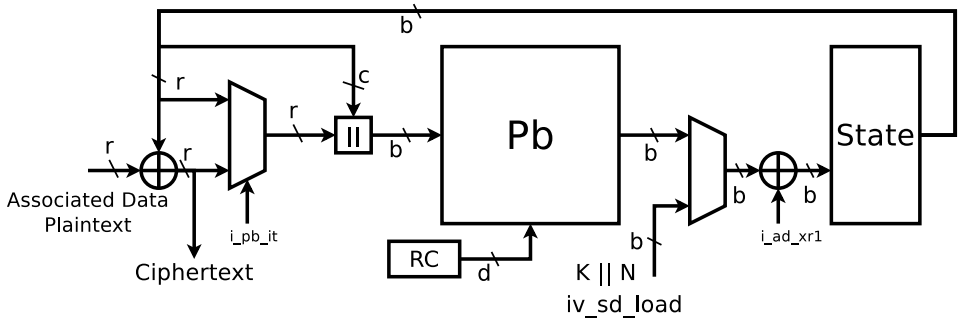

Fig. 1. The datapath of a crypto_core of KNOT-Pair I

Figure 1 illustrates the core datapath of KNOT-AEAD(128,256,64). In this figure, the KNOT permutation and round constant generator are denoted by *Pb* and *RC*. The key and

Table 14. Hardware results of the crypto_core of KNOT-AEAD(128,256,64)

KNOT-AEAD(128, 256, 64)	Area (GE)	Latency (ns)
crypto_core	3628	0.89
Permutation	1206.67	0.33
one S-box	18.67	0.27
Round Constant 6	48.67	0.12
State	1450.67	N/A

Table 15. Hardware results of the crypto_core of KNOT members

crypto_core	Area (GE)	Latency (ns)
KNOT-AEAD(128, 256, 64)	3628	0.89
KNOT-AEAD(128, 384, 192)	5905	0.92
KNOT-AEAD(192, 384, 96)	5421	0.91
KNOT-AEAD(256, 512, 128)	7218	0.89

nonce are loaded via a multiplex to initialize the state. The associated data, plaintext and ciphertext are fed into the datapath by XORing with specific segments of the state.

To evaluate the hardware performance of KNOT-AEAD(128,256,64), the design shown in Figure 1 was synthesized with Synopsys Design Compiler G-2012.06-SP5 to the NANGATE 45 open cell library(PDKv1 3 v2010 12). Table 14 summarizes the results. The core datapath consumes in total around 3.7 kGE(gate equivalent), while the consumption of the permutation part and the state are 1.2 kGE and 1.4 kGE respectively. The critical path of this core datapath is around 0.89 ns, implying that a theoretical upper bound of maximal working frequency is around 1.12 GHz.

We repeated the same evaluation approach for all the other KNOT-AEAD members. The result is listed in Table 15. Thanks to their similar architectures, the hardware implementations of the cryptographic cores for all KNOT-AEAD members have similar critical path length. As shown in Table 15, the hardware area of these implementations varies, mainly due to their different state sizes. It is known that the throughput of an AEAD algorithm greatly depends on sizes of its input and output interface, which should be specified according to a target application. Therefore, instead of analyzing the throughput, we studied latencies of the critical path of all proposed implementations. We note that these latencies can be improved to achieve a higher maximal working frequency by techniques such as pipeline, multiple clock domains and etc. In the ideal case, there is no further throughput reduction caused by interfaces, KNOT-AEAD(128, 256, 64) could achieve a throughput of 0.642 - 2.568 Gbps.

The same core datapath of KNOT-AEAD members can be reused by corresponding KNOT-Hash members from the same pair. Under the ideal case assumption, the KNOT-Hash(256, 256, 32, 128) could reach a throughput of 175.39 - 528.75 Mbps.

11.2 Software Implementation on 64-bit Platforms

The platform used to evaluate the software implementation of KNOT is an Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz based system running Ubuntu 18. We follow the requirement on

Table 16. KNOT-AEAD Encryption Speed

	cycles/byte (for 8 bytes)	cycles/byte (for 2048 bytes)
KNOT-AEAD(128, 256, 64)	74	23
KNOT-AEAD(128, 384, 192)	137	17
KNOT-AEAD(192, 384, 96)	159	51
KNOT-AEAD(256, 512, 128)	198	47

Table 17. KNOT-Hash Message Processing Speed (Reference Code)

	cycles/byte (for 8 bytes)	cycles/byte (for 2048 bytes)
KNOT-Hash(256, 256, 32, 128)	234	111
KNOT-Hash(256, 384, 128, 128)	310	74
KNOT-Hash(384, 384, 48, 192)	685	299
KNOT-Hash(512, 512, 64, 256)	769	253

compiler and flags:

GCC 7.3.0 using flags: -std=c99 -Wall -Wextra -Wshadow -fsanitize=address,undefined -O2

The main purpose of the reference implementation is to support public understanding, we provide well-documented and straightforward implementations for all the KNOT members. In the following, we present the results of optimized implementations of KNOT. Table 16 presents the encryption speed of the 4 KNOT-AEAD members for 8-byte and 4096-byte messages respectively. Table 17 presents the message processing speed of the 4 KNOT-Hash members, also for 8-byte and 4096-byte messages respectively.

11.3 Software Implementation on 8-bit Platforms

All members of KNOT are expected to have quite small code size (ROM) and very low RAM. To show this, we implemented all members of KNOT. The implementations were written in assembly code and compiled using AVR macro assembler 2.2.7 in Atmel Studio 7.0 with AVR ATmega128 as the targeted device. The code sizes and the RAM usage were also measured using Atmel Studio 7.0. Table 18 presents the measured results of the implementations of all KNOT members.

In all implementations, the size of the required RAM equals the size of the state (excluding the RAM required to store test vectors, i.e., key, nonce, messages, outputs). The implementations of the core permutations is in the bit-sliced way, in which the execution of the (eight) S-boxes are implementated using a sequence of 13 instructions. In addition, the execution of the S-boxes are combined with the row rotation to minimize the number of memory accesses.

From Table 18, all members of KNOT-AEAD can be implmented with code size less than 800 bytes, and all members of KNOT-Hash can be implemented with code size less than 650 bytes. It is worth mentioning that, from the last column of Table 18, supporting hash functionality on top of AEAD costs very limited additional resources, that is, only 70-114 ROM bytes are added.

KNOT Members	Memory Type	AEAD	Hash	AEAD+Hash
KNOT-Pair I: AEAD(128, 256, 64) + Hash(256, 256, 32, 128)	RAM	32	32	32
	ROM	652	506	756
KNOT-Pair II: AEAD(128, 384, 192) + Hash(256, 384, 128, 128)	RAM	48	48	48
	ROM	730	570	800
KNOT-Pair III: AEAD(192, 384, 96) + Hash(384, 384, 48, 192)	RAM	48	48	48
	ROM	716	570	786
KNOT-Pair IV: AEAD(256, 512, 128) + Hash(512, 512, 64, 256)	RAM	64	64	64
	ROM	786	650	900

ROM was measured excluding the codes for generating test vectors. RAM was measured excluding those used for storing test vectors (i.e., key, nonce, messages, outputs).

Table 18. Implementation costs in 8-bit AVR processors (memory costs in bytes)

11.4 Capability of Integrating Side-Channel Countermeasures

For application scenarios where side-channel resistance is critical, KNOT by design can be implemented efficiently. Implementations of the KNOT families could follow the bit-slice style without using the look-up tables, which helps to mitigate the threat of cache-timing attacks. The 4-bit S-box used in KNOT comes from the same class as RECTANGLE and PRESENT [7]. The existence of countermeasures for S-boxes of RECTANGLE and PRESENT implies the feasibility of implementing efficient first-order and high-order masking, or threshold implementations for KNOT algorithms.

11.5 Targeted Constrained Devices of KNOT

The bit-slice style makes KNOT well-suited for both hardware and software. Due to small state size, careful selection of the 4-bit S-box and a bit permutation as the diffusion layer, KNOT is extremely hardware-friendly. As for software, the KNOT S-box can be implemented using a sequence of 12 basic logical instructions, the P-layer of each KNOT permutations is composed of 3 $b/4$-bit rotations, which makes KNOT also very friendly for software implementations. Moreover, the bit-sliced design principle allows KNOT for very flexible hardware/software implementations.

The researchers in University of Luxembourg developed an open-source framework-FELICS (Fair Evaluation of Lightweight Cryptographic Systems) [25] in 2015, which aims at fairly evaluating the software performance of lightweight ciphers on embedded devices. By extracting Flash, RAM consumption and execution time on 3 widely used microcontrollers: 8-bit AVR, 16-bit MSP and 32-bit ARM, the ciphers are ranked respectively with an average value under scenario 1 (communication protocol) and scenario 2 (authentication protocol). There are 22 lightweight block ciphers in this evaluation, the RECTANGLE block cipher ranked the 4th in Scenario 1 and 5th in Scenario 2, which indicates that RECTANGLE has very good software performance on embedded devices. Since the KNOT permutations are extensions of RECTANGLE, it can be deduced that KNOT also has very good performances on these 3 microcontrollers.

To sum up, KNOT is well suited in constrained environments, including various hardware and embedded software platforms.

12 Advantages and Limitations of KNOT

We would like to emphasize the following advantages of KNOT:

1. KNOT uses two provably secure modes, that is, Monkey Duplex for KNOT-AEAD and an extended Sponge for KNOT-Hash.
2. KNOT-AEAD has 4 members, KNOT-Hash also has 4 members, which can provide different security requirements.
3. Security of the KNOT permutations against known cryptanalytic approaches can be thoroughly evaluated, especially the security against differential and linear cryptanalysis. Moreover, the KNOT permutations are based on the design of RECTANGLE, hence, the security analysis and hardware/software implementations of KNOT can benefit from the known results on RECTANGLE [40, 25, 5, 30].
4. Due to the bit-slice style, KNOT allows for very efficient and flexible implementations in both hardware and software environments. Moreover, the implementation of the round function can be reused in the KNOT-AEAD and KNOT-Hash of the same KNOT-Pair, which reduces the hardware area or software ROM compared to use of two different primitives. The last but not least, bit-slice style, together with carefully selected S-box, enables efficient side-channel resistant implementations of KNOT.
5. KNOT is especially well-suited for constrained devices, due to its mode selection, small state size, 4-bit Sbox and a bit permutation as the diffusion layer.
6. KNOT is inverse-free, that is, there is no need to implement the inverse of the underlying permutations.
7. No hidden weaknesses in KNOT. All design choices are explained and motivated in this document.

It requires that the nonce of KNOT is unique, the uniqueness of the nonce is as critical as the secrecy of a key. KNOT is a new design, we encourage further security analysis on KNOT.

Acknowledgements

We are very grateful to Chun Guo, Vincent Rijmen, Lei Wang and Peng Wang for their helpful comments.

References

1. Anderson, R., Biham, E., Knudsen, L.R.: Serpent: A Proposal for the Advanced Encryption Standard. NIST AES proposal (1998).
2. Andreeva, E., Bilgin, B., Bogdanov, A., Luykx, A., Mendel, F., Mennink, B., Mouha, N., Wang, Q., Yasuda, K.: PRIMATEs v1 (2014), submission to CAESAR competition.
3. Andreeva, E., Mennink, B., and Preneel, B.: The parazoa family: Generalizing the sponge hash functions. International Journal of Information Security 11(3), 149-165, 2012.
4. Aumasson, J., Jovanovic, P., Neves, S.: NORX v1 (2014), submission to CAESAR competition.
5. Bao Z., Luo P., Lin D.: Bitsliced Implementations of the PRINCE, LED and RECTANGLE Block Ciphers on AVR 8-Bit Microcontrollers. ICICS 2015, Springer-Verlag, LNCS, Vol. 9543, 18-36.

6. Bao, Z., Zhang, W., Lin, D.: Speeding up the Search Algorithm for the Best Differential and Best Linear Trails, Inscrypt'2014, LNCS, vol. 8957, 259–285, Springer (2014).
7. Bilgin, B., Nikova, S., Nikov, V., Rijmen, V., Sttz, G.: Threshold Implementations of All 3×3 and 4×4 S-Boxes. CHES 2012: 76–91. Springer (2012).
8. Biham, E.: A Fast New DES Implementation in Software. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, 260–272. Springer (1997).
9. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, 12–23. Springer (1999).
10. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Cryptographic sponge functions, 2011. http://sponge.noekeon.org/CSF-0.1.pdf
11. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak Specifications. NIST SHA-3 Submission (2008), http://keccak.noekeon.org/
12. Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. Journal of Cryptology 4(1), 3–72 (1991).
13. Bertoni, G., Daemen, J., Peeters, M., and Assche, G.: Ketje v2(2014), submission to CAESAR competition.
14. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., Van Keer, R.: Keyak v1 (2014), submission to CAESAR competition.
15. A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varici, I. Verbauwhede, spongent: A lightweight hash function, in B. Preneel, T. Takagi, (eds.) Cryptographic Hardware and Embedded SystemsCHES 201113th InternationalWorkshop, Nara, Japan, September 28COctober 1, 2011. Proceedings. LNCS, vol. 6917, 312C325. Springer (2011).
16. A. Chakraborti, N. Datta, M. Nandi, K. Yasuda.: Beetle Family of Lightweight and Secure Authenticated Encryption Ciphers, IACR Transactions on Cryptographic Hardware and Embedded Systems. 2018(2), 218–241.
17. Collard, B., Standaert, F.X.: A Statistical Saturation Attack against the Block Cipher PRESENT. In: Fischlin, M. (ed.) CT-RSA 2009. LNCS, vol. 5473, 195–210. Springer (2009).
18. Courtois N., Pieprzyk J.: Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In Zheng Y., ed. 8th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2002), LNCS 2501, 2002, 267– 287. Springer (2002).
19. Daemen, J.: Permutation-based Encryption, Authentication and Authenticated Encryption. DIAC - Directions in Authenticated Ciphers, July 2012.
20. Daemen, J., Knudsen, L. R., Rijmen, V.: The block cipher Square. In Biham, E., editor, Fast Software Encryption, FSE 1997, LNCS, vol. 1267, 149–165. Springer (1997).
21. Daemen, J., Peeters, M., Van Assche, G., Rijmen, V.: Nessie Proposal: the Block Cipher Noekeon, Nessie submission (2000), http://gro.noekeon.org/
22. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer (2002).
23. De Cannière, C., Preneel, B. Trivium. In: Robshaw, M., Billet , O.(eds.), New Stream Cipher Designs - The eSTREAM Finalists, LNCS, vol. 4986. 244–266, Springer (2008).
24. C. Dobraunig, M. Eichlseder, F. Mendel, M. Schl?ffer, Ascon v1.1 (2015), submission to CAESAR competition.
25. FELICS website, https://www.cryptolux.org/index.php/FELICS_Block_Ciphers_Brief_Results.
26. Guo, J., Peyrin, T., and Poschmann, A.: The photon family of lightweight hash functions. In: P. Rogaway (Ed.), CRYPTO 2011. LNCS, vol. 6841, 222-239. Springer (2011).
27. Knudsen, L.R., Wagner, D.: Integral Cryptanalysis. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, 112–127. Springer (2002).
28. Leander, G., Poschmann, A.: On the Classification of 4 bit S-boxes. In: Carlet, C., Sunar, B. (eds.) WAIFI 2007. LNCS, vol. 4547, 159–176. Springer (2007).
29. Li T., Sun Y., Liao M.D., Wang D.K.: Preimage Attacks on the Round-reduced Keccak with Cross-linear Structures. IACR Trans. Symmetric Cryptol. 2017(4): 39–57 (2017).
30. Maene,P.,Verbauwhede,I.:Single-Cycle Implementations of Block Ciphers, Lightweight Cryptography for Security and Privacy:LightSec 2015, LNCS, Vol.9542, 131C-147. Springer (2015).

31. Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, 386–397. Springer (1994).
32. Matsui, M.: On Correlation between the Order of S-Boxes and the Strength of DES. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, 366–375. Springer (1995).
33. Matsui, M., Nakajima, J.: On the Power of Bitslice Implementation on Intel Core2 Processor. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, 121–134. Springer (2007)
34. Nakahara J., Seperhdad P., Zhang B., Wang M.,: Linear (Hull) and Algebraic Cryptanalysis of the Block Cipher PRESENT, CANS' 2009, LNCS 5888, 58C-75, Springer (2009).
35. Philipp, J., Atul, L., Bart, M., Yu, S., Kan, Y.: Beyond conventional security in sponge-based authenticated encryption modes. Journal of Cryptology, 2018.
36. Todo, Y.: Structural evaluation by generalized integral property. In: Advances in Cryptology EUROCRYPT 2015, 287–314. Springer (2015).
37. Mendel, F., Rechberger, C., Schlaffer, M., Thomsen, S.S.: The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Gr?stl. In: Fast Software Encryption - FSE 2009. LNCS, vol. 1008. Springer (2009).
38. Xiang, Z., Zhang, W., Bao, Z., Lin, D.: Applying milp method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In: International Conference on the Theory and Application of Cryptology and Information Security. 648–678. Springer (2016).
39. Wu, H.: The Hash Function JH. Submission to NIST (2008), http://icsd.i2r.a-star.edu.sg/staff/hongjun/jh/jh.pdf
40. Zhang, W., Bao, Z., Rijmen, V., Liu, M.: A New Classification of 4-bit Optimal S-boxes and its Application to PRESENT, RECTANGLE and SPONGENT, In: Leander, G. (ed.) FSE 2015. LNCS, vol.9054, 494–515. Springer (2015).
41. Zhang, W., Bao, Z., Lin, D., Rijmen, V., Yang, B., Verbauwhede, I.: RECTANGLE: a bit-slice lightweight block cipher suitable for multiple platforms. Sci. China Inf. Sci., 2015, 58: 122103(15).