Orthros: A Low-Latency PRF

Subhadeep Banik¹ and Takanori Isobe²,³,⁴ and Fukang Liu²,⁵ and Kazuhiko Minematsu⁶ and Kosei Sakamoto²

¹ LASEC, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland. subhadeep.banik@epfl.ch
² University of Hyogo, Kobe, Japan. takanori.isobe@a1.u-hyogo.ac.jp, liufukang@gmail.com, k.sakamoto0728@gmail.com
³ NICT, Tokyo, Japan
⁴ PRESTO, Japan Science and Technology Agency, Tokyo, Japan
⁵ East China Normal University, Shanghai, China
⁶ NEC, Kawasaki, Japan k-minematsu@nec.com

Abstract. We present Orthros, a 128-bit block pseudorandom function. It is designed with primary focus on latency of fully unrolled circuits. For this purpose, we adopt a parallel structure comprising two keyed permutations. The round function of each permutation is similar to Midori, a low-energy block cipher, however we thoroughly revise it to reduce latency, and introduce different rounds to significantly improve cryptographic strength in a small number of rounds. We provide a comprehensive, dedicated security analysis. For hardware implementation, Orthros achieves the lowest latency among the state-of-the-art low-latency primitives. For example, using the STM 90nm library, Orthros achieves a minimum latency of around 2.4 ns, while other constructions like PRINCE, Midori-128 and QARMA⁻¹₂₈⁻σ₀ achieve 2.56 ns, 4.10 ns, 4.38 ns respectively.

Keywords: Pseudorandom Function · Low Latency · Lightweight Cryptography · Sum of Permutations

1 Introduction

1.1 Low-Latency Encryption

Lightweight cryptography is a subfield of symmetric-key cryptography to study cryptographic core functions usable under strong resource constraints. Hardware circuit size is a typical metric, and there are numerous proposals such as GIFT [BPP⁺¹⁷], KATAN [DDK09], LED [GPPR11], Piccolo [SIH⁺¹¹], PRESENT [BKL⁺⁰⁷], and SIMON [BSS⁺¹³] particularly perform well on this metric. Some other metrics exist, and among them, latency has been increasingly receiving attention. Latency affects the response time of encryption or authentication, and a small latency is highly desirable for applications that require instant response, such as encryption of memory bus, storage systems, automotive communication and industrial control network. Gaining throughput is generally possible with common signal processing techniques (pipelining and parallel processing), while achieving a low latency remains a challenge [KNR12].

To our knowledge, the first lightweight block cipher with explicit focus on latency is PRINCE proposed by Borghoff et al. [BCG⁺¹²]. PRINCE is a 64-bit block cipher comprising multiple round functions to significantly reduce the number of rounds while keeping a moderate complexity of each round. Another proposal is QARMA proposed by Avanzi [Ava¹⁷], which is a family of low-latency tweakable block ciphers (TBCs) [LRW02]. It adopts the design strategy of PRINCE. Mantis [BJK⁺¹⁶] is another family of low-latency...
Orthros: A Low-Latency PRF

Midori is a family of block ciphers proposed by Banik et al. \cite{BBI15}. It primarily aims to reduce energy, however its latency is also quite small.

The current work on low-latency encryption focused on invertible primitives, \textit{i.e.}, (tweakable) block ciphers. We started with a question whether this is an exclusive approach – namely, whether we can do better by not requiring an invertible primitive. Motivated by this question, we initiated a study on designing low-latency (non-invertible) pseudorandom function (PRF) consisting of parallel keyed permutations. We study a sum of two block ciphers denoted as $C = E_K(M) \oplus E'_K(M)$, where $E, E' : K \times M \to M$ are different $n$-bit block ciphers with a key space $K$ and a message space $M = \{0, 1\}^n$. Since $E$ and $E'$ can be computed in parallel, the critical path length of a fully unrolled circuit is a maximum of them instead of the sum. The resulting function has $n$-bit block and is not invertible in general.

The sum of permutations is indeed not new and it has been adopted in the designs of RIPEMD-160 \cite{DBP96} and Grøstl \cite{GKM+09}. In addition, the sum of permutations has also been extensively studied in the context of provable security (see Section 3.1). In particular, the result of Dai \textit{et al.} \cite{DHT17} suggests that it can ideally achieve $n$-bit PRF security, \textit{i.e.}, indistinguishable from a truly random function with $O(2^n)$ complexity. However this requires that $E_K$ and $E'_K$ behave as computationally-secure block ciphers, more formally, (computationally)-independent pseudorandom permutations (PRPs). Instead of requiring this, we explore the setting that $E$ and $E'$ are rather weak as a stand-alone block cipher, using a small number of very simple rounds. The point is that the outputs of $E$ and $E'$ are never given in clear, hence we can hope that both can cover each weakness, and consequently the sum of them can tolerate dedicated attacks as a PRF.

1.2 Our Design

Based on the aforementioned considerations, we present Orthros, which is a 128-bit block pseudorandom function (PRF) with a 128-bit key. The overall structure of Orthros is a sum of two SPN-type keyed permutations called Branch1 and Branch2. The round functions of Orthros are based on Midori. It is already suitable to low-latency ciphers, however we performed a thorough study on it and showed that we can further improve latency by adopting new permutation layers and S-boxes.

In particular, we propose a hybrid use of bit and nibble permutations \textit{i.e.}, a bit permutation is used for some rounds and a nibble permutation for the rest, while Midori-128 \cite{BBI15} uses a single linear layer including both of a bit permutation and a nibble permutation. Consequently, each branch of Orthros achieves the 2.5-round full diffusion and attains 64 active S-boxes over 10 rounds, while Midori-128 requires 3 rounds for the full diffusion and 13 rounds for 64 active S-boxes. In addition, the whole Orthros has more than 64 active S-boxes over only 5 rounds. Importantly, this change of linear layers does not require any additional hardware cost in an unrolled implementation.

For PRF, we do not need an involutory S-box unlike Midori and QARMA. This allows us to develop a new 4-bit S-box that offers about twice smaller delay than that of Midori-128 \cite{BBI15} (see Table 7 of Page 14).

Since we do not rely on provable security of Sum of Permutations, we carried out an extensive security analysis on not only the components of Branch1 and Branch2 but also the whole Orthros.

Motivation for using 128-bit PRF. The lack of invertibility limits applications. For example the classical CBC and XTS (a storage encryption mode) \cite{Dwo10} require the decryption routine of a block cipher. However, many popular modes, \textit{e.g.}, CTR, CMAC \cite{Dwo05} and GCM \cite{NIS07}, do not require the decryption routine (this property is also called inverse-freeness). For these modes Orthros can be used as the cryptographic core. For examples of applications that require low latency, ARM’s pointer authentication code
(PAC) uses QARMA [Qua]. PAC is a MAC for the pointer value and the memory context and it is derived by truncating the output of QARMA in an ECB-like mode. The length of PAC ranges from 11 to 31 bits or 3 to 23 bits, depending on a processor feature [Qua]. The PAC is inserted into a reserved space of the original 64-bit pointer containing the address. This mode is inverse-free, even though QARMA is an invertible primitive.

As another example we consider memory protection schemes based on (a MAC variant of) Merkle Tree. They often use a black-box PRF as a MAC function (e.g., [HJ06]), and a concrete memory encryption scheme inside Intel’s SGX [Gue16] adopts inverse-free modes, namely variants of GMAC and GCM. A notable benefit of using a dedicated PRF instead of a PRP is that it can provide a stronger, beyond-birthday-bound (BBB) security depending on the mode. As observed by [MN17b], by changing the component of GCM or CTR from a PRP to a PRF, the provable security immediately improves from $n/2$ to $n$ bits. In such a case, there is a strong incentive to use a PRF from the security perspective.

If we compare our proposal with PRINCE, the larger input and output extend possible applications. In fact, the first example of PAC requires input larger than 64 bits while not exceeding 128 bits. This excludes the application of PRINCE [Qua, Page 6 (Cryptography)] and makes Orthros usable. In the latter example of tree-based memory protection, a GCM-like authenticated encryption mode with PRINCE will only have 32-bit security while Orthros ensures 128-bit security as described above. We note that the output size is also crucial for 128-bit security in this case, since we need 128 bits of pseudorandom sequence to mask 128-bit universal hash function output.

**Implementations.** We implemented Orthros in four different standard cell libraries along with 2 other constructions Midori-128 and QARMA-128-σ₀ that have a block size of 128 bits and offer at least 128 bits of security. Across all libraries, Orthros performs around 40% better than the above designs with respect to a) the absolute delay between input/output ports and also b) the delay when the circuits are restricted to a certain area/power budget. We even found that Orthros performs marginally better than PRINCE (which has a blocksize of 64 bits and offers $(127-d)$ bits of security give $2^d$ plaintext/ciphertext pairs) with respect to the total circuit delay across all libraries. All our implementations are publicly available.

**Related Work.** Mennink and Neves [MN17b] proposed a generalized EDMD mode [MN17a] and an instantiation of it by a pair of reduced-round block ciphers, $E$ and $E'$. The resulting scheme is a PRF of nearly $n$-bit security [MN17a] if $E$ and $E'$ are PRPs. In [MN17b] they proposed an instantiation based on reduced-round AES, called AES-PRF. It has been analyzed by Derbez et al. [DIS⁺18]. Although a conceptual similarity to us, the generalized EDMD is not suitable for low-latency PRF because it is serial. Besides, they did not consider to use dedicated round functions. In a broader context, reduced-round version of a standard block cipher has been used to build a wide variety of cryptographic functions. Most notably for AES, there are examples such as a MAC function [DR05b,DR05a], a stream cipher [Bir07], a hash function [GM16], an authenticated encryption scheme [HKR15], and a tweakable block cipher [BGGS20] to name a few.

From the provable security perspective, the $n$-bit security of sum of two PRPs has been proved [DHT17] as described. Chen et al. [CLM19] studied the sum of Even-Mansour ciphers and proved its tight security of $2n/3$-bit.

---

1. https://github.com/subhadeep-banik/orthros
2 Specification

Orthros is a 128-bit pseudorandom function (PRF) with a 128-bit key, the overview of which is illustrated in Fig 1. On the whole, Orthros consists of two SPN-based 128-bit keyed permutations called Branch1 and Branch2, each composed of an S-layer, P-layer, the round-key addition and the constant addition. The S-layer is the parallel application of a 4-bit S-box and the P-layer is a linear transform (bit or nibble permutation, followed by a matrix multiplication). Moreover, two key scheduling functions called KSF1 and KSF2 based on two different bit permutations are exploited in Branch1 and Branch2, respectively.

In Orthros, a 128-bit plaintext $M^2$ is first copied to two 128-bit internal states $X^1$ and $X^2$. Then $X^1$ and $X^2$ are respectively given to Branch1 and Branch2. The 128-bit ciphertext $C$ is an XOR of the outputs of Branch1 and Branch2. More details will be given in the following.

Figure 1: Overview of Orthros.

2.1 Key Scheduling Function

Orthros adopts two bit-permutation-based key scheduling functions called KSF1 and KSF2, which are used to generate $RK^1_r$ and $RK^2_r$ ($0 \leq r \leq 12$) from the same 128-bit key $K$ for Branch1 and Branch2, respectively. The whitening keys are $RK^1_0$ and $RK^2_0$, which will be first XORed with $X^1$ and $X^2$, respectively. $RK^1_r$ and $RK^2_r$ ($1 \leq r \leq 12$) are the round keys used in the $r$-th round of Branch1 and Branch2, respectively. The algorithms of KSF1 and KSF2 are shown in Fig 2. The bit-permutation $P_{bk1}$ and $P_{bk2}$ used in KSF1 and KSF2 are shown in Table 2.

In Fig 2, when $K$ and $RK^1_r$ are expressed in bit level, we have $RK^1_r = (rk^1_{r,0}, rk^1_{r,1}, \ldots, rk^1_{r,127})$ and $K = (k_0, k_1, \ldots, k_{127})$, where $rk^1_{r,i}, k_i \in F_2$ (0 $\leq i \leq 127, j \in \{1,2\}$). In addition, $rk^1_{r,0}$ and $k_0$ are the most significant bit of $RK^1_r$ and $K$, respectively.

2.2 Round Function of Branch1 and Branch2

In this section, we present the details of Branch1 and Branch2, each of which is a 128-bit keyed permutation consisting of 12 rounds. The round keys (and whitening keys) ($RK^1_r$, $RK^2_r$) are first generated by KSF1 and KSF2. After adding the whitening keys, the 128-bit input will be processed via Branch1 and Branch2 as follows.

\footnote{For convenience, we may call an input (an output) of Orthros a plaintext (a ciphertext), although it is not a block cipher.}
For the first 4 rounds of Branch1 and Branch2, the round function \( R \) is described as

\[
R = \text{AddConstant} \circ \text{AddRoundKey} \circ \text{matrixMul} \circ \text{bit-permutation} \circ \text{S-box},
\]

where \( \text{AddConstant} \) and \( \text{AddRoundKey} \) represent the constant addition operation and the round key addition operation, respectively.

For the following 7 rounds, the round function \( R' \) is described as

\[
R' = \text{AddConstant} \circ \text{AddRoundKey} \circ \text{matrixMul} \circ \text{nibble-permutation} \circ \text{S-box}.
\]

The sequence of operations in the last round is \( \text{AddConstant} \circ \text{AddRoundKey} \circ \text{S-box} \). As described, the 128-bit outputs of Branch1 and Branch2 are XORed to generate the 128-bit ciphertext. Each component in the round function of Branch1 and Branch2 is described as follows.

**S-box (S-box).** A 4-bit S-box will be applied to each nibbles in parallel for Branch1 and Branch2. The specification of the 4-bit S-box is displayed in Table 1.

| \( x \) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| \( S(x) \) | 1 | 0 | 2 | 4 | 3 | 8 | 6 | d | 9 | a | b | e | f | c | 7 | 5 |

**Permutation (bit-permutation, nibble-permutation).** For the first 4 rounds of Branch1 and Branch2, \( P_{b1} \) and \( P_{b2} \) will be used respectively. From the 5th round to the 11th round, the nibble permutations \( P_{a1} \) and \( P_{a2} \) will be adopted in each branch respectively. The details of the permutation \( P_{bN} \) and \( P_{aN} \), where \( N \in \{1, 2\} \), are shown in Table 3 and Table 4, respectively.

**Matrix Multiplication (matrixMul).** Let \( M_b \) be \( 4 \times 4 \) matrix over nibbles defined as

\[
M_b = \begin{pmatrix}
0 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 1 & 0
\end{pmatrix}.
\]
Four nibbles \((a_0, a_1, a_2, a_3)\) will be updated as follows:

\[
(a_0, a_1, a_2, a_3)^T \leftarrow M_b \cdot (a_0, a_1, a_2, a_3)^T,
\]

where \((a_0, a_1, a_2, a_3)^T\) denotes a transposition. Specifically,

\[
\begin{align*}
  a_0 &= a_1 \oplus a_2 \oplus a_3, \\
  a_1 &= a_0 \oplus a_2 \oplus a_3, \\
  a_2 &= a_0 \oplus a_1 \oplus a_3, \\
  a_3 &= a_0 \oplus a_1 \oplus a_2.
\end{align*}
\]

**AddRoundKey (AddRoundKey).** In the \(r\)-th round, the internal states in Branch1 and Branch2 will be xored with the corresponding round key \(RK^1_r\) and \(RK^2_r\), respectively.

**AddConstant (AddConstant).** The internal state will be xored with the corresponding round constant in each branch. The specification of the round constants is displayed in Table 10 at Appendix A, where \(RC^1_r\) and \(RC^2_r\) represent the round constant used in the \(r\)-th round of Branch1 and Branch2, respectively.

**Figure 3:** The round function of Branch1 and Branch2 in the first 4 rounds.

**Figure 4:** The round function of Branch1 and Branch2 in the last 8 rounds. The nibble permutation and the matrix multiplication in the last round will be omitted.

**Round Constants.** In a similar manner to PRINCE, the round constants are derived from the fraction part of \(\pi = 3.1415926\ldots\). Specifically, if the fraction part of \(\pi = 3.1415926\ldots\) is expressed in binary string, it will be 00010100000101011001\ldots. By shifting left the binary string by 3 bits, we obtain a binary string 1010 0000 1010 1100 1\ldots, which corresponds to a nibble string 0xa, 0x0, 0xfa, 0xc, and so on.

**Illustration of Round Function.** The illustration of the round function in the first 4 rounds of Branch1 and Branch2 is shown in Fig 3. The illustration of the round function in the last 8 rounds is shown in Fig 4.
Pseudocode. The algorithms of Branch1 and Branch2 are shown in Fig 5. In the pseudocode, when the 128-bit internal state is expressed in bits, we have $X = (x_0, x_1, \cdots, x_{127})$ and $x_0$ is the most significant bit of $X$. When the 128-bit internal state is expressed in nibbles, we have $X = (X_0, X_1, \ldots, X_{31})$ and $X_0$ is the most significant nibble of $X$. For the constant addition, the 128-bit round constant $RC_j^t$ is expressed in nibbles as $RC_j^t = (RC_{j,0}^t, RC_{j,1}^t, \ldots, RC_{j,31}^t)$, for $1 \leq r \leq 12$ and $1 \leq j \leq 2$, where $j$ denotes the index of Branch. Similarly for the round-key addition, the 128-bit round key $RK_j^t$ is expressed in nibbles as $RK_j^t = (RK_{j,0}^t, RK_{j,1}^t, \ldots, RK_{j,31}^t)$, for $1 \leq r \leq 12$ and $1 \leq j \leq 2$.

Processing. The 128-bit ciphertext $C$ is generated by XORing the output of Branch1 and Branch2. The processing algorithm of Orthros is shown in Fig 6. We provide test vectors of Orthros in Appendix H.

Claimed Security. Orthros claims single-key security, and does not claim any security in related-key and known/chosen-key settings.

Table 2: Bit permutation $P_{bk,N}$ for key scheduling KSFN, where $N \in \{1, 2\}$.

| $x$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-----|---|---|---|---|---|---|---|---|---|---|-----|-----|-----|-----|-----|-----|
| $P_{bk,1}$ | 0 | 53 | 67 | 73 | 22 | 95 | 99 | 48 | 61 | 36 | 108 | 1 | 24 | 67 | 119 | 93 |
| $P_{bk,2}$ | 76 | 30 | 53 | 35 | 34 | 46 | 2 | 79 | 41 | 125 | 100 | 87 | 40 | 91 | 14 | 101 |

As described in introduction, the overall structure of Orthros is a sum of two keyed permutations. This structure and its variant has been extensively studied in the literature [Lue00, B199, Pat08, DHT17, CML19, Iwa06]. In particular, if two keyed permutations of Orthros (Branch1 and Branch2) were independent PRPs, we could claim $n$-bit provable security – more specifically the PRF advantage of $(q/2^n)^{1.5} + 2\epsilon(q, t + O(q))$ for $n = 128$ and $q$ adaptive queries and time complexity $t$, where $\epsilon(q, t)$ denotes the PRP advantages of Branch1 and Branch2 with $q$ queries and $t$ time [DHT17]. However, this means that either Branch1 or Branch2 could be already usable as a low-latency PRP, implying that they should have a sufficient amount of security margins against known cryptanalysis. Since

3 Design Rationale

3.1 General Construction

As described in introduction, the overall structure of Orthros is a sum of two keyed permutations. This structure and its variant has been extensively studied in the literature [Lue00, B199, Pat08, DHT17, CML19, Iwa06]. In particular, if two keyed permutations of Orthros (Branch1 and Branch2) were independent PRPs, we could claim $n$-bit provable security – more specifically the PRF advantage of $(q/2^n)^{1.5} + 2\epsilon(q, t + O(q))$ for $n = 128$ and $q$ adaptive queries and time complexity $t$, where $\epsilon(q, t)$ denotes the PRP advantages of Branch1 and Branch2 with $q$ queries and $t$ time [DHT17]. However, this means that either Branch1 or Branch2 could be already usable as a low-latency PRP, implying that they should have a sufficient amount of security margins against known cryptanalysis. Since
Algorithm Orthros($K, M$)
1. $X^1 \leftarrow M, X^2 \leftarrow M$
2. $Y^1 \leftarrow $ Branch1($K, X^1$), $Y^2 \leftarrow $ Branch2($K, X^2$)
3. $C \leftarrow Y^1 \oplus Y^2$
4. return $C$

Figure 6: Processing algorithm of Orthros.

Figure 5: Algorithms of Branch1 and Branch2, where $N \in \{1, 2\}$.
Table 3: Bit permutation $P_{k,N}$ for round function $\text{Branch}_N$, where $N \in \{1, 2\}.$

| $x$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-----|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| $P_{k,(1)}$ | 6 | 46 | 62 | 126 | 70 | 52 | 28 | 14 | 36 | 125 | 72 | 83 | 106 | 95 | 4 | 35 |
| $P_{k,(2)}$ | 69 | 122 | 74 | 62 | 119 | 35 | 15 | 66 | 9 | 85 | 32 | 111 | 24 | 84 | 12 | 106 |
| $x$ | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| $P_{k,(1)}$ | 25 | 41 | 10 | 76 | 87 | 74 | 120 | 42 | 88 | 24 | 11 | 67 | 64 | 38 | 112 | 50 |
| $P_{k,(2)}$ | 31 | 98 | 115 | 59 | 71 | 90 | 58 | 26 | 2 | 44 | 103 | 121 | 114 | 107 | 68 | 16 |
| $x$ | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| $P_{k,(1)}$ | 85 | 109 | 24 | 65 | 99 | 0 | 29 | 3 | 8 | 66 | 114 | 37 | 127 | 100 | 36 | 40 |
| $P_{k,(2)}$ | 84 | 1 | 102 | 34 | 80 | 52 | 76 | 36 | 27 | 94 | 37 | 55 | 82 | 12 | 112 | 64 |
| $x$ | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
| $P_{k,(1)}$ | 13 | 117 | 78 | 86 | 92 | 58 | 124 | 101 | 55 | 89 | 97 | 9 | 18 | 116 | 59 | 15 |
| $P_{k,(2)}$ | 105 | 14 | 81 | 12 | 108 | 124 | 6 | 58 | 89 | 124 | 20 | 72 | 34 | 15 | 56 | 78 |
| $x$ | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 |
| $P_{k,(1)}$ | 20 | 45 | 75 | 2 | 77 | 27 | 1 | 60 | 115 | 107 | 26 | 69 | 119 | 3 | 84 | 54 |
| $P_{k,(2)}$ | 50 | 18 | 81 | 73 | 67 | 88 | 4 | 64 | 114 | 49 | 24 | 45 | 57 | 78 | 100 | 22 |
| $x$ | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 |
| $P_{k,(1)}$ | 124 | 110 | 31 | 82 | 113 | 63 | 84 | 102 | 63 | 118 | 95 | 12 | 30 | 94 | 108 | 32 |
| $P_{k,(2)}$ | 110 | 37 | 116 | 54 | 60 | 10 | 97 | 39 | 3 | 4 | 83 | 96 | 24 | 32 | 42 | 113 | 87 |
| $x$ | 90 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |
| $P_{k,(1)}$ | 5 | 111 | 29 | 43 | 91 | 19 | 79 | 33 | 73 | 44 | 98 | 48 | 22 | 61 | 68 | 105 |
| $P_{k,(2)}$ | 126 | 13 | 41 | 40 | 51 | 25 | 65 | 125 | 8 | 101 | 118 | 28 | 38 | 89 | 5 | 104 |
| $x$ | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 |
| $P_{k,(1)}$ | 34 | 71 | 54 | 104 | 17 | 57 | 80 | 103 | 96 | 121 | 23 | 39 | 122 | 90 | 7 | 16 |
| $P_{k,(2)}$ | 109 | 120 | 69 | 43 | 7 | 77 | 58 | 34 | 10 | 64 | 30 | 95 | 75 | 46 | 0 | 92 |

Table 4: Nibble permutation $P_{n,N}$ for round function $\text{Branch}_N$, where $N \in \{1, 2\}.$

| $x$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-----|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| $P_{n,(1)}$ | 10 | 27 | 5 | 1 | 30 | 21 | 16 | 13 | 21 | 31 | 6 | 14 | 0 | 25 | 11 | 18 |
| $P_{n,(2)}$ | 20 | 13 | 7 | 11 | 23 | 9 | 22 | 23 | 5 | 18 | 23 | 12 | 10 | 23 | 22 | 22 |
| $x$ | 30 | 17 | 15 | 19 | 20 | 22 | 22 | 22 | 22 | 22 | 22 | 22 | 22 | 22 | 22 | 22 |
| $P_{n,(1)}$ | 15 | 28 | 19 | 24 | 3 | 8 | 22 | 3 | 8 | 29 | 9 | 2 | 26 | 20 | 4 | 17 |
| $P_{n,(2)}$ | 14 | 19 | 24 | 22 | 1 | 8 | 4 | 31 | 15 | 6 | 27 | 9 | 16 | 30 | 20 | 3 |
each Branch never gives its outputs in clear, we expect that a pair of weak permutations can suffice to have a desired, $n$-bit secure PRF. Generally this approach is described as “prove-then-prune” [HKR15]. This means that the provable security reduction does not hold anymore, and an implication of the security bound is more or less fuzzy and depends on the scheme. In our analysis we did not find a full-round attack against Branch1 and Branch2 as a PRP, however they have rather slim margins as a standalone block cipher. Therefore, we bolster our security claim with an extensive security analysis on the whole construction.

We note that realizing Branch1 and Branch2 as Even-Mansour ciphers [EM93] can reduce the circuit size thanks to the absence of key schedule. However, it can provide $2n/3$-bit PRF security at best, as proved by Chen et al. [CLM19]. Instead, we adopt bit permutation-based key scheduling functions for its hardware friendliness.

In order to investigate the initial security of the sum of permutations from the perspective of cryptanalysis, we compare in Appendix B the differential and linear behaviour for two toy ciphers adopting single branch and double branches, respectively. However, it should be emphasized that the security of Orthros never relies on our experiments on the toy ciphers but rather a comprehensive study of Orthros, as will be detailed in Section 4.

3.2 Linear Layer

As underlying matrices in the linear layer, we adopt $4 \times 4$ almost MDS binary matrix used in Midori [BBI+15], whose delay is much smaller than MDS matrices. However, as discussed in [BBI+15], its diffusion speed is slower and the lower bounds of the number of active S-boxes in each round is smaller than those of ciphers employing MDS matrices due to its lower branch number. To improve the diffusion speed and to increase active S-boxes in each round, we utilize bit and nibble permutations in a hybrid manner. We will see that this enables to guarantee security with a relatively small number of rounds.

Midori-128 [BBI+15] also adopted a bit and a nibble permutations, however, our design is more efficient in term of the diffusion speed and the number of active S-boxes while keeping the same hardware cost in an unrolled implementation. Specifically, we adopt two different linear layers that consists of a bit permutation and a nibble permutation, i.e., a bit permutation is used for some rounds and a nibble permutation for the rest, while Midori-128 [BBI+15] uses a single linear layer including both of a bit permutation and a nibble permutation. Importantly, this change of linear layers does not require any additional hardware cost in an unrolled implementation, as observed by [BCG+12].

3.2.1 Bit Permutation vs Nibble Permutation.

To see the advantage of our hybrid use of bit and nibble permutations over the consistent use of a bit or a nibble permutation, we compare the diffusion effect and the lower bound of the number of active S-boxes of two different 128-bit block SPN structures, which we call SPN-B and SPN-N. Here, SPN-B (SPN-N) consistently uses a bit (nibble) permutation. Both use the same 4-bit S-box and the matrix as those used in a branch of Orthros. In addition, we used the full-diffusion property of S-box when evaluating the diffusion effect.

**Diffusion.** To evaluate the minimum number of rounds that achieves the full diffusion for each SPN-B and SPN-N, we look into the propagation of one active input bit and count the upper bounds of the number of active bits through each operation. Here, we only need to consider the number of active bits after `S-box` and `matrixMul` as the remaining operations do not affect its value. The upper bounds of the number of active bits after each operation over some rounds are shown in Table 5.

According to Table 5, SPN-B and SPN-N require at least 2.5 rounds (2 rounds plus `S-box`) and 4 rounds, respectively, i.e., the optimal numbers of rounds for the full diffusion
Table 5: The upper bound of the number of active bits after each operation.

| Round | Operation | Structures |
|-------|-----------|------------|
|       |           | SPN-B | SPN-N |
| -     | Input     | 1     | 1     |
| 1     | S-box     | 4     | 4     |
|       | matrixMul | 12    | 12    |
| 2     | S-box     | 48    | 12    |
|       | matrixMul | 120   | 36    |
| 3     | S-box     | 128   | 36    |
|       | matrixMul | 128   | 100   |
| 4     | S-box     | 128   | 100   |
|       | matrixMul | 128   | 128   |

of SPN-B and SPN-N are estimated as 2.5 and 4 rounds, respectively. We conclude that there is a clear gap between bit permutations and nibble permutations in terms of the diffusion. Note that a class of bit permutations covers all nibble permutations. Thus, to achieve the 2.5-round full diffusion, we need to find a class of bit permutations that are not included in a class of nibble permutations.

Active S-box. Mixed Integer Linear Programming (MILP) is used to obtain the lower bound of the number of active S-boxes in each round. Unfortunately, it is computationally infeasible to estimate the lower bound of the number of active S-boxes of SPN-B, except for a very small number of rounds, due to the large search space of 128-bit bitwise differential and linear trails. This problem about the use of bit permutation was also pointed out by the designers of QARMA-128 [Ava17]. As a consequence, QARMA-128 does not claim a lower bound of active S-boxes. Indeed, it was infeasible to compute a lower bound of the number of active S-boxes for more than 5 rounds for SPN-B, even with a computer equipped with 48 cores and 256 GB RAM. Therefore, SPN-B can only guarantee a very small number of active S-boxes for a moderately large number of rounds (say 10), since we have to combine the bounds obtained for a small number of rounds, which generally yields a loose bound. For example, the lower bound for 8 rounds is obtained by the sum of the bounds for 4 rounds. In our experiment, the best lower bound for 4 rounds is 16, therefore the obtained bound for 8 rounds is only 32.

On the other hand, for SPN-N, it is feasible to obtain a tight lower bound of the number of active S-boxes up to 8 rounds, using the aforementioned 48-core computer. The evaluation of one candidate requires about 2 days by the same computer. As a result, we found a class of nibble permutations that attain 60 active S-boxes over 8 rounds. We will explain the details in the following section.

Conclusion. Based on the above discussions, we conclude that a structure employing a single permutation cannot achieve a fast full diffusion and a guaranteed large number of active S-boxes simultaneously. Hence, we decided to use a bit permutation for the first few rounds, and use a nibble permutation for the rest of the rounds. Consequently, Orthros reaches the full diffusion after first 2.5 rounds and guarantees more than 64 active S-boxes over 10 rounds.

3.2.2 Finding Optimal Bit Permutations for Diffusion.

We take a two-step approach to find a class of bit permutations that achieves 2.5-round full diffusion for SPN-B, which turns out to be optimal. Let $S$ denote the 128-bit internal
state $S$ of SPN-B. It is also viewed as a $4 \times 8$ two-dimensional nibble array:

$$S = \begin{bmatrix}
S_0 & S_4 & S_8 & S_{12} & S_{16} & S_{20} & S_{24} & S_{28} \\
S_1 & S_5 & S_9 & S_{13} & S_{17} & S_{21} & S_{25} & S_{29} \\
S_2 & S_6 & S_{10} & S_{14} & S_{18} & S_{22} & S_{26} & S_{30} \\
S_3 & S_7 & S_{11} & S_{15} & S_{19} & S_{23} & S_{27} & S_{31}
\end{bmatrix},$$

where $S_i$ ($0 \leq i \leq 31$) is a 4-bit value defined as

$$S_i = [s_{4i}, s_{4i+1}, s_{4i+2}, s_{4i+3}]^T.$$

Note that $S$ can also be viewed as a $16 \times 8$ two-dimensional binary array by seeing each column of $S$ as a 16-bit sequence. Hereafter, bit-cell means a binary cell in the $16 \times 8$ array and nibble-cell means a nibble cell in the $4 \times 8$ array, that is, $S_i$.

We first try to reduce the search space of target 128-bit permutations as it is computationally infeasible to test all possible $2^{716.16}$ ($= 128!$) candidates. Specifically, we focus on a class satisfying Condition 1 to efficiently find the class of bit permutations having the 2.5 rounds full diffusion property.

**Condition 1.** For any nibble-cell $S_i$ ($0 \leq i \leq 31$), the corresponding 4 bits, $s_{4i}, s_{4i+1}, s_{4i+2}, s_{4i+3}$, are mapped to the bit-cells in different columns after applying the bit permutation.

A detailed description of Condition 1 can be seen in Appendix C.1.

Condition 1 can be justified as follows. If a bit permutation satisfies Condition 1, for the 4 output bits of each S-box, they will be mapped to 4 different groups of inputs to the binary matrix. Observe that in the MatrixMul operation, the binary matrix $M_6$ is independently applied to 8 different groups of inputs, each of which consists of 4 consecutive nibbles. This is expected to be a key feature for the fast diffusion since 1 input active bit is expanded to 4 active bits via the S-box, and the 4 active bits are subsequently expanded to 12 active bits after the bit permutation and MatrixMul operations. This matches the upper bound of the number of active bits after one-round permutation, as shown in Table 5.

In the class of the bit permutations satisfying Condition 1, we obtain the following sufficient condition for the 2.5-round full diffusion. It should be emphasized that for each active bit in the nibble-cell $S_i$ ($0 \leq i \leq 31$), after applying the S-box and the bit permutation satisfying Condition 1, there will exist 4 different groups of inputs to the binary matrix, each of which will contain exactly one active nibble. Therefore, after MatrixMul there will be 12 active nibbles. We add the following condition on these 12 active nibbles.

**Condition 2.** After applying the bit permutation, in each column of the $4 \times 8$ array, there exist at least 2 nibble-cells containing the bits coming from those in the active 12 nibbles.

The proof that Condition 2 is sufficient for the 2.5 round full diffusion in the class of the bit permutations satisfying Condition 1 is described in Appendix C.2.

**3.2.3 Finding Good Nibble Permutation for Active S-boxes.**

As in the case of bit permutations, we take the following two-step approach to find nibble permutations that can activate as many S-boxes as possible over a certain number of rounds, which is expected to outperform that used in Midori-128. To explain our approach, we use the same $4 \times 8$ array to express the 128-bit state as above.

In the first step, we reduce the search space by focusing on the class of nibble permutations satisfying Condition 3 since it is computationally infeasible to estimate the lower bound of the number of active S-boxes for all possible $2^{171.06}$ ($= 32!$) permutations. Condition 3 is chosen to achieve fast diffusion for differences and linear masks.
Table 6: Comparison of lower bounds of the number of active S-boxes.

| Target                    | 4 | 5 | 6 | 7 | 8 |
|---------------------------|---|---|---|---|---|
| Midori-128 [BBI+15]      | 16| 20| 30| 35| 38|
| Our nibble permutation    | 16| 25| 36| 51| 60|

Condition 3. For each column \((S_{4i}, S_{4i+1}, S_{4i+2}, S_{4i+3})\) in the \(4 \times 8\) array, after applying the nibble permutation, they will be mapped to four nibble-cells in different columns.

In the second step, we first randomly choose 7,000 nibble permutations satisfying Condition 3. Then, we compute the lower bound of the number of active S-boxes after 5, 6, 7 and 8 rounds for these nibble permutations. To efficiently find good permutations among these nibble permutations, we first find a class of nibble permutations that have the best lower bound after 5 rounds. Then, we focus on this class and evaluate the lower bound of the number of active S-boxes after 6 rounds, which will be repeated until the 8th round.

As a result, we find three nibble permutations that can achieve 60 active S-boxes over 8 rounds. Table 6 shows the comparison of the lower bound of the number of active S-boxes for Midori-128 and our structure. Compared with Midori-128, our nibble permutations guarantee a much larger number of active S-boxes after 6 rounds, about a factor of 1.5.

3.2.4 Hybrid Use of Bit and Nibble Permutations.

Consequently, we obtain a set of bit and nibble permutations. For Orthros, we pick a bit and nibble permutations from the set, and use the bit permutation for the first 4 rounds, in order to achieve a fast full diffusion. Specifically, it achieves full diffusion in 2.5 rounds, while Midori-128 requires 3 rounds. For the rest of 8 rounds we used the nibble permutation to guarantee a large number of active S-boxes. Indeed, 10 rounds of Orthros starting from the 3rd round, i.e., the 3rd to the 12th round, achieve 64 active S-boxes. We note that Midori-128 needs 13 rounds to obtain 64 active S-boxes, thus the gain is 3 rounds.

3.3 S-box

We search a small-delay and lightweight 4-bit S-box which fulfills the following requirements: (1) the maximal probability of a differential is \(2^{-2}\), (2) the maximal absolute bias of a linear approximation is \(2^{-2}\) and (3) full diffusion, i.e., any input bit difference diffuses to all output bits. We use a metric called depth [BBI+15] to estimate the path delay of S-boxes.

Definition 1. (depth): The depth is defined as the sum of the sequential path delays of basic operations, namely AND, OR, NAND, NOR and NOT.

Following the assumption of [BBI+15], our search assumes that depths of XOR, AND/OR, NAND/NOR, and NOT are weighted as 2, 1.5, 1 and 0.5, respectively, and the required gates of NOT, NAND/NOR, AND/OR and XOR/XNOR are estimated as 0.5, 1, 1.5 and 2 Gate Equivalents (GEs), respectively. We search over the set of all 4-bit S-boxes, whose size is \(2^{25.5}\), sort them in order of small depth, and check whether they satisfy our security requirements.

We remark that our construction does not require the involution property of S-box unlike Midori’s Sb1. It allows us to expand the number of possible candidates from \(2^{25.5}\) (the number of all involution 4-bit S-boxes) to \(2^{44.3}\). As a result, we found an S-box (see Table 1) whose depth and gate size are the lowest and the smallest in our search. Specifically, the depth is 3.5 and the area is 19 GE under the aforementioned assumption of [BBI+15]. The S-box can be expressed as follows, where inputs and outputs are defined as \(\{x_0, x_1, x_2, x_3\}\) and \(\{y_0, y_1, y_2, y_3\}\), and \(x_3\) and \(y_3\) are the most significant bits.
Table 7: Comparison of S-boxes.

|          | Orthros | Midori [BBI+15] | QARMA [Ava17] | PRINCE [BCG+12] |
|----------|---------|-----------------|---------------|-----------------|
| S        | Sb₁     | σ₀              | σ₁            | σ₂             | σ⁻¹           |
| Area [GE] | 10.7    | 12.0            | 11.0          | 12.2           | 15.7          |
| Delay [ps]| 285.7   | 367.2           | 380.6         | 412.3          | 328.3         |
|          |         |                 |               | 341.9          | 390.7         |

Area Optimized

|          | 40.4    | 32.2            | 43.2          | 36.4           | 49.2          | 65.4          | 51.1          | 41.2          |
| Delay [ps]| 37.2    | 71.1            | 39.2          | 89.3           | 53.3          | 52.7          | 80.6          |
| Full diffusion | Yes | Yes            | No            | Yes           | Yes          | Yes          | Yes           |
| Involution  | No      | Yes             | Yes           | Yes           | No           | No           | No            |

Delay Optimized

Compared to the S-box of Midori-128, the depth and area can be reduced to 3.5 and 10.7 GE from 4 and 12 GE, respectively, when synthesized with the standard cell library of the STM 90nm CMOS logic process (as shown in Table 7) with area optimization. The table also shows detailed comparisons with S-boxes of the QARMA and PRINCE family when the circuit is optimized with respect to area as well as delay. The S-box of Orthros performs well when optimized across both metrics. Note that σ₀ does not have the full diffusion property.

3.4 Key Scheduling Function

To minimize the hardware cost, key scheduling functions of Orthros are realized by only bit permutations, whose hardware overhead, such as area and delay, is essentially free. We use a class of bit permutations that satisfy both Condition 1 and 2 in the key scheduling functions of Branch1 and Branch2 as shown in Table 2, although it cannot guarantee the full diffusion property as there is no S-box and Matrix in the key scheduling functions. One reason to introduce two different key scheduling functions for each branch is to increase the hardness of the key-recovery attack, as will be discussed in the next section.

4 Security Evaluation

4.1 Differential/Linear Attack

To evaluate the resistance against differential attacks and linear attacks, one way is to obtain the lower bound of the number of differentially and linearly active S-boxes in each round, which can be efficiently computed with a MILP-based method [MWGP11]. In the following, we will present lower bounds of the number of differentially and linearly active S-boxes for Branch1, Branch2 and the whole Orthros. Since the maximal differential and linear probability of the S-box is $2^{-2}$, it is sufficient to guarantee the security against differential attacks and linear attacks if there are 64 active S-boxes, as it gives $2^{-2\times64} = 2^{-128}$ as an estimate of a differential probability. In our evaluation, we only consider the single-key setting.
As discussed in Section 3.2, we observed that it is computationally infeasible to obtain a lower bound for more than 5 rounds starting from the first round of Branch 1, Branch 2, and Orthros, even with a computer equipped with 48 cores and 256 GB RAM. The search space of 128-bit bitwise differential and linear trails for the first 4 rounds is huge. On the other hand, for the last 8 rounds of Branch 1 and Branch 2 starting from the 5th round, where the nibble permutation is adopted, we can obtain tight lower bounds of the number of active S-boxes for the nibble-wise differential and linear trails. In addition, we can obtain tight lower bounds of the number of active S-boxes of 5 rounds of Orthros starting from the 5th round, i.e., 5 to 9 rounds.

In our evaluation, each of Branch 1, Branch 2 and Orthros is first divided into two parts, i.e., the first 4 rounds and the remaining 8 rounds. We compute a lower bound of the number of active S-boxes for each part. The lower bound of Orthros is obtained by the sum of those of Branch 1 and Branch 2.

The corresponding results are displayed in Table 8. Table 8 shows that there are at least 68 active S-boxes in 5 rounds of Orthros starting from the 5th round, i.e., 5 to 9 rounds. In addition, the last 10 rounds of Branch 1 and Branch 2 including 2 bit-permutation rounds and 8 nibble-permutation rounds, i.e., 3 to 12 rounds, have at least 64 (= 4 + 60) active S-boxes. Although we do not claim any security for Branch 1 and Branch 2 as a full-fledged block cipher, each has a sufficient number of active S-boxes in the full 12 rounds.

| Construction      | bit/nibble | Rounds   |
|-------------------|------------|----------|
| Orthros (the first 4 rounds) | bit        | 2 8 12 16 |
| Orthros (from the 5th round)   | nibble     | - - - -   |
| Branch 1 (the first 4 rounds)   | bit        | 1 4 6 8   |
| Branch 2 (the first 4 rounds)   | bit        | 1 4 5 8   |
| Branch 1 (from the 5th round)   | nibble     | - - - -   |
| Branch 2 (from the 5th round)   | nibble     | - - - -   |

It should be emphasized that the lower bounds of Orthros in Table 8 are not tight, i.e., actually the full rounds of Orthros includes more active S-boxes. This is because the number of active S-boxes of Orthros after 10 rounds is computed as the sum of those of Branch 1 and Branch 2. Besides, those of first 4 rounds and the last 8 rounds are independently obtained. Thus, we expect that the full-round Orthros can resist against the differential attack and the linear attack.

### 4.2 Impossible Differential Attack

The impossible differential attack can be estimated by the required number of rounds for the full diffusions. In the forward direction, both Branch 1 and Branch 2 require 2.5 rounds for the full diffusion, while it is 5 rounds in the backward direction. Consequently, we expect that there is no any probability-one impossible differential characteristic over 8 rounds of Branch 1 and Branch 2, respectively. Since Orthros take a sum of the outputs of Branch 1 and Branch 2, we believe that the number of rounds of an impossible differential characteristic for Orthros is much lower than that of Branch 1 and Branch 2.

To obtain actual impossible differential characteristics, we utilize the MILP-aided automatic searching tool proposed by Sasaki and Todo [ST17]. Taking DDT (differential distribution table) of the S-box into consideration, we searched bit-wise impossible differential characteristics of Orthros that have one active bit for both of a plaintext and a ciphertext. The details of modeling S-boxes is in Appendix D.2.

As a result, we found 3/5/5-round impossible differential characteristics of Orthros, Branch 1 and Branch 2, respectively, as shown in Appendix D.3. Thus, we expect that the full-round Orthros is secure against impossible differential attacks.
4.3 Integral Attack

We present integral distinguishers on round-reduced Orthros. Since the division property [Tod15,TM16] was proposed, it has become an efficient tool to evaluate the resistance against integral attacks. Moreover, with the development of the MILP model for the bit-based division property [XZBL16], the attacker now only needs to focus on modeling the propagation of the division property.

The round function of Orthros consists of a nonlinear layer (S-box), a bit/nibble permutation layer, another linear layer (matrixMul) and the constant/key addition. To model the propagation of the division property through each component, we only need to consider the S-box and the binary matrix used in matrixMul. The bit/nibble permutation only has an influence on the coordinates of the variables used in the MILP model. The modelling of the S-box and binary matrix can be referred to Appendix D.4.

Based on our model, the longest integral distinguisher can reach up to at most 7 rounds with 127 active bits in the input. For example, when the most significant bit of the plaintext is constant and the remaining 127 bits take all possible $2^{127}$ values, for 7-round Orthros, all output bits are balanced.

Remark. Although there are several 7-round integral distinguishers, it is difficult to mount a key-recovery attack on 8 rounds of Orthros. This is different from usual key-recovery attacks on block ciphers, where the attacker is able to add several rounds after the integral distinguisher and guess partial key bits to decrypt the ciphertext. It is quite costly to guess the key bits and reverse the ciphertext for Orthros since the final output is the sum of the outputs of two branches, i.e., the attacker further needs to guess the output of the other branch.

4.4 Invariant Subspace Attack

Beierle et al. [BCLR17] showed that an invariant subspace attack can be mounted on a block cipher if one finds a non-trivial invariant for the substitution layer whose linear space is invariant under the linear layer matrix L that it uses and contains all the differences between the round keys. For block ciphers without a dedicated key schedule function, say when the $i$-th round key $rk_i = k \oplus rc_i$ is simply the xor of the master key and the $i$-th round constant, the difference of all round keys is the difference of the round constants. If $D$ denotes the set of all round constant differences, the authors of [BCLR17] computed $W_L(D)$, which denotes the smallest $L$-invariant subspace containing $D$. If the dimension of $W_L(D)$, $\dim(W_L(D))$, satisfies $\dim(W_L(D)) \geq n - 1$, where $n$ is the block size of the cipher in bits, then the authors showed that there is no non-trivial invariant of the substitution layer, provided that the S-box is well designed and does not have any linear component.

If this condition is not satisfied, one must further investigate the properties of the substitution layer. The authors then showed that, for every subspace $Z$ of the 0-linear space of the invariant of the substitution layer $S$, the invariant, $g$, takes the same value on each coset of $Z$ in $\mathbb{F}_2^n$ and also on each element of the set $S(Z)$. To show that $g$ is trivial, the authors computed the S-box layer at some points in $Z$ and hoped that all cosets would be hit when evaluating $S$ at a few points in $Z$. If $g$ takes the same value on all the corresponding cosets, we would conclude that $g$ must be a constant function and thus trivial. This can be done if we take $Z = W_L(D)$ and if $\dim(W_L(D))$ is close to $n$, since one would only need to hit $2^{n-\dim(W_L(D))}$ cosets.

Since our construction uses a key schedule function for both branches, we can not directly construct the set $D$ as the difference of round constants. From this fact, we use four different linear layers (two different in each branch) in our construction. However for any randomly chosen value of the secret key $k$ one can construct the sets $D_1, D_2, D_3, D_4$, one each for the difference of the round keys used in each of the four linear layers $L_1, L_2, L_3, L_4$. 
and then try to compute $\dim(W_L(D_i))$ for each $i$. We found that the linear matrices composed by a bit permutation and a matrix multiplication (used in the first to 4th rounds of each branch, call them $L_1$ (left), $L_3$ (right)) have extremely high multiplicative orders – around $2^{48}$ to $2^{60}$ – and thus it is not directly possible to find $W_L(D_i)$ for these matrices. Thus we limited ourselves to find $W_L(D_i) = \sum_{<\cdot, c> \leq 10000} L^i(c)$ for $L_1$, $L_2$ and $L_3$ (where $<\cdot, \cdot>$ denotes the subspace generated by the constituent vectors). We did an experiment with 1000 randomly chosen keys, and computed $W_L(D_i), W_L(D_i)'$. The dimension of these spaces is almost always more than 127 for $L_1, L_3$ and always more than 123 for $L_2, L_4$. Whenever the dimension of these spaces was less than 127, we tried to run Algorithm 1 of [BCLR17] to see if all cosets are hit when trying to evaluate the S-layer. For all choices of the random key, we find that all the cosets are always hit, and thus we conclude that it is highly unlikely that an invariant subspace attack can be mounted on our construction.

4.5 Other Attacks

We have evaluated the security against meet-in-the-middle [SA09], Yoyo [RBH17], exchange [BR19] and mixture-differential [Gra18] attacks on Orthros. The details are described in Appendix E. Moreover, we also reported the difficulty of key recovery from the statistical distinguishers for Orthros as well as a key-recovery attack framework for the initial design of Orthros in Appendix F.

5 Hardware Evaluation

Since our target construction is a low-latency PRF, the most useful hardware evaluation of the design is a fully unrolled circuit that optimizes the signal delay from the input to output ports. Such a circuit would be able to evaluate the PRF in one clock cycle itself, and naturally the clock frequency can be increased till the clock period is just above the total critical path of the circuit, affording a maximum throughput of $\frac{\text{blocksize}}{\text{critical path}}$ bits per second. To perform a fair evaluation we compare our construction with two other low-latency primitives that provide at least 128-bit block and 128-bit security. The first is Midori-128 and the second is QARMA_128-σ_0 (in [Ava17], QARMA with 9 forward and backward rounds was recommended for applications targeting 192 bit security). QARMA_128-σ_0 is a particular instantiation of QARMA with 9 forward and backward rounds and a low-delay S-box $\sigma_0$. For an added comparison, we also include the 64-bit block cipher PRINCE in our results. On the other hand we also benchmark some permutation based constructions that can be used as a PRF. For example the Kangaroo12-XOF [BDP18] which is based on the 12-round Keccak-f[1600] permutation can be used as a PRF: one could absorb the key and plaintext in the permutation state and extract 128 bits from the resulting XOF. Let us call this construction Kangaroo12-PRF[1600]. Since any design based on a 1600-bit state would naturally be hardware-intensive we also consider a lightweight version of the above construction Kangaroo12-PRF[400] based on the 12-round Keccak-f[400] permutation. We can also use the Subterranean-Deck function [DMMR20] to extract a 128 bit MAC from a 128 bit key and message. We also benchmark this design which we call Subterranean-PRF. It is even more lightweight as it has only a 257-bit state.

We found that across all libraries, Orthros even outperforms PRINCE (see Tables 9, 18, 19, 17) when it comes to the absolute signal delay between the input/output ports. We remark that PRINCE is a 64-bit block cipher.

For a fair evaluation we adhered to the following design flow for all the ciphers:

1. The RTL source codes for the circuit of the ciphers were first written in the Verilog HDL, and a functional simulation was done using the Modelsim software to ensure correctness.
2. The RTL codes were synthesized by the *Synopsys Design Vision* circuit compiler, with the compiler command set to `compile_ultra`. No other optimizations are done at this stage. For this process we used the standard cell libraries of the following CMOS logic processes: a) STM 90nm, b) TSMC 90nm, c) Nangate 45nm and d) Nangate 15nm.

3. A timing simulation was done on the synthesized netlist to confirm the correctness of the design, by comparing the output of the timing simulation with known test vectors.

4. The switching activity of each gate of the circuit was collected while running post-synthesis simulation. The average power was obtained using *Synopsys Power Compiler*, using the back annotated switching activity.

5. Step 2 outputs the critical path of the circuit. We repeat steps 2-4 (for each of the libraries) but this time by asking the circuit compiler to constrain the total signal delay between the input/output ports to some value less than the critical path computed in step 2.

6. We repeat the above processes, with progressively lower values of total signal delay, till such time as the circuit compiler is unable to construct a circuit with given delay. We stop the flow at this point. All results have been tabulated in Tables 9, 17, 18, 19 (for space constraints Tables 17, 18, 19 showing results for Nangate 15nm, TSMC 90nm and Nangate 45 nm processes are shifted to Appendix G).

**Why area increases with decrease in latency:** A cell library typically has several drive strengths of cells that implement a given logic function. These drive strengths correspond to the capacitive load that a cell can drive without excessive delay and with acceptable signal characteristics. Thus when we force the circuit compiler to construct a circuit with lower delay, it starts introducing higher drive strength gates, that typically occupy more area while offering the same functionality. For example, in the TSMC 90nm library 2-input xor gates of drive strength 1, 2, 4 occupy around 2.5, 3, 5 GE respectively. Thus it is natural for the area of a circuit to progressively increase as we constrain the circuit compiler to construct circuits of increasingly lower delay as shown in Tables 9, 17, 18, 19.

For a better illustrative purposes, we provide *Area vs Delay* (see Fig. 7). The plots tell us that not only does Orthros perform around 40% better across all standard cell libraries when it comes to the absolute delay value, it also outperforms QARMA₀-128-σ₀ and Midori-128 when it comes to achieving a) lower area figures and power consumption given a particular delay budget and b) lower or competitive delay given a particular area/power budget.

6 Conclusions

We have presented a new low-latency PRF of 128-bit block, dubbed Orthros. The design is essentially a sum of keyed permutations, which has been studied in the context of provable security. We found this design is suitable to a low-latency cryptographic primitive, which is intuitive, however, to our knowledge never seriously considered before. We made it real by thoroughly revising the current state-of-the-art low-latency, lightweight building blocks, together with an extensive security analysis and comprehensive hardware benchmarks.

For further directions, it would be interesting to extend our design, say having more branches (with even simpler round functions or fewer rounds), to even reduce latency. Software performance and related-key/side-channel security would also be interesting topics.
Table 9: Results for the STM 90nm library. Power measured at 10 MHz. "The core implementation of the underlying permutations in these constructions were taken from [BDH*,DMMR]."

| Cipher             | Area (µm²) | Power (mW) | Energy (µJ) | Latency (ns) | Max TP (Gbps) |
|--------------------|------------|------------|-------------|--------------|---------------|
| Orthros            | 98150.7    | 6.647      | 664.69      | 10.60        | 11.246        |
|                    | 99286.4    | 6.214      | 621.40      | 8.00         | 14.901        |
|                    | 108828.2   | 7.220      | 722.02      | 7.00         | 17.030        |
|                    | 123160.6   | 9.612      | 961.24      | 6.00         | 19.868        |
|                    | 133913.1   | 10.751     | 1075.07     | 5.00         | 23.842        |
|                    | 148432.8   | 12.614     | 1261.35     | 4.00         | 29.802        |
|                    | 177991.2   | 16.591     | 1659.12     | 3.00         | 39.736        |
|                    | 298855.6   | 27.628     | 2762.70     | 2.40         | 49.671        |
| Midori-128         | 85435.0    | 10.205     | 1020.49     | 18.54        | 6.430         |
|                    | 86470.0    | 9.671      | 967.14      | 16.00        | 7.415         |
|                    | 96225.5    | 11.275     | 1127.54     | 12.00        | 9.934         |
|                    | 107839.6   | 15.687     | 1568.7      | 10.00        | 11.921        |
|                    | 122584.4   | 17.887     | 1788.72     | 8.00         | 14.901        |
|                    | 144109.4   | 24.011     | 2401.09     | 5.99         | 19.868        |
|                    | 277060.7   | 46.464     | 4646.39     | 4.10         | 29.075        |
| QARMA_{128-σ₀}    | 104118.3   | 10.044     | 1004.38     | 19.41        | 6.142         |
|                    | 104686.9   | 9.810      | 981.04      | 17.00        | 7.012         |
|                    | 106848.1   | 9.911      | 991.05      | 15.00        | 7.947         |
|                    | 112157.2   | 11.571     | 1157.09     | 13.00        | 9.170         |
|                    | 128032.8   | 16.816     | 1681.62     | 11.00        | 10.837        |
|                    | 147874.2   | 20.438     | 2043.83     | 9.00         | 13.245        |
|                    | 182268.6   | 27.714     | 2771.37     | 6.96         | 17.128        |
|                    | 234453.9   | 41.508     | 4150.79     | 4.99         | 23.890        |
|                    | 319634.3   | 58.119     | 5811.87     | 4.38         | 27.217        |
| PRINCE             | 27897.7    | 1.867      | 186.73      | 11.35        | 10.503        |
|                    | 28773.6    | 1.791      | 179.13      | 9.00         | 13.245        |
|                    | 31905.0    | 2.057      | 205.67      | 7.00         | 17.030        |
|                    | 38941.7    | 3.274      | 327.44      | 5.00         | 23.842        |
|                    | 63795.8    | 6.023      | 602.25      | 3.00         | 39.736        |
|                    | 87285.5    | 8.151      | 815.12      | 2.56         | 46.566        |
| Kangaroo12-PRF[1600] * | 129809.7   | 59.533     | 5953.29     | 17.52        | 6.804         |
|                    | 55281.2    | 58.170     | 5817.01     | 12.00        | 9.934         |
|                    | 590922.6   | 57.350     | 5735.03     | 8.00         | 14.901        |
|                    | 1184290.6  | 1460.64    | 14606.38    | 3.99         | 29.877        |
| Kangaroo12-PRF[400] * | 133662.4   | 16.674     | 1667.36     | 18.47        | 6.454         |
|                    | 144690.0   | 14.388     | 1438.84     | 12.00        | 9.934         |
|                    | 167258.9   | 18.260     | 1825.96     | 8.00         | 28.725        |
|                    | 339274.6   | 39.311     | 3931.14     | 4.15         | 14.901        |
| Subterranean-PRF * | 130206.1   | 9.976      | 997.59      | 17.45        | 6.831         |
|                    | 139879.2   | 8.467      | 846.06      | 12.00        | 9.934         |
|                    | 177843.0   | 18.475     | 1847.46     | 8.00         | 14.901        |
|                    | 375592.1   | 40.271     | 4027.13     | 3.63         | 32.840        |
Orthros: A Low-Latency PRF

Figure 7: Delay vs Area comparisons.

Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions. Subhadeep Banik is supported by the Swiss National Science Foundation (SNSF) through the Ambizione Grant PZ00P2_179921. Takanori Isobe is supported by JST, PRESTO Grant Number JPMJPR2031, Grant-in-Aid for Scientific Research (B)(KAKENHI 19H02141) for Japan Society for the Promotion of Science, and Support Center for Advanced Telecommunications Technology Research (SCAT). Kosei Sakamoto is supported by Grant-in-Aid for JSPS Fellows (KAKENHI 20J25526) for Japan Society for the Promotion of Science.

References

[Ava17] Roberto Avanzi. The QARMA block cipher family. *IACR Trans. Symm. Cryptol.*, 2017(1):4–44, 2017. doi:10.13154/tosc.v2017.i1.4-44.
[BBD+99] Eli Biham, Alex Biryukov, Orr Dunkelman, Eran Richardson, and Adi Shamir. Initial observations on Skipjack: Cryptanalysis of Skipjack-3XOR (invited talk). In Stafford E. Tavares and Henk Meijer, editors, SAC 1998, volume 1556 of LNCS, pages 362–376. Springer, Heidelberg, August 1999. doi:10.1007/3-540-48892-8_27.

[BBI+15] Subhadeep Banik, Andrey Bogdanov, Takahiro Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A block cipher for low energy. In Tetsu Ikawa and Jung Hee Cheon, editors, AISTACRYPT 2015, Part II, volume 9453 of LNCS, pages 411–436. Springer, Heidelberg, November/December 2015. doi:10.1007/978-3-662-48800-3_17.

[BCG+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Kněžević, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombout, Soren S. Thomsen, and Tolga Yalçın. PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In Xiaoyun Wang and Kazue Sako, editors, ASIACRYPT 2012, volume 7658 of LNCS, pages 208–225. Springer, Heidelberg, December 2012. doi:10.1007/978-3-642-34961-4_14.

[BCLR17] Christof Beierle, Anne Canteaut, Gregor Leander, and Yann Rotella. Proving resistance against invariant attacks: How to choose the round constants. In Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II, pages 647–678, 2017. doi:10.1007/978-3-319-63715-0_22.

[BDH+] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Team keccak: Hardware resources. https://keccak.team/hardware.html.

[BDK+18] Achiya Bar-On, Orr Dunkelman, Nathan Keller, Eyal Ronen, and Adi Shamir. Improved key recovery attacks on reduced-round AES with practical data and memory complexities. In Hovav Shacham and Alexandra Boldyreva, editors, CRYPTO 2018, Part II, volume 10992 of LNCS, pages 185–212. Springer, Heidelberg, August 2018. doi:10.1007/978-3-319-96881-0_7.

[BDP+18] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, Ronny Van Keer, and Benoît Viguier. KangarooTwelve: Fast hashing based on keccak-p. In Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Lewen, Belgium, July 2-4, 2018, Proceedings, pages 400–418, 2018. doi:10.1007/978-3-319-93357-0_21.

[BGGS20] Zhenzhen Bao, Chun Guo, Jian Guo, and Ling Song. TNT: How to tweak a block cipher. In Vincent Rijmen and Yuval Ishai, editors, EUROCRYPT 2020, Part II, LNCS, pages 641–673. Springer, Heidelberg, May 2020. doi:10.1007/978-3-030-45724-2_22.

[BI99] M. Bellare and R. Impagliazzo. A tool for obtaining tighter security analyses of pseudorandom function based constructions, with applications to PRP to PRF conversion. Cryptology ePrint Archive, Report 1999/024, 1999. http://eprint.iacr.org/1999/024.

[Bir07] Alex Biryukov. The design of a stream cipher LEX. In Eli Biham and Amr M. Youssef, editors, SAC 2006, volume 4356 of LNCS, pages 67–75. Springer, Heidelberg, August 2007. doi:10.1007/978-3-540-74462-7_6.
Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In Matthew Robshaw and Jonathan Katz, editors, CRYPTO 2016, Part II, volume 9815 of LNCS, pages 123–153. Springer, Heidelberg, August 2016. doi:10.1007/978-3-662-53008-5_5.

Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, CHES 2007, volume 4727 of LNCS, pages 450–466. Springer, Heidelberg, September 2007. doi:10.1007/978-3-540-74735-2_31.

Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A small present - towards reaching the limit of lightweight encryption. In Wieland Fischer and Naofumi Homma, editors, CHES 2017, volume 10529 of LNCS, pages 321–345. Springer, Heidelberg, September 2017. doi:10.1007/978-3-319-66787-4_16.

Navid Ghaedi Bardeh and Sondre Rønjom. The exchange attack: How to distinguish six rounds of AES with $2^{88.2}$ chosen plaintexts. LNCS, pages 347–370. Springer, Heidelberg, December 2019. doi:10.1007/978-3-030-34618-8_12.

Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. In Alfred J. Menezes and Scott A. Vanstone, editors, CRYPTO’90, volume 537 of LNCS, pages 2–21. Springer, Heidelberg, August 1991. doi:10.1007/3-540-38424-3_1.

Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. http://eprint.iacr.org/2013/404.

Yu Long Chen, Eran Lambooij, and Bart Mennink. How to build pseudorandom functions from public random permutations. In Alexandra Boldyreva and Daniele Micciancio, editors, CRYPTO 2019, Part I, volume 11692 of LNCS, pages 266–293. Springer, Heidelberg, August 2019. doi:10.1007/978-3-030-26948-7_10.

Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. RIPEMD-160: A strengthened version of RIPEMD. In Dieter Gollmann, editor, FSE’96, volume 1039 of LNCS, pages 71–82. Springer, Heidelberg, February 1996. doi:10.1007/3-540-60865-6_44.

Christophe De Cannièرة, Orr Dunkelman, and Miroslav Kněžević. KATAN and KTANTAN - a family of small and efficient hardware-oriented block ciphers. In Christophe Clavier and Kris Gaj, editors, CHES 2009, volume 5747 of LNCS, pages 272–288. Springer, Heidelberg, September 2009. doi:10.1007/978-3-642-04138-9_20.

Wei Dai, Viet Tung Hoang, and Stefano Tessaro. Information-theoretic indistinguishability via the chi-squared method. In Jonathan Katz and Hovav Shacham, editors, CRYPTO 2017, Part III, volume 10403 of LNCS, pages 497–523. Springer, Heidelberg, August 2017. doi:10.1007/978-3-319-63697-9_17.
[DIS+18] Patrick Derbez, Tetsu Iwata, Ling Sun, Siwei Sun, Yosuke Todo, Haoyang Wang, and Meiqin Wang. Cryptanalysis of AES-PRF and its dual. *IACR Trans. Symm. Cryptol.*, 2018(2):161–191, 2018. doi:10.13154/tosc.v2018.i2.161-191.

[DKRS20] Orr Dunkelman, Nathan Keller, Eyal Ronen, and Adi Shamir. The retracing boomerang attack. In Vincent Rijmen and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, LNCS, pages 280–309. Springer, Heidelberg, May 2020. doi:10.1007/978-3-030-45721-1_11.

[DMMR] Joan Daemen, Pedro Maat Costa Massolino, Alireza Mehrdad, and Yann Rotella. The subterranean 2.0 cipher suite. https://cs.ru.nl/~joan/subterranean.html.

[DMMR20] Joan Daemen, Pedro Maat Costa Massolino, Alireza Mehrdad, and Yann Rotella. The subterranean 2.0 cipher suite. *IACR Trans. Symmetric Cryptol.*, 2020(S1):262–294, 2020. doi:10.13154/tosc.v2020.iS1.262-294.

[DR05a] Joan Daemen and Vincent Rijmen. A new MAC construction ALRED and a specific instance ALPHA-MAC. In Henri Gilbert and Helena Handschuh, editors, *FSE 2005*, volume 3557 of LNCS, pages 1–17. Springer, Heidelberg, February 2005. doi:10.1007/11502760_1.

[DR05b] Joan Daemen and Vincent Rijmen. The Pelican MAC function 2.0. Cryptology ePrint Archive, Report 2005/088, 2005. http://eprint.iacr.org/2005/088.

[Dwo05] Morris Dworkin. Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. Standard, National Institute of Standards and Technology., 2005.

[Dwo10] Morris Dworkin. Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices. Standard, National Institute of Standards and Technology., 2010.

[EM93] Shimon Even and Yishay Mansour. A construction of a cipher from a single pseudorandom permutation. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *ASIACRYPT’91*, volume 739 of LNCS, pages 210–224. Springer, Heidelberg, November 1993. doi:10.1007/3-540-57332-1_17.

[GKM+09] Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Sören S. Thomsen. Gostl - a SHA-3 candidate. In Helena Handschuh, Stefan Lucks, Bart Preneel, and Phillip Rogaway, editors, *Symmetric Cryptography, 11.01. - 16.01.2009*, volume 09031 of Dagstuhl Seminar Proceedings. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2009. URL: http://drops.dagstuhl.de/opus/volltexte/2009/1955/.

[GM16] Shay Gueron and Nicky Mouha. Simpira v2: A family of efficient permutations using the AES round function. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of LNCS, pages 95–125. Springer, Heidelberg, December 2016. doi:10.1007/978-3-662-53887-6_4.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES 2011*, volume 6917 of LNCS, pages 326–341. Springer, Heidelberg, September / October 2011. doi:10.1007/978-3-642-23951-9_22.
[Gra18] Lorenzo Grassi. Mixture differential cryptanalysis: a new approach to distinguishers and attacks on round-reduced AES. *IACR Trans. Symm. Cryptol.*, 2018(2):133–160, 2018. doi:10.13154/tosc.v2018.12.133-160.

[Gue16] Shay Gueron. A memory encryption engine suitable for general purpose processors. Cryptology ePrint Archive, Report 2016/204, 2016. http://eprint.iacr.org/2016/204.

[HJ06] W. Eric Hall and Charanjit S. Jutla. Parallelizable authentication trees. In Bart Preneel and Stafford Tavares, editors, *SAC 2005*, volume 3897 of *LNCS*, pages 95–109. Springer, Heidelberg, August 2006. doi:10.1007/11693383_7.

[HKR15] Viet Tung Hoang, Ted Krotetz, and Phillip Rogaway. Robust authenticated-encryption AEZ and the problem that it solves. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 15–44. Springer, Heidelberg, April 2015. doi:10.1007/978-3-662-46800-5_2.

[Iwa06] Tetsu Iwata. New blockcipher modes of operation with beyond the birthday bound security. In Matthew J. B. Robshaw, editor, *FSE 2006*, volume 4047 of *LNCS*, pages 310–327. Springer, Heidelberg, March 2006. doi:10.1007/11799313_20.

[KNR12] Miroslav Knežević, Ventzislav Nikov, and Peter Rombouts. Low-latency encryption - is “lightweight = light + wait”? In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 426–446. Springer, Heidelberg, September 2012. doi:10.1007/978-3-642-33027-8_25.

[LRW02] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 31–46. Springer, Heidelberg, August 2002. doi:10.1007/3-540-45708-9_3.

[Luc00] Stefan Lucks. The sum of PRPs is a secure PRF. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 470–484. Springer, Heidelberg, May 2000. doi:10.1007/3-540-45539-6_34.

[Mat94] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In Tor Helleseth, editor, *EUROCRYPT’93*, volume 765 of *LNCS*, pages 386–397. Springer, Heidelberg, May 1994. doi:10.1007/3-540-48285-7_33.

[MN17a] Bart Mennink and Samuel Neves. Encrypted Davies-Meyer and its dual: Towards optimal security using mirror theory. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 556–583. Springer, Heidelberg, August 2017. doi:10.1007/978-3-319-63697-9_19.

[MN17b] Bart Mennink and Samuel Neves. Optimal PRFs from blockcipher designs. *IACR Trans. Symm. Cryptol.*, 2017(3):228–252, 2017. doi:10.13154/tosc.v2017.i3.228-252.

[MWGP11] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In Chuankun Wu, Moti Yung, and Dongdai Lin, editors, *Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*, volume 7537 of *Lecture Notes in Computer Science*, pages 57–76. Springer, 2011. doi:10.1007/978-3-642-34704-7_5.
[NIS07] Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. Technical report, 2007. National Institute of Standards and Technology.

[Pat08] Jacques Patarin. A proof of security in O(2n) for the xor of two random permutations. In Reihanef Safavi-Naini, editor, ICITS 08, volume 5155 of LNCS, pages 232–248. Springer, Heidelberg, August 2008. doi:10.1007/978-3-540-85093-9_22.

[Qua] Qualcomm Technologies Inc. Pointer Authentication on ARMv8.3. https://www.qualcomm.com/media/documents/files/whitepaper-pointer-authentication-on-armv8-3.pdf.

[RBH17] Sondre Rønjom, Navid Ghaedi Bardeh, and Tor Helleseth. Yoyo tricks with AES. In Tsuyoshi Takagi and Thomas Peyrin, editors, ASIACRYPT 2017, Part I, volume 10624 of LNCS, pages 217–243. Springer, Heidelberg, December 2017. doi:10.1007/978-3-319-70694-8_8.

[SA09] Yu Sasaki and Kazumaro Aoki. Finding preimages in full MD5 faster than exhaustive search. In Antoine Joux, editor, EUROCRYPT 2009, volume 5479 of LNCS, pages 134–152. Springer, Heidelberg, April 2009. doi:10.1007/978-3-642-01001-9_8.

[SHW+14] Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. Automatic security evaluation and (related-key) differential characteristic search: Application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In Palash Sarkar and Tetsu Iwata, editors, ASIACRYPT 2014, Part I, volume 8873 of LNCS, pages 158–178. Springer, Heidelberg, December 2014. doi:10.1007/978-3-662-45611-8_9.

[SIH+11] Kyoji Shibutani, Takanori Isobe, Harumaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo: An ultra-lightweight blockcipher. In Bart Preneel and Tsuyoshi Takagi, editors, CHES 2011, volume 6917 of LNCS, pages 342–357. Springer, Heidelberg, September / October 2011. doi:10.1007/978-3-642-23951-9_23.

[ST17] Yu Sasaki and Yosuke Todo. New impossible differential search tool from design and cryptanalysis aspects - revealing structural properties of several ciphers. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, EUROCRYPT 2017, Part III, volume 10212 of LNCS, pages 185–215. Springer, Heidelberg, April / May 2017. doi:10.1007/978-3-662-56617-7_7.

[TM16] Yosuke Todo and Masakatu Morii. Bit-based division property and application to simon family. In Thomas Peyrin, editor, FSE 2016, volume 9783 of LNCS, pages 357–377. Springer, Heidelberg, March 2016. doi:10.1007/978-3-662-52993-5_18.

[Tod15] Yosuke Todo. Structural evaluation by generalized integral property. In Elisabeth Oswald and Marc Fischlin, editors, EUROCRYPT 2015, Part I, volume 9056 of LNCS, pages 287–314. Springer, Heidelberg, April 2015. doi:10.1007/978-3-662-46800-5_12.

[XZBL16] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, ASIACRYPT 2016, Part I, volume 10031 of LNCS, pages 648–678. Springer, Heidelberg, December 2016. doi:10.1007/978-3-662-53887-6_24.
A Round Constants

Table 10 shows the round constants of Branch1 and Branch2.

| Round Number $r$ | $RC_{1r}^1$ | $RC_{2r}^2$ |
|------------------|-------------|-------------|
| 1                | 0xa0ac9329ac4bc991c2313219c193ca81 | 0xa34a8ca0a88b04a1982b9381b2bacac8 |
| 2                | 0x4420cb8b49cc9b8a82c104ba4a22c918 | 0xca98490c308b9c0c99308bc988288c2a |
| 3                | 0x3c0b2031431044cc31401a4129a108b8 | 0x0403a2311bccb13a4ab39a8c42ba93924 |
| 4                | 0x33cc10a4043289941183323849c22304 | 0x48913c9c0c1808ca4894c19b399b1220 |
| 5                | 0xaa82c111b929aca040942408bba2814 | 0x32b3218430109ca4a31ca9123b8c838 |
| 6                | 0x2081380c9c290882aacb223114a49aa4 | 0x10bccc04a1b813b829c90b8bb149bb3 |
| 7                | 0x981c0cb2214048ab32c9a2309423a | 0x9a91c233a40c233b34a028990002b4093 |
| 8                | 0xb24119bc33c18b2938900c848a2b242b | 0x8a2931ab4113bc2bb9a13aabcb4b048b |
| 9                | 0x391a301a430822a1933241099c9b039 | 0x9b1b8bc39a3422048012a91a80a32 |
| 10               | 0x301248a0939b922c380333018a4cc0ba | 0xa4ac29b88283c913cb4492491aa100c |
| 11               | 0x44090490b14192a048b8a9b21b3c4 | 0xcab08904810c1043201a20c0acc09b1 |
| 12               | 0x92c81b00089982982a44102332909c20 | 0x4bb3b8984cb028c3839089a4cccccc1 |
The unique feature of our design is the use of two parallel branches (effectively block ciphers). In order to investigate the generic security of this design, we introduce two toy ciphers using single branch and double branches as shown in Fig. 8. We focus on the maximal differential probability (MDP) [BS91] and linear bias [Mat94] as they are two of the most fundamental security metrics. For the underlying branches, we consider both SPN and Feistel structures.

**Experiments for a SPN-based toy cipher.** First, we consider the case of SPN with 16-bit internal state and 16-bit round key. The basic design of this SPN-based toy cipher is similar to Midori. For our SPN-based toy cipher, the round function is composed of the following operations: S-box, Shuffle, Mix and AK. The state is organized as a $4 \times 4$ two-dimensional Boolean array $A$. The $(4j + i)$-th bit of the internal state is placed at $A[j][i] (0 \leq i < 3, 0 \leq j < 3)$. For the S-box, each column $A[i][j] (0 \leq j < 3)$ is viewed as a 4-bit value $x = 8 \cdot A[3][j] + 4 \cdot A[2][j] + 2 \cdot A[1][j] + A[0][j]$ and the internal state is updated as $A[i][j] = \text{S-box}(A[i][j])$. The S-box is the same with that used in Orthros. For the Shuffle operation, the state is reorganized as $A[i][j] = A[i'][j'] (0 \leq i < 3, 0 \leq j < 3)$, where $4j + i' = \text{SF}[4j + i]$ and the array SF is defined as

$$\text{SF}[16] = [0, 10, 5, 15, 14, 4, 11, 1, 9, 3, 12, 6, 7, 13, 2, 8].$$

For the Mix operation, each column is updated by multiplying the binary matrix $M_b$, i.e.,

$$(A[0][j], A[1][j], A[2][j], A[3][j])^T = M_b \cdot (A[0][j], A[1][j], A[2][j], A[3][j])^T (0 \leq j < 3).$$

For the AK operation, a random 16-bit round key will be XORed with the 16-bit internal state. Since the construction of the underlying block cipher used in Orthros is somewhat similar to that of Midori, we adopt the same Shuffle and Mix operations as in Midori for the toy cipher in order to construct a 16-bit toy cipher.

To compute MDP over a certain number of rounds for each construction, we first generate a random value for the round keys. Then, the whole block cipher can be viewed as a large 16-bit S-box. By exhausting all possible $2^{16} \times (2^{16} - 1)/2$ input pairs, we count the number of occurrences for each possible output difference and obtain the maximum frequency of the output difference, which we denote by $\text{CNT}_o$. In this way, MDP is calculated as $\text{CNT}_o/2^{16}$. With this method, we carried out 100 experiments and compute MDP for each experiments, and take the maximum for all experiments. The results are displayed in Table 11. Table 11 shows that the maximal value becomes stable after about 4 rounds in the construction using double branches. For the construction using a single branch, it becomes stable in about 5 rounds.

**Experiments for a Feistel-based toy cipher.** For the case of Feistel-based cipher, we consider 4-GFS (generalized Feistel structure with 4 sub-blocks), as shown in Fig. 9. The
Table 11: The maximal differential probability of SPN-based toy ciphers.

| Rounds | Max Pro. Single Branch | Max Pro. Double Branches |
|--------|------------------------|--------------------------|
| 1      | $2^{-3}$               | $2^{-1}$                 |
| 2      | $2^{-8}$               | $2^{-6}$                 |
| 3      | $2^{-8}$               | $2^{-6.6}$               |
| 4      | $2^{-11.3}$            | $2^{-12.3}$              |
| 5      | $2^{-12.5}$            | $2^{-12.5}$              |
| 6      | $2^{-12.5}$            | $2^{-12.5}$              |
| 7      | $2^{-12.5}$            | $2^{-12.5}$              |
| 8      | $2^{-12.5}$            | $2^{-12.5}$              |

The round function consists of two parallel 4-bit S-boxes with random round keys, where the S-box is the same with that used in Orthros. We carried out 100 experiments and compute the maximum of the MDP for all experiments, for both the constructions using a single branch and double branches. The corresponding results are displayed in Table 12. It shows that the maximal value becomes stable after about 7 rounds in the construction using double branches. For the construction using single branch, it becomes stable in about 10 rounds.

Figure 9: 4-GFS. Dotted lines denote (random) round keys.

Table 12: The maximal differential probability for each GFS-based construction.

| Rounds | Max Pro. Single Branch | Max Pro. Double Branches |
|--------|------------------------|--------------------------|
| 1      | $2^{-1}$               | $2^{-1}$                 |
| 2      | $2^{-3}$               | $2^{-1}$                 |
| 3      | $2^{-5}$               | $2^{-1}$                 |
| 4      | $2^{-7}$               | $2^{-8.4}$               |
| 5      | $2^{-7}$               | $2^{-7.3}$               |
| 6      | $2^{-9.6}$             | $2^{-11.1}$              |
| 7      | $2^{-9.4}$             | $2^{-12.5}$              |
| 8      | $2^{-11}$              | $2^{-12.5}$              |
| 9      | $2^{-12}$              | $2^{-12.5}$              |
| 10     | $2^{-12.5}$            | $2^{-12.5}$              |
| 11     | $2^{-12.5}$            | $2^{-12.5}$              |
| 12     | $2^{-12.5}$            | $2^{-12.5}$              |

Experiments on linear masks. Similar experiments have also been performed to evaluate the maximal linear bias for the toy ciphers. Due to the high time complexity to accurately
compute the maximal linear bias, we turn to calculating it in a probabilistic way. Specifically, we randomly choose some input and output masks and select the maximal linear bias from them. For the SPN-based toy cipher, it is found that the maximal linear bias \((2^{-0.4})\) becomes stable after 4 rounds if using a single branch, while it becomes stable in 3 rounds for double branches. For the GFS-based toy cipher, the maximal linear bias \((2^{-0.4})\) becomes stable in 9 rounds and 6 rounds for a single branch and double branches, respectively.

**Summary.** In our experiments, both the maximal differential probability and linear bias of the double branches reach a stable value in a smaller number of rounds than that of the single branch. Of course the scale of experiment is limited and a more theoretical support should be desired. However, these results suggest that the double branch enhances the security of the single branch. We also emphasize that the security of Orthros is never ensured based on such a simple simulation. Instead, a comprehensive study is performed.

### C Detailed Explanations for Conditions 1 and 2

#### C.1 Condition 1

Fig 10 shows the transition of the state after applying \(P_{bk1}\), which satisfies Condition 1. The state is represented as a 16 × 8 bit array, where \(S_0, S_1, S_2, S_3\) are nibbles consisting of the first state column. Let us focus on the leftmost column. After applying a bit-permutation satisfying Condition 1, for \(0 \leq i \leq 3\), Fig 10 shows that the 4 bits of \(S_i\) are mapped to different columns. The same applies to the remaining 7 columns.

\[
\begin{bmatrix}
80 & 81 & 82 & 83 & 84 & 85 & 86 & 87 \\
88 & 89 & 90 & 91 & 92 & 93 & 94 & 95 \\
96 & 97 & 98 & 99 & 100 & 101 & 102 & 103 \\
104 & 105 & 106 & 107 & 108 & 109 & 110 & 111 \\
112 & 113 & 114 & 115 & 116 & 117 & 118 & 119 \\
120 & 121 & 122 & 123 & 124 & 125 & 126 & 127 \\
\end{bmatrix}
\]

\[P_{bk1} = \begin{bmatrix}
\begin{bmatrix}
80 & 81 & 82 & 83 & 84 & 85 & 86 & 87 \\
88 & 89 & 90 & 91 & 92 & 93 & 94 & 95 \\
96 & 97 & 98 & 99 & 100 & 101 & 102 & 103 \\
104 & 105 & 106 & 107 & 108 & 109 & 110 & 111 \\
112 & 113 & 114 & 115 & 116 & 117 & 118 & 119 \\
120 & 121 & 122 & 123 & 124 & 125 & 126 & 127 \\
\end{bmatrix}
\end{bmatrix}
\]

\[S_0 = (80, 81, 82, 83)^T \quad S_1 = (84, 85, 86, 87)^T \quad S_2 = (88, 89, 91, 811)^T \quad S_3 = (812, 813, 814, 815)^T\]

**Figure 10:** Transition of a state after applying \(P_{bk1}\).

#### C.2 Proof of Condition 2

For each active bit in the nibble cell \(S_i\) \((0 \leq i \leq 31)\) in the input, after applying the S-box and the bit permutation satisfying Condition 1, there will exist 4 different groups of inputs to the binary matrix, each of which will contain exactly one active nibble, as shown in Fig. 11. Therefore, there will be 12 active nibbles after MatrixMul in the first round, which will activate 12 nibbles located in the same positions in the second round after S-box operation, as depicted in Fig. 12. Therefore, there are 12 nibbles (48 bits in
total) in the state after the S-box (with full-diffusion property) operation in the second round, independent of the value of the one active bit in the input to the first round.

After applying a permutation satisfying Condition 2 for these 48 bits, in each column of the $4 \times 8$ array, there exist at least 2 nibble cells containing the bits coming from these 48 bits, as shown in Fig. 13 for bit level and in Fig. 14 for nibble level. In other words, after applying the bit permutation satisfying Condition 2 in the second round, in each column of the nibble array, there are at least 2 nibbles dependent of the one active bit in the input to the first round. When the $\text{MatrixMul}$ operation is further applied to each column of the $4 \times 8$ nibble array, the values of all the four nibbles in each column will therefore dependent of the one active bit in the input to the first round. However, it cannot be guaranteed that the value of each bit will be dependent of the one active bit. Thus, after further applying the S-box with a full-diffusion property in the third round, all 128 bits become dependent of the one active bit. This means that the full diffusion is achieved by 2.5 rounds.

An example to explain the 2.5-round diffusion can be referred to Fig. 11, 12, 13, and Fig. 14.

$$S_0 = (s_0, s_1, s_2, s_3)^T$$

Figure 11: The 4 active bits after the bit permutation in the first round, as marked in red.

$$\begin{bmatrix}
S_0 & S_1 & S_2 & S_3 & S_4 & S_5 & S_6 & S_7 & S_8 & S_9 & S_{10} & S_{11} & S_{12} & S_{13} & S_{14} & S_{15} & S_{16} & S_{17} & S_{18} & S_{19} & S_{20} & S_{21} & S_{22} & S_{23} & S_{24} & S_{25} & S_{26} & S_{27} & S_{28} & S_{29} & S_{30} & S_{31} & S_{32}
\end{bmatrix}$$

Figure 12: The 12 active nibbles after $\text{MatrixMul}$ in the first round, marked in red.

## D Details of Security Evaluation

### D.1 DDT of S-box

The aforementioned DDT table of our S-box is shown shown in Table 13, where $d_{in}$ and $d_{out}$ denote the input and output difference of 4-bit S-box, respectively.
Figure 13: The 48 active bits after bit permutation in the second round, marked in red.

![Figure 13](image)

Figure 14: The active nibbles after bit permutation in the second round, marked in red.

![Figure 14](image)

Table 13: The differential distribution table of S-box.

| $d_{in}$ | $d_{out}$ | 0x0 | 0x1 | 0x2 | 0x3 | 0x4 | 0x5 | 0x6 | 0x7 | 0x8 | 0x9 | 0xA | 0xB | 0xC | 0xD | 0xE | 0xF |
|----------|----------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 0x0      |          | 16  | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| 0x1      |          | 0   | 2   | 2   | 4   | 0   | 2   | 2   | 0   | 0   | 4   | 0   | 0   | 0   | 0   | 0   | 0   |
| 0x2      |          | 0   | 0   | 2   | 2   | 4   | 4   | 0   | 2   | 2   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| 0x3      |          | 0   | 2   | 2   | 0   | 0   | 2   | 0   | 0   | 2   | 2   | 0   | 0   | 4   | 0   | 2   | 2   |
| 0x4      |          | 0   | 0   | 2   | 0   | 2   | 0   | 2   | 0   | 0   | 2   | 2   | 2   | 0   | 0   | 0   | 0   |
| 0x5      |          | 0   | 0   | 2   | 2   | 0   | 4   | 0   | 0   | 4   | 0   | 0   | 0   | 0   | 0   | 2   | 2   |
| 0x6      |          | 0   | 2   | 2   | 0   | 2   | 0   | 2   | 0   | 0   | 2   | 2   | 2   | 0   | 2   | 2   | 2   |
| 0x7      |          | 0   | 2   | 0   | 0   | 0   | 2   | 0   | 0   | 2   | 2   | 0   | 4   | 2   | 0   | 0   | 0   |
| 0x8      |          | 0   | 0   | 2   | 0   | 0   | 2   | 2   | 2   | 2   | 2   | 0   | 4   | 0   | 0   | 0   | 0   |
| 0x9      |          | 0   | 2   | 0   | 0   | 2   | 0   | 0   | 2   | 2   | 2   | 0   | 4   | 0   | 0   | 0   | 0   |
| 0xA      |          | 0   | 0   | 2   | 0   | 0   | 2   | 0   | 0   | 2   | 2   | 2   | 0   | 2   | 0   | 2   | 2   |
| 0xB      |          | 0   | 2   | 2   | 0   | 2   | 0   | 2   | 2   | 2   | 2   | 0   | 4   | 2   | 0   | 0   | 0   |
| 0xC      |          | 0   | 2   | 2   | 0   | 2   | 0   | 2   | 2   | 2   | 2   | 0   | 4   | 2   | 0   | 0   | 0   |
| 0xD      |          | 0   | 2   | 2   | 0   | 2   | 0   | 2   | 2   | 2   | 2   | 0   | 4   | 2   | 0   | 0   | 0   |
| 0xE      |          | 0   | 2   | 2   | 0   | 2   | 0   | 2   | 2   | 2   | 2   | 0   | 4   | 2   | 0   | 0   | 0   |
| 0xF      |          | 0   | 2   | 2   | 0   | 2   | 0   | 2   | 2   | 2   | 2   | 0   | 4   | 2   | 0   | 0   | 0   |
D.2 Modeling S-box

Based on the method of [SHW+14], we derive following 37 linear inequalities from Table 13, where $d_{in} = (d_{in0}, d_{in1}, d_{in2}, d_{in3})$ and $d_{out} = (d_{out0}, d_{out1}, d_{out2}, d_{out3})$.

\[
\begin{align*}
-d_{in0} & - d_{in1} + 2d_{in2} - d_{in3} - 2d_{out0} + 2d_{out1} - d_{out2} + d_{out3} + 3 \quad \geq 0 \\
d_{in0} + d_{in1} - d_{in2} + d_{in3} + d_{out0} + d_{out1} + d_{out2} \quad \geq 0 \\
d_{in0} + 2d_{in2} + d_{in3} + d_{out0} - d_{out1} - d_{out2} - 2d_{out3} + 2 \quad \geq 0 \\
2d_{in0} - d_{in1} + d_{in2} + 2d_{in3} + d_{out0} - d_{out1} + d_{out2} - 2d_{out3} + 2 \quad \geq 0 \\
d_{in0} - 2d_{in1} - 2d_{in2} - d_{in3} + 2d_{out0} - 2d_{out1} + 2d_{out2} - d_{out3} + 6 \quad \geq 0 \\
-d_{in0} + 2d_{in1} - 2d_{in2} - d_{in3} + 2d_{out0} + d_{out1} - d_{out2} - 2d_{out3} + 5 \quad \geq 0 \\
d_{in0} + d_{in1} + 2d_{in2} + 2d_{in3} - d_{out1} - d_{out3} \quad \geq 0 \\
d_{in0} + 2d_{in1} + d_{in2} + 2d_{in3} - d_{out0} - d_{out1} - d_{out2} - d_{out3} + 1 \quad \geq 0 \\
d_{in0} + d_{in1} - d_{in2} + d_{out0} - d_{out1} - d_{out2} + d_{out3} + 2 \quad \geq 0 \\
d_{in1} + d_{in2} + 2d_{in3} - d_{out1} - d_{out2} + d_{out3} + 1 \quad \geq 0 \\
2d_{in0} + d_{in2} + d_{in3} - 2d_{out0} + 2d_{out1} - 2d_{out2} + 2d_{out3} + 2 \quad \geq 0 \\
-d_{in0} - d_{in2} - d_{in3} - 2d_{out0} - d_{out1} + d_{out2} - d_{out3} + 5 \quad \geq 0 \\
d_{in0} + 2d_{in1} + 2d_{in3} + 2d_{out0} - 2d_{out1} - d_{out2} + 2d_{out3} + 2 \quad \geq 0 \\
d_{in0} + d_{in1} + d_{in2} + d_{out0} + d_{out1} - d_{out2} + d_{out3} + 1 \quad \geq 0 \\
d_{in0} + d_{in1} + d_{in2} - d_{out0} - d_{out1} + d_{out2} - d_{out3} + 2 \quad \geq 0 \\
-d_{in0} + d_{in1} + d_{in3} + d_{out0} - d_{out1} - d_{out2} + 2 \quad \geq 0 \\
d_{in0} + d_{in1} + d_{in2} + d_{out0} - d_{out1} - d_{out2} - d_{out3} + 4 \quad \geq 0 \\
d_{in0} - d_{in2} + d_{in3} + d_{out0} - d_{out1} + d_{out3} + 2 \quad \geq 0 \\
d_{in0} - d_{in1} - d_{in2} + d_{out2} - d_{out3} + 3 \quad \geq 0 \\
-d_{in0} - 2d_{in3} - d_{out0} - 2d_{out1} - 2d_{out2} - d_{out3} + 5 \quad \geq 0 \\
-3d_{in0} - 2d_{in2} - 3d_{in3} + 4d_{out0} + 3d_{out1} + d_{out2} + 2d_{out3} + 4 \quad \geq 0 \\
2d_{in0} + d_{in1} + d_{in2} - 3d_{in3} + 2d_{out0} + d_{out1} + 4d_{out2} + 3d_{out3} \quad \geq 0 \\
-d_{in0} - 2d_{in1} - 2d_{in2} - 2d_{out0} + 2d_{out1} + d_{out2} - 1d_{out3} + 5 \quad \geq 0 \\
-3d_{in0} + d_{in1} + d_{in2} - 2d_{in3} + 2d_{out0} - 2d_{out1} + 3d_{out2} - 1d_{out3} + 5 \quad \geq 0 \\
-2d_{in0} - 2d_{in1} + d_{in2} + d_{in3} + 3d_{out0} + 2d_{out1} + 4d_{out2} + 4d_{out3} \quad \geq 0 \\
-d_{in0} - d_{in1} - 2d_{in2} - 3d_{in3} + d_{out0} - d_{out1} + 2d_{out2} + 3d_{out3} + 2 \quad \geq 0 \\
-d_{in0} - d_{in1} - 2d_{in2} + d_{in3} - 2d_{out0} - d_{out1} + 2d_{out2} + 2d_{out3} + 4 \quad \geq 0 \\
-2d_{in0} - 2d_{in1} + d_{in2} + d_{in3} + 2d_{out0} - 1d_{out1} + d_{out2} + 2d_{out3} + 3 \quad \geq 0 \\
-d_{in0} - 2d_{in1} - d_{in2} + d_{in3} - d_{out0} + 2d_{out1} + d_{out2} - 2d_{out3} + 5 \quad \geq 0 \\
-d_{in0} - 2d_{in2} - d_{out0} + d_{out1} + d_{out2} - d_{out3} + 2 \quad \geq 0 \\
-d_{in0} - d_{in1} - d_{in2} + d_{out0} - d_{out1} - d_{out2} + d_{out3} + 4 \quad \geq 0 \\
-3d_{in0} - 4d_{in1} - 2d_{in2} - 2d_{in3} + d_{out0} + 3d_{out1} - d_{out2} + 4d_{out3} + 8 \quad \geq 0 \\
-d_{in0} - d_{in2} - d_{in3} - d_{out0} - d_{out1} - d_{out2} - d_{out3} + 6 \quad \geq 0 \\
-d_{in0} + d_{in1} - d_{in3} - d_{out0} - d_{out1} + d_{out2} + 4 \quad \geq 0 \\
d_{in0} - d_{in1} - d_{in2} - d_{in3} - d_{out0} - d_{out1} - d_{out2} + 5 \quad \geq 0 \\
-d_{in0} + d_{in2} - d_{in3} - d_{out0} - d_{out1} - d_{out2} + d_{out3} + 4 \quad \geq 0
\end{align*}
\]
D.3 Impossible Differential Characteristics of Orthros/Branch1/Branch2

3/5/5-round impossible differential characteristics of Orthros/Branch1/Branch2 are shown as follows.

Orthros:

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

Branch1:

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

Branch2:

\[
\begin{array}{cccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

D.4 Modeling for Division Property

Modeling S-box. Similar to Section 4.2, one could build a table to describe the propagation of the division property through S-box, as shown in Table 14. In this table, \( u \) and \( v \) denote the input and output division property of S-box, respectively. The entry at \((u, v)\) is \(*\) when the propagation \(u \rightarrow v\) is possible. Otherwise, the propagation is impossible. Based on the method proposed by [SHW+14], such a table is equivalent to the linear inequalities as shown below, where \( u = (u_0, u_1, u_2, u_3) \) and \( v = (v_0, v_1, v_2, v_3) \).
Table 14: The propagation of the division property for the S-box.

| w | 0x0 | 0x1 | 0x2 | 0x4 | 0x8 | 0x3 | 0x5 | 0x9 | 0x6 | 0x0a | 0x0c | 0x0d | 0x0e | 0x0f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x0 | * | * | * | * | * | * | * | * | * | * | * | * | * | * |
| 0x1 | * | * | * | * | * | * | * | * | * | * | * | * | * | * |
| 0x2 | * | * | * | * | * | * | * | * | * | * | * | * | * | * |
| 0x4 | * | * | * | * | * | * | * | * | * | * | * | * | * | * |
| 0x5 | * | * | * | * | * | * | * | * | * | * | * | * | * | * |
| 0x3 | * | * | * | * | * | * | * | * | * | * | * | * | * | * |
| 0x6 | * | * | * | * | * | * | * | * | * | * | * | * | * | * |
| 0x0a | * | * | * | * | * | * | * | * | * | * | * | * | * | * |
| 0x0c | * | * | * | * | * | * | * | * | * | * | * | * | * | * |
| 0x0d | * | * | * | * | * | * | * | * | * | * | * | * | * | * |
| 0x0e | * | * | * | * | * | * | * | * | * | * | * | * | * | * |
| 0x0f | * | * | * | * | * | * | * | * | * | * | * | * | * | * |

\[
\begin{align*}
-u_0 - u_1 + 2v_0 + 2v_1 + 2v_2 + v_3 & \quad \geq 0 \\
-u_0 - u_2 + v_0 + v_1 + v_3 + 1 & \quad A \geq 0 \\
-u_2 - u_3 + v_0 + 2v_1 + 2v_2 + 2v_3 & \quad A \geq 0 \\
-u_0 - u_1 - u_2 + 3v_0 + 2v_1 + 3v_2 + 2v_3 & \quad A \geq 0 \\
-u_0 - u_3 + 2v_0 + v_1 + v_2 + 2v_3 & \quad A \geq 0 \\
-u_0 - u_1 - u_2 + v_0 + v_1 + 2 & \quad A \geq 0 \\
-4u_0 - 3u_1 - 4u_2 - 4u_3 + v_0 + v_1 + v_2 + 2v_3 + 10 & \quad A \geq 0 
\end{align*}
\]

Modeling the Binary Matrix. For the matrixMul operation, it can also be viewed that the binary matrix $M_b$ works on four bits independently. Let $B = M_b \cdot A$, where $A, B \in \mathbb{F}_2^4$. Therefore, we could pre-compute a table to describe the mapping from $A$ to $B$, as specified in Table 15.

Table 15: The mapping of the binary matrix.

| A  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| B  | 0 | 14 | 13 | 3 | 11 | 5 | 6 | 8 | 7 | 9 | 10 | 4 | 12 | 2 | 1 | 15 |

To model the propagation of the division property through the binary matrix, similar to the way we model $S$-box, we first build a table to describe the propagation rule (Table 16), where $w$ and $z$ denote the input and output division property. The entropy $(w, z)$ is marked as * when the propagation $w \rightarrow z$ is possible. Otherwise, the propagation is impossible. By using the method of [SHW+14], Table 16 can be expressed by the linear inequalities as displayed below, where $w = (w_0, w_1, w_2, w_3)$ and $z = (z_0, z_1, z_2, z_3)$.
Table 16: The propagation of the division property for the binary matrix.

| $w$   | 0x0 | 0x1 | 0x2 | 0x3 | 0x4 | 0x5 | 0x6 | 0x7 | 0x8 | 0x9 | 0xa | 0xb | 0xc | 0xd | 0xe | 0xf |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 0x0   | x   | x   | x   | x   | x   | x   | x   | x   | x   | x   | x   | x   | x   | x   | x   | x   |
| 0x1   |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x2   | x   |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x3   |     | x   |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x4   |     |     | x   |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x5   |     |     |     | x   |     |     |     |     |     |     |     |     |     |     |     |     |
| 0x6   |     |     |     |     | x   |     |     |     |     |     |     |     |     |     |     |     |
| 0x7   |     |     |     |     |     | x   |     |     |     |     |     |     |     |     |     |     |
| 0x8   |     |     |     |     |     |     | x   |     |     |     |     |     |     |     |     |     |
| 0x9   |     |     |     |     |     |     |     | x   |     |     |     |     |     |     |     |     |
| 0xa   |     |     |     |     |     |     |     |     | x   |     |     |     |     |     |     |     |
| 0xb   |     |     |     |     |     |     |     |     |     | x   |     |     |     |     |     |     |
| 0xc   |     |     |     |     |     |     |     |     |     |     | x   |     |     |     |     |     |
| 0xd   |     |     |     |     |     |     |     |     |     |     |     | x   |     |     |     |     |
| 0xe   |     |     |     |     |     |     |     |     |     |     |     |     | x   |     |     |     |
| 0xf   |     |     |     |     |     |     |     |     |     |     |     |     |     | x   |     |     |

\[
\begin{align*}
-w_0 - w_1 - w_2 + z_3 & \geq 0 \\
-w_0 - w_1 - w_3 + z_2 & \geq 0 \\
-w_0 - w_2 - w_3 + z_0 + z_1 + z_2 + z_3 & \geq 0 \\
-w_0 - w_2 - w_3 + z_1 + 2 & \geq 0 \\
-w_1 - w_2 - w_3 + z_0 + 2 & \geq 0 \\
-w_0 - w_2 + z_0 + z_2 + 1 & \geq 0 \\
-w_0 - w_1 + z_0 + z_1 + 1 & \geq 0 \\
-w_1 - w_2 - z_1 + z_2 + 1 & \geq 0 \\
-w_1 - w_3 + z_1 + z_3 + 1 & \geq 0 \\
-w_2 + z_0 + z_1 + z_3 & \geq 0 \\
-w_1 + z_0 + z_1 + z_2 & \geq 0 \\
-w_1 + z_0 + z_2 + z_3 & \geq 0 \\
-w_0 - w_3 + z_0 + z_3 + 1 & \geq 0 \\
-w_0 + z_1 + z_2 + z_3 & \geq 0 \\
-w_0 - w_3 + z_0 + z_2 + z_3 & \geq 0 \\

\end{align*}
\]

E Additional Security Evaluation

E.1 Meet-in-the-Middle Attack

To mount a meet-in-the-middle attack, the adversary has to compute the inverse of the cipher, i.e., computing intermediate states (matching states) from the corresponding ciphertexts by guessing the involved round keys. As discussed above, the adversary needs to guess one of two 128-bit outputs of branches to go through the final XOR operation in the backward computation. As it requires at least $2^{128}$ iterations, the attack is not efficient than the brute force attack.

Another possibility is to use the splice-and-cut technique [SA09]. With this technique, the adversary guesses the intermediate values as the start point from which the adversary starts the computation toward both directions. However, once she guesses a 128-bit intermediate of one branch, she has to correctly guess the corresponding 128-bit intermediate of the other branch. Therefore, we believe that Orthros is secure against meet-in-the middle
attacks.

E.2 Yoyo and Mixture-Differential Attacks

The yoyo attack was first introduced by Biham [BBD99]. Recently, it has been applied to the cryptanalysis of AES by Ronjom et al. [RBH17], where generic attacks on up to 3 rounds of SPNs have been discussed. Since 2-round AES can be viewed as 1-round SPN with the concept of super S-box, distinguishing attack on up to 6 rounds of AES are derived by [RBH17]. However, there is one major step in the yoyo attack, that is, the attacker needs to make a decryption query. For the design of Orthros, since the final output is the xor sum of the outputs of Branch1 and Branch2, it is infeasible to make a decryption query. In addition, due to the fast diffusion of the bit permutation and the fact that each branch adopts a different bit permutation, it is quite difficult to construct an efficient super S-box for Orthros in the first four rounds. Based on these reasons, we believe that Orthros is resistant against such an attack.

In Asiacrypt 2019, a different view of the yoyo attack on AES, called exchange attack, was proposed by Bardeh and Ronjom [BR19]. It does not require decryption queries and can reach up to 6 rounds of AES. However, due to the similarity in the underlying idea between the yoyo attack and exchange attack, we believe that the resistance against the yoyo attack implies the resistance against exchange attack.

The mixture differential introduced by Grassi [Gra18] is an efficient tool to analyze a reduced-round AES, as its contribution to the recent progress of key-recovery attacks on 5-round AES [BDK18,DKRS20]. An important factor which makes the mixture differential efficient is that AES adopts a word-wise permutation. Due to the effect of the bit permutation in the first four rounds, we are not able to find a useful mixture differential for Orthros.

F Difficulty of Key-Recovery Attacks

As we repeated several times, the unique feature of Orthros (as a cryptographic primitive) is that it takes the sum of two branch outputs. To mount a key-recovery attack with a statistical distinguisher for block ciphers, such as a differential/linear/integral distinguisher, it is common to append a few rounds after a certain number of rounds for which a distinguisher exists, and guess partial key bits by partially decrypting the ciphertext and verifying whether the distinguishing property holds. However, such a common strategy is quite hard for Orthros since the attacker even needs to guess the outputs of each branch in order to reverse the ciphertext. In addition, it is required to construct two distinguishers for two different branches with the same plaintext set simultaneously if the attacker wants to append a few rounds after the distinguishable rounds. Even if it is possible to construct a distinguisher for one block cipher with an advanced cryptanalysis method as discussed above, it would be challenging to construct two different distinguishers for two different block ciphers for the same plaintext set simultaneously. In such a situation, we think generally the most promising direction is to find integral distinguishers. This will be discussed later.

Another attacking strategy is to prepend some rounds before the distinguishable rounds. However, this implies that there exists a distinguisher for each branch simultaneously. When extending two distinguishers backwards to the plaintext, the attacker can derive which key bits should be guessed in order to compute the desired value of the intermediate internal state of both branches. Since each branch adopts a different linear layer in its round function, and the key schedules of two branches also differ, a lot of to-be-guessed key bits will be involved. Moreover, the whitening keys in two branches are different as well, which further increases the complexity to prepend some rounds before a distinguisher.
For better understanding, we present a framework to recover the secret key by extending an integral distinguisher backwards.

**A Framework for Recovering the Secret Key.** This framework was once applied to a preliminary version of Orthros. Therefore, we omit the details of the design and explain a high-level idea. First, we denote the states after the S-box layer in the first round of Branch1 and Branch2 by $XL_{0.5}$ and $XR_{0.5}$, respectively. Suppose there is a set of bit positions denoted by $PSet \subseteq \{i | 0 \leq i \leq 127\}$ and the size of $PSet$ is $PSize$. In addition, let us denote the final output after $r$ rounds of (an old version of) Orthros by $C^r$, which is the sum of the outputs of (old versions of) Branch1 and Branch2. Suppose there is an integral distinguisher such that

$$\sum_{XL_{0.5} \in PS} \text{Branch1}(XL_{0.5}) \oplus \sum_{XR_{0.5} \in PS'} \text{Branch2}(XR_{0.5}) = 0,$$

where $PS, PS' \in \{e \in \mathbb{F}_{2}^{128} | e[i] \in \{0, 1\}, i \in PSet\}$, i.e., the set of values whose bits located at the positions belonging to $PSet$ take all the possible $2^{PSize}$ values and the remaining $(128 - PSize)$ bits take constant values.

To mount a key recovery attack, the attacker first derives from $PSet$ the active bits in the plaintext. Specifically, if $i \in PSet$, the $(4 \times i/4)$-th, $(4 \times i/4 + 1)$-th, $(4 \times i/4 + 2)$-th and $(4 \times i/4 + 3)$-th bits of the plaintext are all active. Let $ASize$ be the size of the active bits in the plaintext. The attacker then prepares a plaintext set whose active bits take all possible values and make encryption queries with the $r$-round Orthros. It is easily detected that the sum of the ciphertexts is zero. Record the corresponding $2^{ASize}$ pairs of plaintext and ciphertext in a table.

Suppose the whitening keys used by Branch1 and Branch2 are the same. In this case, the attacker guesses $2^{ASize}$ different values of the whitening key which is xored with the active bits in the input. For each guess, the attacker can partially know the corresponding $XL_{0.5}$ and $XR_{0.5}$ and can divide the plaintext set into $2^{ASize - PSize}$ different subsets according to the value of the nonactive bits of $XL_{0.5}$ and $XR_{0.5}$ via the constructed integral distinguisher. For each subset, compute the sum of the corresponding ciphertexts. For the correct key, the sum will be zero for all subsets. However, for a wrong key, the sum is zero for a subset with a probability $2^{-128}$. Therefore, the attacker can recover the key bits by checking the sum of the ciphertexts for the plaintexts in each subset.

Consider the case when different whitening keys are used for Branch1 and Branch2. In this case, when the attacker guesses the key bits in the left branch to obtain the corresponding $2^{ASize - PSize}$ subsets of the plaintexts, the sum of the ciphertexts for the plaintexts in each subset is not clear even the guess is correct. This is because the set of $XR_{0.5}$ behaves randomly for each subset of plaintext obtained according to the guess of the whitening key used in Branch1.

Obviously, it can be interpreted that this framework for Orthros is to convert a $r$-round distinguisher into a $r$-round key-recovery attack. Therefore, a long distinguisher should be prevented in our design.

**G Additional Hardware Results**
Table 17: Results for the Nangate 15nm library. Power measured at 10 MHz.

| Cipher            | Area ($\mu$m²) | Power (mW) | Energy (pJ) | Latency (ps) | Max TP (Gbps) |
|-------------------|----------------|------------|-------------|--------------|---------------|
| Orthros           | 5766.1         | 2.332      | 233.18      | 485.72       | 245.428       |
|                   | 5792.8         | 2.187      | 218.74      | 450.00       | 264.910       |
|                   | 6013.5         | 2.038      | 203.85      | 400.00       | 298.023       |
|                   | 7439.2         | 2.637      | 263.73      | 351.55       | 339.096       |
| Midori-128        | 5102.3         | 3.354      | 335.36      | 850.55       | 140.156       |
|                   | 5116.7         | 3.251      | 325.12      | 800.00       | 149.012       |
|                   | 5156.9         | 3.132      | 313.15      | 750.00       | 158.946       |
|                   | 5298.5         | 3.138      | 313.80      | 700.00       | 170.299       |
|                   | 5731.8         | 3.305      | 330.49      | 650.00       | 183.299       |
|                   | 6076.4         | 4.466      | 446.60      | 603.78       | 197.438       |
| QARMA$_{128-\sigma_0}$ | 6263.3     | 4.160      | 415.98      | 908.09       | 131.275       |
|                   | 6304.8         | 3.810      | 380.95      | 800.00       | 149.012       |
|                   | 7085.8         | 3.861      | 386.12      | 700.00       | 170.299       |
|                   | 8869.9         | 5.611      | 561.05      | 640.00       | 186.265       |
| PRINCE            | 1664.3         | 0.671      | 67.14       | 536.37       | 222.252       |
|                   | 1671.5         | 0.657      | 65.71       | 500.00       | 238.419       |
|                   | 1698.5         | 0.604      | 60.36       | 450.00       | 264.910       |
|                   | 1889.6         | 0.599      | 59.98       | 400.00       | 298.023       |
|                   | 2337.9         | 0.894      | 89.43       | 371.62       | 320.783       |
| Kangaroo12-PRF[1600] | 20957.8   | 13.470     | 1347.04     | 722.32       | 166.036       |
|                   | 27298.5        | 12.223     | 1222.25     | 650.00       | 183.400       |
|                   | 28803.4        | 12.884     | 1288.39     | 600.00       | 198.682       |
|                   | 31052.8        | 13.096     | 1309.60     | 576.65       | 206.727       |
| Kangaroo12-PRF[400] | 7332.0   | 3.812      | 381.22      | 765.95       | 155.636       |
|                   | 7407.3         | 3.711      | 371.98      | 700.00       | 170.300       |
|                   | 7584.1         | 3.501      | 350.11      | 650.00       | 183.400       |
|                   | 8278.4         | 5.571      | 557.07      | 602.81       | 197.756       |
| Subterranean-PRF  | 7582.4         | 3.548      | 354.78      | 692.84       | 172.659       |
|                   | 7586.4         | 3.385      | 338.53      | 650.00       | 183.400       |
|                   | 7661.4         | 3.122      | 312.22      | 600.00       | 198.682       |
|                   | 8392.3         | 2.878      | 287.80      | 543.21       | 219.454       |
| Cipher | Area ($\mu m^2$) | Power (mW) | Energy (pJ) | Latency (ns) | Max TP (Gbps) |
|--------|-----------------|------------|-------------|--------------|---------------|
| Orthros | 76712.1 | 27179 | 2.452 | 245.22 | 9.37 | 12.722 |
|        | 77555.6 | 27407 | 2.525 | 252.49 | 8.00 | 14.901 |
|        | 83566.3 | 29067 | 3.478 | 347.79 | 7.00 | 17.030 |
|        | 95896.7 | 33976 | 4.447 | 444.73 | 6.00 | 19.868 |
|        | 124746.6 | 44197 | 5.393 | 539.28 | 5.29 | 22.535 |
| Midori-128 | 67710.8 | 29990 | 3.817 | 381.72 | 15.89 | 7.502 |
|        | 67938.7 | 24070 | 4.090 | 408.97 | 14.00 | 8.515 |
|        | 73455.8 | 26025 | 5.374 | 537.38 | 12.00 | 9.934 |
|        | 90043.0 | 31902 | 7.200 | 719.96 | 10.00 | 11.921 |
|        | 122156.3 | 43279 | 9.138 | 913.82 | 9.05 | 13.172 |
| QARMA$_{128}$ | 80258.5 | 28435 | 4.2252 | 422.52 | 17.40 | 6.851 |
|        | 80844.1 | 28043 | 5.2578 | 525.78 | 15.00 | 7.947 |
|        | 89048.8 | 31550 | 6.8206 | 682.06 | 13.00 | 9.170 |
|        | 107814.3 | 38198 | 8.4356 | 843.56 | 11.00 | 10.837 |
|        | 153286.0 | 54399 | 10.3186 | 1031.86 | 9.43 | 12.641 |
| PRINCE | 22036.6 | 7807 | 0.699 | 69.94 | 9.79 | 12.177 |
|        | 22674.5 | 8033 | 0.911 | 91.09 | 8.00 | 14.901 |
|        | 25879.3 | 9169 | 1.135 | 113.47 | 7.00 | 17.030 |
|        | 31206.6 | 11056 | 1.421 | 142.12 | 6.00 | 19.868 |
|        | 42512.8 | 15064 | 1.898 | 189.81 | 5.52 | 21.596 |
| Kangaroo12-PRF[1600] | 372818.5 | 132088 | 21.558 | 2155.83 | 16.21 | 7.354 |
|        | 379860.1 | 134564 | 20.788 | 2078.78 | 14.00 | 8.515 |
|        | 424274.4 | 150319 | 24.212 | 2421.24 | 11.00 | 10.837 |
|        | 504947.1 | 178901 | 29.376 | 2937.59 | 9.12 | 13.071 |
| Kangaroo12-PRF[400] | 104761.8 | 37117 | 5.259 | 525.93 | 15.15 | 7.869 |
|        | 106882.5 | 37159 | 5.661 | 566.09 | 14.00 | 8.515 |
|        | 121185.4 | 42935 | 7.248 | 724.75 | 11.00 | 10.837 |
|        | 151044.3 | 53514 | 9.119 | 911.92 | 9.27 | 12.860 |
| Subterranean-PRF | 103337.2 | 36612 | 3.367 | 336.72 | 14.28 | 8.348 |
|        | 105067.4 | 37225 | 3.388 | 338.77 | 12.00 | 9.934 |
|        | 123273.3 | 43675 | 5.149 | 514.85 | 10.00 | 11.921 |
|        | 156786.4 | 55549 | 5.227 | 522.74 | 7.98 | 14.938 |
| Cipher          | Area ($\mu m^2$) | Power (mW) | Energy (pJ) | Latency (ns) | Max TP (Gbps) |
|----------------|-----------------|------------|-------------|--------------|---------------|
| Orthros        | 21404.5         | 15.952     | 1595.20     | 3.81         | 31.289        |
|                | 21466.2         | 14.858     | 1485.80     | 3.50         | 34.060        |
|                | 21548.4         | 13.386     | 1338.60     | 3.00         | 39.736        |
|                | 24028.6         | 13.237     | 1323.70     | 2.68         | 44.481        |
| Midori-128     | 18784.7         | 23.088     | 2308.79     | 6.29         | 18.952        |
|                | 18808.6         | 22.071     | 2207.10     | 6.00         | 19.868        |
|                | 18971.7         | 21.533     | 2153.30     | 5.50         | 21.674        |
|                | 19666.7         | 21.014     | 2101.40     | 5.00         | 23.842        |
|                | 20396.6         | 21.066     | 2106.60     | 4.72         | 25.256        |
| QARMA$_{128-\sigma_0}$ | 22866.4 | 28.055     | 2805.50     | 6.68         | 17.846        |
|                | 22903.4         | 27.896     | 2789.60     | 6.50         | 18.340        |
|                | 22956.6         | 25.678     | 2567.80     | 6.00         | 19.868        |
|                | 23044.9         | 23.804     | 2380.4      | 5.50         | 21.674        |
|                | 24699.2         | 21.174     | 2117.40     | 4.92         | 23.842        |
| PRINCE         | 6037.4          | 4.371      | 437.11      | 3.87         | 30.803        |
|                | 6072.8          | 4.135      | 413.54      | 3.50         | 34.060        |
|                | 6473.9          | 3.994      | 399.43      | 3.00         | 39.736        |
|                | 6695.1          | 4.005      | 400.55      | 2.92         | 40.825        |
| Kangaroo12-PRF[1600] | 99007.3     | 85.880     | 8588.00     | 5.08         | 23.466        |
|                | 101193.1        | 82.437     | 8243.70     | 4.70         | 25.364        |
|                | 100167.0        | 79.505     | 7950.50     | 4.50         | 26.491        |
|                | 102999.7        | 78.764     | 7876.40     | 4.34         | 27.468        |
| Kangaroo12-PRF[400] | 26726.0     | 24.496     | 2449.60     | 5.49         | 21.714        |
|                | 26756.7         | 23.986     | 2398.60     | 5.20         | 22.925        |
|                | 27033.3         | 22.812     | 2281.20     | 4.80         | 24.835        |
|                | 27699.4         | 21.814     | 2181.40     | 4.50         | 26.491        |
| Subterranean-PRF | 27671.7     | 22.760     | 2276.03     | 5.01         | 23.794        |
|                | 27696.2         | 22.225     | 2222.50     | 4.70         | 25.364        |
|                | 28071.5         | 21.406     | 2140.60     | 4.30         | 27.723        |
|                | 29602.2         | 20.762     | 2076.20     | 3.98         | 29.952        |
Table 20: Test vectors for Orthros in hex.

| Plaintext | Key | Ciphertext |
|-----------|-----|------------|
| 00000000000000000000000000000000 | 00000000000000000000000000000000 | 6060acb118f411e434ba4e01984de0de |
| a947436710924cc47f2d571deea8f05 | 4a2be60e3db6abe0c03eae66fd05d0c | e4cec0d077a3401d8c4d07b6d5196e5f |