IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2025, No. 2, pp. 705–742. DOI:10.46586/tches.v2025.i2.705-742

AETHER: An Ultra-High Throughput and Low Energy Authenticated Encryption Scheme

Subhadeep Banik[1], Andrea Caforio[2], Tatsuya Ishikawa[3], Takanori Isobe[3], Mostafizar Rahman[3] and Kosei Sakamoto[4,3]

[1] University of Lugano, Lugano, Switzerland
subhadeep.banik@usi.ch
[2] lowRISC C.I.C., Cambridge, United Kingdom[†]
andrea.caforio@lowrisc.org
[3] University of Hyogo, Kobe, Japan
t.ishikawa037@gmail.com,takanori.isobe@ai.u-hyogo.ac.jp,mrahman454@gmail.com
Mitsubishi Electric Corporation, Kamakura, Japan
[4] Sakamoto.Kosei@dc.MitsubishiElectric.co.jp

Abstract. In this paper, we introduce AETHER, an authenticated encryption scheme that achieves ultra-high throughput and low energy consumption, supporting a 256-bit key and a 128-bit tag. While inspired by an AEGIS-like structure, AETHER stands out with a completely redesigned round-update function. We replace the AES round function with a new inner function optimized for ultra-low latency and energy consumption. This function incorporates Orthros's S-box and a 16×16 binary matrix from Akleylek et al., leading to a 1.56 times reduction in energy consumption and a 1.25 times reduction in delay compared to the AES round function. To further optimize hardware performance, we design the general construction of the round-update function to be more hardware-friendly, allowing parallel execution of the inner function on all 128-bit words, thereby enhancing both throughput and security against collision-based forgery attacks. AETHER achieves a throughput of 2.1 Tbit/s and an energy consumption of only 204.31 nJ, in the Nangate 15 nm standard cell library and a throughput of 5.23 Tbit/s and energy consumption of 1.83 nJ using the CNFET-OCL 5nm library, outperforming all existing AEADs.

Keywords: Authenticated encryption · Low energy · High throughput · AEGIS-like construction

1 Introduction

1.1 Background

Demanding Ultra High-Speed Data Processing. The rapid growth of data consumption, fueled by high-definition streaming, cloud computing, and the Internet of Things (IoT), demands unprecedented data processing speeds from data centers. Cisco's Annual internet report forecasts global IP traffic to reach 3 zettabytes in 2021, while cloud-stored data is projected to swell to 100 zettabytes by 2025, which is half of the total amount of data in the world[1]. To meet this surging demand for real-time data processing and high-bandwidth applications, data center networks must evolve to handle terabits-per-second (Tbps) throughput. Consequently, an ultra-high-throughput encryption scheme is essential

[†]This work was partially done while the author was at TuneInsight SA, Lausanne, Switzerland.
[1]https://cybersecurityventures.com/the-world-will-store-200-zettabytes-of-data-by-2025/

Received: 2024-10-15 Accepted: 2024-12-15 Published: 2025-03-04

for data centers managing large-scale data while ensuring the privacy of processed data. Specifically, to guarantee confidentiality, integrity, and authenticity, an authenticated encryption scheme is mandatory. Additionally, the emerging 6G landscape and the pursuit of Tbps wireless communications by 2030[2] necessitate terabit-per-second encryption speeds across various applications.

Looming Energy Problem. Data centers are the backbone of the digital age, but their ever-growing energy demands pose a significant environmental, societal, and economic challenge. The annual electricity report from the International Energy Agency (IEA) states that data centers consumed 460 TWh in 2022, a figure that could rise to more than 1,000 TWh by 2026 in a worst-case scenario[3]. The European Union's energy efficiency directive, published in September 2023, imposes new obligations on data center operators on the continent. The first of these is a requirement for emissions reports to be filed by any data center larger than 500 kW. The U.S. Department of Energy emphasizes the importance of developing energy-efficient technologies to mitigate the environmental footprint of data centers. Beyond environmental concerns, energy efficiency is crucial for economic viability. Lower energy consumption translates to reduced operational costs, enhancing the long-term sustainability of data centers. Consequently, encryption should be performed with low energy consumption.

Towards Low-Energy Authenticated Encryption. Energy consumption in cryptographic circuits (and general in ASIC implementations) is characterized by the total switching activity, i.e., the number of 0/1 transistions of logic elements, in a synthesized netlist over a specified time interval, with power consumption being the average thereof. Although some adjustments are necessary after the place-and-routing of the netlist to account for the circuit wiring and parasitic effects, performing energy measurements on a netlist is a reasonably precise affair and thus gives rise to the optimisation discipline of tailoring circuits as to reduce their overall energy consumption. In cryptographic hardware, optimizing energy requires a multi-pronged approach on both the algorithmic and circuit level; a problematic which is best illustrated by looking at some existing AEAD circuits:

- High-throughput AEAD circuits such as Rocca-S [ABC+23] and AEGIS-256 [WP13a] can process large chunks of data in few clock cycles and thus achieve impressive performance levels in the 1-2 Tbps range. They achieve this by duplicating important modules such as the AES round function as part of the AEGIS update function in parallel which is reflected in a area footprint which naturally increases the switching activity of the circuit. Hence, this kind of parallelization does not result in an energy-efficient circuit if the area/switching activity is not optimised concurrently (both Rocca-S and AEGIS exclusively focus on the optimization of the throughput). On the other hand, simplifying functions as to produce concise circuits is cryptanalytically delicate endeavour. At the present time, the canon of high-throughput encryption circuits lacks a dedicated low-energy solution.
- Standard AEAD schemes, in the same vein as AES-OCB or AES-GCM can also be parallelized and unrolled to decrease the latency and increase the throughput but again, this results in circuits that switch considerably offsetting the energy-gains.
- Stream ciphers were an early target for energy-optimization efforts. In fact, it was shown that when it comes to the encryption of large amounts of the Trivium-like constructions appear to be the most energy-efficient constructions [CP08, BMA+18,

[2] https://www.6gflagship.com/white-paper-on-rf-enabling-6g-opportunities-and-challenges-from-technology-to-spectrum/

[3] https://www.datacenterdynamics.com/en/news/global-data-center-electricity-use-to-double-by-2026-report/

CBT+21], which is the consequence of a lightweight state update function which can be massively unrolled without incurring a forbidding area penalty. Naturally, Trivium only offers 80-bit security and thus quantum resistance is not achievable without significant design changes. Additionally, achieving terabit-per-second-level throughput with these ciphers appears challenging.

Open Problem. Considering these points, there is currently no authenticated encryption that offers terabit-per-second-level throughput while maintaining low energy consumption for the 6G era, ensuring sustainability and privacy protection, especially for data centers. At the same time, security against quantum computing threats must be addressed.

1.2 Our Contribution

In this paper, we present an ultra-high throughput yet low energy-consumption authenticated encryption scheme dubbed AETHER, featuring a 256-bit key and a 128-bit tag. AETHER is based on an AEGIS-like structure [WP13a], which absorbs the message and outputs the ciphertext for every state update. To achieve ultra-high throughput and low energy consumption, we redesigned the round-update function from scratch. The redesigning procedure is summarized as follows:

- We replace the AES round function in the round-update function with a novel cryptographic function, termed the inner function, optimized for ultra-low latency and energy consumption. This enhancement directly benefits the overall AEAD scheme's performance since AEGIS-like constructions process message and ciphertext blocks synchronously with state updates, akin to stream ciphers. To minimize both delay and energy consumption, we devise an SPN-based inner function utilizing Orthros's S-box and a 16×16 binary matrix from Akleylek et al. [ARSÖ17]. Our choice of components is grounded in a comprehensive evaluation of various S-boxes and matrices for their delay and energy efficiency. The resulting inner function achieves approximately 1.56 and 1.25 times lower energy consumption and delay, respectively, compared to the AES round function, while preserving the security properties essential for thwarting internal collision-based forgery attacks.

- To be more hardware-friendly, we revisit the general construction of the round-update function in Rocca [SLN+21] and Rocca-S [ABC+23]. Specifically, the inner function is applied to all 128-bit words in our general construction because these inner functions can be fully executed in parallel in a hardware environment, whereas Rocca and Rocca-S limit the number of AES round functions to take advantage of hardware acceleration for AES, such as AESNI on Intel CPUs. Additionally, we apply both the inner function and an XOR to each 128-bit word, while Rocca and Rocca-S apply either the AES round function or an XOR. This approach can be realized with negligible overhead in a hardware environment and enhances security against internal collision-based forgery attacks. Finally, we explore the class of round-update functions that meet our security and implementation requirements with the redesigned inner function.

In short, AETHER is a AEAD circuit whose parallelized state update module provides the basis for high throughput rates while, at the same time, offsetting the area/switching activity with a carefully chosen round function that allows for a concise circuit after synthesis. AETHER thus is the first construction that fills that gap in the literature with respect to low-energy, high-throughput encryption. More specifically, by integrating these novel design strategies, AETHER achieves a remarkable throughput of 2.1 Tbit/s while consuming only 204.31 nJ of energy (for processing 1024 bits of AD and 1.28 Mbits of

plaintext) when implemented using the Nangate 15nm library. When implemented using the CNFET-OCL 5nm library, it offers a throughput of 5.23 Tbps and consumes only 1.83 nJ for processing the same amount of data. This achieves both the highest throughput and the lowest energy consumption among AEGIS-like and existing AEADs (a fact, established in a comprehensive comparison with related schemes), while also providing 128-bit security against key-recovery attacks in a Q1 setting[4] (see Table 9 and 10). AETHER is the first authenticated encryption scheme targeting 6G technology, while maintaining low energy consumption.

2 Specification

In this paper, a *block* denotes a 16-byte value as the round update function of AETHER consists of 9 16-bytes values denoted by $S = (S[0], S[1], \ldots, S[8])$.

2.1 The Round-Update Functions

The round-update function of AETHER consists of nine internal states denoted $\boldsymbol{S} = (S[0], S[1], \cdots, S[8])$ and accepts three states denoted $\boldsymbol{X} = (X_0, X_1, X_2)$ in every state update. The round-update function $R(\boldsymbol{S}, \boldsymbol{X})$is shown in Fig. 1 and defined as

$$
\begin{aligned}
&S^{new}[0] = F(S[8]) \oplus X_0, && S^{new}[1] = F(S[0]) \oplus S[3], && S^{new}[2] = F(S[1]) \oplus S[6],\\
&S^{new}[3] = F(S[2]) \oplus X_1, && S^{new}[4] = F(S[3]) \oplus S[4], && S^{new}[5] = F(S[4]) \oplus X_2,\\
&S^{new}[6] = F(S[5]) \oplus S[8], && S^{new}[7] = F(S[6]) \oplus S[2], && S^{new}[8] = F(S[7]) \oplus S[0],
\end{aligned}
$$

where F denotes the inner function described in Sect. 2.2.

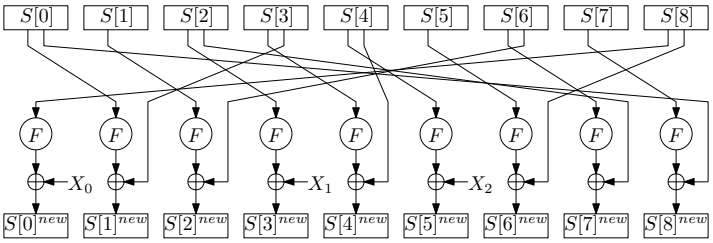


Figure 1: The round-update function of AETHER.

2.2 The Inner Functions

The inner function consists of a 128-bit state; the functions ApplySbox, MatrixMul, and Permutation are applied consecutively; they are defined below:

$$F = \mathsf{Permutation} \circ \mathsf{ApplySbox} \circ \mathsf{MatrixMul} \circ \mathsf{ApplySbox}.$$

The illustration of the inner function is shown in Fig. 2. We provide a detailed explanation of ApplySbox, MatrixMul, and Permutation as follows:

ApplySbox. 32 4-bit S-boxes are applied to the 128-bit state in parallel. We use Orthros's 4-bit S-box in ApplySbox as shown in Table 1.

[4]The adversary can access the offline quantum computer but cannot query the quantum oracle.

Table 1: The 4-bit S-box in F.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$Sbox(x)$	1	0	2	4	3	8	6	d	9	a	b	e	f	c	7	5

MatrixMul. The 128-bit state is first divided into two sets of 16 nibbles. Then, the same two 16×16 matrices over nibbles are applied to them in parallel. 16×16 matrix $\boldsymbol{M_b}$ applied in MatrixMul is defined as

$$\boldsymbol{M_b} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

16 nibbles $(x_0, x_1, x_2, \ldots, x_{15})$ will be updated as follows:

$$(x_0, x_1, x_2, \ldots, x_{15})^T \leftarrow \boldsymbol{M_b} \cdot (x_0, x_1, x_2, \ldots, x_{15})^T,$$

where $(x_0, x_1, x_2, \ldots, x_{15})^T$ denotes a transposition matrix.

Permutation. The nibble permutation P_n shown in Table 2 is applied to the 128-bit state.

Table 2: The nibble permutation P_n.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P_n(x)$	0	16	1	17	2	18	3	19	4	20	5	21	6	22	7	23
x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P_n(x)$	8	24	9	25	10	26	11	27	12	28	13	29	14	30	15	31

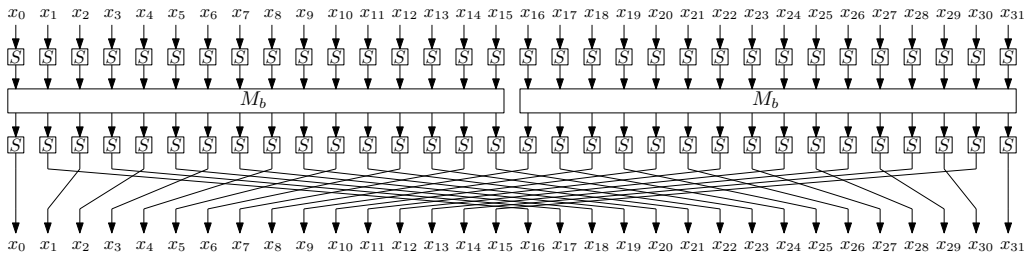


Figure 2: The illustration of F.

The detailed procedure of the inner function F is given in Algorithm 1.

Algorithm 1 Procedure of F. X denotes the 128-bit state.

function $F(X)$ in $R(\boldsymbol{R}, \boldsymbol{X})$
  $(x_0 \| x_1 \| x_2 \| \ldots \| x_{31}) \leftarrow X$
  for $i = 0$ to 31 **do**
    $x_i \leftarrow Sbox(x_i)$
  end for
  $(x_0, x_1, x_2, \ldots, x_{15})^T \leftarrow \boldsymbol{M_b} \cdot (x_0, x_1, x_2, \ldots, x_{15})^T$
  $(x_{16}, x_{17}, x_{18}, \ldots, x_{31})^T \leftarrow \boldsymbol{M_b} \cdot (x_{16}, x_{17}, x_{18}, \ldots, x_{31})^T$
  for $i = 0$ to 31 **do**
    $x_i \leftarrow Sbox(x_i)$
  end for
  $(x_1^*, x_2^*, x_3^*, \ldots, x_{15}^*) \leftarrow (x_1, x_2, x_3, \ldots, x_{15})$
  for $i = 0$ to 31 **do**
    $x_{P_n(i)} \leftarrow x_i^*$
  end for
  $X \leftarrow (x_0 \| x_1 \| x_2 \| \leftarrow \| x_{31})$
  return X
end function

2.3 Specification of AETHER

AETHER consists of four phases: initialization, processing the associated data, encryption, and finalization. We first explain the padding process for the associated data and message.

Padding Process. The padding is applied independently to associated data AD and the message M when their sizes are not a multiple of 384 bits. Each AD and M are padded by appending `0x100...000` independently into the end to be their sizes as a multiple of 384 bits. The padded associated data and message are denoted $\overline{AD}$ and $\overline{M}$, respectively. For a better understanding, we give examples of the padding regarding four cases: 1) the sizes of both AD and M are not a multiple of 384 bits, 2) the size of only AD is not a multiple of 384 bits, 3) the size of only M is not a multiple of 384 bits, and 4) the sizes of both AD and M are a multiple of 384 bits.

1) $\overline{AD} = \underbrace{\texttt{0x9495a...1c0d97}}_{AD\ (336\ \text{bits})}\underbrace{\texttt{100000000000}}_{\text{The padding}}$, $\overline{M} = \underbrace{\texttt{0x6f8a...993ab}}_{M\ (328\ \text{bits})}\underbrace{\texttt{10000000000000}}_{\text{The padding}}$.

2) $\overline{AD} = \underbrace{\texttt{0xb3987...8cce0}}_{AD\ (704\ \text{bits})}\underbrace{\texttt{1000000000000000}}_{\text{The padding}}$, $\overline{M} = \underbrace{\texttt{0x492ba20ff...8fa38bcf}}_{M\ (1152\ \text{bits})}$.

3) $\overline{AD} = \underbrace{\texttt{0x49bd02fff...ce9aa234}}_{AD\ (384\ \text{bits})}$, $\overline{M} = \underbrace{\texttt{0x93bd238...9abc38f}}_{M\ (344\ \text{bits})}\underbrace{\texttt{10000000}}_{\text{The padding}}$.

4) $\overline{AD} = \underbrace{\texttt{0xabff7eeca...1100f80d}}_{AD\ (384\ \text{bits})}$, $\overline{M} = \underbrace{\texttt{0x95f2b5129...ec5819df}}_{M\ (768\ \text{bits})}$.

Initialization. The input consists of the 256-bit key K and the 128-bit nonce N. Additionally, three constant blocks are used for the initialization process as given below. Z_0 =`0x428a2f98d728ae227137449123ef65cd`, Z_1 =`0xb5c0fbcfec4d3b2fe9b5dba58189dbbc`, and Z_2 =`0x7137449123ef65cd428a2f98d728ae22`. Z_0 and Z_1 are the same as in Rocca-S, and Z_2 is generated by the concatenation of the last half and first half parts of

Z_0. K is divided into two 128-bit keys, i.e., $K = K_0 \| K_1$. K_0, K_1, and N are first loaded into the state S as follows:

$$\begin{array}{llll} S[0] = Z_1, & S[1] = K_0, & S[2] = N + K_0, & S[3] = 0, \quad S[4] = Z_0, \\ S[5] = 0, & S[6] = N, & S[7] = K_1, & S[8] = Z_2. \end{array}$$

Then, 20 iterations of the round-update function $R(\boldsymbol{S}, \boldsymbol{Z})$ where $\boldsymbol{Z} = (Z_0, Z_1, Z_2)$ are applied to the state S. After 20 iterations of the round update function, K_0 and K_1 are XORed with the state S as follows:

$$\begin{array}{lll} S[0] = S[0] \oplus K_0, & S[1] = S[1] \oplus K_0, & S[2] = S[2] \oplus K_0, \\ S[3] = S[3] \oplus K_0, & S[4] = S[4] \oplus K_1, & S[5] = S[5] \oplus K_0, \\ S[6] = S[6] \oplus K_1, & S[7] = S[7] \oplus K_1, & S[8] = S[8] \oplus K_1. \end{array}$$

Processing the Associated Data. The associated data AD is first padded for $|AD|$ being a multiple of 384 bits. Then, AD is divided into the set of 128-bit words $\overline{AD}$ as follows:

$$\overline{AD} = (\overline{AD_0} \| \overline{AD_1} \| \overline{AD_2} \| \ldots \| \overline{AD_{i-1}}), \; i = \frac{|\overline{AD}|}{128}.$$

$\overline{AD}$ is loaded into the round update function, i.e., $R(\boldsymbol{R}, \overline{AD_{3 \cdot j}}, \overline{AD_{3 \cdot j+1}}, \overline{AD_{3 \cdot j+2}})$ for $0 \leq j < \frac{|\overline{AD}|}{384}$. If AD is empty, this phase will be skipped.

Encryption. The message M is first padded for $|M|$ being a multiple of 384 bits. Then, M is divided into the set of 128-bit words $\overline{M}$ as follows:

$$\overline{M} = (\overline{M_0} \| \overline{M_1} \| \overline{M_2} \| \ldots \| \overline{M_{i-1}}), \; i = \frac{|\overline{M}|}{128},$$

The divided message blocks are absorbed into the round update functions, i.e., $R(\boldsymbol{R}, \overline{M_{3 \cdot j}}, \overline{M_{3 \cdot j+1}}, \overline{M_{3 \cdot j+2}})$ for $0 \leq j < \frac{|\overline{M}|}{384}$. During this procedure, the ciphertext blocks $\overline{C_i}$ are generated in the following way:

$$\begin{aligned} \overline{C_j} &= F(S[0] \oplus S[1]) \oplus S[4] \oplus \overline{M_j} \\ \overline{C_{j+1}} &= F(S[2] \oplus S[6]) \oplus S[7] \oplus \overline{M_{j+1}} \\ \overline{C_{j+2}} &= F(S[3] \oplus S[5]) \oplus S[8] \oplus \overline{M_{j+2}} \end{aligned}$$

If the message is padded p bits, the ciphertext blocks generated before the last round update function are truncated to $384 - p$ bits. If M is empty, this phase will be skipped.

Finalization. K_0 and K_1 are first XORed with the internal state $\boldsymbol{S}$ as follows:

$$\begin{array}{lll} S[0] = S[0] \oplus K_0, & S[1] = S[1] \oplus K_0, & S[2] = S[2] \oplus K_1, \\ S[3] = S[3] \oplus K_1, & S[4] = S[4] \oplus K_0, & S[5] = S[5] \oplus K_0, \\ S[6] = S[6] \oplus K_1, & S[7] = S[7] \oplus K_0, & S[8] = S[8] \oplus K_1. \end{array}$$

Then, 20 iterations of the round-update function $R(\boldsymbol{R}, K_0, Z_0, K_1)$ are applied to the state S. Before the tag generation, K_0 and K_1 are XORed with the internal state $\boldsymbol{S}$ again as follows:

$$\begin{array}{lll} S[0] = S[0] \oplus K_1, & S[1] = S[1] \oplus K_0, & S[2] = S[2] \oplus K_0, \\ S[3] = S[3] \oplus K_0, & S[4] = S[4] \oplus K_1, & S[5] = S[5] \oplus K_0, \\ S[6] = S[6] \oplus K_0, & S[7] = S[7] \oplus K_1, & S[8] = S[8] \oplus K_1. \end{array}$$

Lastly, the authentication tag T is generated as follows:

$$\bigoplus_{i=0}^{8} S[i] = T.$$

The detailed algorithm and overview of AETHER are given in Algorithm 2 and Fig. 3, respectively. In Algorithm 2, we omit the descriptions of padding and truncating denoted by `Padding`() and `Truncate`(), both of which follow the explanations in Initialization and Encryption, respectively. In the decryption of AETHER, if the received T does not match the tag calculated by the ciphertext C, the message M decrypted by C is never returned. Test vectors of AETHER will be given in Appendix A.

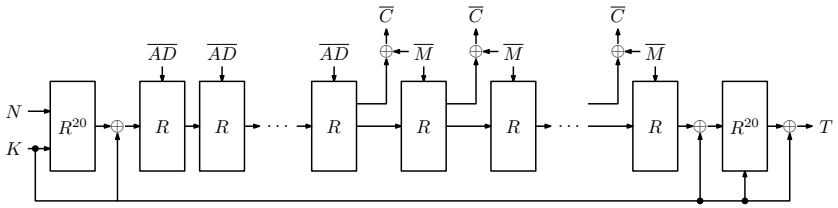


Figure 3: Overview of AETHER.

Security Claims. AETHER provides 256-bit security against key recovery and 128-bit security against distinguishing and forgery attacks in the nonce-respecting setting. We do not claim any security in the related-key and known/chosen-key settings. For the length of the message and associated data, we limit them up to 2^{128} and 2^{64} for a fixed key, respectively. We also limit the number of different messages produced for a fixed key up to 2^{128}. AETHER has not been designed to be key-committing since AEGIS-like AEADs are inherently hard to ensure it as shown in [DFI+24, TTI24]. To meet the demand for a key-committing scheme, we provide a key-committing variant of AETHER with the methodology in [ADG+22] in Appendix F.

3 Design Rationale

Our primary goal is to design an Authenticated Encryption with Associated Data (AEAD) scheme that achieves high throughput and low-energy consumption in hardware while maintaining 128-bit security against key recovery attacks in a quantum setting.

3.1 Existing Results

Regarding energy efficiency, Banik et al. investigate low-energy cryptographic primitives and find that stream ciphers implemented on unrolled circuits consume less energy due to their ability to produce multiple keystream bits per clock cycle [BMA+18]. Caforio et al. further explore the relationship between state-update functions and energy consumption, presenting energy-optimized variants of several existing stream ciphers, namely Trivium-LE and Triad-LE [CBT+21]. However, these do not provide post-quantum security, namely a 256-bit key, and are not optimized for high throughput.

For the throughput aspect, Rocca-S, designed for ultra-high throughput in software, achieves impressive results exceeding 1 Tbps even in hardware with 128-bit quantum security. Rocca-S is based on AEGIS-like structure [WP13b], which typically outputs many ciphertext bits while updating the round function only once. Caforio et al. highlight the connection between high throughput (achieved through small-delay round functions)

Algorithm 2 Encryption procedure of AETHER.

function AETHER $_{enc}(K, N, AD, M)$
  $\boldsymbol{S} \leftarrow \texttt{Initialization}(K, N)$
  if $|AD| > 0$ **then**
    $\overline{AD} \leftarrow \texttt{Padding}(AD)$
    $\boldsymbol{S} \leftarrow \texttt{ProcessAD}(\boldsymbol{S}, \overline{AD})$
  end if
  if $|M| > 0$ **then**
    $\overline{M} \leftarrow \texttt{Padding}(M)$
    $(\boldsymbol{S}, C) \leftarrow \texttt{Encryption}(\boldsymbol{S}, \overline{M})$
    $C \leftarrow \texttt{Truncate}(\overline{C})$
  end if
  $T \leftarrow \texttt{Finalization}(\boldsymbol{S}, K)$
  return (C, T)
end function
function AETHER $_{dec}(K, N, AD, C, T)$
  $\boldsymbol{S} \leftarrow \texttt{Initialization}(K, N)$
  if $|AD| > 0$ **then**
    $\overline{AD} \leftarrow \texttt{Padding}(AD)$
    $\boldsymbol{S} \leftarrow \texttt{ProcessAD}(\boldsymbol{S}, \overline{AD})$
  end if
  if $|C| > 0$ **then**
    $\overline{C} \leftarrow \texttt{Padding}(C)$
    $(\boldsymbol{S}, M) \leftarrow \texttt{Encryption}(\boldsymbol{S}, \overline{C})$
    $M \leftarrow \texttt{Truncate}(\overline{M})$
  end if
  if $T = \texttt{Finalization}(\boldsymbol{S}, K)$ **then**
    return M
  else
    return $\perp$
  end if
end function
function $\texttt{Initialization}(K, N)$
  $K_0 \| K_1 \leftarrow K$
  $(S[0], S[1], S[2], S[3]) \leftarrow (Z_1, K_0, N + K_0, 0)$
  $(S[4], S[5], S[6], S[7], S[8]) \leftarrow (Z_0, 0, N, K_1, Z_2)$
  for $i = 0$ to 19 **do**
    $\boldsymbol{S} \leftarrow R(\boldsymbol{S}, \boldsymbol{Z})$
  end for
  $(S[0], S[1], S[2], S[3]) \leftarrow (S[0] \oplus K_0, S[1] \oplus K_0, S[2] \oplus K_0, S[3] \oplus K_0)$
  $(S[4], S[5], S[6], S[7], S[8]) \leftarrow (S[4] \oplus K_1, S[5] \oplus K_0, S[6] \oplus K_1, S[7] \oplus K_1, S[8] \oplus K_1)$
  return $\boldsymbol{S}$
end function
function $\texttt{ProcessAD}(\boldsymbol{S}, \overline{AD})$
  $d \leftarrow \frac{|\overline{AD}|}{384}$
  for $i = 0$ to $d - 1$ **do**
    $\boldsymbol{S} \leftarrow R(\boldsymbol{R}, \overline{AD_{3i}}, \overline{AD_{3i+1}}, \overline{AD_{3i+2}})$
  end for
  return $\boldsymbol{S}$
end function
function $\texttt{Encryption}(\boldsymbol{S}, \overline{M})$
  $m \leftarrow \frac{|\overline{M}|}{384}$
  for $i = 0$ to $m - 1$ **do**
    $\overline{C_{3i}} \leftarrow F(S[0] \oplus S[1]) \oplus S[4] \oplus \overline{M_{3i}}$
    $\overline{C_{3i+1}} \leftarrow F(S[2] \oplus S[6]) \oplus S[7] \oplus \overline{M_{3i+1}}$
    $\overline{C_{3i+2}} \leftarrow F(S[3] \oplus S[5]) \oplus S[8] \oplus \overline{M_{3i+2}}$
    $\boldsymbol{S} \leftarrow R(\boldsymbol{S}, \overline{M_{3i}}, \overline{M_{3i+1}}, \overline{M_{3i+2}})$
  end for
  $\overline{C} \leftarrow (\overline{C_0} \| \overline{C_1} \| \ldots \| \overline{C_{2m-1}})$
  return $(\boldsymbol{S}, \overline{C})$
end function
function $\texttt{Finalization}(\boldsymbol{S}, K)$
  $(S[0], S[1], S[2], S[3]) \leftarrow (S[0] \oplus K_0, S[1] \oplus K_0, S[2] \oplus K_1, S[3] \oplus K_1)$
  $(S[4], S[5], S[6], S[7], S[8]) \leftarrow (S[4] \oplus K_0, S[5] \oplus K_0, S[6] \oplus K_1, S[7] \oplus K_0, S[8] \oplus K_1)$
  for $i = 0$ to 19 **do**
    $\boldsymbol{S} \leftarrow R(\boldsymbol{S}, K_0, Z_0, K_1)$
  end for
  $(S[0], S[1], S[2], S[3]) \leftarrow (S[0] \oplus K_1, S[1] \oplus K_0, S[2] \oplus K_0, S[3] \oplus K_0)$
  $(S[4], S[5], S[6], S[7], S[8]) \leftarrow (S[4] \oplus K_1, S[5] \oplus K_0, S[6] \oplus K_0, S[7] \oplus K_1, S[8] \oplus K_1)$
  $T \leftarrow \bigoplus_{i=0}^{8} S[i]$
  return T
end function

and low energy consumption [CBT+21], noting that energy is roughly estimated as the product of delay and area required to process the message. Recognizing this connection, we believe AEGIS-like structures offer an opportunity for redesign to achieve both ultra-high throughput and energy efficiency while maintaining a sufficient security level. Therefore, we start by revisiting AEGIS-like structure to reduce energy consumption.

3.2 Revisiting AEGIS-like Structures

The round-update function of an AEGIS-like AEAD typically employs a combination of XOR operation and the AES round function. This design leverages the efficiency of the XOR operation and the availability of hardware acceleration for AES (e.g., AESNI on Intel CPUs) in software environments. However, while the XOR operation remains low-cost in both hardware and software, the delay, area, and energy consumption of the AES round function, particularly due to the complex S-box lookups, become bottlenecks for hardware implementations (as shown in Table 3).

In response to this limitation, we propose a redesign of the round-update function. This redesign aims to significantly improve hardware performance, as detailed in this section. This redesign involves replacing the AES round function with a more hardware-friendly alternative, referred to as the inner function. This inner function will be optimized for various metrics, such as latency, area, and energy consumption. In the following section, we will explore how to construct such a hardware-friendly inner function and how to design the round-update function suitable for this inner function.

3.2.1 The Inner Function.

To achieve both low energy consumption and high throughput in our AEAD design, the round-update function needs to have minimal delay and area as indicated in [BBI+15]. Since the inner function significantly impacts these aspects, we aim to design an inner function with ultra-low delay and minimal area footprint (small energy consumption) while providing strong security.

Performance requirements. Banik et al. demonstrate that Substitution-Permutation Networks (SPNs) offer better energy efficiency and smaller delay than Feistel networks [BBI+15]. This is further supported by the fact that many low-latency primitives, such as the block ciphers PRINCE [BCG+12] and QARMA [Ava17, ABD+23], utilize SPNs for their performance benefits. Moreover, since the AEGIS-like structure absorbs message blocks and outputs ciphertext blocks every state update, achieving a small delay in the inner function plays a crucial role in increasing throughput. Therefore, to optimize implementation performance for energy efficiency and throughput, we need to develop an SPN-based inner function consisting of a low-delay S-box and matrix with a small area footprint to achieve small energy consumption.

Security requirements. A critical challenge in designing AEGIS-like AEADs is ensuring security against internal collision-based forgery attacks. Existing AEGIS-like AEADs, such as Tiaoxin [Nik14], Rocca [SLN+21], and Rocca-S [ABC+23], provide this security by leveraging the strong security property of the AES round function. Specifically, the cascading two AES rounds ensures 5 active S-boxes with probability 2^{-30}. This significantly increases the difficulty of mounting a forgery attack.

While the overall security against forgery attacks depends on both the inner function and the round-update function, it is crucial to maintain a strong security property within the inner function itself. This minimizes the risk of collision-based forgery attacks due to the inner function replacement. Therefore, our security requirement for the inner function is that the two cascaded inner functions ensure a better differential probability than that

of the two cascaded AES round functions, i.e., the maximum differential probabilities over the two cascaded inner functions must be less than 2^{-30}.

S-box. Table 3 shows delay, area, and energy consumption for the AES S-box and several low-energy and -latency S-boxes. According to Table 3, it is evident that employing a 4-bit S-box is a favorable choice for our design due to its inherent advantages in terms of the energy consumption and delay. For delay and energy consumption, Sb0 used in Midori appears to be a suitable choice for our design. However, Sb0 lacks the full-diffusion property, which ensures that each input bit can affect all output bits.

Since the inner function must consist of as few non-linear (and linear) layers as possible to minimize delay and energy consumption, our S-box used in the inner function should have optimal security properties. Then, among S-boxes with optimal security property, Orthros's S-box and ρ used in QARMAv2 are the best choices regarding delay and energy consumption, respectively. Particularly, for Orthros's S-box, energy consumption is close to that of ρ while delay is much better than that of ρ. For the area, all 4-bit S-boxes with optimal security are almost the same. Therefore, considering the balance between delay and energy consumption, we choose Orthros's S-box as the component for the inner function.

Table 3: Security and implementation properties of low-latency S-boxes. $\mathcal{E}$ and $\mathcal{L}$ denote energy-optimized and delay-optimized circuits for the NanGate 15 nm cell library at a clock frequency of 100 MHz, respectively. We used the Synopsys Design Compiler synthesis directive `compile_ultra` in combination with manually restricting the critical path to obtain the latency-optimised circuits $\mathcal{L}$. An approach that was already used in [BIL+21, ABC+24] and `compile` command for the $\mathcal{E}$ circuits leading to energy-efficient variants.

Scheme	Width	Area (μm^2)		Delay (ps)		Energy (pJ)		DP	C^2	Degree	Full
		$\mathcal{L}$	$\mathcal{E}$	$\mathcal{L}$	$\mathcal{E}$	$\mathcal{L}$	$\mathcal{E}$				Diffusion
AES	8	248.17	136.15	24.96	96.88	0.5908	0.1105	2^{-6}	2^{-6}	7	✓
BipBip	6	17.89	12.04	13.70	24.99	0.0414	0.0358	2^{-4}	2^{-4}	2	-
SPEEDY	6	15.08	9.73	7.43	11.97	0.0389	0.0198	2^{-3}	$2^{-2.83}$	5	✓
Gleeok (χ)	5	9.04	9.73	6.61	12.97	0.0244	0.0088	2^{-2}	2^{-2}	2	-
Orthros	4	5.99	3.34	5.17	9.74	0.0144	0.0080	2^{-2}	2^{-2}	3	✓
Midori (Sb0)	4	6.93	3.04	4.81	8.96	0.0178	0.0059	2^{-2}	2^{-2}	3	-
Midori (Sb1)	4	5.99	3.29	7.37	10.48	0.0138	0.0080	2^{-2}	2^{-2}	3	✓
QARMAv2 (ρ)	4	5.55	3.29	7.37	10.86	0.0137	0.0079	2^{-2}	2^{-2}	3	✓
QARMAv2 (σ_0)	4	9.58	3.19	4.68	16.51	0.0262	0.0075	2^{-2}	2^{-2}	3	-
PRINCE	4	5.94	3.78	6.05	15.62	0.0131	0.0093	2^{-2}	2^{-2}	3	✓
Gleeok (χ)	3	4.27	2.31	6.61	13.21	0.0112	0.0053	2^{-2}	2^{-2}	2	-

Matrix. Banik et al. demonstrate that almost MDS matrices (binary matrices) offer advantages over MDS matrices regarding delay and area [BBI+15]. This is why it is popular choice for low-latency cryptographic primitives like QARMA [Ava17, ABD+23] and Orthros [BIL+21]. In the inner function, however, it is essential to ensure as small a maximum differential characteristic probability (DCP_{max}) as possible on a few non-linear (and linear) layers to achieve a small delay. Besides, a 4-bit S-box ensures only a maximum differential probability of 2^{-2}, while an 8-bit S-box can ensure that of 2^{-6}. Therefore, considering the use of Orthros's 4-bit S-box with a maximum differential probability of 2^{-2}, we need a binary matrix with strong diffusion property while minimizing overhead.

Several studies, such as [SA14, AS14, ARSÖ17], explore binary matrices and demonstrate that 8×8, 16×16, and 32×32 binary matrices have the maximum branch number 5, 8, and 12, with maximum minimum Hamming weights of each row and column being

5, 7, and 11, respectively. Due to the page limitation, we show these binary matrices in Appendix B. As delay of almost MDS, 8×8, 16×16, and 32×32 binary matrices can be roughly estimated to 3, 4.5, 4.5, and 6, respectively, by *depth* proposed by Banik et al. [BBI+15], these three binary matrices have a potential for adoption in our design.

Table 4 shows the security and detailed implementation properties of 4×4, 8×8, 16×16, 32×32 binary matrices. In this evaluation, we consider a 128-bit linear layer consisting of eight 4×4, four 8×8, two 16×16, and one 32×32 binary matrices with matrix elements being 4 bits and a linear layer of AES consisting of four MDS matrices.

Table 4: Security and implementation properties of the binary matrices and MDS matrix from AES. All designs were for the NanGate 15 nm cell library at a clock frequency of 100 MHz. BR denotes the branch number.

Implementation properties						
Matrix	Area (μm^2)		Delay (ps)		Energy (pJ)	
	$\mathcal{L}$	$\mathcal{E}$	$\mathcal{L}$	$\mathcal{E}$	$\mathcal{L}$	$\mathcal{E}$
MDS matrix in AES	70.19	47.77	16.00	28.64	0.1574	0.1463
4×4 binary matrix [BBI+15]	28.31	21.23	9.82	12.39	0.0558	0.0514
8×8 binary matrix [AS14]	35.83	29.63	15.62	21.67	0.1141	0.0825
16×16 binary matrix [ARSÖ17]	113.69	88.03	21.01	37.22	0.3030	0.3220
32×32 binary matrix [ARSÖ17]	368.09	307.59	25.34	42.78	0.9023	1.2540
Security properties						
Matrix	Branch number (BR)		Delay / BR*		Energy / BR**	
MDS matrix in AES	5		3.2		0.1170 (0.0292×4)	
4×4 binary matrix [BBI+15]	4		2.45		0.1028 (0.0128×8)	
8×8 binary matrix [AS14]	5		3.12		0.0660 (0.0165×4)	
16×16 binary matrix [ARSÖ17]	8		2.62		0.0757 (0.0378×2)	
32×32 binary matrix [ARSÖ17]	12		2.11		0.0751	

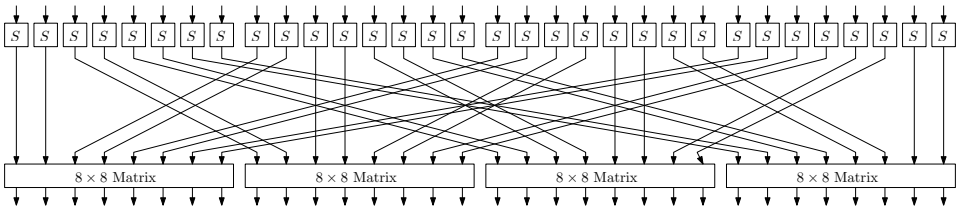
* The number of matrices is assumed to the case of applying 128-bit linear layer.
* The better results are used.

According to Table 4, the 4×4 binary matrix is the best for delay. For the other binary matrices, the delay to branch number ratio improves as the matrix size increases regarding binary matrices. Conversely, the 8×8 matrix stands out with its superior energy consumption to branch number ratio, making it the most efficient choice. Notably, the 4×4 matrix costs the worst among the four binary matrices, an undesirable property for our design. For area, the 4×4 matrix naturally achieves the smallest area footprint among all matrices we evaluate. However, the total area of the 4×4 matrix is not the best since we apply 8 4×4 matrices in the inner function. Considering the balance between delay and energy consumption, 8×8, 16×16, and 32×32 matrices are better choices for our design. Therefore, we choose these 3 matrices as the candidates for the inner function.

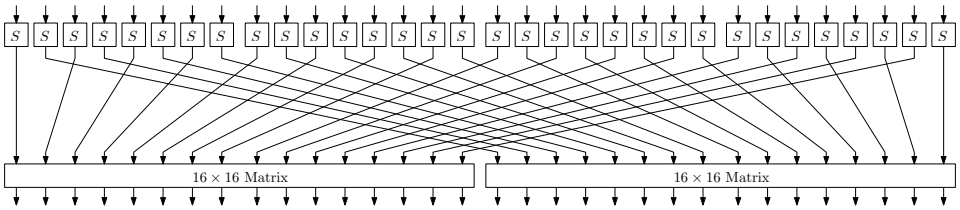
Structure of the Inner Function. We evaluate three candidates for our SPN-based inner function, consisting of the aforementioned 32×32, 16×16, and 8×8 binary matrices as a linear layer with Orthros's 4-bit S-box as a non-linear layer. Table 5 shows the lower bound for the number of active S-boxes, which is denoted # AS in this paper, after each layer regarding these three candidates and AES, where DCP_{max} is calculated by the lower bound for the number of active S-boxes. According to Table 5, none of the three candidates reach DCP_{max} of less than 2^{-30} after two SPN-based rounds and one S-box layer.

To fulfill the security requirement, we consider an inner function with a single SPN-based round followed by one S-box layer with 32×32 and 16×16 matrices. For the 8×8 matrix, we use two SPN-based rounds followed by one S-box layer. Then, two cascaded such inner functions can ensure DCP_{max} of $2^{-48} (= 2^{-24 \times 2})$, $2^{-32} (= 2^{-16 \times 2})$, and $2^{-32} (= 2^{-16 \times 2})$ with 32×32, 16×16, and 8×8 matrices, respectively. Fig 4a, 4b, and

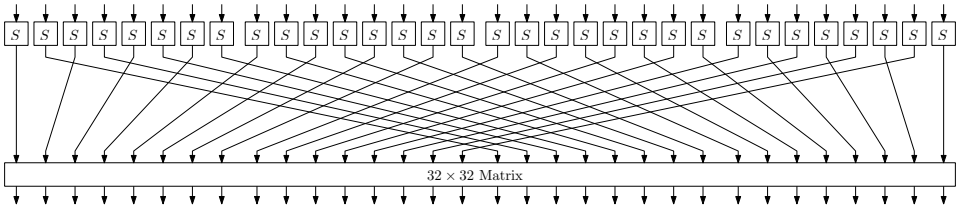
4c illustrate a single SPN-based round with 8×8, 16×16, and 32×32 binary matrices, respectively. Note that we apply nibble permutations before the matrices to have the maximum # AS in the nibble-wise evaluations, which can be realized without the overhead.



(a) A single SPN-based round with 8×8 matrix.



(b) A single SPN-based round with 16×16 matrix.



(c) A single SPN-based round with 32×32 matrix.

Figure 4: A single SPN-based round with 32×32, 16×16, and 8×8 matrices.

Then, we evaluate the hardware performance for these candidates and the AES round function without a key addition. Table 6 shows the implementation results where F_{32}, F_{16}, and F_8 denote a single SPN-based round followed by one S-box layer with 32×32 matrix, a single SPN-based round followed by one S-box layer with 16×16 matrix, and two SPN-based rounds followed by one S-box layer with 8×8 matrix, respectively. According to Table 6, F_{16} is the best in all evaluations in terms of area, delay, and energy consumption.

Considering the fact that all candidates meet the security requirement and the implementation results, we finally choose F_{16} as the inner function of our design[5].

3.2.2 The Round-Update Function

Difference in Throughput between Software and Hardware. In a software environment, the number of the applied AES round functions and the inserted message blocks are closely related to the throughput of overall AEAD. Let the number of the applied AES round functions and the inserted message blocks in a single round-update function be $\#AES$ and $\#M$, respectively. Jean and Nikolić introduce *rate* that is a metric to estimate the throughput calculated by $(\#AES)/(\#M)$ [JN16]. To further reduce the critical path of the round-update function, Sakamoto et al. apply either the AES round function or an XOR for each 128-bit word in a single round-update function and propose the ultra-high throughput AEAD Rocca, which has rate 2.

[5]In the final specification of the inner function, we apply P_n after the second S-box layer. We would emphasize that this change affects neither implementation nor security properties.

Table 5: Lower bounds for the number of active S-boxes and differential characteristic probability after each operation. # AS and DCP_{max} denote the lower bound for the number of active S-boxes and the maximum differential characteristic probability, respectively.

Operation	AES		32×32 matrix		16×16 matrix*		8×8 matrix⋆	
	# AS	DCP_{max}	# AS	DCP_{max}	# AS	DCP_{max}	# AS	DCP_{max}
Non-linear (S-box)	1	2^{-6}	1	2^{-2}	1	2^{-2}	1	2^{-2}
Linear (Matrix)	1	2^{-6}	1	2^{-2}	1	2^{-2}	1	2^{-2}
Non-linear (S-box)	5	2^{-30}	12	2^{-24}	8	2^{-16}	5	2^{-10}
Linear (Matrix)	5	2^{-30}	12	2^{-24}	8	2^{-16}	5	2^{-10}
Non-linear (S-box)	9	2^{-54}	13	2^{-26}	9	2^{-18}	8	2^{-16}

* The identical two 16×16 binary matrices are applied in parallel to proceed 32 4-bit words.
⋆ The identical four 8×8 binary matrices are applied in parallel to proceed 32 4-bit words.

Table 6: The area, delay, and energy consumption of each candidate for the inner function. F_{AES} denotes the AES round function without a key addition.

Inner Functions	Construction	DCP_{max} (2 consecutive rounds)	Area (μm^2)		Delay (ps)		Energy (pJ)	
			$\mathcal{L}$	$\mathcal{E}$	$\mathcal{L}$	$\mathcal{E}$	$\mathcal{L}$	$\mathcal{E}$
F_{AES}	Single SPN-based round	2^{-30}	3126.06	2368.73	54.79	131.79	8.7880	2.8040
F_8	Two SPN-based rounds + one S-box layer	2^{-32}	811.69	557.97	70.85	91.85	2.8320	2.5440
F_{16}	Single SPN-based round + one S-box layer	2^{-32}	597.83	389.97	43.81	67.49	1.9190	1.7966
F_{32}	Single SPN-based round + one S-box layer	2^{-48}	696.82	610.27	50.90	79.51	2.1936	2.9580

In contrast, on a hardware environment, the throughput is primarily determined by the critical path of a single round-update function and the number of message blocks processed per round ($\#M$). The number of inner functions applied within a single round-update function ($\#F$) has less impact. This is because hardware can often execute all inner functions in parallel within a single round.

Towards Low Energy. To achieve small energy consumption, the round-update function should minimize both delay and area. As mentioned above, the small delay is already realized by designing the inner function. To reduce the implementation area, we need to reduce $\#F$. However, it should be noted that reducing the implementation area poses a challenge in ensuring security against forgery attacks.

An alternative approach to reduce energy consumption is to increase the number of message blocks processed per round ($\#M$). This improves the energy efficiency per encrypted bit of plaintext. Additionally, it contributes to increased throughput. Therefore, to maintain a required level of security, we prioritize increasing $\#M$ over reducing $\#F$.

Furthermore, we allow the round-update function to apply both the inner function and an XOR for each 128-bit word because the overhead of applying one XOR is quite small compared to applying the inner function, this plays a more significant role in enhancing security against internal collision-based forgery attacks. Therefore, the general construction of our round-update function becomes as illustrated in Fig. 5.

In our general construction, if a 128-bit word is XORed with a message block, this word does not accept XORing with an additional 128-bit word coming from a permutation, similar to Rocca. Therefore, let the number of the 128-bit words be $\#S$, the total number of candidates in the class of $\#S = s$ and $\#M = m$ is as follows:

$$s! \times \binom{s}{m} \times (m!)^{-1}. \tag{1}$$

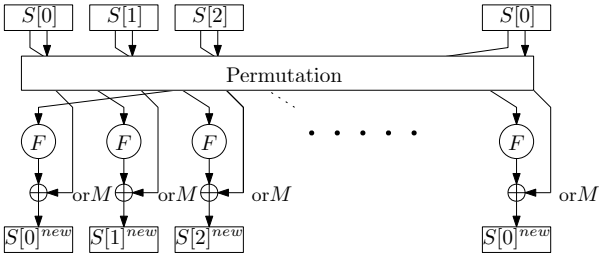


Figure 5: General construction of the round-update function where F denotes the application of the inner function.

Requirements of Round-Update Function. In summary, the requirements for the round-update function are as follows:

Performance requirement.

1. The round-update function should absorb as many message blocks as possible, i.e., $\#M$ should be as large as possible.
2. The number of the states in the round-update function, denoted $\#S$, should be as small as possible.

Security requirement.

1. The round-update function ensures the same security level as that of AEGIS-128, Tiaoxin, and Rocca, i.e., 128-bit security against forgery attacks is required.

Finding Optimal Round-Update Function. The designers of AEGIS-128 and Rocca demonstrate security against internal collision-based forgery attacks by evaluating the lower bound for the number of active S-boxes, denoted # AS, using a byte-wise truncated difference approach. However, we must evaluate it using a nibble-wise truncated difference due to applying a 4-bit S-box in the inner function, making the evaluation more time-consuming. Given the need to evaluate numerous candidates for the round-update function, conducting a nibble-wise evaluation for all of them is impractical.

Moreover, the first performance requirement implies that we must explore a large class of round-update functions. Therefore, we must adopt an alternative method to evaluate security against internal collision-based forgery attacks with a much smaller computational cost. Consequently, we conduct a word-wise evaluation instead of a nibble-wise one, which requires significantly less computational effort than a byte-wise evaluation. Since DCP_{max} of the inner function is 2^{-16}, it is enough to ensure 8 active inner functions, i.e., $2^{-128} \leq 2^{-16\times 8}$ to guarantee 128-bit security against the internal collision-based forgery attacks.

As the number of the candidates for the round-update function with $\#S = s$ and $\#M = m$ is calculated by Eq. (1), it is infeasible to evaluate all the candidates in a practical time for $\#S \geq 10$. Thus, we randomly evaluate the candidates with $\#S \leq 11$. We employ a SAT-based automatic search method and carry out the evaluation on a workstation equipped with Intel Xeon Platinum 8380 and 1 TB memory.

Search Result. Table 7 shows the summary of our search results. We found 8478 candidates with $\#S = 9$ and $\#M = 3$, which has the maximum $\#M$ and ensures 128-bit security against internal collision-based forgery attacks. Note that these candidates ensure $\#\mathrm{AF} = 8$, where #AF denotes the lower bound for the number of active inner functions.

To further narrow down the candidates, we evaluate the diffusion property regarding each 128-bit word and found that 162 out of 8478 candidates achieve full diffusion after 6 rounds. We then investigate the diffusion property of each 128-bit word after 5 rounds for 162 candidates. We identified that 5 out of 162 candidates achieve almost full diffusion after 5 rounds, meaning that each 128-bit state after 5 rounds can be affected by at least 8 128-bit input words. After a comprehensive evaluation, we select 1 out of the 5 candidates as the round-update function of AETHER shown in Fig. 1.

Table 7: The summary of our evaluation to find the candidates with #AF ≥ 8.

$\#S$	$\#M$	Total	# of searched	# of found	$\#S$	$\#M$	Total	# of searched	# of found
4	**1**	**96**	**All**	**16**	8	3	376320	All	0
4	2	72	All	0	8	4	117600	All	0
4	3	16	All	0	8	5	18816	All	0
5	2	600	All	0	8	6	1568	All	0
5	3	200	All	0	8	7	64	All	0
5	4	25	All	0	**9**	**3**	**5080320**	**All**	**8478**
6	**2**	**5400**	**All**	**54**	9	4	1905120	All	0
6	3	2400	All	0	9	5	381024	All	0
6	4	450	All	0	9	6	42336	All	0
6	5	36	All	0	9	7	2592	All	0
7	3	29400	All	0	9	8	81	All	0
7	4	7350	All	0	10	4	31752000	All	0
7	5	882	All	0	10	5	38102400	All	0
7	6	49	All	0	11	4	548856000	2^{26}	0

3.3 Loading Scheme and Output Functions

AEGIS-like AEADs first load the nonce and key into the internal states in the initialization. This loading scheme has a significant impact on the security of the initialization, as reported in [LIMS21]. As with Rocca and Rocca-S, we would like to have the property that the whole internal states cannot be expressed only in terms of $S(N)$, K_0, and K_1 after some rounds. If these terms exist, the attacker can construct one useless round in the initialization, resulting in stronger attacks. After a careful investigation into the formulas of each 128-bit word after several rounds, we decided to load N, K_0, K_1, and $N + K_0$ into $S[6]$, $S[4]$, $S[7]$, and $S[2]$, respectively. This can avoid a useless round, as reported in [LIMS21].

Since the structure of the output function influences a linear bias that is a powerful distinguishing attack for AEGIS-like AEADs [Min14]. To resist it, we proactively explore the output function with the most robust resistance against this type of attack, fitting the following form by constructing a SAT model.

$$\begin{aligned}\overline{C_i} &= F(S[j_0] \oplus S[j_1]) \oplus S[j_2] \oplus \overline{M_i},\\ \overline{C_{i+1}} &= F(S[j_3] \oplus S[j_4]) \oplus S[j_5] \oplus \overline{M_{i+1}},\\ \overline{C_{i+2}} &= F(S[j_6] \oplus S[j_7]) \oplus S[j_8] \oplus \overline{M_{i+2}},\end{aligned}$$

where $j_{k_0} \neq j_{k_1}$ for $k_0 \neq k_1$ and $0 \leq j_0, j_1, j_2, j_3.j_4, j_5, j_6, j_7, j_8 \leq 8$. Therefore, the number of candidate output functions equals 7560 $(= \binom{9}{3} \cdot 3! \cdot (2!)^{-1} \times \binom{6}{3} \cdot 3! \cdot (2!)^{-1} \times \binom{3}{3} \cdot 3! \cdot (2!)^{-1} \times (3!)^{-1})$. We evaluate the word-wise truncated linear characteristics for all patterns regarding the lower bounds for the number of active inner functions and choose one that ensures 13 active S-boxes, guaranteeing a linear bias of less than $2^{-208}(2^{-16\times 13})$. As the designers of Rocca and Rocca-S mentioned, there is a large gap between our truncated model and the actual characteristics that can be exploited in the attack. Considering that our model is truncated word-wise, we expect that a linear bias of the actual best characteristics is much smaller than 2^{-208}, ensuring 128-bit security against distinguishing attacks.

3.4 Two Key Feed-Forward Operations

Hosoyamada et al. demonstrated the potential risk of AEGIS-like AEADs, showing that an imbalance between tag length and key-recovery security can lead to internal state recovery [HII+22]. This vulnerability is inherent to designs with tag sizes smaller than the claimed key-recovery security. Successful internal state recovery can potentially enable key recovery attacks by inverting the initialization process [HII+22]. Additionally, if the finalization process is publicly known, forgery attacks may become feasible by generating tags using the recovered state information.

Since AETHER has a 128-bit tag and a 256-bit key-recovery security level, it falls into this category. To address these issues, we propose two key feed-forward operations in order to reduce key recovery problem after recovering the internal state to well-analyzed hard problem of symmetric cryptography.

Key Forwarding to Initialization. We employ the key feed-forward operation at the end of the initialization, similar to Rocca [SLN+22]. As mentioned in Sect. 3.2.2, the cost of applying one XOR operation is negligible compared to that of the inner function. Hence, we XOR K_0 and K_1 to all 128-bit words at the end of the initialization, making inverting the initialization phase impossible without guessing K_0 or K_1. Essentially, inverting the initialization becomes a preimage problem for the underlying permutation with the forward operation. As the initialization phase is sufficiently strong permutation, mounting key recovery attacks is computationally infeasible.

Key Forwarding to Finalization. To prevent forgery attacks following internal state recovery, incorporating the key into the finalization process is essential. However, Takeuchi et al. demonstrated a key recovery attack against a variant of Rocca that incorporates key addition only before the finalization step [TTI24]. Essentially, recovering the internal state reduces the finalization step to an Even-Mansour cipher, enabling key recovery by observing the ciphertext-tag.

To address this, we design the finalization in AETHER that involves the key even in the state update process, similar to a block cipher. This approach makes it difficult to recover the key even when the internal state is fully recovered. Specifically, recovering the key becomes equivalent to a key recovery problem for an underlying block cipher, treating the finalization as a large block cipher.

4 Security Evaluation

4.1 Differential Attack

The differential attack is one of the applicable attacks to the initialization of AEGIS-like AEADs. The attacker can exploit the differences with a high probability in the nonce in the differential attack. To evaluate its probability, we explore the word-wise truncated differential characteristics and compute the lower bound for the number of active inner functions in each round. Because the maximum differential characteristic probability of the inner function is 2^{-16}, it is sufficient to ensure 16 active inner functions ($2^{-16\times 16} \leq 2^{-256}$) for achieving 256-bit security against key-recovery attacks. Table 8 shows the lower bounds for the number of active inner functions in each round. As can be viewed in Table 8, the initialization of AETHER can ensure 16 active inner functions after 5 rounds, and the number of active inner functions is over double of 16 after 10 rounds. Considering the number of rounds in the initialization is 20, we expect that AETHER can resist the differential attack.

Table 8: The lower bounds for the number of active inner function in the initialization phase.

Rounds	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
# AF	2	4	8	12	16	19	22	24	28	34	36	39	43	45	48	50	53	57	60	65

4.2 Forgery Attack

A main threat to AEGIS-like AEADs is internal collision-based forgery attacks as shown in [Nik14] because the message blocks are absorbed after every one-round update. All AEGIS-like AEADs have round-update functions designed to resist this type of attack. In AETHER, we design the round-update function to ensure 128-bit security against the internal collision-based attacks by ensuring 8 active inner functions, i.e., $2^{-16\times 8} \leq 2^{-128}$. However, since our round-update function can ensure only 8 active inner functions in a truncated SAT model, we cannot ensure that AETHER has enough security margin against this type of attack so far. Although there is a large gap from the actual characteristics that the attacker can exploit to that obtained from the word-wise truncated model, it must be shown that AETHER certainly has the security margin against this type of attack. To this end, we construct the bit-wise evaluation model and evaluate a tight differential characteristic probability of internal collisions. In this evaluation, we take a two-step approach due to the high computational cost for the bit-wise model as follows:

Step 1. Extract the number of rounds that ensure only 8 active inner functions in the word-wise truncated model.

Step 2. Evaluate the bit-wise differential characteristics on the number of rounds extracted in Step 1.

After the evaluation of Step 1. we found that only 6-round internal collisions have 8 active inner functions in the truncated model. Then, we evaluate optimal differential characteristics on the 6-round internal collision by the bit-wise model. As a result, we find that the differential characteristic probability of optimal differential characteristics is 2^{-164}. Since there is still enough gap to 2^{-128}, we expect that AETHER can resist the internal collision-based forgery attacks.

4.3 Linear Bias of the Keystream

Minaud showed a linear bias on the keystream can be exploited as the distinguishing attacks for AEGIS-256 [Min14]. Then, Eichlseder et al. improved this attack with an automatic search method [ENP19]. Since AETHER has the AEGIS-like construction, the security against this attack should be evaluated.

As discussed in Sect. 3.3, the output function is designed to have a strong resistance against this type of attack; that is, the strongest output function among possible candidates is chosen. Specifically, we expect the time complexity to exploit a linear bias to be at least $2^{-208}(2^{-16\times 13})$. Therefore, we expect that the linear attack can not violate our security claim.

4.4 Integral Attack

Liu et al. show that the integral attacks work to AEGIS family and Tiaoxin based on the integral distinguisher on 4-round AES. Although AETHER does not employ the AES round function, it is necessary to evaluate the security against the integral attack due to the similarity of the constructions. To estimate the security against integral attacks, we

investigate the longest integral distinguisher on the initialization, which is known to be efficiently evaluated by the division property [Tod15].

To estimate the longest integral distinguisher on the initialization, we take a two-step approach as follows:

Step 1. Evaluate the division property for the inner function.

Step 2. Estimate how many rounds the attacker can construct the integral distinguishers on for the round-update function with the results of Step 1.

After the evaluation of Step 1, we found that the output after 7 applications of the inner function does not have the integral distinguishers, i.e., $F(F(F(F(F(F(F(X)))))))$ is not balanced. Then, in Step 2, we confirmed that each output word after 9 rounds consist of the term in terms of 7 applications of the inner function for the nonce, implying that there are no integral distinguishers on the initialization after 9 rounds.

It should be emphasized that this is a conservative evaluation because we do not consider the impact of other terms that makes constructing the integral distinguishers more difficult. Considering the number of rounds in the initialization is 20, we expect that AETHER can resist the integral attacks.

4.5 Resistance to Key-Recovery

AETHER is analyzed against the key-recovery attack techniques from [TTI24]. Although, in the nonce-misuse setting, it is possible to recover the internal state (see Appendix C) and set the state to an arbitrary value via interpolation (see Appendix D), the key-recovery attack in [TTI24] is still not applicable. In the case of Rocca, once the internal state is recovered, the security of the finalisation is reduced to an Even-Mansour cipher which enables to recover the key in an efficient way. However, in the case of AETHER, due to the introduction of the keys in the state-update function in the finalisation step, offline computation of the tag based on a guessed sub-key (a part of the key) is not possible. Hence, it is viable to believe, AETHER is secure against key-recovery. Note that, some distinguishing attacks also exist but they do not lead to key-recovery. For an instance, with reference to Fig. 8 during the encryption process in the nonce-misuse setting, if a difference is injected into M_0, the same difference propagates to C_8 (such attacks also exists for AEGIS [WP13a, Section 6.2]).

4.6 Security of the Finalization Step

One of the main novelty of AETHER is its unique finalization step. As discussed, due to introduction of the secret key in the finalization step, it becomes difficult to recover the key even when the internal state is completely recovered. Additionally, the offline computation of the tag based on some arbitrary internal state is also not possible which in turn resists the forgery attacks (as in [TTI24]).

5 Hardware Evaluation

The circuit for AETHER is reminiscent of earlier designs for schemes like Rocca-S [ABC+23] and AEGIS [WP13b]. In fact, one can easily convert a given round-based circuit for these schemes and convert it into a viable design for AETHER, simply by replacing the combinatorial round function circuit and adapting some of the wiring. More specifically, the core structure of these algorithms is a layer of duplicated interconnected combinatorial modules that are fed by and wired back to a bank of state registers. In the case of Rocca-S and AEGIS these modules implement the AES round function, while for AETHER this function

is a combination of two tranches of the 4-bit Orthros S-box [BIL+21] interwoven with a novel diffusion layer composed of a 16×16 byte-wise matrix a nibble-wise permutation.

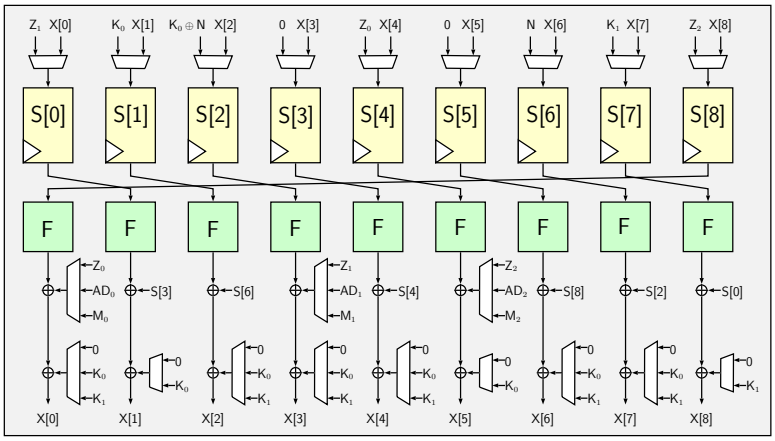


Figure 6: AETHER state update circuit. The ciphertext generation module has been omitted for the sake of conciseness.

5.1 Circuits

At the gate level, an implementation for AETHER can be realized with common circuitry. A row of 128-bit D-flip-flops stores the intermediate cipher state, its input data being switched by an array of 128-bit multiplexers. The register outputs are directly fed to the round function module whose 4-bit S-box is mapped to its corresponding circuit by passing its lookup-table description to the synthesis tool. It has been shown by the designers of Orthros [BIL+21] that the lookup-table-based synthesis approach works particularly well for small 4-bit S-boxes in producing competitive latency figures. As AETHER is using the Orthros S-box for its non-linear layer, we adopted the same technique. Beyond the S-box, the byte-based matrix multiplication can be achieved in a straightforward manner with a low-depth network of XOR gates, and the nibble-based permutation does not increase the circuit area footprint as it solely consists of wiring. Finally, some additional multiplexers steer the addition of data and keys into the before it is fed back to the registers. A schematic depiction of the AETHER state update circuit is shown in Figure 6.

5.2 Results

The measurements of AETHER are compared against existing related algorithms Rocca-S [ABC+23], AEGIS-256 [WP13a], AES-256-GCM [MV04] as well as the recently standardized lightweight AEAD scheme Ascon [DEMS19] in order to establish its competitiveness not only in terms of energy consumption but also in circuit area, throughput and power. Note that our implementation of the permutation for Ascon is similar to the 6-round unrolled circuit used in [GWDE15]. We also compare our design with the NIST lightweight cryptography candidates GIFT-COFB [BCI+19], SUNDAE-GIFT [BBP+19], ROMULUS-N1 [IKMP19], PHOTON-BEETLE [BCD+19] and TINYJAMBU [WH19]. To make the comparisons meaningful, for each of the above designs we unroll the round function of the underlying block-cipher/permutation by a suitable amount so that the energy consumption is optimal.

An important side note in this endeavour is the fact that the hardware design space of Rocca-S and AEGIS are significantly broader due to the their utilization of the AES round function as the core primitive for which a multitude of different circuits exist each aiming

at a different trade-off in the set of hardware metrics. For example, the 8-bit Rijndael S-box can be passed to the synthesis tool as a lookup-table (LUT) as is the case for AETHER or implemented as optimised handwritten circuits as was done by Maximov and Ekdahl [ME19] to achieve the to date both smallest (SMALL) and fastest implementation. If minimization of power is the goal, one can make use of the Decode-Switch-Encode (DSE) technique that adds specific encoders/decoders to the S-box lookup-table input and outputs as to reduce the overall switching activity of the circuitry. Inspired by common AES software implementations, it further possible to implement the entire round function as a combination of several large T-tables that are then passed to the synthesizer.[6][7]

The obtained measurements for the NanGate 15 nm library [MMR+15] are tabulated in Table 9 demonstrating that the design choices of AETHER, i.e., the combination of the conciseness of F and the large cipher state enabling a data absorption rate of 384 bits per clock cycle, result in a competitive construction that cuts the energy consumption by roughly significantly improves the energy consumption compared to related ciphers while being competitive with respect to other hardware metrics. The table shows that for smaller data-lengths some schemes like TINYJAMBU and AEGIS outperform AETHER: however that comes with the cost of lowering of throughput. For longer data-lengths AETHER outperforms all the schemes listed in the table.

Table 9: Measurement comparison between AETHER and related schemes in the Nangate 15 nm OCL. The source code for Rocca-S, AEGIS-256 and AES-256-GCM have been taken from [ABC+23]. The circuit of Ascon-128a stems from the reference implementation with the difference that the permutations ρ_a and ρ_b have been unrolled. The input size for the short energy measurements consisted of 1024 bits of AD and 2048 bits of plaintext while the long input size was 1024 bits of AD and 1.28 Mbits of plaintext. All design were compiled with the regular `compile` routine as part of the Synopsys Design Compiler, and all power measured at 10 MHz. The figure $r =$ `X` indicates that the round function was unrolled `X` times to construct the corresponding circuit.

	Area		Latency	Throughput	TP/Area	Power	Energy			
	μm^2	GE	ns	Tbit/s	Tbit/(sm^2)	mW	#Cycles	Short(nJ)	#Cycles	Long(nJ)
AETHER	10504	53428	0.185	2.066	**196.7**	0.605	49	**2.965**	3377	**204.3**
Ascon-128a	7789	39619	0.583	0.219	28.1	7.874	28	22.047	10012	7883.4
Rocca-S										
LUT	22832	116130	0.179	1.431	62.7	1.401	44	6.165	5036	705.6
DSE	22931	116138	0.177	1.451	63.3	0.765	44	3.368	5036	385.5
SMALL	11184	56889	0.232	1.102	98.5	1.255	44	5.522	5036	632.1
TT	28579	145364	0.154	1.653	57.8	0.881	44	3.876	5036	443.7
AEGIS-256										
LUT	17403	88521	0.167	0.766	44.0	1.106	48	5.309	10032	1109.0
DSE	17520	89116	0.165	0.766	43.7	0.613	48	2.945	10032	615.6
SMALL	8703	44266	0.210	0.610	70.1	1.014	48	4.868	10032	1017.0
TT	21743	110591	0.132	0.970	44.6	0.691	48	3.317	10032	693.3
AES-256-GCM										
LUT	10023	50980	0.349	0.023	2.3	0.521	266	13.85	160010	8328.0
DSE	10101	51381	0.349	0.023	2.3	0.417	266	11.09	160010	6674.0
SMALL	8265	42038	0.349	0.023	2.8	0.502	266	13.36	160010	8035.0
TT	12599	64082	0.349	0.023	1.8	0.577	266	15.35	160010	9224.0
GIFT-COFB($r = 2$)	1454	7397	0.397	0.015	10.3	0.081	500	4.06	200180	1625.6
SUNDAE-GIFT($r = 2$)	1137	5783	0.247	0.012	10.6	0.077	1220	9.42	400180	3090.2
PHOTON-BEETLE($r = 1$)	2038	10365	0.179	0.057	28.0	0.163	300	4.89	120108	1956.2
ROMULUS-N1($r = 3$)	2366	12032	0.292	0.022	9.3	0.143	399	5.73	190095	2727.7
TINYJAMBU ($r = 128$)	526	2676	0.142	0.027	51.3	0.022	700	1.56	320188	712.4

[6] It has been shown in [ABC+23] that the T-table (TT) implementation approach leads counter-intuitively to the most latency-efficient circuits for the AES-based AEAD schemes in the likes of Rocca-S, AEGIS, SNOW-V and AES-256-GCM, albeit at the cost of significant circuit area overhead.

[7] All investigated designs were synthesized with the Synopsys Design Compiler (2017.09) using the academic NanGate 15 nm cell library. Power and Energy measurements were then obtained in post-synthesis with the help of the Synopsys Power Compiler.

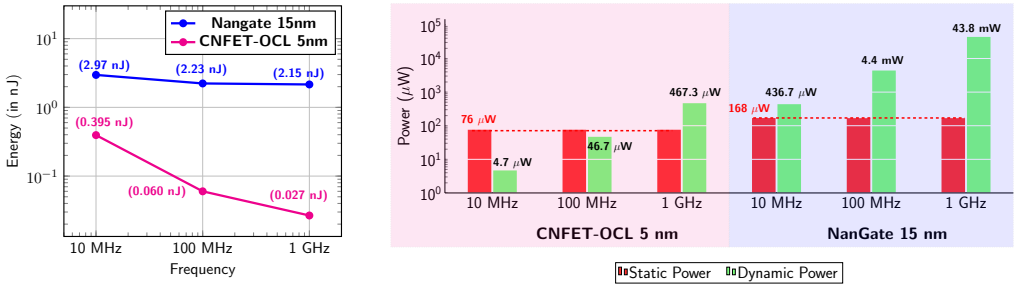


Figure 7: [Left] Energy Consumption of AETHER with respect to Frequency (Short). [Right] Breakup of dynamic and leakage components of the power for the two standard cell libraries w.r.t. clock frequency. Note both axes are plotted in log-scale in both the plots.

5.3 Results using the CNFET-OCL 5nm standard cell library

The CNFET-OCL family of standard cells uses carbon nanotube technology to implement cells using transistors of 5 and 7 nm feature sizes [SMY+23b]. The library is completely open source and can be downloaded from [SMY+23a]. It is very well suited for low energy and high speed designs as the authors report around 96%, 62% and 82% reduction in dynamic and static power consumption and critical-path delay, respectively, when compared with ASAP7 [CVS+16], another 7 nm standard cell library reported in literature.

Figure 7 (left) shows the variation of energy consumption w.r.t. clock frequency for AETHER for the two standard cell libraries (for the "short" amount of plaintext/AD bits). We can see that whereas for the Nangate library, the energy consumption is of the same order, for the CNFET library the energy consumption decreases by almost an order of magnitude for 100 MHz over 10 MHz, and similarly for 1 GHz over 100 MHz. To understand this, recall that in both [KDH+12, BBR15], the authors had concluded that in low leakage environments, the energy consumption for any operation is independent of the clock frequency provided it is high enough. There are two types of power consumed in CMOS circuits: the first is due to the continuous currents drawn by the transistors i.e. mainly due to the short-circuit current and the sub-threshold leakage current when the transistor is OFF. This component is clubbed under the term static or leakage power, and is independent of the frequency at which the circuit is clocked. The second (called dynamic power) is due to the logic transitions every transistor is required to make: to make this happen the power source as to drive electric charge in and out of load capacitances of each transistor. This component varies directly as the clock frequency, since the rate of all such charging and discharging is dictated by the clock period. Figure 7 (right) shows the breakup of these components for AETHER when implemented with both the libraries for frequencies starting from 10 Mhz to 1 GHz. While the static component remains unchanged w.r.t. frequency, the dynamic component increases proportionally, i.e. by a factor of close to 10 for a 10-fold increase in clock frequency. However the total amount of physical time taken to do an operation also decreases proportionally w.r.t. clock frequency. For example, at 10 MHz, it takes 49 cycles which equals 4900 ns. Similarly 490ns at 100 MHz and 49 ns at 1 GHz. Hence the dynamic component of energy remains more or less constant for both the libraries at all frequencies.

- In this example, for the Nangate 15 nm library, E_{dynamic} is around $43.8\mathsf{mW} \times 49\mathsf{ns} \approx 2.146\ \mathsf{nJ}$. And for the CNFET-OCL 5nm library $E_{\mathsf{dynamic}} \approx 0.467\mathsf{mW} \times 49\mathsf{ns} \approx 0.023\ \mathsf{nJ}$.

- However the static component of energy decreases since the static power remains

constant and physical time decreases. In this example for the Nangate 15nm library we have

$$\begin{aligned} E_{\mathsf{static}} &\approx 168\mu\mathsf{W} \times 4900\mathsf{ns} = 0.823\ \mathsf{nJ}\ at\ 10\ \text{MHz} \\ &\approx 168\mu\mathsf{W} \times 490\mathsf{ns} = 0.082\ \mathsf{nJ}\ at\ 100\ \text{MHz} \\ &\approx 168\mu\mathsf{W} \times 49\mathsf{ns} = 0.008\ \mathsf{nJ}\ at\ 1\ \text{GHz} \end{aligned}$$

- The contribution of the static part thus decreases w.r.t. an increase of clock frequency. This lead to the conclusion made by [KDH+12, BBR15].
- The case with the CNFET 5nm library is similar, but it is to be noted that sue to transistor structure it consumes extremely low switching power at lower frequencies, as a result the static energy is the dominant component at clock frequencies lower than 1 GHz. For this library we have

$$\begin{aligned} E_{\mathsf{static}} &\approx 76\mu\mathsf{W} \times 4900\mathsf{ns} = 0.372\ \mathsf{nJ}\ at\ 10\ \text{MHz} \\ &\approx 76\mu\mathsf{W} \times 490\mathsf{ns} = 0.037\ \mathsf{nJ}\ at\ 100\ \text{MHz} \\ &\approx 76\mu\mathsf{W} \times 49\mathsf{ns} = 0.0037\ \mathsf{nJ}\ at\ 1\ \text{GHz} \end{aligned}$$

- For higher frequencies it is reasonable to conclude that for this library too, the static component decreases to make the energy consumption more or less constant w.r.t. frequency.

Since the switching power more closely captures the algorithmic traits of the AEAD under consideration, we report the power and energy in Table 10 at 1 GHz, which captures the dynamic component better. In any way one of the reasons of employing cell libraries of lower feature size, is to be able to clock the circuits faster and so 1 GHz seems to be a fair level to benchmark the energy consumption. The results using the 5 nm library is presented in Table 10. We see that for the long long data benchmark, AETHER outperforms nearly all other schemes using this library except for Rocca-S, whose energy consumption is similar to AETHER. However AETHER comes with around 1.5 times the throughput as it can process 3 blocks per clock cycle, when compared to 2 blocks per cycle for Rocca-S.

Lastly, for the software performance of AETHER, we left it in Appendix G, as it is not our main scope.

6 Conclusion

This paper proposed an ultra-high throughput and low energy authenticated encryption scheme named AETHER. To realize both ultra-high throughput and low energy consumption in a hardware environment simultaneously, we revisited AEGIS-like construction and designed the inner function, which is the AES round function in the other AEGIS-like AEADs, from scratch. With this inner function and the round-update function suitable for it, AETHER achieves a throughput of more than 2/5 Tbps with energy consumption of 204.31/1.83 nJ to encrypt 1.28 Mbits of data using the Nangate 15nm/CNFET-OCL 5nm standard cell library respectively.

Acknowledgments

This work was supported by JST AIP Acceleration Research JPMJCR24U1 Japan and JSPS KAKENHI Grant Number JP24H00696. This result is also obtained from the commissioned research (JPJ012368C05801) by the National Institute of Information and Communications Technology (NICT), Japan.

Table 10: Measurement comparison between AETHER and related schemes in the CNFET-OCL 5nm. The source code for Rocca-S, AEGIS-256 and AES-256-GCM have been taken from [ABC+23]. The circuit of Ascon-128a stems from the reference implementation with the difference that the permutations ρ_a and ρ_b have been unrolled. The input size for the short energy measurements consisted of 1024 bits of AD and 2048 bits of plaintext while the long input size was 1024 bits of AD and 1.28 Mbits of plaintext. All design were compiled with the regular `compile` routine as part of the Synopsys Design Compiler, and all power measured at 1 GHz. The figure $r =$ `X` indicates that the round function was unrolled `X` times to construct the corresponding circuit.

	Area		Latency	Throughput	TP/Area	Power	Energy			
	μm^2	GE	ps	Tbit/s	Tbit/sm^2	mW	#Cycles	Short(pJ)	#Cycles	Long(nJ)
AETHER	4514	53310	69.06	5.234	**1159.5**	0.543	49	26.62	3377	1.83
Ascon-128a	3353	39602	217.04	0.555	165.5	7.067	28	197.88	10012	70.76
Rocca-S										
LUT	9731	114928	65.18	3.719	382.2	0.828	44	36.43	5036	4.170
DSE	9896	116875	69.70	3.478	351.5	0.409	44	18.00	5036	2.060
SMALL	5236	61835	74.75	3.243	619.4	0.320	44	14.06	5036	**1.609**
TT	12328	145595	56.40	4.298	348.6	0.500	44	22.02	5036	2.520
AEGIS-256										
LUT	7334	86612	60.74	2.003	273.1	0.646	48	30.99	10032	6.477
DSE	7477	88304	66.20	1.838	245.8	0.325	48	15.59	10032	3.258
SMALL	3982	47024	71.94	1.691	424.7	0.258	48	12.37	10032	2.586
TT	9301	109844	53.72	2.265	243.5	0.394	48	18.93	10032	3.957
AES-256-GCM										
LUT	4293	50701	103.42	0.074	17.2	0.226	266	60.16	160010	36.187
DSE	4318	50998	103.51	0.074	17.1	0.160	266	42.60	160010	25.625
SMALL	3590	42398	101.94	0.075	20.9	0.146	266	38.86	160010	23.376
TT	5377	63507	105.21	0.073	13.6	0.255	266	67.70	160010	40.726
GIFT-COFB($r = 2$)	625	7379	131.300	0.05	80.0	0.061	500	30.31	200180	12.14
SUNDAE-GIFT($r = 2$	486	5744	90.260	0.03	61.7	0.061	1220	74.35	400180	24.39
PHOTON-BEETLE($r = 1$)	879	10385	64.440	0.16	182.0	0.134	300	40.07	120108	16.04
ROMULUS-N1($r = 3$)	1018	12028	110.41	0.058	57.0	0.110	399	43.80	190095	20.87
TINYJAMBU($r = 128$)	225	2655	49.71	0.077	342.2	0.013	700	**9.02**	320188	4.12

References

[ABC+23] Ravi Anand, Subhadeep Banik, Andrea Caforio, Kazuhide Fukushima, Takanori Isobe, Shinsaku Kiyomoto, Fukang Liu, Yuto Nakano, Kosei Sakamoto, and Nobuyuki Takeuchi. An ultra-high throughput aes-based authenticated encryption scheme for 6g: Design and implementation. In Gene Tsudik, Mauro Conti, Kaitai Liang, and Georgios Smaragdakis, editors, *Computer Security - ESORICS 2023 - 28th European Symposium on Research in Computer Security, The Hague, The Netherlands, September 25-29, 2023, Proceedings, Part I*, volume 14344 of *Lecture Notes in Computer Science*, pages 229–248. Springer, 2023.

[ABC+24] Ravi Anand, Subhadeep Banik, Andrea Caforio, Tatsuya Ishikawa, Takanori Isobe, Fukang Liu, Kazuhiko Minematsu, Mostafizar Rahman, and Kosei Sakamoto. Gleeok: A family of low-latency prfs and its applications to authenticated encryption. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024(2):545–587, 2024.

[ABD+23] Roberto Avanzi, Subhadeep Banik, Orr Dunkelman, Maria Eichlseder, Shibam Ghosh, Marcel Nageler, and Francesco Regazzoni. The qarmav2 family of tweakable block ciphers. *IACR Transactions on Symmetric Cryptology*, pages 25–73, 2023.

[ABL+14] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to securely release unverified plaintext in authenticated

encryption. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 105–125. Springer, 2014.

[ADG+22] Ange Albertini, Thai Duong, Shay Gueron, Stefan Kölbl, Atul Luykx, and Sophie Schmieg. How to Abuse and Fix Authenticated Encryption Without Key Commitment. In Kevin R. B. Butler and Kurt Thomas, editors, *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 3291–3308. USENIX Association, 2022.

[ARSÖ17] Sedat Akleylek, Vincent Rijmen, Muharrem Tolga Sakalli, and Emir Öztürk. Efficient methods to generate cryptographically significant binary diffusion layers. *IET Inf. Secur.*, 11(4):177–187, 2017.

[AS14] Bora Aslan and Muharrem Tolga Sakalli. Algebraic construction of cryptographically good binary linear transformations. *Secur. Commun. Networks*, 7(1):53–63, 2014.

[Ava17] Roberto Avanzi. The QARMA block cipher family. almost MDS matrices over rings with zero divisors, nearly symmetric even-mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes. *IACR Trans. Symmetric Cryptol.*, 2017(1):4–44, 2017.

[BBI+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A block cipher for low energy. In *ASIACRYPT (2)*, volume 9453 of *Lecture Notes in Computer Science*, pages 411–436. Springer, 2015.

[BBP+19] Subhadeep Banik, Andrey Bogdanov, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, Elmar Tischhauser, and Yosuke Todo. SUNDAE-GIFT v1.0. *NIST Lightweight Cryptography Project*, 2019.

[BBR15] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Exploring energy efficiency of lightweight block ciphers. In *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers*, pages 178–194, 2015.

[BCD+19] Zhenzhen Bao, Avik Chakraborti, Nilanjan Datta, Jian Guo, Mridul Nandi, Thomas Peyrin, and Kan Yasuda. Photon-beetle. *NIST Lightweight Cryptography Project*, 2019.

[BCG+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, et al. Prince–a low-latency block cipher for pervasive computing applications. In *Advances in Cryptology–ASIACRYPT 2012: 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings 18*, pages 208–225. Springer, 2012.

[BCI+19] Subhadeep Banik, Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, Mridul Nandi, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. Gift-cofb v1.0. *NIST Lightweight Cryptography Project*, 2019.

[BH22] Mihir Bellare and Viet Tung Hoang. Efficient Schemes for Committing Authenticated Encryption. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part II*, volume 13276 of *Lecture Notes in Computer Science*, pages 845–875. Springer, 2022.

[BIL+21] Subhadeep Banik, Takanori Isobe, Fukang Liu, Kazuhiko Minematsu, and Kosei Sakamoto. Orthros: A low-latency PRF. *IACR Trans. Symmetric Cryptol.*, 2021(1):37–77, 2021.

[BMA+18] Subhadeep Banik, Vasily Mikhalev, Frederik Armknecht, Takanori Isobe, Willi Meier, Andrey Bogdanov, Yuhei Watanabe, and Francesco Regazzoni. Towards low energy stream ciphers. *IACR Trans. Symmetric Cryptol.*, 2018(2):1–19, 2018.

[CBT+21] Andrea Caforio, Subhadeep Banik, Yosuke Todo, Willi Meier, Takanori Isobe, Fukang Liu, and Bin Zhang. Perfect trees: Designing energy-optimal symmetric encryption primitives. *IACR Trans. Symmetric Cryptol.*, 2021(4):36–73, 2021.

[CP08] Christophe De Cannière and Bart Preneel. Trivium. In *The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 244–266. Springer, 2008.

[CR22] John Chan and Phillip Rogaway. On Committing Authenticated-Encryption. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian Damsgaard Jensen, and Weizhi Meng, editors, *Computer Security - ESORICS 2022 - 27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26-30, 2022, Proceedings, Part II*, volume 13555 of *Lecture Notes in Computer Science*, pages 275–294. Springer, 2022.

[CVS+16] Lawrence T. Clark, Vinay Vashishtha, Lucian Shifren, Aditya Gujja, Saurabh Sinha, Brian Cline, Chandarasekaran Ramamurthy, and Greg Yeric. Asap7: A 7-nm finfet predictive process design kit. *Microelectronics Journal*, 53:105–115, 2016.

[DEMS19] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2. Submission to Round 1 of the NIST Lightweight Cryptography project, 2019.

[DFI+24] Patrick Derbez, Pierre-Alain Fouque, Takanori Isobe, Mostafizar Rahman, and André Schrottenloher. Key committing attacks against aes-based AEAD schemes. *IACR Trans. Symmetric Cryptol.*, 2024(1):135–157, 2024.

[DGRW18] Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. Fast Message Franking: From Invisible Salamanders to Encryptment. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 155–186. Springer, 2018.

[ENP19] Maria Eichlseder, Marcel Nageler, and Robert Primas. Analyzing the linear keystream biases in AEGIS. *IACR Trans. Symmetric Cryptol.*, 2019(4):348–368, 2019.

[Fac16] Facebook. *Messenger Secret Conversations technical whitepaper*. https://fbnewsroomus.files.wordpress, 2016.

[GLR17] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. Message Franking via Committing Authenticated Encryption. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 66–97. Springer, 2017.

[GWDE15] Hannes Groß, Erich Wenger, Christoph Dobraunig, and Christoph Ehrenhöfer. Suit up! made-to-measure hardware implementations of ascon. Cryptology ePrint Archive, Paper 2015/034, 2015.

[HII+22] Akinori Hosoyamada, Akiko Inoue, Ryoma Ito, Tetsu Iwata, Kazuhiko Minematsu, Ferdinand Sibleyras, and Yosuke Todo. Cryptanalysis of rocca and feasibility of its security claim. *IACR Trans. Symmetric Cryptol.*, 2022(3):123–151, 2022.

[IKMP19] Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, and Thomas Peyrin. Romulus v1.2. *NIST Lightweight Cryptography Project*, 2019.

[JN16] Jérémy Jean and Ivica Nikolic. Efficient design strategies based on the AES round function. In *FSE*, volume 9783 of *Lecture Notes in Computer Science*, pages 334–353. Springer, 2016.

[KDH+12] Stéphanie Kerckhof, François Durvaux, Cédric Hocquet, David Bol, and François-Xavier Standaert. Towards green cryptography: A comparison of lightweight ciphers from the energy viewpoint. In *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, pages 390–407, 2012.

[LIMS21] Fukang Liu, Takanori Isobe, Willi Meier, and Kosei Sakamoto. Weak keys in reduced AEGIS and tiaoxin. *IACR Trans. Symmetric Cryptol.*, 2021(2):104–139, 2021.

[ME19] Alexander Maximov and Patrik Ekdahl. New circuit minimization techniques for smaller and faster AES sboxes. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(4):91–125, 2019.

[Mil17] Jon Millican. *Challenges of E2E Encryption in Facebook Messenger*. Real World Cryptography conference, 2017.

[Min14] Brice Minaud. Linear biases in AEGIS keystream. In *Selected Areas in Cryptography*, volume 8781 of *Lecture Notes in Computer Science*, pages 290–305. Springer, 2014.

[MMR+15] Mayler Martins, Jody Maick Matos, Renato P. Ribas, André Reis, Guilherme Schlinker, Lucio Rech, and Jens Michelsen. Open cell library in 15nm freepdk technology. In *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, ISPD '15, page 171–178, New York, NY, USA, 2015. Association for Computing Machinery.

[MV04] David A. McGrew and John Viega. The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In Anne Canteaut and Kapalee Viswanathan, editors, *Progress in Cryptology - INDOCRYPT 2004, 5th International Conference on Cryptology in India, Chennai, India, December 20-22,*

2004, Proceedings, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2004.

[Nik14] Ivica Nikolić. Tiaoxin-346: Version 2.0. CAESAR Competition, 2014.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.

[SA14] Muharrem Tolga Sakalli and Bora Aslan. On the algebraic construction of cryptographically good 32×32 binary linear transformations. *J. Comput. Appl. Math.*, 259:485–494, 2014.

[SLN+21] Kosei Sakamoto, Fukang Liu, Yuto Nakano, Shinsaku Kiyomoto, and Takanori Isobe. Rocca: An efficient aes-based encryption scheme for beyond 5g. *IACR Trans. Symmetric Cryptol.*, 2021(2):1–30, 2021.

[SLN+22] Kosei Sakamoto, Fukang Liu, Yuto Nakano, Shinsaku Kiyomoto, and Takanori Isobe. Rocca: An efficient AES-based encryption scheme for beyond 5G (full version). Cryptology ePrint Archive, Paper 2022/116, 2022. https://eprint.iacr.org/2022/116.

[SMY+23a] Chenlin Shi, Shinobu Miwa, Tongxin Yang, Ryota Shioya, Hayato Yamaki, and Hiroki Honda. Cnfet-ocl. Github Repository, Available at https://github.com/uec-hpc-lab/CNFET-OCL, 2023.

[SMY+23b] Chenlin Shi, Shinobu Miwa, Tongxin Yang, Ryota Shioya, Hayato Yamaki, and Hiroki Honda. Cnfet7: An open source cell library for 7-nm cnfet technology. In *2023 28th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 763–768, 2023.

[Tod15] Yosuke Todo. Structural Evaluation by Generalized Integral Property. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 287–314. Springer, 2015.

[TTI24] Ryunouchi Takeuchi, Yosuke Todo, and Tetsu Iwata. Key recovery, universal forgery, and committing attacks against revised rocca: How finalization affects security. *IACR Transactions on Symmetric Cryptology*, 2024(2):85–117, Jun. 2024.

[WH19] Hongjun Wu and Tao Huang. Tinyjambu. *NIST Lightweight Cryptography Project*, 2019.

[WP13a] Hongjun Wu and Bart Preneel. AEGIS: A fast authenticated encryption algorithm. *IACR Cryptol. ePrint Arch.*, page 695, 2013.

[WP13b] Hongjun Wu and Bart Preneel. AEGIS: A fast authenticated encryption algorithm. In *Selected Areas in Cryptography*, volume 8282 of *Lecture Notes in Computer Science*, pages 185–201. Springer, 2013.

A Test Vectors

Table 11 shows test vectors of AETHER.

Table 11: Test vectors of AETHER.

Key	K_0	K_1	N
Nonce	0x1640224596795a4c54550546722fc76b	0x16d3059dfc04066657a839d2d5be827b	0x51af4471e8bcf6a2704d71163021f4fd
M	M_0	M_1	M_2
	0x84070aa5e8afb486da0561b6a1b88164	0x78b67f4be34cc1f34a02ecce30813270	0x9556a6ca986a3d6d7f9c5bb40c99aa61
AD	AD_0	AD_1	AD_2
	0xe098499961971de22fec2235b24c1309	0xbab0da98c3a386daa98d918b8cd88d5d	0x2fbe54a2c06d135bf0fdc7cc81f47625
C	C_0	C_1	C_2
	0xa47cdb80f4946a4afdc25993de8708c6	0x002b1646a6c79cc8d73cd433167b216f	0xaff7f36089f14c36f49504331b998aa9
T	0xebb69bfbc2d9b02b3af0cbdaae73fd5d		
Key	K_0	K_1	N
Nonce	0x1640224596795a4c54550546722fc76b	0x16d3059dfc04066657a839d2d5be827b	0x51af4471e8bcf6a2704d71163021f4fd
M	M_0	M_1	M_2
	0x84070aa5e8afb486da0561b6a1b88164	0x78b67f4be34cc1f34a02ecce30813270	0x9556a6ca986a3d6d7f9c5bb40c99aa61
AD	AD_0	AD_1	AD_2
	-	-	-
C	C_0	C_1	C_2
	0xfefca57ffb6cb1f1715e3810698cfd49	0x91b9b892892136e7c34f0bb7d13183cb	0x526940c921e30c98b0bf74d5f0c55683
T	0xfccf3692b1af11e7f2868032f5b78fa2		
Key	K_0	K_1	N
Nonce	0x1640224596795a4c54550546722fc76b	0x16d3059dfc04066657a839d2d5be827b	0x51af4471e8bcf6a2704d71163021f4fd
M	M_0	M_1	M_2
	-	-	-
AD	AD_0	AD_1	AD_2
	0xe098499961971de22fec2235b24c1309	0xbab0da98c3a386daa98d918b8cd88d5d	0x2fbe54a2c06d135bf0fdc7cc81f47625
C	C_0	C_1	C_2
	-	-	-
T	0xf46a8b9bba90b93f4232653333b0a6d0		

B Binary Matrices with Optimal Branch Number

We show 4×4 (almost MDS), 8×8, 16×16, 32×32 binary matrices compared in Sect. 3.2.1, whose branch numbers are optimal

4×4 (almost MDS) binary matrix [BBI+15]. Eq. (2) shows 4×4 (almost MDS) binary matrix used in Midori, whose branch number is 4.

$$\boldsymbol{M_4} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}. \tag{2}$$

8×8 binary matrix [AS14]. Eq. (3) shows 8×8 binary matrix proposed by Aslan et al.,

whose branch number is 5.

$$\boldsymbol{M_8} = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}. \tag{3}$$

16 × 16 binary matrix [ARSÖ17]. Eq. (4) shows 16 × 16 binary matrix proposed by Akleylek et al., whose branch number is 8.

$$\boldsymbol{M_{16}} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4}$$

32 × 32 binary matrix [ARSÖ17]. Eq. (5) shows 32 × 32 binary matrix proposed by Akleylek et al., whose branch number is 12.

$$\boldsymbol{M_{32}} = \begin{pmatrix} \boldsymbol{M_a} & \boldsymbol{M_b} & \boldsymbol{M_c} & \boldsymbol{M_d} \\ \boldsymbol{M_b} & \boldsymbol{M_a} & \boldsymbol{M_d} & \boldsymbol{M_c} \\ \boldsymbol{M_c} & \boldsymbol{M_d} & \boldsymbol{M_a} & \boldsymbol{M_b} \\ \boldsymbol{M_d} & \boldsymbol{M_c} & \boldsymbol{M_b} & \boldsymbol{M_a} \end{pmatrix}, \tag{5}$$

where $\boldsymbol{M_a}$, $\boldsymbol{M_b}$, $\boldsymbol{M_c}$, and $\boldsymbol{M_d}$ are as follows:

$$\boldsymbol{M_a} = \begin{pmatrix} 0&0&0&0&0&0&0&1\\ 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0 \end{pmatrix}, \boldsymbol{M_b} = \begin{pmatrix} 0&0&0&0&0&0&1&1\\ 0&0&0&0&0&0&0&1\\ 1&0&0&0&0&0&0&0\\ 1&1&0&0&0&0&0&0\\ 0&1&1&0&0&0&0&0\\ 0&0&1&1&0&0&0&0\\ 0&0&0&1&1&0&0&0\\ 0&0&0&0&1&1&0&0 \end{pmatrix},$$

$$\boldsymbol{M_c} = \begin{pmatrix} 1&0&0&0&0&0&1&0\\ 0&1&0&0&0&0&0&1\\ 1&0&1&0&0&0&0&0\\ 0&1&0&1&0&0&0&0\\ 0&0&1&0&1&0&0&0\\ 0&0&0&1&0&1&0&0\\ 0&0&0&0&1&0&1&0\\ 0&0&0&0&0&1&0&1 \end{pmatrix}, \boldsymbol{M_d} = \begin{pmatrix} 0&1&1&1&1&1&0&1\\ 1&0&1&1&1&1&1&0\\ 0&1&0&1&1&1&1&1\\ 1&0&1&0&1&1&1&1\\ 1&1&0&1&0&1&1&1\\ 1&1&1&0&1&0&1&1\\ 1&1&1&1&0&1&0&1\\ 1&1&1&1&1&0&1&0 \end{pmatrix}.$$

C State Recovery Attack

Here, we analyze AETHER on the basis of the state-recovery attack on Rocca provided in [HII+22]. First, we estimate the number of pair of states that can be recovered from a given input and output difference of the F function and the corresponding complexity.

Estimation on the number of possible pair of states for a fixed input and output difference of F function. Refer to the illustration of F function shown in Fig. 2. Consider the initial (input to F) and final state as T_1 and T_4, respectively. The state after the application of first S-box and the linear layer M_b be T_2 and T_3, respectively. Assume that a differential at T_1 and a corresponding differential at T_4 is known. We want to estimate the possible number of pair of 128-bit states that satisfies the differentials while propagating through the F function.

Consider the difference distribution table of the S-box (in Table 1) given in Table 12. There are six input differences for which there are six different output differences of the S-box. Additionally, there are six and three input differences of the S-box which corresponds to seven and eight different output differences, respectively. For a fixed input difference, on an average there are $(\frac{6}{15} \times 6 + \frac{6}{15} \times 7 + \frac{3}{15} \times 8) \approx 2^{2.77}$ possible output differences. Thus, for a fixed input differential at T_1, there are $(2^{2.77})^{32} = 2^{88.64}$ possible differences at T_2. Due to the application of a linear layer, this number remains the same at T_3. For a fixed output difference, a random input of the S-box is valid with probability $\frac{2^{2.77}}{15} \approx 2^{-1.14}$. Hence, out of $2^{88.64}$ possible differences at T_3, $(2^{-1.14})^{32} \times 2^{88.64} = 2^{52.16}$ are valid corresponding to the output difference.

For a valid input-output difference of the S-box, on an average there are $\frac{84*2+18*4}{84+18} \approx 2^{1.23}$ solutions. Thus, there will be in total $2^{52.16} \times (2^{1.23})^{32} = 2^{71.84}$ pair of 128-bit states at T_2 which conforms to the difference at T_1. Due to the application of linear-layer, same number of pair of states remain at T_3. Note that, all the differences corresponding to these pair of states are valid with respect to the difference at T_4. However, if we consider

Table 12: Difference Distribution Table (DDT) of S-box

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

the individual S-boxes, the input-output difference is known along with the input nibble pairs which allows to perform additional filtering. Given a pair of nibbles with fixed input difference to the S-box, the probability that those nibbles satisfies the fixed output difference of the S-box is $\frac{2^{1.23}}{16} = 2^{-2.77}$. Hence, after this filtering $2^{71.84} \times (2^{-2.77})^{32} = 2^{-16.8}$ pairs, i.e., a unique pair survives. Due to the application of the linear layer on 64 bits, the state-recovery operation of the F function can be partitioned into two parallel steps where each step is applied on half the size of the state (64 bits). Time complexity of the step depends on the handling of the $2^{88.64}$ possible output differences which can be done in parallel for each of the 16 S-boxes. Thus, the total time complexity is $2 \times (2^{2.77})^{16} = 2^{45.32}$.

Recovering the State. For mounting the state-recovery attack, we followed the strategy in [HII+22]. We consider the nonce-misuse setting for this analysis. The attack idea is represented in Fig. 8.

In the following, when we say a X is recovered/determined/fixed, we mean the exact 128-bit state is recovered. By determining ΔX, we refer to determining the difference at state X. The attack idea is outlined below:

1. Fix $\Delta C_0 \neq 0, \Delta C_1 \neq 0, \Delta C_2 \neq 0$ which gives $\Delta M_0 \neq 0, \Delta M_1 \neq 0, \Delta M_2 \neq 0$. Fix $\Delta C_i = 0$ for $3 \leq i \leq 11$. The differences $\Delta S_1[0], \Delta S_1[3], \Delta S_1[5]$ are known.

2. $\Delta U_1 = \Delta S_1[0]$, $\Delta V_1 = \Delta M_3$, $\Delta Y_1 = \Delta S_1[3] \oplus \Delta S_1[5]$ and $\Delta Z_1 = \Delta M_5$. By analyzing the input-output difference ΔU_1 and ΔV_1 of the F function, pair of states satisfying $(S_1[0] \oplus S_1[1])$ can be determined. From the relation, $S_1[4] = F(S_1[0] \oplus S_1[1]) \oplus (M_3 \oplus C_3)$, two possible states corresponding to $S_1[4]$ can be recovered. Similarly, analyzing ΔY_1 and ΔZ_1, a pair of states corresponding to $S_1[3] \oplus S_1[5]$ can be recovered which subsequently gives two possible states for $S_1[8]$ (by using the relation $S_1[8] = F(S_1[3] \oplus S_1[5]) \oplus (M_5 \oplus C_5)$).

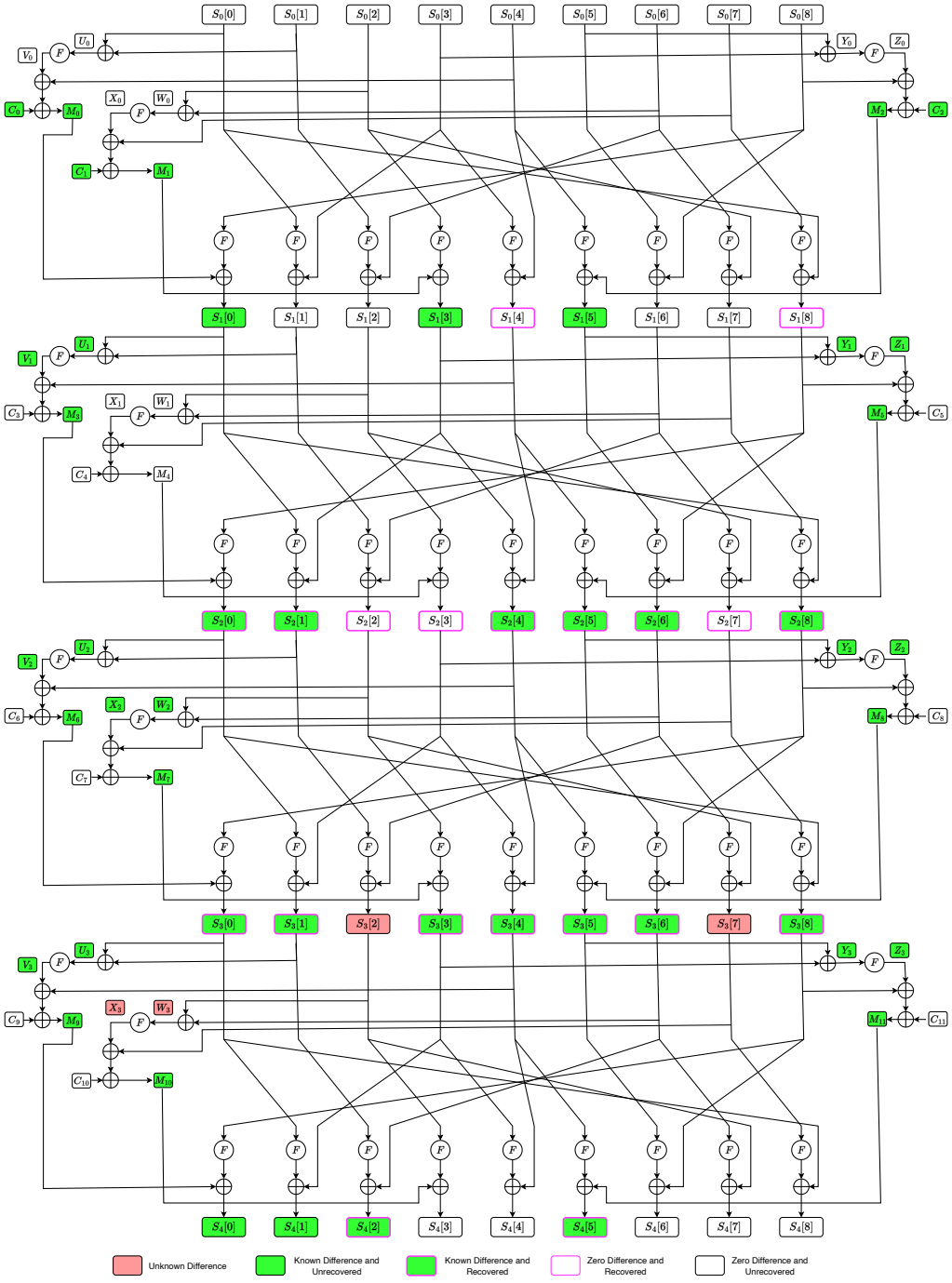


Figure 8: State-recovery Attack on AETHER.

3. From the following relations-

$$S_2[0] = F(S_1[8]) \oplus M_3$$
$$S_2[5] = F(S_1[4]) \oplus M_5,$$

values corresponding to $S_2[0]$ and $S_2[5]$ can be determined, respectively.

4. Now, $\Delta Y_2 = \Delta S_2[5]$ and $\Delta Z_2 = \Delta M_8 \oplus \Delta S_2[8]$. Again, analyzing the input-output difference of the F function, values corresponding to $(S_2[5] \oplus S_2[3])$ and $S_2[8]$ can be recovered. As $S_2[5]$ is already known, $S_2[3]$ can be determined.

5. From the following relations-

$$S_3[1] = F(S_2[0]) \oplus S_2[3]$$
$$S_3[6] = F(S_2[5]) \oplus S_2[8]$$
$$S_3[0] = F(S_2[8]) \oplus M_6,$$

values corresponding to $S_3[1]$, $S_3[6]$ and $S_3[0]$ can be determined, respectively.

6. As both $S_3[0]$ and $S_3[1]$ are known, we can determine $F(S_3[0] \oplus S_3[1])$ (i.e., V_3 is known). Thus, using M_9, C_9 and V_3, the states in $S_3[4]$ can be recovered.

7. Subsequently, values corresponding to $S_2[4]$ and $S_3[5]$ can be recovered (step-by-step) by analyzing the following relations

$$S_3[4] = F(S_2[3]) \oplus S_2[4]$$
$$S_3[5] = F(S_2[4]) \oplus M_8.$$

8. As $\Delta S_3[3]$ and $S_3[5]$ are known, ΔY_3 can be determined. As ΔZ_3 is also known, we can determine the values corresponding to $S_3[8]$ and $(S_3[3] \oplus S_3[5])$ by analyzing the function F. As $S_3[5]$ is known, we can determine $S_3[3]$.

9. From the relation $S_2[7] = F^{-1}(S_3[8] \oplus S_2[0])$, we can determine $S_2[7]$. Similarly, using the relation $S_3[3] = F(S_2[2]) \oplus M_7$, $S_2[2]$ can be determined.

10. As $S_2[7]$, M_7 and C_7 are known, we can retrieve the states corresponding to X_2. This allows to determine the states for W_2. From $S_2[2]$, $S_2[6]$ can be determined.

11. Similarly using $S_2[4]$, M_6 and C_6, first V_2 and then U_2 can be determined. Again, using $S_2[0]$, $S_2[1]$ can be determined.

The attack idea involves determining the states corresponding to the input-output difference of the function F in Step 2, Step 4 and Step 8. Thus, using two decryption queries several candidate states corresponding to $S[3]$ can be determined at a time complexity of $4 \times 2^{45.32} = 2^{47.32}$. However, to determine the exact state for a particular nonce (out of the possible states), more nonce-repeated queries are required.

D State Interpolation

We apply here the state interpolation technique [TTI24] which is shown in Fig. 9. By using this strategy, we can fix the internal state to an arbitrary state from a given state by fixing the appropriate values for M_i $(0 \leq i \leq 8)$. In the following, we show the substates $S_{3,i}$ in terms of $S_{0,i}$ and M_i's. By subsequently fixing M_3, M_4, M_1, M_2, M_0, M_5, M_7, M_8 and M_6, the substates $S_3[8]$, $S_3[1]$, $S_3[4]$, $S_3[7]$, $S_3[2]$, $S_3[6]$, $S_3[3]$, $S_3[5]$ and $S_3[0]$ can be fixed to an arbitrary value, respectively.

$$
\begin{aligned}
S_3[0] &= M_6 \oplus F(F(F(S_0[6]) \oplus S_0[2]) \oplus M_0 \oplus F(S_0[8])) \\
S_3[1] &= F(M_3 \oplus F(F(S_0[7]) \oplus S_0[0])) \oplus M_4 \oplus F(F(S_0[1]) \oplus S_0[6]) \\
S_3[2] &= F(F(M_0 \oplus F(S_0[8])) \oplus M_1 \oplus F(S_0[2])) \oplus F(M_2 \oplus F(S_0[4])) \oplus F(S_0[7]) \oplus S_0[0] \\
S_3[3] &= M_7 \oplus F(F(F(S_0[0]) \oplus S_0[3]) \oplus F(S_0[5]) \oplus S_0[8]) \\
S_3[4] &= F(M_4 \oplus F(F(S_0[1]) \oplus S_0[6])) \oplus F(M_1 \oplus F(S_0[2])) \oplus F(S_0[3]) \oplus S_0[4] \\
S_3[5] &= M_8 \oplus F(F(M_1 \oplus F(S_0[2])) \oplus F(S_0[3]) \oplus S_0[4]) \\
S_3[6] &= F(M_5 \oplus F(F(S_0[3]) \oplus S_0[4])) \oplus F(F(S_0[6]) \oplus S_0[2]) \oplus M_0 \oplus F(S_0[8]) \\
S_3[7] &= F(F(M_2 \oplus F(S_0[4])) \oplus F(S_0[7]) \oplus S_0[0]) \oplus F(F(S_0[0]) \oplus S_0[3]) \oplus F(S_0[5]) \oplus S_0[8] \\
S_3[8] &= F(F(F(S_0[5]) \oplus S_0[8]) \oplus F(S_0[1]) \oplus S_0[6]) \oplus M_3 \oplus F(F(S_0[7]) \oplus S_0[0])
\end{aligned}
$$

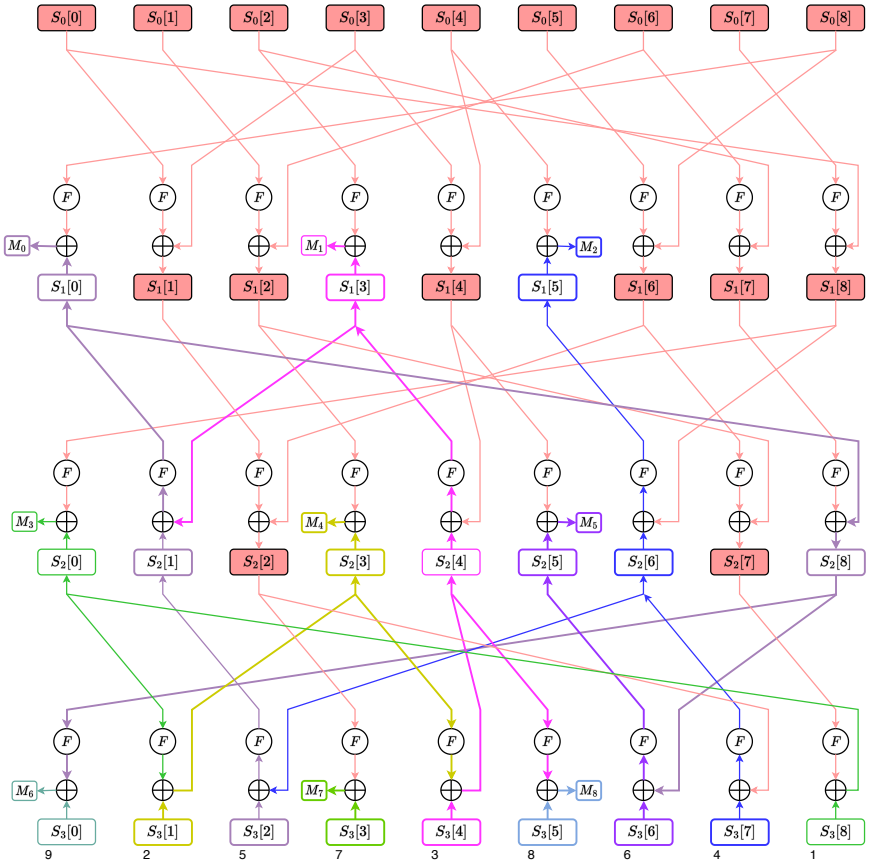


Figure 9: State Interpolation on AETHER

E Security in the RUP Settings

Typically, for an authenticated encryption (AE) cipher, when the tag is computed over the plaintext instead of the ciphertext, secure memory must retain the decrypted plaintext until verification is complete. This presents challenges for resource-constrained IoT devices, as attackers may exploit insecure memory to access unverified plaintexts, compromising

security. Some real-time protocols also release plaintext in segments to reduce latency and storage demands. Thus, analyzing the security of AE under such scenarios is essential for these applications. The security analysis in such contexts is formalized in [ABL+14] and is referred to as the *releasing unverified plaintext* (RUP) setting. In this setting, an adversary can decrypt several ciphertexts under the same nonce. For a state-recovery attack in RUP settings, analysis similar to the one discussed in Appendix C can be followed. However, as discussed in Section 4.5, such state recovery does not lead to key-recovery attacks due to the introduction of the keys in the finalization step. Therefore, we believe that AETHER resists key-recovery attacks in the RUP setting.

F Key-Committing Variant

Key commitment ensures that a ciphertext C can be decrypted successfully only with the exact key used to encrypt the corresponding plaintext. In cryptographic systems, if a ciphertext could be decrypted to valid plaintexts using two different keys, it would violate the key commitment principle. This principle is crucial for AEAD schemes. The requirement for an AEAD scheme to be key-committing, while also providing confidentiality and integrity, is studied in [GLR17, DGRW18] within the context of Facebook message franking [Fac16, Mil17].

The proposed scheme AETHER is not key-committing. In fact, by applying the technique described in [DFI+24], a deterministic attack can be mounted on AETHER (the attack directly follows from the state interpolation technique described in Appendix D). However, several strategies have been proposed in the literature to provide key-committing security while using a non-committing AEAD [CR22, BH22, ADG+22]. In the current context, we suggest the methodology in [ADG+22] which proposes a solution involving two PRF calls and ciphertext expansion by one block. Let F_{enc} and F_{com} be two independent PRFs, and let $ENC(K, N, A, M)$ be a non-committing AEAD where K, N, A and M are the key, nonce, associated data and plaintext, respectively. Then ENC can be transformed into a key-committing AEAD in the following way:

$$
\begin{aligned}
&K_{enc} \leftarrow F_{enc}(K, N)\\
&K_{com} \leftarrow F_{com}(K, N)\\
&C \leftarrow Enc(K_{enc}, N, A, M)\\
&ret(C, K_{com})
\end{aligned}
$$

For PRF calls, Gleeok [ABC+24] can be employed. Note that, as AETHER is specifically designed for encryption of very large data, in that context, two PRF calls will have negligible impact.

G Software Evaluation

In this section, we evaluate the performance of AETHER in software. For the comparison to the existing AEADs, we also evaluate AEGIS-256, Rocca-S, and AES-256-GCM on the X86_64 processor (Intel Core i9-12900K) and ARM processor (Apple M2). The evaluation environment is the same as in [SLN+21, ABC+23], i.e., all evaluations are carried out on OpenSSL 3.1.0-dev and measured their performances with `speed` command. To implement the existing AEAD schemes for the X86_64 processor, we use the publicly available codes of Rocca-S[8] and AEGIS-256[9] and the pre-provided code by OpenSSL of AES-256-GCM, all

[8] https://github.com/yt-nakano/Rocca-S-openssl
[9] https://github.com/floodyberry/supercop

of which are implemented by SIMD instructions. For the implementation of AETHER, we use SIMD instructions only in the round-update function because the inner function consists of the nibble-wise operations that are hard to implement by SIMD instructions. For ARM implementations, we implement AETHER, Rocca, AEGIS-256, and AES-256-GCM by NEON instructions, but the inner function of AETHER does not use them due to the same reason as the X86_64 implementation.

Table 13 shows the results of our evaluations on Intel Core i9-12900K and Apple M2. As can be seen in Table 13, the performance of AETHER is not good in software because AETHER is not designed to make use of the advantage of SIMD and NEON instructions, while other evaluated AEAD schemes are designed to leverage these fully. Moreover, they can use the special instruction sets of AES, such as AES-NI, to conduct the AES round function quite fast in software. In contrast, for AETHER, we can use SIMD and NEON instructions only for the part of the round-update function, not including the inner function. The significant gaps in the results between AETHER and the other AEAD schemes come from the fact that the parallel application of the inner functions takes much more cost than conducting the AES round functions. We expect this is inevitable because the round-update function of AETHER has nibble-wise operations to realize the high throughput and low-energy consumption in hardware, making software-friendly implementations, especially using SIMD and NEON operations, much harder.

Table 13: Measurement comparison between AETHER and related AEAD schemes in software. The input size for the short measurements consisted of 1024 bits of AD and 2048 bits of plaintext while the long input size was 1024 bits of AD and 1.28 Mbits of plaintext. All results are given in Gbps.

Algorithms	Intel Core i9-12900K		Apple M2	
	Short	Long	Short	Long
AETHER	0.0083	0.1104	0.0132	0.1736
AES-256-GCM	5.5564	39.5520	9.7123	48.1241
AEGIS-256	8.0177	31.6024	7.0807	43.5146
Rocca-S	8.2823	132.0542	7.9294	88.0408

H Side-Channel Hardening

We briefly sketch a first-order Threshold Implementation [NRR06] of AETHER that is based on the four-share decomposition of its S-box guaranteeing the security of the circuit in the glitch-extendend probing model.

Any Threshold Implementation can be reduced down to the splitting of t-degree Boolean functions into at least $t+1$ component functions such that each one of them is independent of at least one input share, which is a straightforward task for both linear and non-linear functions. However, the real obstacle in Threshold Implementation is making sure these component function do not skew the distribution of the output shares, i.e., uniformly distributed input shares result in output shares that equally follow a uniform distribution. Often, to fulfill this requirement, Threshold Implementations resort to increasing the number of shares resulting in decompositions of t-degree functions with $> t+1$ shares. In the case of the AETHER 4-bit S-box, which it shares with Gleeok [ABC+24] whose output bits are all cubic functions a straightforward 4-shared Threshold Implementation is possible (the reader is referred to [ABC+24] for a more thorough description of the derivation of this sharing).

Having decomposed the S-box into four TI-conforming component functions and the trivial sharing of the remaining linear functions in AETHER, we can derive a straightforward first-order protected circuit where the state is quadrupled into four shares (one for each component function).