IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2021, No. 2, pp. 1–30. DOI:10.46586/tosc.v2021.i2.1-30

Rocca: An Efficient AES-based Encryption Scheme for Beyond 5G

Kosei Sakamoto[1] and Fukang Liu[1,2] and Yuto Nakano[3] and Shinsaku Kiyomoto[3] and Takanori Isobe[1,4,5]

[1] University of Hyogo, Kobe, Japan. takanori.isobe@ai.u-hyogo.ac.jp, liufukangs@gmail.com, k.sakamoto0728@gmail.com
[2] East China Normal University, Shanghai, China
[3] KDDI Research, Fujimino, Japan. yuto@kddi-research.jp, kiyomoto@kddi-research.jp
[4] National Institute of Information and Communications Technology (NICT), Tokyo, Japan
[5] PRESTO, Japan Science and Technology Agency, Tokyo, Japan

Abstract. In this paper, we present an AES-based authenticated-encryption with associated-data scheme called Rocca, with the purpose to reach the requirements on the speed and security in 6G systems. To achieve ultra-fast software implementations, the basic design strategy is to take full advantage of the AES-NI and SIMD instructions as that of the AEGIS family and Tiaoxin-346. Although Jean and Nikolić have generalized the way to construct efficient round functions using only one round of AES (aesenc) and 128-bit XOR operation and have found several efficient candidates, there still seems to exist potential to further improve it regarding speed and state size. In order to minimize the critical path of one round, we remove the case of applying both aesenc and XOR in a cascade way for one round. By introducing a cost-free block permutation in the round function, we are able to search for candidates in a larger space without sacrificing the performance. Consequently, we obtain more efficient constructions with a smaller state size than candidates by Jean and Nikolić. Based on the newly-discovered round function, we carefully design the corresponding AEAD scheme with 256-bit security by taking several reported attacks on the AEGIS family and Tiaxion-346 into account. Our AEAD scheme can reach 138Gbps which is 4 times faster than the AEAD scheme of SNOW-V. Rocca is also much faster than other efficient schemes with 256-bit key length, e.g. AEGIS-256 and AES-256-GCM. As far as we know, Rocca is the first dedicated cryptographic algorithm targeting 6G systems, i.e., 256-bit key length and the speed of more than 100 Gbps.

Keywords: AES-NI · Fast Software Implementation · 6G · AEAD

1 Introduction

1.1 Background

The fifth-generation mobile communication systems (5G) have been launched in several countries for commercial services since 2020. Besides, researches for beyond-5G or 6G have been already started in some research institutes. As the first white paper of 6G, [LaL19] was published by the 6Genesisi project in 2019, which is mainly organized by the University of Oulu in Finland. In the white paper, several requirements for 6G systems are raised. For the data transmission speed, it says that 6G achieves more than 100 Gbps, which is more than 10 times faster than that of 5G.

For the 4G system, as underlying cryptographic algorithms to ensure confidentiality and integrity, SNOW 3G [SAG06], AES [Nat01], and ZUC-128 [SAG11] are employed, which

Received: 2021-03-01 Accepted: 2021-05-01 Published: 2021-06-11

are specified as 128-EEA1 (EIA1), 128-EEA2 (EIA2), 128-EEA3 (EIA3), respectively, and these algorithms are also selected cryptographic algorithms for the 5G system as 128-NEA1 (NIA1), 128-NEA2 (NIA2), 128-NEA3 (NIA3). However, for the 5G system, the 3GPP standardization organization requires to increase the security level to 256-bit key lengths. In 2018, ZUC-256 [The18] was proposed as the 256-bit key version of ZUC-128. ZUC-256 was revised only in the initialization phase and in the MAC generation phase from ZUC-128. By this revise, ZUC-256 improves the security level against the key-recovery attack to the 256-bit security from the 128-bit security. On the other hand, the performance of the encryption/decryption speed is not quite improved because the key-stream generation phase is the same as ZUC-128, and a structural weakness was found [YJM20]. In FSE 2020, Ekdahl *et al.* proposed SNOW-V that is the 256-bit key version of SNOW 3G, and they showed that SNOW-V achieves more than 38 Gbps at an AEAD (Authenticated Encryption with Associated Data) mode on OpenSSL [EJMY19]. The performances of SNOW-V are sufficient for them to be used in the 5G system.

However, when taking requirements in 6G systems into account, we have to tackle some challenges. The biggest one is the encryption/decryption speed. For 6G systems, as the data transmission speed is expected to reach more than 100 Gbps, we have to design a cryptographic algorithm with the encryption/decryption speed of more than 100 Gbps, which is at least three times faster than SNOW-V. Besides, achieving 256-bit security against key-recovery attacks is essential as in 5G systems [3GP18]. In addition, due to the increase of data transmissions in 6G systems, it is necessary to ensure at least 128-bit security against distinguishing attacks while SNOW-V only claims 64-bit security against distinguishing attacks. Therefore, there is no doubt that a new cryptographic algorithm is needed in 6G systems.

For symmetric-key primitives targeting high-performance applications, there are several interesting cryptographic algorithms. The most tempting ones are those employing AES-NI [Gue10, Corb], which is a new AES instruction set equipped on many modern CPUs from Intel and AMD. Some SoCs for mobile devices are also equipped with an instruction set for AES [arm21], and more and more SoCs will support the instruction by the time 6G system is realized. Hence employing AES-NI seems reasonable in designing cryptographic algorithms for 6G systems. The AEGIS family and Tiaoxin-346 belongs to such a category, which are two submissions to the CAESAR competition [cae18] and AEGIS-128 has been selected in the final portfolio for high-performance applications. The round functions of the AEGIS family and Tiaoxin-346 are quite similar. Specifically, they are only based on the usage of one AES round and the 128-bit XOR operation, both of which have been realized with one instruction on SIMD (Single Instruction, Multiple Data) instructions. As a result, both the AEGIS family and Tiaoxin-346 are competitive in terms of encryption/decryption speed in a pure software environment, if compared with many primitives.

Jean and Nikolić generalized the method to design efficient round functions as that used in AEGIS and Tiaoxin-346 in [JN16]. After a thorough search, they discovered round functions that can achieve a faster speed than any of the round functions adopted in the AEGIS family and Tiaoxin-346 and provide the 128-bit security against forgery attacks. However, they did not propose a concrete AEAD scheme [JN16].

Obviously, AEGIS-128, AEGIS-128L and Tiaoxin-346 do not meet the security requirement of the 256-bit key length in 6G systems. In addition, according to our experiments, AEGIS-256 does not reach more than 100 Gbps (See Sect. 5). However, those researches leave us the potential of designing the faster cryptographic algorithm based on AES round functions for 6G.

1.2 Our Design

In this paper, we present an AES-based encryption scheme with a 256-bit key and 128-bit tag called Rocca, which provides both a raw encryption scheme and an AEAD scheme

with a 128-bit tag. The goal of Rocca is to meet the requirement in 6G systems in terms of both performance and security. For performance, Rocca achieves an encryption/decryption speed of more than 100 Gbps in both raw encryption scheme and AEAD scheme. For security, Rocca can provide 256-bit and 128-bit security against key-recovery attacks and forgery attacks, respectively.

Optimized AES-NI-Friendly Round Function To achieve such a dramatically fast encryption/decryption speed, Rocca is designed for a pure software environment that can fully support both the AES-NI and SIMD instructions. The design of the round function of Rocca is inspired by the work of Jean and Nikolić [JN16]. To further increase its speed and reduce the state size, we explore a new class of AES-based structures. Specifically, we take the following different approaches.

- To minimize the critical path of the round function, we focus on the structure where each 128-bit block of the internal state is updated by either one AES round or XOR while Jean and Nikolić consider the case of applying both aesenc and XOR in a cascade way for one round, and most efficient structures in [JN16] are included in this class.
- We introduce a permutation between the 128-bit state words of the internal state in order to increase the number of possible candidates while keeping efficiency as executing such a permutation is a cost-free operation in the target software, which was not taken into account in [JN16].

We search for round functions that can ensure 128-bit security against forgery attacks in a class of our general constructions as with [JN16]. Consequently, we succeed in discovering more efficient constructions with a smaller state size than those in [JN16]. The internal state of Rocca consists of eight 128-bit words and its round function is composed of 4 aesencs and 4 128-bit XOR operations, which is significantly faster than those of the AEGIS family, Tiaxion-346 and Jean and Nikolić's structure [JN16].

Encryption and Authentication Scheme. To resist against the statistical attack in [Min14], generating each 128-bit ciphertext block will additionally require one AES round, while it is generated with simple quadratic boolean functions in the AEGIS family and Tiaxion-346. However, such a way will have few overhead by AES-NI (See Sect. 3). Moreover, a study on the initialization phases for both reduced AEGIS-128 and Tiaoxin-346 has been reported recently [LIMS21]. To further increase the resistance against the reported attacks, how to place the nonce and the key at the initial state is carefully chosen in our scheme.

Performance The encryption/decryption speed of Rocca is dramatically improved compared with other AES-based encryption schemes. Rocca is more than three and four times faster than SNOW-V and SNOW-V-GCM, respectively, i.e. the speed reaches 160 and 138 Gbps, respectively. Compared to other schemes with 256-bit key, Rocca is more than two times faster than AEGIS-256 and more than four times faster than AES-256-GCM in our evaluations (See Sect. 5). Moreover, Rocca is also faster than AEGIS-128, AEGIS-128L, and Tiaoxin-346 even though Rocca provides a higher security level. To the best of our knowledge, Rocca is the first dedicated cryptographic algorithm targeting 6G systems and we hope it can inspire future designs.

Organization This paper is organized as follows. We first present the specification of Rocca in Sect. 2. Then, we describe the design rationale, such as the general construction based on AES-NI, criteria for performance and security, and how to find efficient round functions in Sect. 3. In Sect. 4, we provide the details of security evaluations against

possible attacks on Rocca. Sect. 5 shows our software implementation results. Finally, we conclude this paper in Sect. 6.

2 Preliminaries

In this section, the notations and the specification of our designs will be described.

2.1 Notations

The following notations will be used in the paper. Throughout this paper, a block means a 16-byte value. For the constants Z_0 and Z_1, we utilize the same ones as Tiaoxin-346 [Nik14].

1. S: The state of Rocca, which is composed of 8 blocks, i.e. $S = (S[0], S[1], \ldots, S[7])$, where $S[i]$ $(0 \leq i \leq 7)$ are blocks and $S[0]$ is the first block.
2. Z_0: A constant block defined as $Z_0 =$ `428a2f98d728ae227137449123ef65cd`.
3. Z_1: A constant block defined as $Z_1 =$ `b5c0fbcfec4d3b2fe9b5dba58189dbbc`.
4. $\mathsf{AES}(X, Y)$: One AES round applied to the block X, where the round constant is Y, as defined below:

$$\mathsf{AES}(X, Y) = (\text{MixColumns} \circ \text{ShiftRows} \circ \text{SubBytes}(X)) \oplus Y,$$

   where MixColumns, ShiftRows and SubBytes are the same operations as defined in AES.
5. $A(X)$: The AES round function without the constant addition operation, as defined below:

$$A(X) = \text{MixColumns} \circ \text{ShiftRows} \circ \text{SubBytes}(X),$$

6. $|X|$: The length of X in bits.
7. 0^l: A zero string of length l bits.
8. $X||Y$: The concatenation of X and Y.
9. $R(S, X_0, X_1)$: The round function used to update the state S.

2.2 The Round Update Function

The input of the round function $R(S, X_0, X_1)$ of Rocca consists of the state S and two blocks (X_0, X_1). If denoting the output by S^{new}, $S^{new} \leftarrow R(S, X_0, X_1)$ can be defined as follows:

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

The corresponding illustration can be referred to Figure 1.

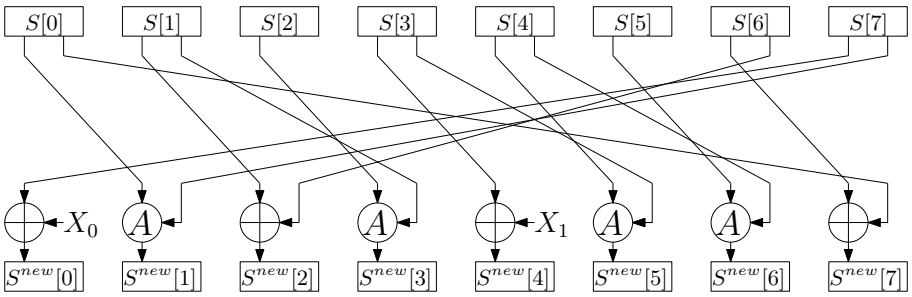

Figure 1: Illustration of the round function

2.3 Specification of Rocca

Rocca is an authenticated-encryption with associated-data scheme composed of four phases: initialization, processing the associated data, encryption and finalization. The input consists of a 256-bit key $K_0||K_1 \in \mathbb{F}_2^{128} \times \mathbb{F}_2^{128}$, a 128-bit nonce N, the associated data AD and the message M. The output is the corresponding ciphertext C and a 128-bit tag T. Define $\overline{X} = X||0^l$ where l is the minimal non-negative integer such that $|\overline{X}|$ is a multiple of 256. In addition, write X as $X = X_0||X_1||\ldots||X_{\frac{|X|}{256}-1}$ with $|X_i| = 256$. Further, X_i is written as $X_i = X_i^0||X_i^1$ with $|X_i^0| = |X_i^1| = 128$.

Initialization. First, (N, K_0, K_1) is loaded into the state S in the following way:

$$S[0] = K_1, S[1] = N, S[2] = Z_0, S[3] = Z_1,$$
$$S[4] = N \oplus K_1, S[5] = 0, S[6] = K_0, S[7] = 0.$$

Then, 20 iterations of the round function $R(S, Z_0, Z_1)$ is applied to the state S.

Processing the associated data. If AD is empty, this phase will be skipped. Otherwise, AD is padded to $\overline{AD}$ and the state is updated as follows:

$$\begin{aligned}&\textsf{for } i = 0 \textsf{ to } d-1\\ &\quad R(S, \overline{AD}_i^0, \overline{AD}_i^1),\\ &\textsf{end for}\end{aligned}$$

where $d = \frac{|\overline{AD}|}{256}$.

Encryption. The encryption phase is similar to the phase to process the associated data. If M is empty, the encryption phase will be skipped. Otherwise, M is first padded to $\overline{M}$ and then $\overline{M}$ will be absorbed with the round function. During this procedure, the ciphertext C is generated. If the last block of M is incomplete and its length is b bits, i.e. $0 < b < 256$, the last block of C will be truncated to the first b bits. A detailed description is shown below:

$$\begin{aligned}&\textsf{for } i = 0 \textsf{ to } m-1\\ &\quad C_i^0 = \textsf{AES}(S[1], S[5]) \oplus \overline{M}_i^0,\\ &\quad C_i^1 = \textsf{AES}(S[0] \oplus S[4], S[2]) \oplus \overline{M}_i^1,\\ &\quad R(S, \overline{M}_i^0, \overline{M}_i^1),\end{aligned}$$

end for

where $m = \frac{|\overline{M}|}{256}$.

Finalization. After the above three phases, the state S will again pass through 20 iterations of the round function $\mathsf{R}(S, |AD|, |M|)$ and then the tag is computed in the following way:

$$T = \sum_{i=0}^{7} S[i].$$

A formal description of Rocca can be seen in Algorithm 1 and the corresponding illustration is shown in Figure 2.

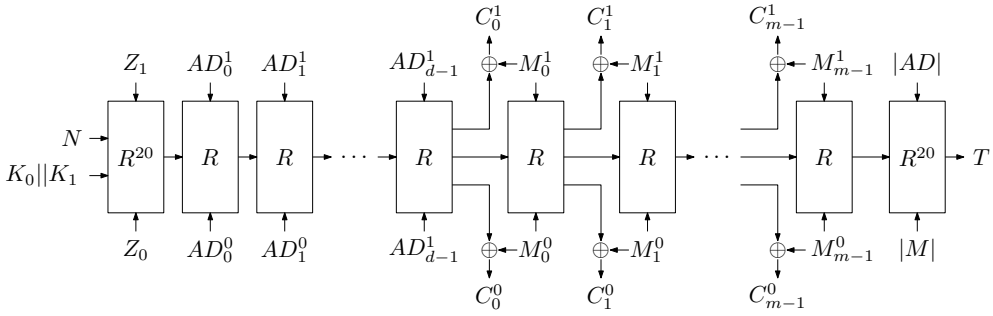

Figure 2: The procedure of Rocca

A raw encryption scheme. If the phases of processing the associated data and finalization are removed, a raw encryption scheme is obtained.

Security claims. Rocca provides 256-bit security against key-recovery and distinguishing attacks and 128-bit security against forgery attacks in the nonce-respecting setting. We do not claim its security in the related-key and known-key settings.

3 Design Rationale

3.1 General Construction

SIMD instruction. The prime design goal of Rocca is to meet the requirements of processing/transmission speed for 6G applications, namely more than 100 Gbps [LaL19]. In order to realize fast encryption/decryption speed (hereafter, we simply call "speed") on a pure software environment, we take full advantage of the SIMD instructions and the AES-NI, both of which are equipped on most of modern CPUs. The SIMD instructions contains some fundamental instructions such as XOR and AND, and can execute them by 32/64/128-bit units as one instruction, where the AES-NI is a special set of the SIMD instructions, which is first rolled out by Intel [Cora] and available on modern processors. The AES-NI can execute AES about 10 times faster than non-AES-NI in parallelizable modes such as CTR mode. In this paper, we utilize on aesenc, which is one of instruction sets of AES-NI, and performs one regular (not the last) round of AES on an input state S with a subkey K:

$$\mathsf{aesenc}(S, K) = (\mathsf{MixColumns} \circ \mathsf{ShifRows} \circ \mathsf{SubBytes}(S)) \oplus K.$$

Algorithm 1 The specification of Rocca

procedure $\texttt{RoccaEncrypt}(K_0, K_1, N, AD, M)$
  $S \leftarrow \texttt{Initialization}(N, K_0, K_1)$
  if $|AD| > 0$ **then**
    $S \leftarrow \texttt{ProcessAD}(S, \overline{AD})$
  if $|M| > 0$ **then**
    $S \leftarrow \texttt{Encrypt}(S, \overline{M}, C)$
    Truncate C
  $T \leftarrow \texttt{Finalization}(S, |AD|, |M|)$
  return (C, T)
procedure $\texttt{RoccaDecrypt}(K_0, K_1, N, AD, C, T)$
  $S \leftarrow \texttt{Initialization}(N, K_0, K_1)$
  if $|AD| > 0$ **then**
    $S \leftarrow \texttt{ProcessAD}(S, \overline{AD})$
  if $|C| > 0$ **then**
    $S \leftarrow \texttt{Encrypt}(S, \overline{C}, M)$
    Truncate M
  if $T = \texttt{Finalization}(S, |AD|, |C|)$ **then**
    return M
  else
    return $\perp$
procedure $\texttt{Initialization}(N, K_0, K_1)$
  $(S[0], S[1], S[2], S[3]) \leftarrow (K_1, N, Z_0, Z_1)$
  $(S[4], S[5], S[6], S[7]) \leftarrow (N \oplus K_1, 0, K_0, 0)$
  for $i = 0$ to 19 **do**
    $S \leftarrow R(S, Z_0, Z_1)$
  return S
procedure $\texttt{ProcessAD}(S, AD)$
  $d \leftarrow \frac{|AD|}{256}$
  for $i = 0$ to $d - 1$ **do**
    $S \leftarrow R(S, AD_i^0, AD_i^1)$
  return S
procedure $\texttt{Encryption}(S, M, C)$
  $m \leftarrow \frac{|M|}{256}$
  for $i = 0$ to $m - 1$ **do**
    $C_i^0 \leftarrow \mathsf{AES}(S[1], S[5]) \oplus M_i^0$
    $C_i^1 \leftarrow \mathsf{AES}(S[0] \oplus S[4], S[2]) \oplus M_i^1$
    $S \leftarrow R(S, M_i^0, M_i^1)$
  return S
procedure $\texttt{Finalization}(S, |AD|, |M|)$
  for $i = 0$ to 19 **do**
    $S \leftarrow R(S, |AD|, |M|)$
  $T \leftarrow 0$
  for $i = 0$ to 7 **do**
    $T \leftarrow T \oplus S[i]$
  return T

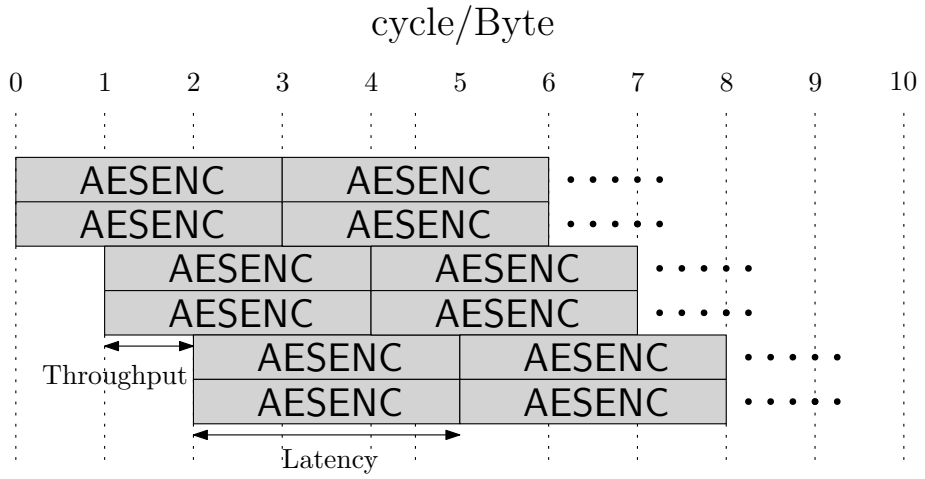

Figure 3: The process of aesenc for Intel Ice-lake.

The execution speed of these instructions can be evaluated by *latency* and *throughput*, where latency is the number of clock cycles required to execute a single instruction and throughput is the required number of clock cycles before the same instruction to be executed. It is important when considering the parallel execution. Table 1 shows latency and throughput of aesenc [RTL] in each architecture. Among existing architectures, we focus the latest architecture Intel Ice-Lake series that has the fastest AES-NI whose latency and throughput of aesenc are 3 and 0.5, respectively. Figure 3 illustrates an example of the process in the parallel execution of aesenc for Intel Ice-lake whose latency and throughput are 3 and 0.5[1], respectively.

Employing one AES round as an underlying component for future designs has a great merit for performance compared to employing other cryptographic primitives. Many software and libraries support AES-NI natively, e.g OpenSSL. Thus, it seems to be very reasonable that devices connected to 6G services will still support such instructions. SNOW-V also takes advantage of AES-NI for the same reason.

Table 1: Latency and throughput of aesenc for some architectures by Intel and AMD referred by [RTL].

Vendor	Architecture	Latency	Throughput
Intel	Sky-lake	4	1
	Kaby-lake	4	1
	Coffee-lake	4	1
	Cannon-lake	4	0.5
	Cascade-lake	4	1
	Comet-lake	unknown	unknown
	Ice-lake	3	0.5
AMD	Zen +	4	0.5
	Zen 2	4	0.5

Permutation-based Structure. As a reference point, we consider a stream cipher SNOW-V, which is designed for 5G applications. SNOW-V is based on linear feedback shift register (LFSR) and Finite State Machine(FSM) with AES-based round functions. As discussed in Section 1, if we follow this design strategy, we need to accelerate the

[1]Throughput 0.5 means that there are two ports for aesenc with throughput 1.

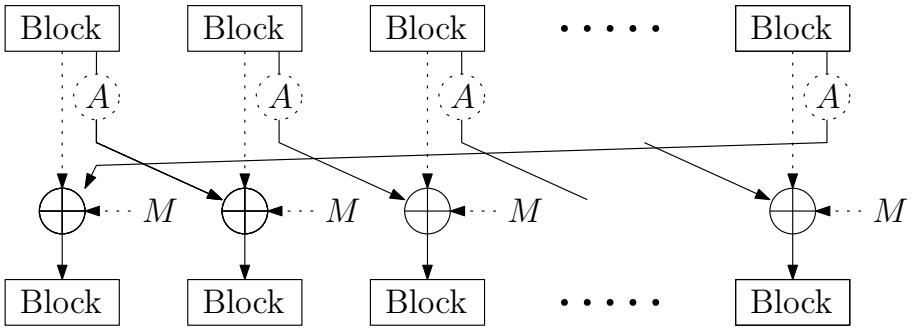

Figure 4: The general construction considered of the round function in [JN16]. Dash lines mean that it can be possible to be absent or present in the design.

performance approximately at least three times faster than SNOW-V to achieve the required performance of 100 Gbps. Thus, we decide to choose other design strategies based on AES round functions.

Specifically, we focus on AEGIS family [WP13] and Tiaoxin-346 [Nik14], which are permutation-based authenticated encryption schemes using AES round functions and submitted to CAESAR competition [cae18]. These allow a full parallelization and can achieve the outstanding speed compared to AES-CTR.

However, as it has been pointed out that there exists a linear bias in the ciphertext blocks for AEGIS-256 [Min14], it seems insecure to adopt the similar quadratic boolean function to generate the ciphertexts, especially for the purpose to reach 256-bit security. This fact motivates us to design different ways to generate the ciphertext blocks and finally involving 1 AES round function into generating each ciphertext block is chosen. Such a way is efficient due to the parallel calls to AES-NI. Moreover, a study on the initialization phases for both reduced AEGIS-128 and Tiaoxin-346 has been reported recently [LIMS21]. To further increase the resistance against the reported attacks, how to place the nonce and the key at the initial state is carefully chosen in our scheme, which is little discussed in AEGIS and Tiaoxin-346.

Efficient AES-Based Round Function. Round functions of AEGIS family [WP13] and Tiaoxin-346 [Nik14] consist of the 128-bit XOR operation and one AES round that is executed by the processor instruction aesenc. Jean and Nikolić have generalized the way to construct efficient round functions using only the one AES round (aesenc) and 128-bit XOR and have found several more efficient candidates [JN16]. Figure 4 shows the general construction of the round function considered in [JN16].

To push the limitation further of efficiency of their structures, we explore a new class of AES-based structures shown in Fig 5. Compared to the structures considered by Jean and Nikolić results [JN16], our constructions remove the case of applying both aesenc and XOR to each block in a cascade way for one round to minimize the critical path of one round. Specifically, we only consider the case of applying only either aesenc or 128-bit XOR to each block in one round, where aesenc takes a state block or message block as input of AddRoundKey and 128-bit XOR takes state block or message block as inputs, respectively as shown in Figure 5.

Moreover, we apply a block permutation to state blocks, which was not considered by Jean and Nikolić (See Fig 4). This sufficiently increases the number of possible candidates. Indeed, as described in later section, it enables us to find more efficient constructions than Jean and Nikolić's results, which is not covered by their target classes. It should be emphasized that executing the block permutation in register size is a cost-free operation, that is, the permutation only changes the order of blocks. More strictly, a permutation

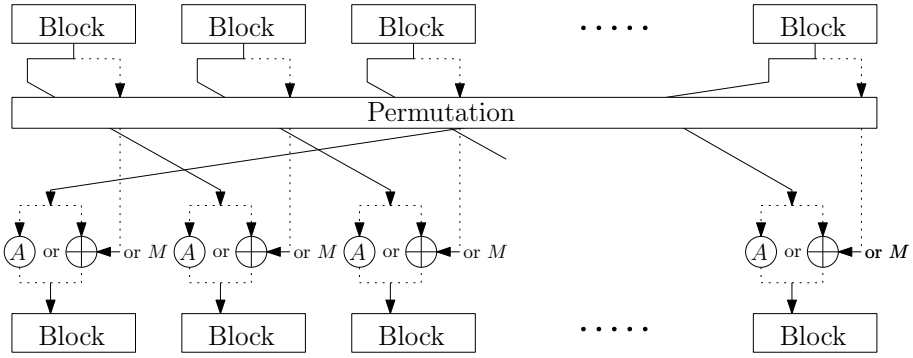

Figure 5: General construction of the round function. Dash lines mean that it can be possible to be absent or present in the design.

needs some temporary registers. However, these registers almost do not affect the speed if the total number of registers used in process of the scheme is lower than 16, which is the total number of xmm-registers equipped in almost all modern CPUs. Hence, applying a block permutation does not affect the speed of the round function. For a block that will be inputted into aesenc or XOR, we use one-block right rotation as in [JN16].

3.2 Criteria for Performance and Security

For designing efficient round functions, we need to choose several parameters such as the number of aesencs, the number of inserted message blocks, and a block permutation for our structure in Fig. 5. We clarify requirements of performance and security for target applications to choose these parameters.

Requirements for Performance. To theoretically estimate speed, we utilize a metric called *rate*, which is proposed by Jean and Nikolić [JN16].

Definition 1 (Rate [JN16])**.** *The rate p of a design is the number of* AES *rounds (calls to* aesenc*) used to process a 128-bit message.*

For our general construction of Fig 5, the *rate p* is estimated as a ratio of (# of aesencs)/(# of the inserted 128-bit messages) in one round. Since a smaller *rate* leads to more efficient design [JN16], we should design the round function that have as small *rate* as possible. The *rate* is the most important parameter for speed.

The number of aesenc in one round is also important factor to maximize the efficiency. Jean and Nikolić claim that the number of aesenc in one round should be close to (latency)/(throughput) ratio [JN16] for the efficient design, e.g. if the latency and throughput of aesenc are respectively 3 and 0.5, the number of aesenc should be 6 in one round. The reason is when the number of aesencs is less than a (latency)/(throughput) ratio, there are empty cycles in process of aesenc. On the other hand, if the number of aesencs is the same as (latency)/(throughput) ratio, there is no empty cycles as shown in Figure 3. Since our target architecture is Ice-lake, the number of aesenc in a round should be 6.

Another important factor related to speed is the number of blocks of round functions, namely the state size. Smaller state size significantly improves the efficiency because it can reduce registers used for encryption and makes a whole process of encryption easier. We experimentally confirmed that reducing the number of blocks leads to increasing speed when the *rate* is the same. Table 2 shows our experimental result that compares three types of round functions of the *rate* 2 with the number of blocks of 8, 9, and 10, each of which is measured on Intel(R) Core(TM) i7-1068NG7 CPU @ 2.30GHz with 16 GB RAMs.

Details of these round functions are given in Appendix B. Besides, a smaller state size is a preferable feature to be deployed in wider classes of devices with keeping the efficiency. It is because this, that some CPUs, such as ones from AMD, do not support the large size register like AVX512, and the process requiring the use of many registers tends to become more complicated on these CPUs. Since the number of blocks of SNOW-V, which is our reference point, is 7, the state size should be competitive.

Table 2: Comparison of the performance of the round function having different number of blocks at the same *rate*.

# of blocks	Speed (in cycle/Byte)	rate
8	0.126717	2
9	0.147397	2
10	0.155584	2

Requirements for Security. Since evaluating the resistance to all possible attacks for all possible candidates is practically infeasible, we focus on the security against the forgery attack by the internal collision as a criteria of security when finding candidates, as with [JN16]. Especially, we impose the 128-bit security against the forgery attack on our design, i.e. our security requirement is that there are no internal collisions with a probability more than 2^{-128}. Through this paper, "forgery attacks" is meant to be a universal forgery in the nonce-respecting setting.

To evaluate the probability of the internal collision, we search the lower bound for the number of active S-boxes by a Mixed Integer Linear Programming (MILP) solver [MWGP11]. Since the maximum probability of an S-box is 2^{-6}, it is sufficient to guarantee the security against internal collisions if there are 22 active S-boxes, as it gives $2^{(-6\times 22)} < 2^{-128}$ as an estimate of differential probability. For the security against other possible attacks, we evaluate after designing a whole design, and it will be described in Sect. 4.

Summary of Our Criteria. Requirements for AES-based round function are as follows.

For speed.

Requirement 1. The lowest *rate* round function as possible that leads to faster speed.

Requirement 2. The number of aesencs in one round is close to 6.

Requirement 3. A round function with a smaller number of blocks (around 7).

For security.

Requirement 4. 128-bit security to the forgery attack by internal collision, i.e. the lower bound of active S-boxes is 22.

For comparison, Table 3 shows parameters of the round function in the AEGIS [WP13] family, Tiaoxin-346 [Nik14] and structure by Jean and Nikolić [JN16].

3.3 Finding Efficient Structures

We choose several parameters such as the number of aesencs, the number of inserted message blocks, and a block permutation to meet requirements given in Sect. 3.2. The number of possible candidates is estimated as $s! \times \binom{s}{a} \times \binom{s}{m}$ candidates where s, a, and m are # of blocks, # of aesenc, and # of message blocks, respectively. For example, it reaches $2^{35.00}$ candidates when $s = 10$, $a = 4$, and $m = 2$.

Table 3: Round functions of AEGIS family and Tiaoxin-346

Primitive	# of aesenc	# of blocks	# of inserted message blocks	rate
AEGIS-128	5	5	1	5
AEGIS-256	6	6	1	6
AEGIS-128L	8	8	2	4
Tiaoxin-346	6	13	2	3
[JN16]	6	12	3	2

Our Approach. According to Table 3, the most efficient design is Jean and Nikolić's structure whose *rate* is 2. However, their state size is quite large for our requirement. In our experiments, the round functions with a smaller *rate* require a larger number of blocks to meet the security requirement. Indeed, we cannot find any structure of *rate* 2 and less than 12 internal blocks by Jean and Nikolić's constructions (Fig.4) [JN16]. To address it, our approach is as follow.

- To expand possible candidates while keeping efficiency, we introduce a block permutation to state blocks in the round function, while Jean and Nikolić did not consider any permutation. It should be emphasized that executing the block permutation in register size is a cost-free operation.
- To further improve the efficiency, we focus on the structure in which each block in one round is applied only either aesenc or XOR to minimized the critical path of the round function.

Search Targets. When the number of inserted message blocks is m, the number of aesencs in one round should be $(6 - m)$ to satisfy requirement 2 as m aesenc is used for generating ciphertext blocks for our design to the resistance to the linear bias (details in Section 3.5). Considering requirement 1 ($rate = 2$), the only choice of m is 2, thus the number of aesencs is 4. Following requirement 3, we consider the case where # of blocks are from 6 to 8. Besides, we consider the case where $rate = 1.5$ that can not satisfy requirement 2, because the low *rate* round function might be possible to more efficient even if it does not meet requirement 2. Table 4 shows our candidates of the round function.

Table 4: Candidates of round functions which we search.

Round function	# of aesenc	# of blocks	# of message blocks	rate	# of candidates	# of searched candidates
Candidates-1	4	6	2	2	$2^{17.30}$	ALL
Candidates-2	4	7	2	2	$2^{21.82}$	ALL
Candidates-3	4	8	2	2	$2^{26.23}$	$2^{19.93}$
Candidates-4	3	6	2	1.5	$2^{17.72}$	ALL
Candidates-5	3	7	2	1.5	$2^{21.82}$	ALL
Candidates-6	3	8	2	1.5	$2^{25.91}$	$2^{19.93}$

We evaluate the lower bounds for the number of active S-boxes for Candidate-1, 2, 3, 4, 5, and 6 by a MILP solver. We can conduct exhaustive searches for Candidates-1, 2, 4, and 5 while exhaustive searches for Candidates-3 and 6 are infeasible due to too large candidates that reach $2^{26.23}$ and $2^{25.91}$ for Candidates-3 and 6, respectively. Thus, we randomly search $2^{19.93}$ candidates for both Candidate-3 and 6.

Results. As a result of an exhaustive search over Candidates-1, 2, 4, and 5, there are no round functions that satisfy the requirement 4. For candidates-6, we could not find round functions meeting requirement 4 either. For Candidates-3, we found that 100 out of $2^{19.93}$ candidates ensure active S-boxes of ≥ 22. We then evaluate a diffusion property for these 100 candidates. Then we find 22 out of 100 candidates achieve the full diffusion

after 7 rounds in nibble-wise while round functions of AEGIS-128, AEGIS-256, AEGIS-128L, and [JN16] require 7, 8, 10, and 12 rounds for the full diffusion, respectively, and the one of Tiaoxin-346 never achieve the full diffusion as the state consists of three independent chucks.

We finally choose the round function shown in Fig 1 as the one of Rocca, which ensures active S-boxes of 24 that is the largest number of active S-boxes among 22 candidates. This evaluation requires about 23 days on three computers equipped with 48/64/64 cores and 256/256/256 GB RAMs.

Table 5 compares the speed of round functions of Rocca and other primitives, where speed is estimated as the average value of the round function executed 1000000 times with 64kB messages on Intel(R) Core(TM) i7-1068NG7 CPU @ 2.30GHz with 16 GB RAMs. Our round function is the fastest one and the number of blocks is smaller than ones whose *rate* is 2 or 3.

It should be mentioned that the comparison of the speed of round functions does not always reflect directly to the speed of the whole design. This is because that the overhead of the ciphertext generation depends on the structure of the round function, especially the empty cycle in process of XOR/aesenc.

Table 5: Speed (in cycles / Byte) of round functions of Rocca, AEGIS-128, AEGIS-128L, AEGIS-256, Tiaxion-346, and JN16 (not include a generation part of a ciphertext).

Primitive	Speed (in cycles / Byte)	# of blocks	rate
AEGIS-128	0.384482	5	5
AEGIS-256	0.388125	6	6
AEGIS-128L	0.191072	8	4
Tiaoxin-346	0.192413	13	3
[JN16]	0.140433	12	2
Rocca	0.124609	8	2

3.4 Loading the Nonce and Key

It has been pointed by Liu et al. that there is one useless round in Tiaoxin-346 by expressing the internal states in terms of the nonce and the key at the initialization phase [LIMS21]. The main reason is that the nonce and the key are not well diffused, i.e. after a certain number of rounds, the internal state can be expressed in terms of $A(N)$ and the key. To avoid it in Rocca, we carefully investigate how to place the nonce and the key.

In Rocca, the initial state is loaded as follows:

$$S[0] = K_1, S[1] = N, S[2] = Z_0, S[3] = Z_1,$$
$$S[4] = N \oplus K_1, S[5] = 0, S[6] = K_0, S[7] = 0.$$

After one-round update, the state $(S[0], \ldots, S[7])$ becomes:

$$S[0] = Z_0, S[1] = A(K_1), S[2] = \underline{N \oplus K_0}, S[3] = \underline{N \oplus A(Z_0)},$$
$$S[4] = 0, S[5] = \underline{A(N \oplus K_1) \oplus Z_1}, S[6] = \underline{N \oplus K_1}, S[7] = K_0 \oplus K_1.$$

It can be observed that N is xored with K_0 and K_1, respectively. Moreover, N is involved in the expressions of each state block in a very different way, which can avoid the useless rounds and, at the same time, strengthen the resistance against the key-recovery attacks applied to round-reduced AEGIS-128 and Tiaoxin-346 as described in [LIMS21]. Further evidence can be seen from the expressions of the state blocks after 3 rounds of update, as shown below:

$$S[0] \quad = \quad \underline{N \oplus K_1},$$

$$
\begin{aligned}
S[1] &= A(K_0 \oplus K_1 \oplus Z_0) \oplus Z_0 \oplus N \oplus K_1,\\
S[2] &= A(Z_0) \oplus K_0 \oplus K_1 \oplus A(A(\underline{N \oplus K_1}) \oplus Z_1),\\
S[3] &= A(\underline{A(K_1) \oplus N \oplus K_1}) \oplus A(Z_0) \oplus K_0 \oplus K_1,\\
S[4] &= A(\underline{N \oplus K_0}) \oplus A(K_1) \oplus Z_1,\\
S[5] &= A(\underline{N \oplus A(Z_0) \oplus Z_1}) \oplus A(\underline{N \oplus K_0}) \oplus A(K_1),\\
S[6] &= A(\underline{N \oplus A(Z_0)}) \oplus N \oplus A(Z_0) \oplus Z_1,\\
S[7] &= K_0 \oplus K_1 \oplus Z_0 \oplus A(A(\underline{N \oplus K_1}) \oplus Z_1).
\end{aligned}
$$

3.5 Generating the Ciphertext Blocks

In both AEGIS and Tiaoxin-346, each ciphertext block is computed based on a simple quadratic boolean function in terms of the several internal state blocks where the number of AND operations is 1. However, such a way to generate the output seems to be insecure against the statistical attack proposed by [Min14], especially for the scheme targeting 256-bit security.

At the initial design phase, we tried many possible combinations to compute each ciphertext block with a similar quadratic boolean function. However, with the MILP-based method [ENP19] to automatically evaluate the security against this statistical attack, the lower bound for the time complexity is always below 2^{128}, which is far smaller than 2^{256}. Therefore, new strategies are essential for Rocca.

The basic idea is to utilize a complex nonlinear function and finally the AES round function is chosen as the only nonlinear function. Due to the parallel way to perform the AES round function, such a way is indeed rather efficient and can simultaneously strengthen the security of our scheme. To reduce the overall overheads, computing each ciphertext block only utilizes 1 aesenc.

The basic principle to choose the state blocks to compute the ciphertext is that the state blocks $(S[0], S[2], S[4], S[5])$ passing through the AES round function in the round updated function should be involved, which can increase the number of active S-boxes in the first round. In addition, we expect that they should be processed in a different way from that in the round update function. Intuitively, this can prevent the ciphertext blocks from being related to the updated internal state blocks.

Moreover, as $(S[4], S[5])$ passes through the AES round function in the round update function and the two state blocks are next to each other, considering the fact that several rounds are needed, it is better to choose additional state blocks from $(S[0], S[1], S[2], S[3], S[4])$, which will be shifted to $(S[4], S[5])$ after some rounds. A detailed study of the security of our choice can be found in the following section.

We emphasize that the overhead of executing these two aesencs is few since we can assign them into empty cycles of aesenc in the round function.

4 Security Evaluation

4.1 Differential Attack

The differential attack is one of the possible attacks on the initialization phase of Rocca. Specifically, the differences in the *nonce* (and key) can propagate to the ciphertext. If there is a differential characteristic with a high probability, it can be exploited for the differential attack. Hence, we can compute the lower bound for the number of active S-boxes in the initialization phase to evaluate the resistance against the differential attack. To compute the lower bound, we utilize a MILP-aided method proposed by Mouha et al. [MWGP11] and focus on the byte-wise truncated differences. We evaluate it in both the single-key

setting where differences can only be injected into the *nonce* and the related-key setting where differences can be injected into the key and *nonce*.

Table 6 shows the lower bounds for the number of active S-boxes up to 14 rounds in the single-key setting and up to 11 rounds in the related-key setting in the initialization phase. Since the maximal differential probability of the S-box of AES is 2^{-6}, it is sufficient to guarantee the security against differential attacks if there are 43 active S-boxes, as it gives $2^{(-6\times 43)} < 2^{-256}$ as an estimate of the differential probability. As shown in Table 6, there are 54 active S-boxes over 6 rounds in the single-key setting and 44 active S-boxes over 7 rounds in the related-key setting in the initialization phase. It should be emphasized that we do not claim the security in the related-key setting, although we evaluated the number of active S-boxes in the related-key setting.

Since there is a large security margin, we expect that Rocca can resist against differential attacks in the initialization phase.

Table 6: The lower bound for the number of active S-boxes in the initialization phase where AS_{sk} and AS_{rk} mean an active S-box in the single-key setting and in the related-key setting, respectively.

Rounds	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# of AS_{sk}	1	6	9	30	38	54	62	82	85	93	100	104	111	115
# of AS_{rk}	0	1	2	11	21	36	44	48	68	73	79	-	-	-

4.2 Forgery Attack

It has been shown in [Nik14] that the forgery attack is a main threat to the constructions like Tiaoxin-346 and AEGIS as only one-round update is used to absorb each block of associated data and message. Such a concern has been taken into account in our design phase, as reported in Sect. 3.

Specifically, in the forgery attack, the aim is to find a differential trail where the attackers can arbitrarily choose differences at the associated data and expect that such a choice of difference can lead to a collision in the internal state after several number of rounds. The resistance against this attack vector can be efficiently evaluated with an automatic method [MWGP11]. As Rocca is based on the AES round function, it suffices to prove that the number of active S-boxes in such a trail is larger than 22 as the length of the tag is 128 bits. With the MILP-based method, it is found that the lower bound is 24. Consequently, Rocca can provide 128-bit security against the forgery attack.

4.3 Integral Attack

One of the most efficient attacks on round-reduced AES is integral attacks. Recently, Liu et al. presented some attacks [LIMS21] on round-reduced AEGIS-128 and Tiaoxin-346 based on the integral distinguisher on 4-round AES. To understand the security of our construction, it is necessary to evaluate the resistance against integral attacks. Similar to [LIMS21], the internal state will be first expressed in terms of the initial state and then we study the expressions.

For simplicity, denote the state after r iterations of the round function at the initialization phase by S_r. In addition, when writing the expressions, we omit the constants and use $A(X)$ to represent that X passes through one AES round, i.e. $A(X)$ can represent $A(X \oplus \epsilon)$ where ϵ is a 128-bit constant. In this way, the internal state S_4 can be expressed as follows:

$$S_4[0] = A(A(N)), S_4[1] = A(N) \oplus A(A(N)),$$

$$S_4[2] = A(N), S_4[3] = A(A(A(N))) \oplus N,$$
$$S_4[4] = A(N), S_4[5] = A(A(N)) \oplus A(N),$$
$$S_4[6] = A(A(N) \oplus A(N)) \oplus A(N), S_4[7] = A(N).$$

As our construction can provide 256-bit security, it is necessary to evaluate the case when N traverses all the 2^{128} possible values under the same 256-bit key. According to [LIMS21], some terms in the expressions can be eliminated by adding proper conditions and the expressions can be simplified. However, according to the expression of $S_4[3]$, when N takes all the possible values, it is impossible that $S_4[3]$ will also take all the 2^{128} possible values. In other words, the multiset of $S_4[3]$ tends to be unstructured. Therefore, by considering the propagation of $S_4[3]$ and the way to compute the ciphertext, we believe that 20 rounds are sufficient to resist against integral attacks.

On the other hand, consider the expressions for S_6, as shown below:

$$\begin{aligned}
S_6[0] &= A(A(N)) \oplus A(A(N) \oplus A(N)) \oplus A(N),\\
S_6[1] &= A(A(N)) \oplus A(A(N)) \oplus A(A(N)) \oplus A(A(N) \oplus A(N)) \oplus A(N),\\
S_6[2] &= A(A(A(N))) \oplus A(N) \oplus A(A(A(N)) \oplus A(N)) \oplus A(N),\\
S_6[3] &= A(A(N) \oplus A(A(N)) \oplus A(A(N) \oplus A(N)) \oplus A(N)) \oplus A(N))\\
&\quad \oplus A(A(A(N))) \oplus A(N),\\
S_6[4] &= A(A(N)) \oplus A(N) \oplus A(A(N)),\\
S_6[5] &= A(A(A(A(N))) \oplus N) \oplus A(A(N)) \oplus A(N) \oplus A(A(N)),\\
S_6[6] &= A(A(A(N)) \oplus A(A(A(N))) \oplus N) \oplus A(A(A(N))) \oplus N,\\
S_6[7] &= A(N) \oplus A(A(A(N)) \oplus A(N)) \oplus A(N).
\end{aligned}$$

As

$$\begin{aligned}
S_8[0] \oplus S_8[4] &= S_6[0] \oplus S_6[6] \oplus A(S_6[2]) \oplus S_6[1] \oplus Z_0 \oplus Z_1,\\
S_8[1] &= A(S_6[7] \oplus Z_0) \oplus S_6[0] \oplus S_6[7],
\end{aligned}$$

it can be found that in the expressions of $A(S_8[1])$ and $A(S_8[0] \oplus S_8[4])$, N will pass through 5 AES rounds and there seems to be no way to add proper conditions to prevent N from passing through 5 AES rounds. Moreover, as N passes through 5 AES rounds in very different ways in $A(S_8[1])$ and $A(S_8[0] \oplus S_8[4])$, it is also impossible to prevent it by considering the sum $A(S_8[1]) \oplus A(S_8[0] \oplus S_8[4])$. Consequently, we further believe that 20 rounds are secure against integral attacks.

4.4 State-recovery Attack

Different from AEGIS and Tiaoxin-346, the output in our construction only involves a few state blocks, i.e. the attackers are able to know $A(S[1]) \oplus S[5]$ and $A(S[0] \oplus S[4]) \oplus S[2]$. As the internal state consists of 8 blocks and the output in each round only leaks 256-bit information, the attackers at least need to consider 4 consecutive rounds in order to recover the whole secret internal state.

Guess-and-determine attack. The guess-and-determine attack is a common tool to achieve state recovery. Consider four consecutive rounds at the encryption phase and denote the 4 internal states used to generate the ciphertexts by S_t, S_{t+1}, S_{t+2} and S_{t+3}, respectively. In this case, the attackers can compute

$$A(S_i[1]) \oplus S_i[5], A(S_i[0] \oplus S_i[4]) \oplus S_i[2],$$

where $t \leq i \leq t+3$.

Assuming the message blocks are all zero, we thus have

$$\begin{aligned}
A(S_{t+1}[1]) &= A(A(S_t[0]) \oplus S_t[7]),\\
S_{t+1}[5] &= A(S_t[4]) \oplus S_t[3],\\
A(S_{t+1}[0] \oplus S_{t+1}[4]) &= A(S_t[7] \oplus S_t[3]),\\
S_{t+1}[2] &= S_t[1] \oplus S_t[6],
\end{aligned}$$

$$\begin{aligned}
A(S_{t+2}[1]) &= A(A(S_{t+1}[0]) \oplus S_{t+1}[7])\\
&= A(A(S_t[7]) \oplus S_t[0] \oplus S_t[6]),\\
S_{t+2}[5] &= A(S_{t+1}[4]) \oplus S_{t+1}[3]\\
&= A(S_t[3]) \oplus A(S_t[2]) \oplus S_t[1],\\
A(S_{t+2}[0] \oplus S_{t+2}[4]) &= A(S_{t+1}[7] \oplus S_{t+1}[3])\\
&= A(S_t[0] \oplus S_t[6] \oplus A(S_t[2]) \oplus S_t[1]),\\
S_{t+2}[2] &= S_{t+1}[1] \oplus S_{t+1}[6]\\
&= A(S_t[0]) \oplus S_t[7] \oplus A(S_t[5]) \oplus S_t[4],
\end{aligned}$$

$$\begin{aligned}
A(S_{t+3}[1]) &= A(A(S_{t+1}[7]) \oplus S_{t+1}[0] \oplus S_{t+1}[6])\\
&= A(A(S_t[0] \oplus S_t[6]) \oplus S_t[7] \oplus A(S_t[5]) \oplus S_t[4]),\\
S_{t+3}[5] &= A(S_{t+1}[3]) \oplus A(S_{t+1}[2]) \oplus S_{t+1}[1],\\
&= A(A(S_t[2]) \oplus S_t[1]) \oplus A(S_t[1] \oplus S_t[6]) \oplus A(S_t[0]) \oplus S_t[7],\\
A(S_{t+3}[0] \oplus S_{t+3}[4]) &= A(S_{t+1}[0] \oplus S_{t+1}[6] \oplus A(S_{t+1}[2]) \oplus S_{t+1}[1]),\\
&= A(A(S_t[5]) \oplus S_t[4] \oplus S_t[1] \oplus S_t[6] \oplus A(S_t[0])),\\
S_{t+3}[2] &= A(S_{t+1}[0]) \oplus S_{t+1}[7] \oplus A(S_{t+1}[5]) \oplus S_{t+1}[4],\\
&= A(S_t[7]) \oplus S_t[0] \oplus S_t[6] \oplus A(A(S_t[4]) \oplus S_t[3]) \oplus S_t[3].
\end{aligned}$$

Therefore, the attackers at least need to consider the following 1024 nonlinear boolean equations in terms of 1024 boolean variables $(S_t[0], \ldots, S_t[7])$ in order to recover the secret state:

$$\begin{aligned}
\alpha_0 &= A(S_t[1]) \oplus S_t[5],\\
\alpha_1 &= A(S_t[0] \oplus S_t[4]) \oplus S_t[2],\\
\alpha_2 &= A(A(S_t[0]) \oplus S_t[7]) \oplus A(S_t[4]) \oplus S_t[3],\\
\alpha_3 &= A(S_t[7] \oplus S_t[3]) \oplus S_t[1] \oplus S_t[6],\\
\alpha_4 &= A(A(S_t[7]) \oplus S_t[0] \oplus S_t[6]) \oplus A(S_t[3]) \oplus A(S_t[2]) \oplus S_t[1],\\
\alpha_5 &= A(S_t[0] \oplus S_t[6] \oplus A(S_t[2]) \oplus S_t[1]) \oplus A(S_t[0]) \oplus S_t[7] \oplus A(S_t[5]) \oplus S_t[4],\\
\alpha_6 &= A(A(S_t[0] \oplus S_t[6]) \oplus S_t[7] \oplus A(S_t[5]) \oplus S_t[4])\\
&\quad \oplus A(A(S_t[2]) \oplus S_t[1]) \oplus A(S_t[1] \oplus S_t[6]) \oplus A(S_t[0]) \oplus S_t[7],\\
\alpha_7 &= A(A(S_t[5]) \oplus S_t[4] \oplus S_t[1] \oplus S_t[6] \oplus A(S_t[0]))\\
&\quad \oplus A(S_t[7]) \oplus S_t[0] \oplus S_t[6] \oplus A(A(S_t[4]) \oplus S_t[3]) \oplus S_t[3],
\end{aligned}$$

where $\alpha_i \in \mathbb{F}_2^{128}$ $(0 \leq i \leq 7)$ are known constants. It is obvious that the attackers should not completely guess 2 state blocks as the time complexity of guess will be 2^{256}. A clever way is to guess a column and a diagonal of the state blocks, which fits well with the form of the outputs. Such a strategy will allow attackers to guess at most 8 columns and diagonals. However, only in the conditions imposed by $(\alpha_0, \alpha_1, \alpha_3)$, one AES round is involved, i.e. the clever way is only applicable to these conditions. For the remaining conditions, two

AES rounds are involved, which implies that the attackers at least need to guess a complete 128-bit block due to the full diffusion. For such reasons, we believe the time complexity of the guess-and-determine attack cannot be lower than 2^{256}.

Algebraic attack. It is well-known that the S-box of AES can be expressed as a set of quadratic boolean equations if the input zero is discarded. Therefore, the above equation system can be described as quadratic boolean equations by introducing extra intermediate variables to represent the outputs of the S-box for each AES round function. Notice that for different ciphertext blocks $(\alpha_0, ..., \alpha_7)$, the attackers have to introduce different variables due to the big difference between the equations. Although the system of equations is overdefined, the number of equations is only slightly larger than the number of variables and the number of variables is much larger than 256. As far as we know, such a system of equations can not be solved with time complexity 2^{256}.

4.5 The Linear Bias

Exploiting the fact that the output (keystream) of AEGIS is quadratic in terms of several state blocks and only one-round update is used to process each message block, Minaud proposed a statistical attack [Min14] on the keystream of AEGIS-256. Such an attack was improved in [ENP19] with an automatic search method based on [SSS+19]. Specifically, the authors first utilized a simple truncated model and evaluated the minimal number of active S-boxes. It is found that for AEGIS-128, AEGIS-128L and AEGIS-256, all the results obtained in the simple truncated model suggest they are insecure against such a statistical attack. However, when searching for compatible linear trails in the bit level, almost all of them are incompatible. In addition, the results obtained in the refined model is far larger than that obtained in the simple truncated model.

To evaluate the resistance of our construction against such a statistical attack, we also adopted the simple truncated model as in [ENP19]. According to our results, the best case is to consider 4 consecutive rounds and the minimal number of active S-boxes is 38, which suggests that the time complexity of the distinguishing attack is at least 2^{228}. Achieving 42 active S-boxes is ambitious without affecting the performance and we believe 38 is enough to resist against such an attack considering the big gap between the truncated model and bitwise model as reported in [ENP19]. To further verify whether there is a compatible linear trail to the best solution obtained with the truncated model, we also implemented the bitwise model where there is no additional constraint on the input mask and output mask of the S-box except the trivial infeasible pairs caused by the zero input mask or zero output mask. When searching for a compatible linear trail based on the truncated pattern, it is soon shown to be infeasible. One main reason is that compared with the attack on AEGIS-256 requiring 2 consecutive rounds, this statistical attack on Rocca requires 4 consecutive rounds, which makes the contradictions in the solutions obtained with the simple truncated model occur more easily if verified with the bitwise model. Taking this fact into account, we further believe Rocca is secure against this attack vector.

4.6 Other Attacks

While there are many attack vectors for block ciphers, their application to Rocca is restrictive as the attackers can only know partial information of the internal state from the ciphertext blocks. In other words, reversing the round update function is impossible in Rocca without guessing many secret state blocks. For this reason, only the above potential attacks vectors are taken into account. In addition, due to the usage of the constant (Z_0, Z_1) at the initialization phase, the attack based on the similarity in the four columns of the AES state is also excluded.

4.7 No Claims

We do not claim the security of our scheme in the nonce-misuse setting and it seems trivial to achieve the state recovery in this setting as the output is computed with only one-round update function at the encryption phase. In addition, we do not claim the security of our scheme in the related-key and known-key setting, which is far from meaningful in real-world applications. For the attacks on the initialization phase, we emphasize that the attackers can only derive information from the restricted outputs and cannot know the full secret internal state.

5 Software Implementation

According to [ITU17], target peak data rates for 5G communication are 10 Gbps for uplink and 20 Gbps for downlink. SNOW-V [EJMY19] is a new version of SNOW-family designed for 5G communication with 256-bit key support and achieves 58.25 Gbps on Intel(R) Core(TM) i7 8650U CPU @1.90GHz in encryption only mode. In the next generation (*i.e.* 6G), the target peak data rate is further increased to 100 Gbps to 1 Tbps [LaL19]. In order to realize this high peak data rate, a new encryption algorithm is required.

We evaluate the performance of Rocca and show that Rocca can achieve 160 Gbps when encrypting data of large size. Modern CPUs are equipped with a dedicated instructions set for AES called AES New Instructions (AES-NI). As Rocca has the AES round function as its component, we can optimize the implementation by utilizing AES-NI. Specifically, we use `_mm_aesenc_si128()` for AES's round function. For XORing two 128-bit values, we use `_mm_xor_si128()`. We also compare the performance with existing algorithms and demonstrate Rocca's advantage in terms of the performance. All evaluations were performed on a PC with Intel(R) Core(TM) i7-1068NG7 CPU @ 2.30GHz with 32GB RAM. For the fair comparison, we included Rocca as well as SNOW-V, Tiaoxin and AEGIS to Openssl (3.0.0-alpha3-Dev) and measured their performances. We used SNOW-V reference implementation with SIMD, which was given in [EJMY19]. For Tiaoxin-346 and AEGIS, we used implementations available at https://github.com/floodyberry/supercop. The results are given in Table 7, and all performance results are given in Gbps. In TLS, data will be divided into chunks of $2^{14} = 16384$ bytes or less before it is encrypted, the values in Table 7 are close to what we expect in practice. As shown, Rocca is 2.99 times faster than SNOW-V, and 2.72 times faster than AES-256-CTR in processing 16384 bytes message. It also outperforms both 128-bit algorithms which we tested. In encryption only mode, the initialization is performed once and only the encryption is iterated. While in AEAD mode, the initialization, associated data addition, encryption, tag generation and finalization are iterated. Here, the size of associated data is fixed to 13 bytes. In case of Rocca, the round function is iterated 20 times in the initialization and finalization, respectively, which is equivalent to processing 1280 bytes of input. As a result, we expect $1280/16384 \approx 8\%$ overhead to the encryption mode for 16384 bytes input. Additional overhead will be incurred by calling functions for the initialization, tag generation and finalization. The performance results on other CPUs are given in Appendix A, and Rocca achieves the best performance in other CPUs as well.

The performance can be further improved by using new instructions set and/or optimizing the implementation. The new instructions set AVX512 contains `_mm512_aesenc_epi128()`, which runs four 128-bit AES round functions in parallel. As Rocca uses four AES round functions in one state update, using `_mm512_aesenc_epi128()` instead of four `_mm_aesenc_epi128()`s can be improved by up-to 4 times.

Table 7: Performance Evaluation

Algorithms	Key length	Size of input (bytes)				
		16384	8192	1024	256	64
Encryption only						
AEGIS-128	128-bit	66.09 Gbps	65.53 Gbps	60.14 Gbps	46.38 Gbps	29.55 Gbps
AEGIS-128L		113.05 Gbps	107.95 Gbps	70.48 Gbps	33.65 Gbps	14.85 Gbps
Tiaoxin-346 v2		131.47 Gbps	125.81 Gbps	83.21 Gbps	35.11 Gbps	13.77 Gbps
AEGIS-256	256-bit	67.97 Gbps	68.47 Gbps	60.96 Gbps	42.98 Gbps	28.53 Gbps
AES-256-CBC		10.25 Gbps	10.48 Gbps	10.49 Gbps	10.47 Gbps	9.92 Gbps
AES-256-CTR		58.91 Gbps	61.14 Gbps	52.77 Gbps	38.75 Gbps	19.00 Gbps
ChaCha20		12.20 Gbps	12.82 Gbps	12.41 Gbps	11.54 Gbps	5.25 Gbps
SNOW-V		53.64 Gbps	45.64 Gbps	44.37 Gbps	41.99 Gbps	29.96 Gbps
Rocca		**160.41 Gbps**	**156.47 Gbps**	**112.19 Gbps**	**68.45 Gbps**	**30.10 Gbps**
AEAD						
AEGIS-128	128-bit	59.61 Gbps	60.58 Gbps	33.91 Gbps	13.06 Gbps	4.06 Gbps
AEGIS-128L		102.00 Gbps	90.13 Gbps	33.50 Gbps	10.91 Gbps	3.14 Gbps
Tiaoxin-346 v2		118.26 Gbps	104.12 Gbps	34.72 Gbps	10.50 Gbps	3.00 Gbps
AEGIS-256	256-bit	66.32 Gbps	61.84 Gbps	31.35 Gbps	11.76 Gbps	3.58 Gbps
AES-256-GCM		30.59 Gbps	30.70 Gbps	19.97 Gbps	8.93 Gbps	2.77 Gbps
ChaCha20-Poly1305		7.57 Gbps	7.57 Gbps	6.36 Gbps	3.77 Gbps	1.33 Gbps
SNOW-V-GCM		28.97 Gbps	28.07 Gbps	18.41 Gbps	8.56 Gbps	2.71 Gbps
Rocca		**138.22 Gbps**	**117.58 Gbps**	**37.38 Gbps**	**11.55 Gbps**	**3.13 Gbps**

6 Conclusions

To fulfill the basic requirements on the speed and security in 6G systems, i.e. 100 Gbps and 256-bit security, we are motivated to further study the generalized method to construct round functions based on the parallel calls to the AES round function, which was first studied by Jean and Nikolić in FSE 2016. As a result, an efficient AES-based AEAD scheme called Rocca is proposed, whose construction is only based on the AES round function and the 128-bit XOR operation supported by the SIMD instructions on model CPUs. In addition, we have performed a thorough study to understand the security of Rocca. According to the software implementation, Rocca can reach 138 Gbps in the AEAD mode, which is more than four times faster than SNOW-V designed for 5G systems. To the best of our knowledge, Rocca is the first dedicated scheme targeting 6G systems and it also shows the potential to reach the basic requirements in such systems.

As future work, a parallelizable mode of Rocca would be interesting and beneficial for both environments equipped with multiple cores and not supported AES-NI.

Acknowledgments

The authors would like to thank Stefan Kölbl and the anonymous ToSC reviewers for the valuable comments and suggestions. Takanori Isobe is supported by JST, PRESTO Grant Number JPMJPR2031, Grant-in-Aid for Scientific Research (B)(KAKENHI 19H02141) and SECOM science and technology foundation. Fukang Liu is supported by Invitation Programs for Foreigner-based Researchers of NICT. Kosei Sakamoto is supported by Grant-in-Aid for JSPS Fellows (KAKENHI 20J23526) for Japan Society for the Promotion of Science.

References

[3GP18] 3GPP SA3. Study on the support of 256-bit algorithms for 5G. https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3422, 2018.

[arm21] arm. Arm® architecture reference manual armv8, for armv8-a architecture profile, 2021.

[cae18] CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness,. https://competitions.cr.yp.to/caesar.html, 2018.

[Cora] Intel Corporation. Intel Advanced Encryption Standard (AES) New Instructions Set. Official webpage, https://www.intel.com/content/dam/doc/white-paper/advanced-encryption-standard-new-instructions-set-paper.pdf.

[Corb] Intel Corporation. Intel intrinsics guide. Official webpage, https://software.intel.com/sites/landingpage/IntrinsicsGuide/.

[EJMY19] Patrik Ekdahl, Thomas Johansson, Alexander Maximov, and Jing Yang. A new SNOW stream cipher called SNOW-V. *IACR Trans. Symmetric Cryptol.*, 2019(3):1–42, 2019.

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

[Gue10] Shay Gueron. Intel advanced encryption standard (aes) new instructions set, 2010.

[ITU17] ITU. Minimum requirements related to technical performance for IMT-2020 radio interface(s), 2017.

[JN16] Jérémy Jean and Ivica Nikolic. Efficient design strategies based on the AES round function. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 334–353. Springer, 2016.

[LaL19] Matti Latva-aho and Kari Leppänen. Key drivers and research challenges for 6G ubiquitous wireless intelligence, 2019.

[LIMS21] Fukang Liu, Takanori Isobe, Willi Meier, and Kosei Sakamoto. Weak keys in reduced aegis and tiaoxin. Cryptology ePrint Archive, Report 2021/187, 2021. https://eprint.iacr.org/2021/187.

[Min14] Brice Minaud. Linear biases in AEGIS keystream. In Antoine Joux and Amr M. Youssef, editors, *Selected Areas in Cryptography - SAC 2014 - 21st International Conference, Montreal, QC, Canada, August 14-15, 2014, Revised Selected Papers*, volume 8781 of *Lecture Notes in Computer Science*, pages 290–305. Springer, 2014.

[MWGP11] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In Chuankun Wu, Moti Yung, and Dongdai Lin, editors, *Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*, volume 7537 of *Lecture Notes in Computer Science*, pages 57–76. Springer, 2011.

[Nat01] National Institute of Standards and Technology. FIPS 197 Advanced encryption standard., 2001.

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

[RTL] Real-Time and Embedded Sys Lab. uops.info. Official webpage, https://www.uops.info/.

[SAG06] SAGE. Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2. Version 1.1, ETSI/SAGE, 2006. https://www.etsi.org/deliver/etsi_ts/133500_133599/133501/15.02.00_60/ts_133501v150200p.pdf, 2006.

[SAG11] SAGE. Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3. document 2: ZUC specification. Version 1.6, ETSI/SAGE, 2011. https://www.etsi.org/deliver/etsi_ts/133500_133599/133501/15.02.00_60/ts_133501v150200p.pdf, 2011.

[SSS+19] Danping Shi, Siwei Sun, Yu Sasaki, Chaoyun Li, and Lei Hu. Correlation of quadratic boolean functions: Cryptanalysis of all versions of full \mathsf MORUS. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 180–209. Springer, 2019.

[The18] The ZUC design team. The ZUC-256 Stream Cipher. http://www.is.cas.cn/ztzl2016/zouchongzhi/201801/W020180126529970733243.pdf, 2018.

[WP13] Hongjun Wu and Bart Preneel. AEGIS: A fast authenticated encryption algorithm. In Tanja Lange, Kristin E. Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*, volume 8282 of *Lecture Notes in Computer Science*, pages 185–201. Springer, 2013.

[YJM20] Jing Yang, Thomas Johansson, and Alexander Maximov. Spectral analysis of ZUC-256. *IACR Trans. Symmetric Cryptol.*, 2020(1):266–288, 2020.

A Software Implementation Results on Other CPUs

We show software implementation results on other CPUs in Table 8 and Table 9. Rocca also shows competitive performance on Comet Lake and Coffee Lake.

Table 8: Performance on Intel(R) Core(TM) i9-10910 CPU@3.60GHz with 64 GB RAMs.

Algorithms	Key length	Size of input (bytes)				
		16384	8192	1024	256	64
Encryption only						
AEGIS-128	128-bit	73.92 Gbps	73.55 Gbps	70.38 Gbps	66.02 Gbps	29.06 Gbps
AEGIS-128L		137.92 Gbps	135.67 Gbps	99.32 Gbps	51.97 Gbps	17.75 Gbps
Tiaoxin-346 v2		164.04 Gbps	159.76 Gbps	109.67 Gbps	53.37 Gbps	16.75 Gbps
AEGIS-256	256-bit	88.93 Gbps	88.24 Gbps	79.35 Gbps	67.21 Gbps	28.82 Gbps
AES-256-CBC		9.98 Gbps	9.95 Gbps	9.87 Gbps	9.90 Gbps	9.77 Gbps
AES-256-CTR		41.33 Gbps	41.18 Gbps	39.00 Gbps	33.72 Gbps	19.89 Gbps
ChaCha20		16.01 Gbps	16.02 Gbps	15.75 Gbps	14.59 Gbps	6.71 Gbps
SNOW-V		63.26 Gbps	62.96 Gbps	58.57 Gbps	51.96 Gbps	32.64 Gbps
Rocca		**183.67 Gbps**	**182.24 Gbps**	**155.03 Gbps**	**93.79 Gbps**	**32.57 Gbps**
AEAD						
AEGIS-128	128-bit	68.35 Gbps	65.94 Gbps	37.27 Gbps	15.08 Gbps	4.17 Gbps
AEGIS-128L		124.57 Gbps	110.40 Gbps	41.25 Gbps	13.11 Gbps	3.51 Gbps
Tiaoxin-346 v2		142.49 Gbps	121.86 Gbps	40.27 Gbps	12.36 Gbps	3.26 Gbps
AEGIS-256	256-bit	80.29 Gbps	76.21 Gbps	37.60 Gbps	13.97 Gbps	3.85 Gbps
AES-256-GCM		26.54 Gbps	25.19 Gbps	17.68 Gbps	8.52 Gbps	2.72 Gbps
ChaCha20-Poly1305		10.46 Gbps	10.26 Gbps	8.13 Gbps	4.79 Gbps	1.65 Gbps
SNOW-V-GCM		32.21 Gbps	30.81 Gbps	19.96 Gbps	9.02 Gbps	2.90 Gbps
Rocca		**156.82 Gbps**	**134.36 Gbps**	**43.64 Gbps**	**12.85 Gbps**	**3.35 Gbps**

Table 9: Performance on Intel(R) Core(TM) i5-8279U CPU@2.40GHz with 8 GB RAMs.

Algorithms	Key length	Size of input (bytes)				
		16384	8192	1024	256	64
Encryption only						
AEGIS-128	128-bit	62.02 Gbps	61.71 Gbps	60.41 Gbps	54.35 Gbps	24.08 Gbps
AEGIS-128L		104.94 Gbps	101.70 Gbps	76.06 Gbps	39.53 Gbps	13.23 Gbps
Tiaoxin-346 v2		122.35 Gbps	120.43 Gbps	76.46 Gbps	37.88 Gbps	12.50 Gbps
AEGIS-256	256-bit	72.52 Gbps	72.85 Gbps	67.99 Gbps	56.08 Gbps	23.83 Gbps
AES-256-CBC		8.02 Gbps	7.96 Gbps	8.08 Gbps	8.00 Gbps	7.99 Gbps
AES-256-CTR		34.27 Gbps	34.16 Gbps	32.50 Gbps	27.73 Gbps	16.50 Gbps
ChaCha20		13.22 Gbps	13.26 Gbps	12.90 Gbps	11.94 Gbps	5.63 Gbps
SNOW-V		52.41 Gbps	52.00 Gbps	49.82 Gbps	42.79 Gbps	26.26 Gbps
Rocca		**147.69 Gbps**	**143.73 Gbps**	**122.69 Gbps**	**74.71 Gbps**	**26.29 Gbps**
AEAD						
AEGIS-128	128-bit	57.39 Gbps	55.17 Gbps	30.67 Gbps	11.96 Gbps	3.26 Gbps
AEGIS-128L		103.01 Gbps	91.20 Gbps	33.99 Gbps	10.82 Gbps	2.90 Gbps
Tiaoxin-346 v2		117.34 Gbps	101.32 Gbps	33.60 Gbps	10.25 Gbps	2.69 Gbps
AEGIS-256	256-bit	66.26 Gbps	62.46 Gbps	31.12 Gbps	11.60 Gbps	3.12 Gbps
AES-256-GCM		21.75 Gbps	21.02 Gbps	14.60 Gbps	6.91 Gbps	2.23 Gbps
ChaCha20-Poly1305		8.64 Gbps	8.46 Gbps	6.71 Gbps	3.93 Gbps	1.34 Gbps
SNOW-V-GCM		26.40 Gbps	25.31 Gbps	16.48 Gbps	7.60 Gbps	2.41 Gbps
Rocca		**129.49 Gbps**	**110.77 Gbps**	**36.30 Gbps**	**10.74 Gbps**	**2.76 Gbps**

We also evaluate the performance of Rocca in both encryption only mode and AEAD mode on Android and iOS, implemented with ARM NEON intrinsics. The results are shown in the Table 10 and Rocca achieves very competitive performance on recent mobile platforms. The performance is improved on the newer platforms (*i.e.* Snapdragon 888 and A14) and further improvement is expected in the future.

Table 10: Performance on Smartphones

SoC	Size of input (bytes)				
	16384	8192	1024	256	64
Encryption only					
Qualcomm Snapdragon 888	72.3 Gbps	71.8 Gbps	64.2 Gbps	48.35 Gbps	24.22 Gbps
Qualcomm Snapdragon 845	32.64 Gbps	32.16 Gbps	27.74 Gbps	18.83 Gbps	8.46 Gbps
Apple A14	81.37 Gbps	81.67 Gbps	76.09 Gbps	60.21 Gbps	30.59 Gbps
Apple A11	57.04 Gbps	56.68 Gbps	52.07 Gbps	41.31 Gbps	20.97 Gbps
AEAD					
Qualcomm Snapdragon 888	65.17 Gbps	58.69 Gbps	24.29 Gbps	8.05 Gbps	2.19 Gbps
Qualcomm Snapdragon 845	29.98 Gbps	26.79 Gbps	11.25 Gbps	3.72 Gbps	1.01 Gbps
Apple A14	72.07 Gbps	65.88 Gbps	29.9 Gbps	10.41 Gbps	2.86 Gbps
Apple A11	49.83 Gbps	46.39 Gbps	20.91 Gbps	7.19 Gbps	1.99 Gbps

B Round functions in Table 2

Fig 6, 7, and 8 show round functions whose # of blocks are 8, 9, and 10 in Table 2, respectively. The round function whose # of blocks is 8 is the same as the one of Rocca. Other 2 round functions whose # of blocks is 9 and 10 are the simple extended version of that.

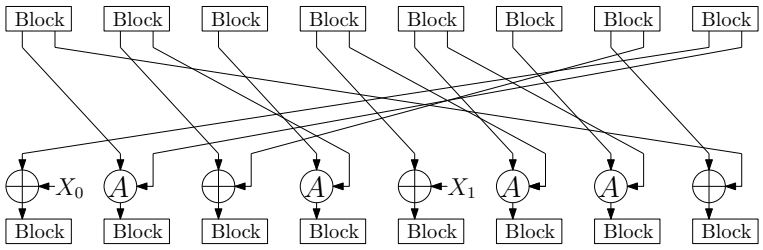

Figure 6: The round function whose # of blocks is 8.

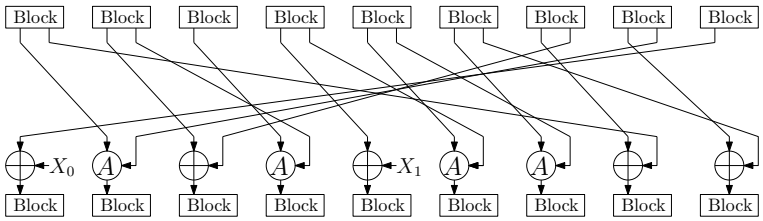

Figure 7: The round function whose # of blocks is 9.

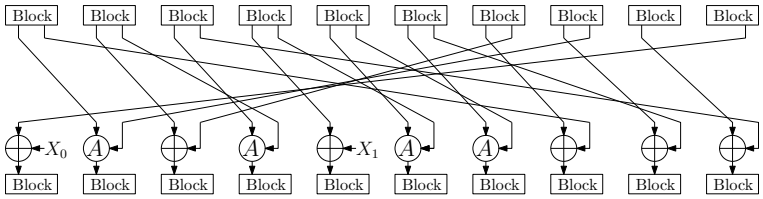

Figure 8: The round function whose # of blocks is 10.

C Reference Implementation with SIMD

```
#include <stdint.h>
#include <immintrin.h>

typedef struct Context{
        __m128i state[8]; // state
        size_t sizeM;     // byte length of  input data
        size_t sizeAD;    // byte length of  associated data
} context;

#define S_NUM 8
#define M_NUM 2
#define BLKSIZE 32
#define NUM_LOOP_FOR_INIT 20

// Z0 = 428a2f98d728ae227137449123ef65cd
#define Z0_3 0x428a2f98
#define Z0_2 0xd728ae22
#define Z0_1 0x71374491
#define Z0_0 0x23ef65cd

// Z1 = b5c0fbcfec4d3b2fe9b5dba58189dbbc
#define Z1_3 0xb5c0fbcf
#define Z1_2 0xec4d3b2f
#define Z1_1 0xe9b5dba5
#define Z1_0 0x8189dbbc

#define enc(m, k) _mm_aesenc_si128(m, k)
#define xor(a, b) _mm_xor_si128(a, b)

#define UPDATE_STATE(X) \
        tmp7  =      S[ 7]; \
        tmp6  =      S[ 6]; \
        S[ 7] = xor( S[ 6], S[ 0] ); \
        S[ 6] = enc( S[ 5], S[ 4] ); \
        S[ 5] = enc( S[ 4], S[ 3] ); \
        S[ 4] = xor( S[ 3], X[ 1] ); \
        S[ 3] = enc( S[ 2], S[ 1] ); \
        S[ 2] = xor( S[ 1], tmp6  ); \
        S[ 1] = enc( S[ 0], tmp7  ); \
        S[ 0] = xor( tmp7 , X[ 0] );

#define LOAD(src, dst) \
        dst[0] = _mm_loadu_si128((const __m128i*)((src)   )); \
        dst[1] = _mm_loadu_si128((const __m128i*)((src)+16));

#define XOR_STRM(src, dst) \
        dst[0] = xor(src[0], enc(         S[1] ,S[5])); \
        dst[1] = xor(src[1], enc(xor(S[0],S[4]),S[2]));

#define STORE(src, dst) \
        _mm_storeu_si128((__m128i*)((dst)   ), src[0]); \
        _mm_storeu_si128((__m128i*)((dst)+16), src[1]);

#define CAST_U64_TO_M128(v) \
        _mm_set_epi32(0, 0, (((uint64_t)(v))>>32)&0xFFFFFFFF, \
```

```
        (((uint64_t)(v))>>0)&0xFFFFFFFF)

void stream_init(context * ctx, const uint8_t * key, \
const uint8_t * nonce) {
        __m128i S[S_NUM], M[M_NUM], tmp7, tmp6;

        // Initialize intenal state
        S[0] = _mm_loadu_si128((const __m128i*)(key+16));
        S[1] = _mm_loadu_si128((const __m128i*)(nonce ));
        S[2] = _mm_set_epi32(Z0_3, Z0_2, Z0_1, Z0_0);
        S[3] = _mm_set_epi32(Z1_3, Z1_2, Z1_1, Z1_0);
        S[4] = _mm_xor_si128(S[1], S[0]);
        S[5] = _mm_setzero_si128();
        S[6] = _mm_loadu_si128((const __m128i*)(key   ));
        S[7] = _mm_setzero_si128();
        M[0] = S[2];
        M[1] = S[3];

        // Update local state
        for(size_t i = 0; i < NUM_LOOP_FOR_INIT; ++i) {
                UPDATE_STATE(M)
        }

        // Update context
        for(size_t i = 0; i < S_NUM; ++i) {
                ctx->state[i] = S[i];
        }
        ctx->sizeM = 0;
        ctx->sizeAD = 0;
}

size_t stream_proc_ad(context * ctx, const uint8_t *ad, \
size_t size) {
        __m128i S[S_NUM], M[M_NUM], tmp7, tmp6;

        // Copy state from context
        for(size_t i = 0; i < S_NUM; ++i) {
                S[i] = ctx->state[i];
        }

        // Update local state with associated data
        size_t proc_size = 0;
        for(size_t size2=size / BLKSIZE * BLKSIZE;
        proc_size < size2; proc_size += BLKSIZE) {
                LOAD(ad + proc_size, M);
                UPDATE_STATE(M);
        }

        // Update context
        for(size_t i = 0; i < S_NUM; ++i) {
                ctx->state[i] = S[i];
        }
        ctx->sizeAD += proc_size;

        return proc_size;
}
```

```
size_t stream_enc(context * ctx, uint8_t *dst, const uint8_t *src, \
size_t size) {
        __m128i S[S_NUM], M[M_NUM], C[M_NUM], tmp7, tmp6;

        // Copy state from context
        for(size_t i = 0; i < S_NUM; ++i) {
                S[i] = ctx->state[i];
        }

        // Generate and output ciphertext
        // Update internal state with plaintext
        size_t proc_size = 0;
        for(size_t size2 = size / BLKSIZE * BLKSIZE; \
        proc_size < size2; proc_size += BLKSIZE) { \
                LOAD(src + proc_size, M);
                XOR_STRM(M, C);
                STORE(C, dst + proc_size);
                UPDATE_STATE(M);
        }

        // Update context
        for(size_t i = 0; i < S_NUM; ++i) {
                ctx->state[i] = S[i];
        }
        ctx->sizeM += proc_size;

        return proc_size;
}

size_t stream_dec(context * ctx, uint8_t *dst, const uint8_t *src, \
size_t size) {
        __m128i S[S_NUM], M[M_NUM], C[M_NUM], tmp7, tmp6;

        // Copy state from context
        for(size_t i = 0; i < S_NUM; ++i) {
                S[i] = ctx->state[i];
        }

        // Generate and output plaintext
        // Update internal state with plaintext
        size_t proc_size = 0;
        for(size_t size2 = size / BLKSIZE * BLKSIZE; \
        proc_size < size2; proc_size += BLKSIZE) { \
                LOAD(src + proc_size, C);
                XOR_STRM(C, M);
                STORE(M, dst + proc_size);
                UPDATE_STATE(M);
        }

        // Update context
        for(size_t i = 0; i < S_NUM; ++i) {
                ctx->state[i] = S[i];
        }
        ctx->sizeM += proc_size;

        return proc_size;
}
```

```
void stream_finalize(context * ctx, uint8_t *tag) {
        __m128i S[S_NUM], M[M_NUM], tmp7, tmp6;

        // Copy state from context
        for(size_t i = 0; i < S_NUM; ++i) {
                S[i] = ctx->state[i];
        }

        // set M[0] to bit length of processed AD
        // set M[1] to bit length of processed M
        M[0] = CAST_U64_TO_M128((uint64_t)ctx->sizeAD << 3);
        M[1] = CAST_U64_TO_M128((uint64_t)ctx->sizeM  << 3);

        // Update internal state
        for(size_t i = 0; i < NUM_LOOP_FOR_INIT; ++i) {
                UPDATE_STATE(M)
        }

        // Generate tag by XORing all S[i]s
        for(size_t i = 1; i < S_NUM; ++i) {
                S[0] = _mm_xor_si128(S[0], S[i]);
        }

        // Output tag
        _mm_store_si128((__m128i*)tag, S[0]);
}
```

D Test Vectors

This section gives three test vectors of Rocca. The least significant byte of the vector is shown on the left and the first 128-bit value is shown on the first line.

```
=== test vector #1===
key =
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

nonce =
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

associated data =
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

plaintext =
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

ciphertext =
15 89 2f 85 55 ad 2d b4 74 9b 90 92 65 71 c4 b8
c2 8b 43 4f 27 77 93 c5 38 33 cb 6e 41 a8 55 29
17 84 a2 c7 fe 37 4b 34 d8 75 fd cb e8 4f 5b 88
bf 3f 38 6f 22 18 f0 46 a8 43 18 56 50 26 d7 55

tag =
cc 72 8c 8b ae dd 36 f1 4c f8 93 8e 9e 07 19 bf

=== test vector #2===
key =
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01

nonce =
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01

associated data =
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01

plaintext =
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

ciphertext =
f9 31 a8 73 0b 2e 8a 3a f3 41 c8 3a 29 c3 05 25
32 5c 17 03 26 c2 9d 91 b2 4d 71 4f ec f3 85 fd
88 e6 50 ef 2e 2c 02 b3 7b 19 e7 0b b9 3f f8 2a
a9 6d 50 c9 fd f0 53 43 f6 e3 6b 66 ee 7b da 69

tag =
ba d0 a5 36 16 59 9b fd b5 53 78 8f da ab ad 78
```

```
=== test vector #3===
key =
01 23 45 67 89 ab cd ef 01 23 45 67 89 ab cd ef
01 23 45 67 89 ab cd ef 01 23 45 67 89 ab cd ef

nonce =
01 23 45 67 89 ab cd ef 01 23 45 67 89 ab cd ef

associated data =
01 23 45 67 89 ab cd ef 01 23 45 67 89 ab cd ef
01 23 45 67 89 ab cd ef 01 23 45 67 89 ab cd ef

plaintext =
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

ciphertext =
26 5b 7e 31 41 41 fd 14 82 35 a5 30 5b 21 7a b2
91 a2 a7 ae ff 91 ef d3 ac 60 3b 28 e0 57 61 09
72 34 22 ef 3f 55 3b 0b 07 ce 72 63 f6 35 02 a0
05 91 de 64 8f 3e e3 b0 54 41 d8 31 3b 13 8b 5a

tag =
66 72 53 4a 8b 57 c2 87 bc f5 68 23 cd 1c db 5a


%=== test vector #4===
%key =
%11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
%22 22 22 22 22 22 22 22 22 22 22 22 22 22 22 22
%nonce =
%44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44
%associated data =
%80 81 82 83 84 85 86 87 88 89 8a 8b 8c 8d 8e 8f 90 91
%plaintext =
%00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
%10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
%20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f
%30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f
%ciphertext =
%34 8b 6f 6e fa d8 07 d2 46 eb f3 45 e7 30 d8 3e
%59 63 bd 6d 29 ee dc 49 a1 35 40 54 5a e2 32 a7
%03 4e d4 ef 19 8a 1e b1 f8 b1 16 a1 76 03 54 b7
%72 60 d6 f2 cc a4 6e fc ad fc 47 65 ff fe 9f 09
%tag =
%a9 f2 06 94 56 55 9d e3 e6 9d 23 3e 15 4b a0 5e
```