TRIAD v1
– A lightweight AEAD and hash function based on stream cipher

Cover sheet

Subhadeep Banik	EPFL, Switzerland
Takanori Isobe	University of Hyogo, Japan
Willi Meier	FHNW, Switzerland
Yosuke Todo	NTT Secure Platform Laboratories, Japan
Bin Zhang	Chinese Academy of Science, China

Contact mail address: `triad-designers@googlegroups.com`

Corresponding submitter: Takanori Isobe
E-mail address : `takanori.isobe@ai.u-hyogo.ac.jp`
Telephone : +81-78-303-1972
Postal address : Takanori Isobe, University of Hyogo, Computational Science Center Building 5-7F 7-1-28 Minatojima-minamimachi, Chuo-ku Kobe, Hyogo 650-0047, Japan

Backup point of contact: Yosuke Todo
E-mail address : `yosuke.todo.xt@hco.ntt.co.jp`
Telephone : +81-422-59-2870
Postal address : Yosuke Todo, NTT Secure Platform Laboratories, 3-9-11, Midori-cho, Musashino-shi, Tokyo 180-8585 JAPAN

Contents

1 Introduction

This document specifies TRIAD, a family of lightweight symmetric-key schemes based on a stream cipher, and it consists of an authenticated encryption mode TRIAD-AE, and a hash function TRIAD-HASH.

There are various characteristics for lightweight cryptography, and many lightweight ciphers are designed such that its area size is as small as possible. The feasibility of low-area implementation is one of the most imporatant aims for lightweight ciphers, but another important index of a lightweight cipher is low energy consumption. Our new authenticated encryption and hash function called TRIAD is designed such that its energy consumption is as low as possible and it can be also implemented in reasonable area size.

In order to realize low energy consumption, we adopt a TRIVIUM-like stream cipher. Unfortunately, we cannot use TRIVIUM directly because its security level is only 80 bits. A simple solution to achieve 112/128-bit security by a TRIVIUM-based construction is to increase the state size, but this make a low-area implementation hard. Therefore, TRIAD adopts a new structure while respecting the global structure of TRIVIUM accepted by the community. Thanks to the new structure, its claimed security level increases to 112 bits even though the state size is reduced to 256 bits (compared to 288 bits in TRIVIUM).

Unlike a block cipher, a stream cipher can accept messages with various bit lengths. This implies that we do not need a "mode of operation" for confidentiality, and we can save an extra state size. Similarly to the confidentiality, we consider a message authentication code (MAC) mode without an extra state size. Our MAC mode is based on a Sponge-like construction, but the internal permutation is very high speed and low energy.

The concrete parameters and claimed security of TRIAD-AE are summarized as follows:

- State size is 256 bits.
- Key size is 128 bits.
- The nonce has 96 bits.
- Tag size for the MAC is 64 bits.
- The number of initialization/finalization rounds is 1024.
- Input data are limited to 2^{50} bytes in total.
- Key recovery attacks on TRIAD-AE require at least 2^{112} computations on a classical computer in a single-key setting.
- Under the related-key attack scenario, we claim that there is no attack that is more efficient than either the generic attack exploiting related keys or 2^{112} computations.
- A nonce respecting scenario is assumed, and neither unverified plaintext nor its corresponding tag are released.
- Nonce misuse or releasing unverified plaintext and its corresponding tag effect loss of confidentiality of plaintext but no loss of security of the key.

The hash function TRIAD-HASH is based in the extended sponge construction [BDPV08, GPP11]. The concrete parameters and claimed security of TRIAD-HASH are summarized as follows:

- The permutation size is 256 bits.
- The input bitrate is 32 bits.

- The output bitrate is 128 bits.
- The output size is 256 bits.
- Input data are limited to 2^{50} bytes in total.
- Collision, second preimage and preimage attacks require at least 2^{112} computations on a classical computer.

1.1 NIST requirements

NIST requirements are reflected as follows:

- A complete specification is found in Section 2
- A design rationale is given in Section 3
- Security arguments and a preliminary cryptanalysis is found in Section 4
- Performace figures in software and hardware are provided in Section 5
- Test vectors are given in Appendix A

2 Specification

2.1 Overview

TRIAD is a family of lightweight symmetric-key schemes based on a stream cipher, and it consists of an authenticated encryption mode TRIAD-AE, and a hash function TRIAD-HASH.

TRIAD-AE has an encrypt-and-mac construction, where a stream cipher TRIAD-SC is used for encryption and a message authentication code TRIAD-MAC is used for the mac. Both TRIAD-SC and TRIAD-MAC accept a 128-bit secret key and a 96-bit nonce, and TRIAD-MAC outputs a 64-bit tag. The same 128-bit secret key is loaded to both TRIAD-SC and TRIAD-MAC, but the claimed security level is up to 112 bits.

TRIAD-HASH is based on the sponge construction with a 256-bit permutation TRIAD-P. The bit security level of the hash function based on the sponge construction depends on the birthday paradox for the bit length of the capacity. To achieve 112-bit security, the bit length of the rate part is 32 bits.

2.2 Notation

TRIAD accepts a 128-bit key K and a 96-bit nonce N, and they are sometimes represented by using byte arrays as

$$K = (K[0], K[1], K[2], \ldots, K[15]),$$
$$N = (N[0], N[1], N[2], \ldots, N[11]),$$

where $K[i]$ and $N[i]$ denote the ith memory of K and N, respectively.

Byte arrays M for plaintext is also defined similarly, and $M[i]$ denote the ith memory of M. Sometimes, the jth bit of $M[i]$ is written as $M[i]_j$, i.e.,

$$M[i] = M[i]_1 \| M[i]_2 \| \cdots \| M[i]_8,$$

where $M[i]_1$ is the msb of $M[i]$. We also use the same notation for the byte array for the associated data A and ciphertext C. We assume that the plaintext and associated data are byte string, and the size of bits is always multiple of 8.

2.3 Update Function and Triad-P

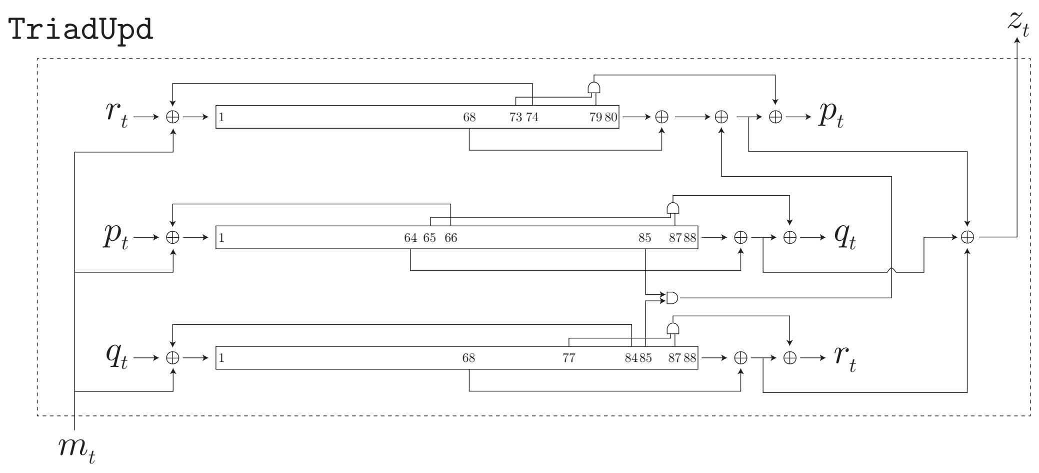

Figure 1: TRIAD update function `TriadUpd`

Algorithm 1 TRIAD Update Function

```
1: procedure TriadUpd($\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, msg$)
2:     $t_1 \leftarrow a_{68} \oplus a_{80} \oplus b_{85} \cdot c_{85}$
3:     $t_2 \leftarrow b_{64} \oplus b_{88}$
4:     $t_3 \leftarrow c_{68} \oplus c_{88}$
5:     $z \leftarrow t_1 \oplus t_2 \oplus t_3$
6:     $t_1 \leftarrow t_1 \oplus a_{73} \cdot a_{79} \oplus b_{66} \oplus msg$
7:     $t_2 \leftarrow t_2 \oplus b_{65} \cdot b_{87} \oplus c_{84} \oplus msg$
8:     $t_3 \leftarrow t_3 \oplus c_{77} \cdot c_{87} \oplus a_{74} \oplus msg$
9:     $(a_1, a_2, \ldots, a_{80}) \leftarrow (t_3, a_1, \ldots, a_{79})$
10:    $(b_1, b_2, \ldots, b_{88}) \leftarrow (t_1, b_1, \ldots, b_{87})$
11:    $(c_1, c_2, \ldots, c_{88}) \leftarrow (t_2, c_1, \ldots, c_{87})$
12:    return $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, z)$
13: end procedure
```

TRIAD contains a 256-bit internal state denoted by $\boldsymbol{a} = (a_1 \| a_2 \| \ldots \| a_{80})$, $\boldsymbol{b} = (b_1 \| b_2 \| \ldots \| a_{88})$, and $\boldsymbol{c} = (c_1 \| c_2 \| \ldots \| c_{88})$, and the internal state is updated by the TRIAD update function `TriadUpd`. The input of `TriadUpd` is 256-bit internal state $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ and 1-bit message *msg*. `TriadUpd` outputs the updated internal state and 1-bit key stream z. Figure 1 shows the update function, and Algorithm 1 shows the detailed algorithm.

TRIAD-P and TRIAD-$\bar{\text{P}}$ are a cryptographic permutation based on `TriadUpd`. They repeats `TriadUpd` 1024 times, and TRIAD-$\bar{\text{P}}$ absorb “1” in the first round. TRIAD-P is used in TRIAD-SC and TRIAD-HASH, and TRIAD-$\bar{\text{P}}$ is used in TRIAD-MAC.

2.4 Triad-AE

Figure 2 summarizes the encryption of TRIAD-AE.

Algorithm 2 Triad-P and Triad-$\bar{\text{P}}$

```
1: procedure TriadP($\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$)
2:     for $i = 1$ to 1024 do
3:         $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, z) \leftarrow$ TriadUpd($\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, 0$)
4:     end for
5:     return $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$
6: end procedure
1: procedure TriadPB($\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$)
2:     $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, z) \leftarrow$ TriadUpd($\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, 1$)
3:     for $i = 2$ to 1024 do
4:         $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, z) \leftarrow$ TriadUpd($\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, 0$)
5:     end for
6:     return $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$
7: end procedure
```

Triad-AE provides authenticated encryption with associated data (AEAD) and accepts a 128-bit key K, 96-bit nonce N, variable-length byte array for message M, and variable-length byte array for associated data A. The encryption of Triad-AE outputs a byte array for ciphertexts C and its length is the same as that of message. Additionally, the encryption of Triad-AE outputs a 8-byte array for the tag T.

Algorithm 3 Encryption and Decryption Algorithms of Triad-AE

```
1: procedure EncryptTriadAE($K, N, M, A$)
2:     $C \leftarrow$ TriadSC($K, N, M$)
3:     $T \leftarrow$ TriadMAC($K, N, M, A$)
4:     return $(C, T)$
5: end procedure
1: procedure DecryptTriadAE($K, N, AD, C, T$)
2:     $M \leftarrow$ TriadSC($K, N, C$)
3:     $V \leftarrow$ TriadMAC($K, N, M, A$)
4:     if $V = T$ then
5:         return $M$
6:     else
7:         return $\perp$
8:     end if
9: end procedure
```

Algorithm 3 shows the encryption and decryption algorithms of Triad-AE. Triad-AE consists of a stream cipher Triad-SC and message authentication code Triad-MAC. The encryption of Triad-AE first encrypt M by using Triad-SC. Then, Triad-MAC generates a 64-bit tag T independently of Triad-SC. The decryption of Triad-AE first decrypt C by using Triad-SC. Then, a 64-bit tag is computed, and it verifies the delivered tag. Decrypted message is returned if the delivered tag is verified, but the message is not returned if it is not verified.

Triad-SC. Triad-SC encrypts message M with mlen bytes and output ciphertext C with mlen bytes. Algorithm 4 shows the algorithm of Triad-SC, where a constant is defined as

$$(con[3] \| con[2] \| con[1] \| con[0]) = \texttt{0xFFFFFFFE}. \quad (1)$$

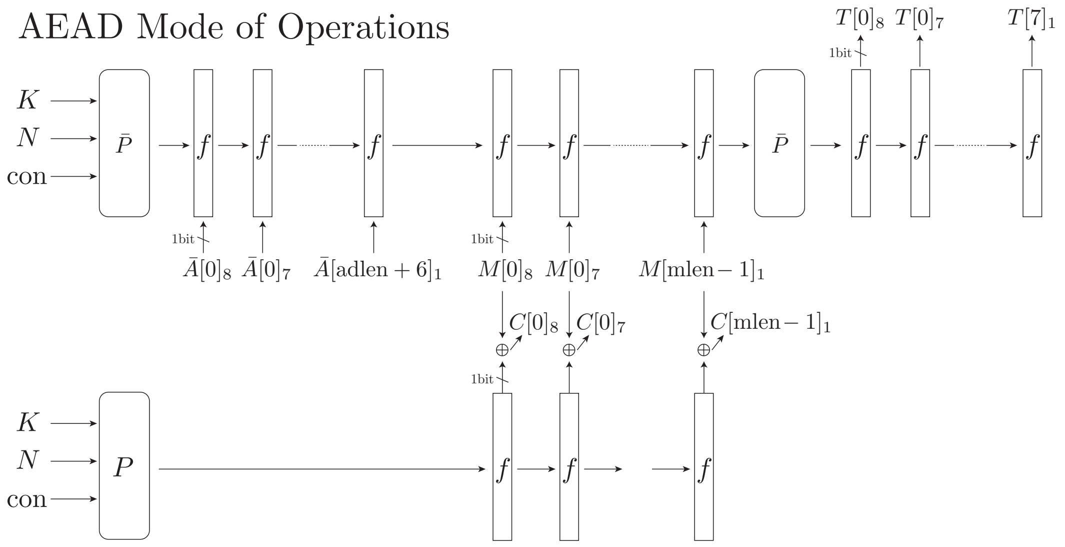

Figure 2: Encryption of Triad-AE

Algorithm 4 Triad-SC

```
1:  procedure TriadSC($(K, N, M)$)
2:      $(a_1\|\cdots\|a_{80}) \leftarrow (N[0]\|K[4]\|con[3]\|K[3]\|con[2]\|K[2]\|con[1]\|K[1]\|con[0]\|K[0])$
3:      $(b_1\|\cdots\|b_{88}) \leftarrow (N[11]\|\cdots\|N[1])$
4:      $(c_1\|\cdots\|c_{88}) \leftarrow (K[15]\|\cdots\|K[5])$
5:      $(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}) \leftarrow$ TriadP$(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c})$
6:      for $i = 0$ to mlen $- 1$ do
7:          for $j = 8$ to $1$ do
8:              $(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c},z) \leftarrow$ TriadUpd$(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c},0)$
9:              $C[i]_j \leftarrow M[i]_j \oplus z$
10:         end for
11:     end for
12:     return $C$
13: end procedure
```

Note that TriadSC can be used to decrypt the ciphertext.

Triad-MAC. Triad-MAC generates a tag T from M with mlen bytes and A with adlen bytes. First, we apply the byte-length padding to the associated data A, and the padded associated data is denoted by $\bar{A}$. Since we restrict the size of the associated data up to $2^{50}-1$ bytes, the byte size of $\bar{A}$ becomes adlen $+ 7$. In detail,

$$\bar{A}[i] = \begin{cases} A[i] & 0 \leq i \leq \text{adlen} \\ \text{adlen}\&\texttt{0xFF} & i = \text{adlen}+1 \\ (\text{adlen} \gg 8)\&\texttt{0xFF} & i = \text{adlen}+2 \\ (\text{adlen} \gg 16)\&\texttt{0xFF} & i = \text{adlen}+3 \\ (\text{adlen} \gg 24)\&\texttt{0xFF} & i = \text{adlen}+4 \\ (\text{adlen} \gg 32)\&\texttt{0xFF} & i = \text{adlen}+5 \\ (\text{adlen} \gg 40)\&\texttt{0xFF} & i = \text{adlen}+6 \\ (\text{adlen} \gg 48)\&\texttt{0xFF} & i = \text{adlen}+7 \end{cases}$$

Algorithm 5 shows the algorithm of Triad-MAC, where a constant is defined as Eq. (1).

Algorithm 5 Triad-MAC

```
procedure TriadMAC($K, N, M$)
    $(a_1\|\cdots\|a_{80}) \leftarrow (N[0]\|K[4]\|con[3]\|K[3]\|con[2]\|K[2]\|con[1]\|K[1]\|con[0]\|K[0])$
    $(b_1\|\cdots\|b_{88}) \leftarrow (N[11]\|\cdots\|N[1])$
    $(c_1\|\cdots\|c_{88}) \leftarrow (K[15]\|\cdots\|K[5])$
    $(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}) \leftarrow$ TriadPB($\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}$)
    for $i = 0$ to adlen $+ 7 - 1$ do
        for $j = 8$ to $1$ do
            $(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c},z) \leftarrow$ TriadUpd($\boldsymbol{a},\boldsymbol{b},\boldsymbol{c},\bar{A}[i]_j$)
        end for
    end for
    for $i = 0$ to mlen $- 1$ do
        for $j = 8$ to $1$ do
            $(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c},z) \leftarrow$ TriadUpd($\boldsymbol{a},\boldsymbol{b},\boldsymbol{c},M[i]_j$)
        end for
    end for
    $(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}) \leftarrow$ TriadPB($\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}$)
    for $i = 0$ to $7$ do
        for $j = 8$ to $1$ do
            $(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c},T[i]_j) \leftarrow$ TriadUpd($\boldsymbol{a},\boldsymbol{b},\boldsymbol{c},0$)
        end for
    end for
    return $T$
end procedure
```

2.5 Triad-HASH

Triad-HASH follows the extended sponge construction [BDPV08, GPP11] with a 256-bit permutation Triad-P with capacity $c = 224$, input bitrate $r = 32$, output bitrate $r' = 128$, digest size $n = 256$, and internal state size $t = 256$ as shown in Fig. 3

The extended sponge construction consists of three phases.

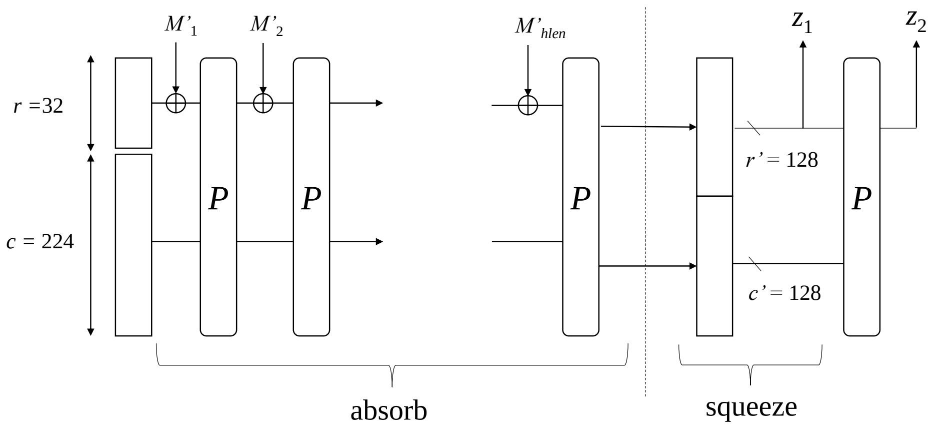

Figure 3: Extended Sponge Construction

Initialization: The message IN with inlen bytes is padded by appending a single 1 bit followed by the minimal number of 0 bits to reach a length that is a multiple of 32, and formatted into $hlen$ 32-bit blocks $M'_1, \ldots, M'_{hlen}$. Note that $M'_{i+1} = (in[4i+3] \| in[4i+2] \| in[4i+1] \| in[4i])$ for any $0 \leq i < hlen - 1$. The initial value of the internal state is given as

$$\begin{aligned}
\{b_1, \ldots, b_{44}\} &= \mathsf{10110111111000010101000101100010100010101110}, \\
&= (e-2) \cdot 2^{44} = \mathsf{0xb7e151628ae}, \\
\{b_{45}, \ldots, b_{88}\} &= \mathsf{00100100001111110110101010001000100001011010}, \\
&= (\pi - 3) \cdot 2^{44} = \mathsf{0x243f6a8885a},
\end{aligned}$$

and the remaining bits are initialized by zero.

Absorbing: the 32-bit input message blocks are XORed into the first 32 bits of the state, interleaved with applications of the permutation TRIAD-P.

Squeezing: 32-byte array $Z = ([31] \| \cdots \| Z[1] \| Z[0])$ is returned as the hash value. Then, the first 128 bits of the state are returned as an output $(Z[15] \| \cdots \| Z[0])$. With a single application of the permutation TRIAD-P, the first 128 bits of the updated states are returned as an output $(Z[31] \| \cdots \| Z[16])$.

3 Design Rationale

3.1 Perspective of Our Design

Our goal is to design an extremely low-energy authenticated encryption algorithm with associated data (AEAD), and to design a lightweight hash function whose components share the same function as the AEAD. According to the observations shown by [BMA+18], a stream-cipher based construction is preferable over one that is based on a block cipher. Further, the fact that a natural unroll implementation can be possible is the most preferable, and they reported that TRIVIUM is one of the most low-energy ciphers. To achieve our design goal, we intended to choose TRIVIUM in the internal primitive, but thus is impossible as it accepts only an 80-bit key. Therefore, we design a TRIVIUM-like stream cipher such that it can accept a 128-bit key and so that its claimed security becomes 112 bits.

Algorithm 6 TRIAD-HASH

```
1: procedure TriadHash($K, N, M$)
2:     $(a_1, \| \cdots \| a_{80}) \leftarrow (0, \ldots, 0)$
3:     $(b_1 \| \cdots \| b_{44}) \leftarrow$ 0xb7e151628ae
4:     $(b_{45} \| \cdots \| b_{88}) \leftarrow$ 0x243f6a8885a
5:     $(c_1, \| \cdots \| c_{88}) \leftarrow (0, \ldots, 0)$
6:     for $i = 1$ to hlen do
7:         $(a_1, \ldots, a_{32}) \leftarrow (a_1, \ldots, a_{32}) \oplus M'_i$
8:         $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) \leftarrow$ TriadP($\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$)
9:     end for
10:    $(Z[15] \| \cdots \| Z[0]) \leftarrow (a_1 \| \cdots \| a_{80} \| b_1 \| \cdots \| b_{48})$
11:    $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) \leftarrow$ TriadP($\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$)
12:    $(Z[31] \| \cdots \| Z[16]) \leftarrow (a_1 \| \cdots \| a_{80} \| b_1 \| \cdots \| b_{48})$
13: end procedure
```

3.2 From Stream Cipher to AEAD

Our goal is to design AEAD, and we construct it by using a stream cipher. One method is to absorb the message into the internal state while generating key stream, but this will cause security risks: Since the update function of the internal state is very sparse, such construction will leak much information of the internal state by controlling absorbed message. Another method is to use the message-authentication code based on the universal hashing such as [FS03] or the Galois Message Authentication Code (GMAC), but the multiplier over Galois field is required. In [FS03], they reported that the multiplier over GF(2^{64}) needs 36,000 gate, and this is not lightweight.

To save area size and energy consumption, our solution is to design a dedicated MAC from the update function of the stream cipher. We guarantee that a large enough number of AND gates must be active to find a collision of the internal state. Please refer to Sect. 4.5 for details. Since the update function for the MAC part can be fully shared with that for the encryption part, the required area size is saved. Besides, it inherits the advantage of the stream cipher, i.e., the energy consumption is dramatically low.

3.3 From Trivium to TRIAD

As already pointed out, TRIVIUM does not accept a 128-bit key. According to a well-known criterion, i.e., the internal state size must be larger than the double of the key size [Bab95, Gol97], TRIVIUM could potentially achieve 128-bit security because the internal state size is 288 bits. Unfortunately, TRIVIUM cannot achieve 128-bit security even if a 128-bit key can be loaded to the internal state, as an attack with complexity of about $2^{99.5}$ is known [MB07]. This attack implies that just a small tweak of TRIVIUM is not sufficient for our use. Therefore, we design a new stream cipher but it inherits the design philosophy of TRIVIUM as much as possible.

From "Multiple of 3" to "Multiple of 2". TRIVIUM adopts the so-called "multiple of 3" property, where the linear tap position is chosen from bits indexed by $3i$, and it is introduced for security against correlation attacks (or distinguishers). Then, the internal state can be decomposed into three linear parts, and these linear parts are connected nonlinearly. To find linear approximations for the correlation attack, one method is to use linear trails in which all AND gates are approximated by 0. However, the linear tap positions are chosen so that such a linear approximation is expected to have at least 72 active AND gates. If we try to find more complicated linear approximations in which AND gates are not always approximated by 0, we can expect that such a linear approximation will not be good because at least two linear parts are active.

While the "Multiple of 3" property is good against the correlation attack, Maximov then reported that it caused a security degradation against the guess-and-determine attack [MB07]. In their advanced guess-and-determine attack, attackers first guess and determine one part of three linear parts, and the complexity is at least $c \times 2^{288/3-66/3} = c \times 2^{74}$. The factor c is the complexity to solve nonlinear system and not negligible. If we additionally guess any bits such that c is small enough, the complexity becomes $c \times 2^{83.5}$ and $c \approx 2^{16}$.

We want to inherit the good property of the "multiple of 3" but such a choice is not always suitable. While the easiest and simplest countermeasure is to increase the state size, such a design is not preferable for the availability of the low-area and low-energy implementation. Therefore, we adopt "multiple of 2" property instead. Although it inherits some good properties of the "multiple of 3" it is not as favorable against the correlation attack but is still reasonable because of it turns out to have a sufficient security margin.

On the positive side, it makes more difficult to mount the advanced guess-and-determine attack. Note that the complexity of the best correlation attack against Trivium is 2^{144}, although that of the advanced guess-and-determine attack is $2^{99.5}$. We believe that our choice is reasonable when we consider a balanced security margin.

From 288-Bit State to 256-Bit State. When we adopt the "multiple of 2" instead of the "multiple of 3," the register size no longer needs to be a multiple of 3. According to the well-known criterion about the internal state size and security, $112 \times 2 = 224$-bit state is the possible minimum. However, there is no security margin in such a construction, and it makes the conversion to the hash function with 112-bit security difficult. Therefore, our choice is to use a 256-bit state for a reasonable security margin and ease of the conversion to the hash function.

Additional AND Gate. Compared with Trivium, the modifications such as smaller register size or "multiple of 2" property might cause new security risks. To compensate for such degradation, one additional AND gate is used in TRIAD, i.e., there are four AND gates in the update function. Then, how to insert the additional AND gate is important. Since the original three AND gates are put in each register, we put the additional AND gate such that it mixes different registers. As shown

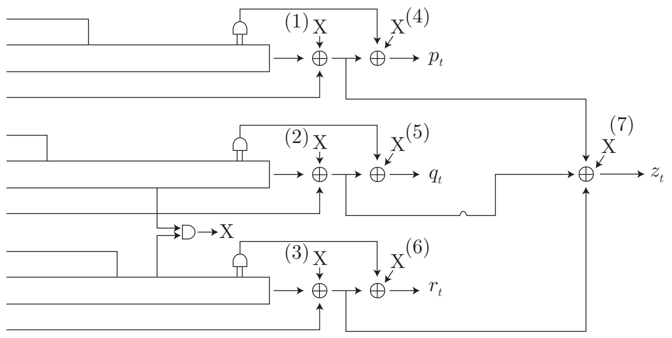

Figure 4: Seven choices to XOR additional AND

in Fig. 4, we have seven choices of positions that the output of the additional AND gate is XORed. Among them, the key stream z_t can have a nonlinear term in the cases of (1), (2), (3), and (7), and such a construction achieves good security against the guess-and-determine attack. However, in the case of (7), the additional AND gate does not contribute to security enhancement of the MAC mode. Therefore, we should choose from the three cases (1), (2), and (3). As a result, we chose the case (1) because it is the lowest energy option on the hardware simulation.

Sizes of Three Registers. Because of the "multiple of 2" property, each state size must be an even number. To determine each state size, we tried various cases under the hardware simulation. We found that the energy consumption, especially when we unrolled the circuit to perform more round function updates in a single clock cycle, was minimized when register sizes are balanced. The reason for this can be intuitively understood as follows: we know that when we unroll round functions, the gates for successive modules stack in front of each other [BMA+18]. The signal transients/glitches produced due to the signal delays traversing these gates are a major source of dynamic energy consumption in the circuit. When the register sizes are unbalanced, while still keeping the first 64 locations of each register untapped for efficient unrolling, the situation reaches a flexing point when we try to unroll the circuit for more than 64 times. For example if we unroll 128 times, the smaller register sees more gates stacked infront of it, due to which it contributes more to teh dynamic power. A balanced register structure evenly distributes the cluster of gates (in the sense of the total signal delay incurred from input to output of teh register) and keeps the dynamic power from increasing disproportionately for one register.

When a 256-bit state is divided into three register sizes equally, sizes of each register become 84, 86, and 86. However, regarding software efficiency, bytewise register size is preferable, and thus, we have chosen sizes of each register as 80, 88, and 88.

Choosing Tap Positions. Finally, we need to choose linear and nonlinear tap positions. As already mentioned in Sect. 3.1, a natural unroll implementation should be possible. Similarly to Trivium, we chose a construction that allows natural unrolling up to 64 rounds. Then, all tap positions must not be chosen from the left 64 bits.

We first choose the linear tap positions, and they are chosen from bits indexed by $2i$ due to "multiple of 2" property. We applied variants of the matrix method in [Gol96]. Note that the counting number of active AND gates becomes independent of nonlinear tap positions when all AND gates are approximated by 0. Thus, we picked parameters whose number of active AND gates is as large as possible. According to exhaustive search, $96 = 24 \times 4$ active AND gates were the maximum possible, and we chose such a tap position.

We next choose the nonlinear tap positions, and they are chosen from bits indexed by $2i + 1$ due to "multiple of 2" property. Unlike Trivium, the nonlinear tap positions are not consecutive. Interestingly, in some of our experiments, when positions of two tapped bits of the AND gate (keeping one of the taps fixed at the second last bit) were not too close and not very far either, the energy consumption was found to be lower.

From the aspect of security, we estimated the correlation of linear trails found by the method that is used to choose linear tap positions. Due to the modification from "multiple of 3" to "multiple of 2", all active AND gates of the linear trails are not always independent. It should not degrade security compared with that of Trivium. In a search for optimizing energy efficiency and low correlation, we found that the present design choice that achieves a correlation that is at most 2^{-72}.

4 Report of Cryptanalysis

4.1 Time-Memory-Data Trade-off Attacks

A time-memory-data trade-off attack (TMDTA) is a well-known generic attack against stream ciphers. There are five parameters:

- N represents the size of the search space.
- P represents the time required by the preprocessing phase of the attack.
- M represents the amount of random access memory available to the attacker.

- T represents the time required by the realtime phase of the attack.
- D represents the amount of realtime data available to the attacker.

The simplest time-memory tradeoff attack on stream ciphers was independently described by Babbage [Bab95] and Golic [Gol97]. Then, the tradeoff curve becomes $TM = N$ with $1 \le T \le D$ and $P = M$. The total attack complexity is minimized when $T = P$, and it implies the so-called birthday bound. In our case of $N = 2^{256}$, T and P are minimized when $T = M = P = 2^{128}$. As it is beyond the claimed security of 112 bits, TRIAD-AE can avoid the simplest time-memory tradeoff attack.

An advanced time-memory-data trade-off attack was proposed by Biryukov and Shamir [BS00]. An improved tradeoff curve is given as $TM^2D^2 = N^2$ with $D^2 \le T \le N$ and $P = N/D$. This tradeoff curve is useful to reduce D and M, but considering the limitations of $D^2 \le T \le N$ and $P = N/D$, the relation of $N^2 \le TP^2$ is obtained. It means that it is impossible to have both P and T less than $2^{n/2}$. Therefore, for $N = 2^{256}$, P or T must be more than 2^{128}.

4.2 Correlation Attack

We follow the security evaluation when TRIVIUM was designed. There are two-types of correlation attacks. The first type exploits correlations between linear combinations of key stream bits and internal state bits and the goal is to recover the internal state. Another type is the distinguishing attack, where correlations between the key stream bits themselves are exploited.

The first-type of correlation attack is useful for the LFSR based stream ciphers. However, it has not been successfully applied to the NFSR based stream ciphers although about one decade has passed since TRIVIUM was proposed. As the designers of TRIVIUM also claimed, it is not clear how attackers exploit the first-type correlation attack.

The second-type correlation attack can be applied. Similarly to the case of TRIVIUM, we evaluated linear trails in which the output of all AND gates is approximated by 0. We chose preferable parameters such that the found linear trail has at least 96 active AND gates. An example of a correlated linear combination of key stream bits is

$$z_0 + z_{12} + z_{20} + z_{24} + z_{36} + z_{44} + z_{56} + z_{98} + z_{106} + z_{116} + z_{172} + z_{182} + z_{190} + z_{256}. \qquad (2)$$

Then, each AND gate is active 24 times, and it has $24 \times 4 = 96$ active AND gates in total. If all active AND gates are independent, the corresponding correlation is 2^{-96}. Unfortunately, since weak property called "multiple of 2" is used in TRIAD instead of "multiple of 3" of TRIVIUM, these exact minimum numbers of active AND gates highly depend. However, the highest correlation of the found linear trails becomes at most 2^{-72} by choosing nonlinear tap positions appropriately.

For example, let us consider the case of Eq. (2). For simplicity, let a_i^r, b_i^r, and c_i^r be the ith bit of $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$ in the rth round, respectively. We define $\tilde{a}_i$, $\tilde{b}_i$, and $\tilde{c}_i$ as $\tilde{a}_{80-i+r} = a_i^r$, $\tilde{b}_{88-i+r} = b_i^r$, and $\tilde{c}_{88-i+r} = c_i^r$, respectively. Then, we have the following relations:

$$\begin{aligned}
f_t &= \tilde{a}_t + \tilde{b}_{t+22} + \tilde{a}_{t+12} + \tilde{b}_{t+88} + \tilde{b}_{t+3} \cdot \tilde{c}_{t+3} + \tilde{a}_{t+1} \cdot \tilde{a}_{t+7} = 0, \\
g_t &= \tilde{b}_t + \tilde{c}_{t+4} + \tilde{b}_{t+24} + \tilde{c}_{t+88} + \tilde{b}_{t+1} \cdot \tilde{b}_{t+23} = 0, \\
h_t &= \tilde{c}_t + \tilde{a}_{t+6} + \tilde{c}_{t+20} + \tilde{a}_{t+80} + \tilde{c}_{t+1} \cdot \tilde{c}_{t+11} = 0, \\
z_t &= (\tilde{a}_t + \tilde{a}_{t+12} + \tilde{b}_{t+3} \cdot \tilde{c}_{t+3}) + (\tilde{b}_t + \tilde{b}_{t+24}) + (\tilde{c}_t + \tilde{b}_{t+20}) \\
&= (\tilde{b}_{t+88} + \tilde{b}_{t+22} + \tilde{a}_{t+1} \cdot \tilde{a}_{t+7}) + (\tilde{c}_{t+88} + \tilde{c}_{t+4} + \tilde{b}_{t+1} \cdot \tilde{b}_{t+23}) + (\tilde{a}_{t+80} + \tilde{a}_{t+6} + \tilde{c}_{t+1} \cdot \tilde{c}_{t+11}).
\end{aligned}$$

Once the linear combination of key stream bits is determined, we can deterministically get the corresponding linear trail. By analyzing the linear trail, we can get three sets $\mathbb{A}$, $\mathbb{B}$, and $\mathbb{C}$ such that

$$z_0 + z_{12} + z_{20} + z_{24} + z_{36} + z_{44} + z_{56} + z_{98} + z_{106} + z_{116} + z_{172} + z_{182} + z_{190} + z_{256} + \sum_{i \in \mathbb{A}} f_{t+i} + \sum_{i \in \mathbb{B}} g_{t+i} + \sum_{i \in \mathbb{C}} h_{t+i}$$

do not have any linear independent term, and it results in the sum of 96 AND in detail. However, these AND are not always independent. For example, the correlation of the sum of 2 AND, $\tilde{a}_t \cdot \tilde{a}_{t+2} + \tilde{a}_{t+2} \cdot \tilde{a}_{t+4}$, is 2^{-1} not 2^{-2}. We analyzed this Boolean function and compute the correlation by using the similar technique shown in [TIM+18]. As a result, the corresponding correlation is 2^{-72}.

Correlation attacks exploiting more complicated linear trails might exist, but it is not exposed to any risk unless the correlation exceeds 2^{-56}. Besides, since we adopt an additional AND gate for the conservative security from the structure of Trivium, we expect that there are no such correlation attacks.

4.3 Cube Attack

The cube attack was proposed by Dinur and Shamir in [DS09] and can be regarded as a variant of a higher-order differential attack [Lai94]. Since the cube attack is eventually the best attack against the initialization of Trivium, we need to guarantee that the initialization of TRIAD is secure against the cube attack. In CRYPTO 2017, Todo et al. proposed a method to evaluate the security against cube attacks theoretically [TIHM17]. This technique evaluates the propagation of the division property [Tod15, TM16] and estimate the reachable algebraic degree. While the estimated degree is an upper bound, the accuracy is high enough and other methods to estimate the degree more accurately have not been known yet.

Among 256 bits in the internal state, a_{80}, b_{88}, and c_{88} can take the lowest degree, and the key stream bit has higher degree than these three bits. We estimated the upper bound of degree for these four bits, where a mixed integer linear programming (MILP) was used to estimate them [XZBL16, Inc15]. Table 1 shows these algebraic degrees.

Table 1: Upper bounds of algebraic degree

	Rounds												
Target	300	350	400	450	500	550	600	650	700	750	800	850	900
a_{80}	9	16	26	33	54	81	110	166	226	**255**	**255**	**255**	**255**
b_{88}	8	16	24	32	54	74	106	161	217	**255**	**255**	**255**	**255**
c_{88}	8	16	24	33	54	79	109	164	220	**255**	**255**	**255**	**255**
z	16	30	44	60	93	139	188	249–252	**255**	**255**	**255**	**255**	**255**

Note that `TriadUpd` is a permutation for given message and control bits. Therefore, the reachable algebraic degree is up to 255, and algebraic degrees of all three bits reach 255 after 750 rounds. The time complexity of cube attacks exploiting degree d is at leat 2^d, and it implies that algebraic degree exploited by attackers is at most 112. The number of rounds in the initialization of TRIAD is 1024, and this gives a large enough security margin.

When the permutation is used as the sponge function, the existence of a zero-sum partition is often discussed. Probably, Triad-P will not have zero-sum partition because the iterated inverse function of `TriadUpd` increases the degree dramatically. On the other hand, we think that we do not need to claim non existence of the zero-sum partition because it does not derive any security risk. Keccak standardized by NIST for SHA3 also has the full-round zero-sum partition [BDPA15, DL11].

4.4 Guess-and-Determine Attack

The goal of a guess-and-determine attack is to guess a part of state bits in an appropriate way, in order to derive equations relating state bits and keystream bits. These equations potentially enable to restrict the state space, so that the complexity of determining the state may be lower than exhaustive search.

As TRIAD resembles TRIVIUM, we check the feasibility of guess-and-determine attacks against TRIAD that are similar to those against TRIVIUM.

In TRIVIUM, 64 equations relating state bits and keystream bits are linear by design. There are essentially two guess-and-determine attacks against Trivium known to get more linear equations, a basic one, and an elaborate one due to [MB07].

The basic attack exploits the property of TRIVIUM that the inputs to the AND gates in the state update function are in consecutive positions. This requires to guess only every second state bit to get linear equations. One of these guess-and-determine attacks on TRIVIUM has complexity 2^{135}. In TRIAD, inputs to the AND gates are non consecutive, and one AND even takes its inputs from different registers. In addition, there are four rather than three AND gates in the state update as in Trivium. Both facts will prevent this type of guess-and-determine attack on TRIAD.

The elaborate attack in [MB07] makes a guess on the state of the shortest of the the three registers, of length 84 bits. As TRIVIUM follows the "multiple of 3" strategy, the state can be divided into three disjoint parts that are connected by AND gates. In [MB07] it is demonstrated that after about 2^{61} update steps a configuration may appear where a sufficient number of ANDs will happen to vanish to get enough linear equations. In principle, a similar approach can be considered for TRIAD, as it follows a "multiple of 2" structure. This would divide the state space of 256 bits into two parts of 128 bits each. The data for a same secret key and nonce are restricted to 2^{50} bytes. This together with the property that TRIAD has four rather than three AND gates, will make such kind of elaborate guess-and-determine very unlikely.

4.5 Forgery Attack

TRIAD-MAC is a message authentication code and is designed by using `TriadUpd`. Attackers first query a message M for a known or chosen nonce N and get a tag T. For any difference Δ, if the internal state absorbing $M \oplus \Delta$ collides with that absorbing M, the forgery attack is successful.

Note that TRIAD-MAC is independent of TRIAD-SC, and the internal state of TRIAD-MAC is secret for attackers. Therefore, attackers need to use probabilistic events for two colliding internal states. In other words, if many active AND gates are forced, we can guarantee that attackers are unlikely to succeed in a forgery attack. Assuming a 1-bit difference is induced at clock t, we evaluate the minimum number of active AND gates under the condition that attackers can induce any difference from clock $t+1$ to clock $t+r-1$. Table 2 shows the number of active AND gates.

Table 2: Number of active AND gates

Rounds r	160	192	224	256	288	320	352	384
# active AND gates	19	26	36	53	62	85	89	89

Since the bit length of the tag of TRIAD-MAC is 64 bits, attackers can trivially forge any ciphertext by trying 2^{64} tags. To avoid the nontrivial forgery attack, we need to guarantee at least 64 active AND gates. As a result, TRIAD-MAC guarantees 89 active AND gates.

4.6 Security of Hash Function

TRIAD-HASH follows the extended sponge construction [GPP11]. If the underlying permutation TRIAD-P does not have any structural weakness, the following bounds apply for its collision, second, preimage and preimage resistance.

Collision $\min\{2^{n/2}, 2^{c/2}\}$

Second-preimage $\min\{2^{n}, 2^{c/2}\}$

preimage $\min\{2^{\min\{n,t\}}, \max\{2^{\min\{n,t\}-r'}), 2^{c/2}\}\}$

For $n = 256$, $c = 224$, $r = 32$, $r' = 128$ and $t = 256$, these concrete bounds are given as in Table. 3.

Table 3: Security of collision, 2nd preimage and preimage attacks.

Collision	2nd preimage	Preimage
2^{112}	2^{112}	2^{128}

4.7 Fault Attacks

Fault attacks on stream ciphers as well as lightweight block ciphers is a well researched topic as is apparent from numerous papers in literature [BMS12, BM13, HR08]. If the attacker is able to apply time and/or location synchronized bit flipping faults to either the state register or the LFSR, he may be able to determine the internal state of the cipher by comparing the faulty and fault-free keystream bits and formulating enough equations to solve for the internal state. Once the internal state is found, the can be calculated, just by clocking back the state to its initial position. Thus it will always be possible to mount fault attacks on unprotected lightweight designs provided the adversary is resourceful enough and he can rekey the cipher multiple times using the same key and nonce. The ability of the adversary to rekey the device is essential as the main attack strategy of the adversary to deduce internal state by ovserving difference between faulty and faultfree ciphertext pairs. Thus the attck must necessarily cause the attacker to abuse nonce misuse privileges. Thus if a fault attack is to be thwarted, the cipher needs to be implemented with adequate protection circuitry.

4.8 Security of Related-Key Attacks

The secret key and nonce are loaded into the initial state of TRIAD-AE, and then TRIAD-P or TRIAD-$\bar{\text{P}}$ is applied. If the underlying permutations do not have any structural weakness, there is no significant difference between the key and nonce. Therefore, we think that TRIAD-AE is also secure against related-key attacks.

4.9 Security under Unpreferable Use

TRIAD-AE should be used under the nonce respecting setting, and security risks will be caused under the unpreferable use. Then, we consider two cases: nonce repeat and releasing unverified plaintexts.

When the nonce is repeated, the same internal state and key stream sequence are generated. It clearly implies that confidentiality is no longer maintained. On the other hand, we claim that the secret key K cannot be recovered even if the nonce is repeated.

Next, we consider the case that unverified plaintexts are released. Attackers can query any ciphertext to a decryption oracle and get corresponding plaintext regardless of the tag verification. Similarly to the case of nonce repeat, confidentiality is no longer maintained. However, the information that attackers can newly get is only key stream, and the internal state is never recovered from the key stream. Therefore, we also claim that the secret key K cannot be recovered if unverified plaintexts are released.

5 Performance

5.1 Hardware Implementation

One of the findings of [BMA+18], was that unrolled implemetations of lightweight stream ciphers like Trivium [CP08] are very energy efficient. [BMA+18] showed that for encrypting multiple blocks of data, an implementation of Trivium that is unrolled to perform 160 round update computations in one clock cycle is more energy efficient than even Midori-128, [BBI+15], a block cipher designed specifically for energy efficiency. This was surprising because Trivium was initially designed to be efficiently unrollable to perform 64 round function computations in one clock cycle. However as the authors of [BMA+18] argued, since the Trivium round function is essentially very lightweight and unrolling even beyond 64 rounds does not increase hardware complexity significantly. It also does not increase the circuit depth that contributes to formation of transient glitches which is the principal source of dynamic power consumption in lightweight circuits [BBR16]. On the other hand, unrolling a circuit n times, brings down the time required to encrypt a given amount of data by a factor of n clock cycles. So although the power consumption of increasingly higher degree of unrolled circuits always increases, the reduction in time required leads inevitably to energy savings, for optimal values of n.

Trivium is a remarkably well designed cipher in many ways. It achieves efficiency in energy consumption, and a decade of intense scrutiny by the community has not been able to successfully cryptanalyze the full version of the cipher. However it is well known that the cipher does not provide 128 bit security, as outlined in [MB07], and so our motivation was to redesign the cipher to guarantee 128 bit security. Any attempt by us to design a cipher similar to Trivium, ran into familiar problems: a candidate cipher with good energy performance had a high linear correlation coefficient i.e. we could find a linear sum of output bits biased towards zero with probability signifcantly more than half, and a cipher having low enough correlation coefficent consumed much more energy than Trivium. We could not seem to do better than to simply increase the state size by a factor of 2 and scale the taps accordingly.

Our initial target was to design a cipher with 256 bit state to have an efficient lightweight implementation. After several experiments we came to the conclusion that in order to guarantee security agaist linear and guess and determine attacks we needed an additional AND gate in the circuit. In the past there have been similar designs proposed that advocate the use of additonal AND gates in the Trivium framework. Two such proposals are those made in the appendix of [MB07] and in Trivia-SC [CCHN15]. [MB07] suggested the use of 3 additional AND gates which increase the energy consumption significantly. [CCHN15] suggested adding the output of the additional AND gate to the output bit: this would not be able to take advantage of modern standard cell libraries that have provision for 3-input XOR gates. The output bit in Trivium is already produced by a 3-input XOR: adding another component would necessitate placement of another gate that leads to increased signal delay and additonal glitching.

Therefore our goal to to design a cipher with a single AND gate with proper security margins in place evolved in the manner as follows. We experimented with parameter sets that consisted of a) size of the registers, b) location of all the taps c) location of addition of the output of the additional AND gate. For parameter sets that met security requirements, we implemented the design in hardware

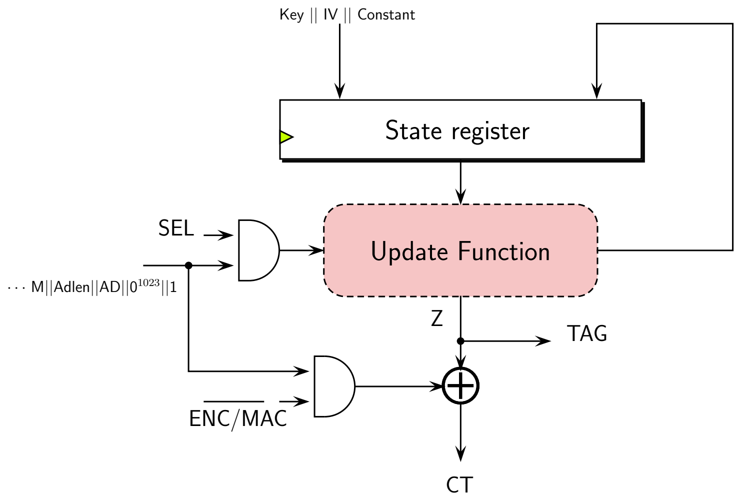

Figure 5: Hardware circuit for Triad-AE

and ran simulations to evaluate the energy consumption. The current design Triad-SC was chosen by experimenting with around 300 candidate designs.

5.1.1 Triad-SC

The first step was naturally to design an energy efficient stream cipher with Trivium as a reference point. We wanted flexibility of unrolling to be one of the design requirements, not only because energy efficiency is generally achieved at higher unrolled circuits, but also it gives the implementer the option to choose design as per the current area, power, throughput requirements. In table 4, we tabulate synthesis results of Triad-SC against lightweight stream ciphers and block ciphers in literature. These results place Triad-SC reasonably well in the design space.

5.1.2 Circuit Details

Figure 5 details the hardware circuit for the encryption and MAC routines of Triad-AE. The mode is designed in a manner so as to not require any additional state bits apart from the ones used in the stream cipher circuit. Thus we have a 256 bit state register which directly feeds the state update function. The initial input (denoted by Key || IV || Constant in the above figure) to the AE routine is loaded onto the state register which is updated by the round function in each clock cycle with additional message/associated data input. During the encryption stage, the output filter of the round function produces keystream that is added to message to produce ciphertext.

The round function circuit can be implemented in an unrolled fashion to perform more update computations and generate more keystream bits per clock cycle. The circuit can indeed be efficiently described in VHDL as per the guidelines given in [BMA+18, Appendix A]. In this document we present results for $n = 1,\ 8,\ 32,\ 64,\ 128,\ 256,\ 1024$. The input data to be absorbed is a bit string of form 10^{1023} (Initialization)$||AD||Adlen||M||10^{1023}$ (Finalization)$||0^{16}$ (MAC) during the MAC generation stage. For, an n times unrolled circuit, the above string is padded with zeros to make it a multiple of n and fed n bits at a time through the input port. For the encryption stage the input is 10^{1023} (Initialization)$||M$. Note that this time the message is not absorbed into the state, but just added with the keystream. So signals to control the addition of the input bitstream to the state, and to the keystream are be generated centrally depending on the length of the message and associated data.

#	Cipher	n	Area	Power	Energy (nJ)		
			(GE)	(μw)	16B	64B	256B
1	Grain v1	1	1005	38.9	1.120	2.614	8.589
		16	2673	86.6	0.156	0.364	1.195
		32	3934	165.1	0.149	0.347	1.139
		64	7474	561.3	0.281	0.617	1.965
2	Grain-128	1	1455	57.8	2.220	4.439	13.317
		32	3579	126.8	0.152	0.304	0.913
		64	6336	282.7	0.170	0.339	1.018
3	Trivium	1	1870	78.4	10.035	13.046	25.088
		64	3051	128.7	0.257	0.335	0.643
		128	4593	207.1	0.207	0.269	0.518
		256	7755	419.5	0.209	0.294	0.545
4	Kreyvium	1	2892	140.8	18.022	23.429	45.056
		64	4579	202.8	0.406	0.527	1.014
		128	5050	221.4	0.221	0.288	0.553
		256	8612	452.6	0.226	0.317	0.588
5	PRESENT	1	1440	52.2	0.345	1.378	5.512
		2	1968	91.3	0.310	1.242	4.967
		3	2500	149.0	0.358	1.430	5.721
6	Midori64	1	1542	60.6	0.206	0.824	3.297
		2	2017	100.6	0.181	0.724	2.897
		3	2826	273.8	0.383	1.533	6.133
7	Triad-SC	1	1425	50.5	5.818	7.756	15.513
		64	3040	122.5	0.220	0.294	0.588
		128	4733	200.2	0.180	0.240	0.480
		256	8165	426.3	0.213	0.256	0.512

Table 4: Comparison of energy for different degrees of unrolling, n denotes # unrolled rounds, Energy/bit figure calculated over 1000 blocks.

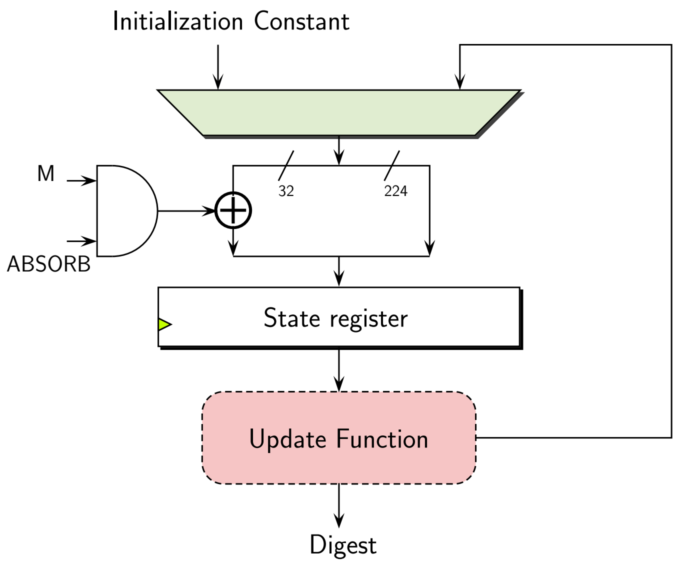

Figure 6: Hardware circuit for Triad-HASH

For Triad-HASH, the circuit is slightly different. Firstly, the round function is a lot more lightweight, since it does not have to generate keystream bits, and so the output filter fuction can be omitted. At the beginning, the Initialization Constant is loaded on to the state register after adding with the first block of message. Thereafter, the Triad-P permutation is run for 1024 rounds before the next block of message is xored. Again the update function can be implemented in an unrolled fashion with various degrees of unrolling. Note that if n is the degree of unrolling, each message block is absorbed after $\frac{1024}{n}$ cycles. Thus an ABSORB signal controlling the addition of the message to the state is generated centrally depending on the degree of unrolling.

5.1.3 Timing

For a dataset with associated data length = a bytes and message length = m bytes, the MAC generation needs to perform $R_{MAC} = 1024 + 8a + 56$ (For absorbing $Adlen$) $+ 8m + 1024 + 16 = 2120 + 8(a + m)$ round update functions. Similarly encryption would take $R_{ENC} = 1024 + 8m$ round updates. Thus any n-round unrolled implementation of Triad-AE would take $\lceil \frac{R_{ENC}}{n} \rceil$ clock cycles to encrypt and $\lceil \frac{R_{MAC}}{n} \rceil$ clock cycles to produce MAC. Triad-HASH needs 1024 round function updates to execute Triad-P, after absorbing every 4 bytes of message, and another 1024 to finalize the digest. Thus if the length of the 10^* padded message is w^* 4-byte blocks then $R_{HASH} = 1024w^* + 1024$ round updates are required. Any n-round unrolled implementation of the round function requires $\lceil \frac{R_{HASH}}{n} \rceil$ clock cycles to hash.

5.1.4 Performance

In Table 5 and 6 we present the synthesis results for Triad-AE and Triad-HASH. The following design flow was used: first the design was implemented in VHDL. Then, a functional verification was first done using Mentor Graphics Modelsim software. The designs were synthesized using the standard cell library of the 90nm logic process of STM (CORE90GPHVT v2.1.a) with the Synopsys Design Compiler, with the compiler being specifically instructed to optimize the circuit for area. A timing simulation was done on the synthesized netlist. The switching activity of each gate of the circuit was collected while running post-synthesis simulation. The average power was obtained using

Synopsys Power Compiler, using the back annotated switching activity.

Design	n	Area (GE)	Power (μW)	Energy(nJ) AD 0B, PT 16B	Energy(nJ) AD 16B, PT 16B	Energy(nJ) AD 16B, PT 32B
TRIAD-AE	1	1354	61.6	20.957	21.746	23.324
	8	1598	69.8	2.967	3.078	3.302
	32	2444	99.0	1.059	1.099	1.139
	64	3573	137.9	0.745	0.772	0.800
	128	5861	226.2	0.611	0.634	0.656
	256	10486	502.7	0.704	0.754	0.754
	1024	38191	5427.5	2.714	2.714	2.714

Table 5: Implementation results for TRIAD-AE. (Power reported at 10 MHz)

Design	n	Area (GE)	Power (μW)	Energy(nJ) 0B	8B	16B	32B	48B	64B
TRIAD-HASH	1	1344	69.1	14.152	21.228	28.303	42.455	56.607	70.758
	8	1466	74.4	1.905	2.857	3.809	5.714	7.619	9.523
	32	1964	91.7	0.587	0.880	1.174	1.761	2.348	2.934
	64	2630	108.7	0.348	0.522	0.696	1.044	1.391	1.739
	128	4016	172.6	0.276	0.414	0.552	0.828	1.105	1.381
	256	6791	357.3	0.286	0.429	0.572	0.858	1.143	1.429
	1024	23467	3436.5	0.687	1.031	1.375	2.062	2.749	3.436

Table 6: Implementation results for TRIAD-HASH. (Power reported at 10 MHz)

Tables 5 and 6 unearth interesting results about the energy consumption of TRIAD-AE and TRIAD-HASH. Table 5 lists the energy consumed for performing the encryption and MAC routines of TRIAD-AE for different datalengths whereas table 6 lists the corresponding energy consumed to hash messages of given lengths. As outlined in [BMA$^+$18], we obtain a quasi-parabolic behaviour wrt to unrolling. For both TRIAD-HASH and TRIAD-AE, the 128x unrolling is the most efficent energywise whereas the 32x or the 64x implementation provides a good balance between number of clock cycles required to process, area and energy. We also provide a 1024x implementation for high throughput requirements.

5.2 Software Implementation

Although the main target of TRIAD is hardware, we can have good performance on the high-end software. We show how to implement TRIAD efficiently by using the SIMD in the Intel architecture.

Since updated three bits are never used after 64 rounds, the specification allows us to compute 64 rounds simultaneously. When each register in the internal state of TRIAD can be stored into each register, BIT-SHIFT, XOR, and AND operations can allow us to compute 64 rounds. In the case of Intel CPU, we store these three registers into 128-bit `xmm` registers. Unfortunately, there is no single BIT-SHIFT instruction for the whole of the `xmm` register. For example, `PSLLQ` instruction shift the top and bottom 64 bits independently. Only BYTE-SHIFT instruction is available for the whole of the `xmm` register. To solve this issue, we restrict that the number of unrolled rounds is up to 56 rounds, and then, we can collect the necessary 56-bit value in the top 64 bits by using two operations, `PSLLDQ` and `PSLLQ`.

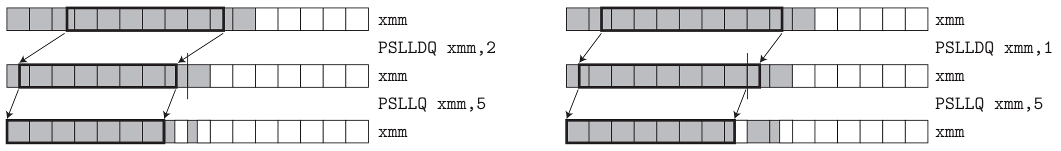

Figure 7: BIT-SHIFT for `TriadUpd`

The left figure of Fig. 7 shows an example to extract 56-bit value from the 77-th bit of the third register. The third register is shifted by 2 bytes to the left in `PSLLDQ`, and it is shifted by 5 bits to the left in `PSLLQ`. Eventually, the 77-th bit is shifted to the 56-th bit. Note that only 1 byte is shifted in `PSLLDQ`, the 77-th bit vanishes in `PSLLQ`.

Another important fact is that Triad-SC and Triad-MAC shares almost the same operation. The difference is only the absorb of message and generation of key stream, and the main update function is the same. Therefore, we use 256-bit `ymm` registers instead of `xmm` registers, and Triad-SC and Triad-MAC are processed in the top and bottom 128 bits, respectively. Namely, while Triad-AE has the two streams, they are fully parallelizable.

Table 7: Cycle per Byte on Intel Core i7-7660U (Kaby Lake)

# of encrypted bytes		32	64	1024	8192	65536
cycles per Byte	Triad-AE	24.46	14.81	3.81	3.18	3.06
	Triad-HASH	94.25	84.10	76.15	74.16	73.86

Table 7 summarizes the benchmarks of Triad-AE and Triad-HASH, where Intel Core i7-7660U (Kaby Lake) is used. The speed of Triad-AE is 3.06 cpb and fast enough. Triad-HASH is slower than Triad-AE, but this is because that the claimed security is 112 bits in spite of 256-bit state. We think that this processing speed is not slow for the hash function with such an extremely small state.

6 Acknowledgment

Subhadeep Banik is supported by the Ambizione grant PZ00P2_179921, awarded by the Swiss National Science Foundation. Takanori Isobe is supported by Grant-in-Aid for Scientific Research (C) (KAKENHI 17K12698) for Japan Society for the Promotion of Science.

References

[Bab95] S. H. Babbage. Improved "exhaustive search" attacks on stream ciphers. In *European Convention on Security and Detection, 1995.*, pages 161–166, May 1995.

[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 Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 411–436. Springer, Heidelberg, November / December 2015.

[BBR16] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Exploring energy efficiency of lightweight block ciphers. In Orr Dunkelman and Liam Keliher, editors, *SAC 2015*, volume 9566 of *LNCS*, pages 178–194. Springer, Heidelberg, August 2016.

[BDPA15] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sha-3 standard: Permutation-based hash and extendable-output functions. FIPS PUB 202, 2015.

[BDPV08] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 181–197. Springer, Heidelberg, April 2008.

[BM13] Subhadeep Banik and Subhamoy Maitra. A differential fault attack on MICKEY 2.0. In Guido Bertoni and Jean-Sébastien Coron, editors, *CHES 2013*, volume 8086 of *LNCS*, pages 215–232. Springer, Heidelberg, August 2013.

[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. Symm. Cryptol.*, 2018(2):1–19, 2018.

[BMS12] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. A differential fault attack on the Grain family of stream ciphers. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 122–139. Springer, Heidelberg, September 2012.

[BS00] Alex Biryukov and Adi Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 1–13. Springer, Heidelberg, December 2000.

[CCHN15] Avik Chakraborti, Anupam Chattopadhyay, Muhammad Hassan, and Mridul Nandi. TriviA: A fast and secure authenticated encryption scheme. In Tim Güneysu and Helena Handschuh, editors, *CHES 2015*, volume 9293 of *LNCS*, pages 330–353. Springer, Heidelberg, September 2015.

[CP08] Christophe De Cannière and Bart Preneel. Trivium. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 244–266. Springer, 2008.

[DL11] Ming Duan and Xuajia Lai. Improved zero-sum distinguisher for full round Keccak-f permutation. Cryptology ePrint Archive, Report 2011/023, 2011. `http://eprint.iacr.org/2011/023`.

[DS09] Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 278–299. Springer, Heidelberg, April 2009.

[FS03] Soichi Furuya and Kouichi Sakurai. Single-path authenticated-encryption scheme based on universal hashing. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002*, volume 2595 of *LNCS*, pages 94–109. Springer, Heidelberg, August 2003.

[Gol96] Jovan Dj. Golic. Linear models for keystream generators. *IEEE Trans. Computers*, 45(1):41–49, 1996.

[Gol97] Jovan Dj. Golic. Cryptanalysis of alleged A5 stream cipher. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 239–255. Springer, Heidelberg, May 1997.

[GPP11] Jian Guo, Thomas Peyrin, and Axel Poschmann. The PHOTON family of lightweight hash functions. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 222–239. Springer, Heidelberg, August 2011.

[HR08] Michal Hojsík and Bohuslav Rudolf. Differential fault analysis of Trivium. In Kaisa Nyberg, editor, *FSE 2008*, volume 5086 of *LNCS*, pages 158–172. Springer, Heidelberg, February 2008.

[Inc15] Gurobi Optimization Inc. Gurobi optimizer 6.5. Official webpage, `http://www.gurobi.com/`, 2015.

[Lai94] Xuejia Lai. Higher order derivatives and differential cryptanalysis. In *Communications and Cryptography*, volume 276 of *The Springer International Series in Engineering and Computer Science*, pages 227–233, 1994.

[MB07] Alexander Maximov and Alex Biryukov. Two trivial attacks on Trivium. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *SAC 2007*, volume 4876 of *LNCS*, pages 36–55. Springer, Heidelberg, August 2007.

[TIHM17] Yosuke Todo, Takanori Isobe, Yonglin Hao, and Willi Meier. Cube attacks on non-blackbox polynomials based on division property. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 250–279. Springer, Heidelberg, August 2017.

[TIM$^+$18] Yosuke Todo, Takanori Isobe, Willi Meier, Kazumaro Aoki, and Bin Zhang. Fast correlation attack revisited - cryptanalysis on full grain-128a, grain-128, and grain-v1. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 129–159. Springer, Heidelberg, August 2018.

[TM16] Yosuke Todo and Masakatu Morii. Bit-based division property and application to simon family. In Thomas Peyrin, editor, *FSE 2016*, volume 9783 of *LNCS*, pages 357–377. Springer, Heidelberg, March 2016.

[Tod15] Yosuke Todo. Structural evaluation by generalized integral property. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 287–314. Springer, Heidelberg, April 2015.

[XZBL16] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 648–678. Springer, Heidelberg, December 2016.

A Test Vectors

A.1 Triad-AE

Key	00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0A, 0B, 0C, 0D, 0E, 0F
Nonce	00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0A, 0B
PT	
AD	
CT	D1, 6D, CC, A6, B3, 34, CB, 84

Key	00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0A, 0B, 0C, 0D, 0E, 0F
Nonce	00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0A, 0B
PT	00
AD	00
CT	F5, D4, 1A, F0, 01, D9, D7, 53, 67

A.2 Triad-HASH

Msg	
MD	BB, 98, 1A, 14, EF, A8, EA, 5D, 30, 8E, 79, 55, F7, 40, 2C, 94
	1B, 20, 9E, A4, 13, 19, FE, EE, 3D, 57, 1A, 4C, 44, 17, 5E, 6E

Msg	00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0A, 0B, 0C, 0D, 0E, 0F
MD	35, 54, 06, 34, 1E, BF, B7, 08, 68, C2, 5A, 10, A7, 18, 3F, B7
	6B, F2, D7, C0, 21, 86, 17, FD, 0B, 41, E1, 1E, 16, 87, EA, 34