ESTATE Authenticated Encryption Mode: Hardware Benchmarking and Security Analysis

Avik Chakraborti[1], Nilanjan Datta[2], Ashwin Jha[2], Cuauhtemoc Mancillas Lopez[3], Mridul Nandi[2], Yu Sasaki[1]

[1] NTT Secure Platform Laboratories, Japan
[2] Indian Statistical Institute, Kolkata, India
[3] Computer Science Department, CINVESTAV-IPN, Mexico
chakraborti.avik@lab.ntt.co.jp,nilanjan_isi_jrf@yahoo.com,ashwin.jha1991@gmail.com,cuauhtemoc.mancillas83@gmail.com,mridul.nandi@gmail.com,sasaki.yu@lab.ntt.co.jp

Abstract. In this draft, we consider the lightweight and low energy authenticated encryption family, called ESTATE, that significantly improves the design of SUNDAE in terms of implementation costs (both hardware area and energy) and efficient processing of short messages. In particular, ESTATE does not require additional multiplication circuit, and it reduces the number of block cipher calls by one. Moreover, it provides integrity security even under the release of unverified plaintext (or RUP) model. ESTATE is based on short-tweak tweakable block ciphers (or tBC) and we instantiate it with two recently designed tBCs: TweAES-128 and TweGIFT-128. We also propose a low latency variant of ESTATE, called sESTATE, that uses a round-reduced (6 rounds) variant of TweAES-128 called TweAES-128-6. We provide comprehensive FPGA based hardware implementation for all the three instances. The implementation results overwhelmingly depict that ESTATE_TweGIFT-128 and ESTATE_TweAES-128 consume lesser area as well as achieve higher throughput for short messages, as compared to SUNDAE_GIFT-128 and SUNDAE_AES-128. We also present concrete security analysis for both the modes.

Keywords: SUNDAE, AES, GIFT, authenticated encryption, lightweight, elastic-tweak

1 Formal Specifications of ESTATE and sESTATE

In this section, we describe the specification of ESTATE [15] mode of operation based on tweakable block ciphers. We also give a detailed algorithmic description for the mode. Finally, we list the recommended instantiations. ESTATE_TweAES-128, sESTATE_TweAES-128-6 and ESTATE_TweGIFT-128. We use the tweakable block ciphers TweAES-128 and TweGIFT-128, used for listed instantiations. The block ciphers are designed by Chakraborti et al. [16]. We use exactly the same specification as proposed in [16].

1.1 ESTATE AEAD Mode

ESTATE is roughly based on the MAC-then-Encrypt paradigm. It is composed of an FCBC like MAC, we call FCBC*, and the OFB mode of encryption. ESTATE is parametrized by its underlying tweakable block cipher $\widetilde{\mathsf{E}}$-$n/\tau/\kappa$. It operates on n-bit data blocks at a time using a tweakable block cipher. Complete specification of ESTATE is presented in Algorithm 1. The pictorial description is given in Figure 1, 2, and 3.

1.1.1 FCBC*: Tag Generation Phase

The tag generation phase is a tweakable variant of FCBC, where distinct tweaks are used to instantiate multiple instantiations of the block cipher. The distinctness in tweaks is used to separate different cases based on the length of associated data and message. We represent a tweak value in 4 bits and the tweak value i represents the 4-bit binary representation of integer i. The processing of first block (i.e. nonce N) uses the tweak value

1. The intermediate blocks are always processed with tweak 0, to minimize the overheads. The last block processing may use tweak tweaks 2, 4, 6, if the block is full, or 3, 5, 7, if the block is partial.

1.1.2 OFB: Encryption Phase

The encryption phase is built on the well-known OFB mode, where we fix the tweak value to 0, again to minimize the tweak injection overhead.

Algorithm 1 ESTATE Authenticated Encryption and Verified Decryption Algorithm

```
1: function ESTATE.Enc[$\widetilde{E}$]($K, N, A, M$)
2:     $T$    MAC[$\widetilde{E}$]($K, N, A, M$)
3:     $C$    OFB[$\widetilde{E}$]($K, T, M$)
4:     return ($C, T$)

5: function MAC[$\widetilde{E}$]($K, N, A, M$)
6:     if $|A| = 0$ and $|M| = 0$ then
7:         return $T$    $\widetilde{E}_K^8(N)$
8:     $T$    $\widetilde{E}_K^1(N)$
9:     if $|A| > 0$ then
10:        $A_{a-1} \| \cdots \| A_0$    $A$
11:        $t$    $(|M| > 0\ ;\ |A_{a-1}| = n)\ ?\ 2 : 3 : 6 : 7$
12:        $T$    FCBC*[$\widetilde{E}$]($K, T, A, t$)
13:    if $|M| > 0$ then
14:        $M_{m-1} \| \cdots \| M_0$    $M$
15:        $t$    $(|M_{m-1}| = n)?\ 4 : 5$
16:        $T$    FCBC*[$\widetilde{E}$]($K, T, M, t$)
17:    return $T$
```

```
1: function ESTATE.DEC[$\widetilde{E}$]($K, N, A, C, T$)
2:     $M$    OFB[$\widetilde{E}$]($K, T, C$)
3:     $T'$    MAC[$\widetilde{E}$]($K, N, A, M$)
4:     return $(T' = T)?\ M : \perp \leftarrow$

5: function FCBC*[$\widetilde{E}$]($K, T, D, t$)
6:     $D_{d-1} \| \cdots \| D_0$    $D$
7:     for $i = 0$ to $d - 2$ do
8:         $T$    $\widetilde{E}_K^0(T \oplus D_i)$
9:     $T$    $\widetilde{E}_K^t\big(T \oplus \mathsf{ozp}(D_{d-1})\big)$
10:    return $T$

11: function OFB[$\widetilde{E}$]($K, T, M$)
12:    $M_{m-1} \| \cdots \| M_0$    $M$
13:    for $i = 0$ to $m - 1$ do
14:        $T$    $\widetilde{E}_K^0(T)$
15:        $C_i$    $\mathsf{chop}(T, |M_i|) \oplus M_i$
16:    return $(C_{m-1} \| \cdots \| C_0)$
```

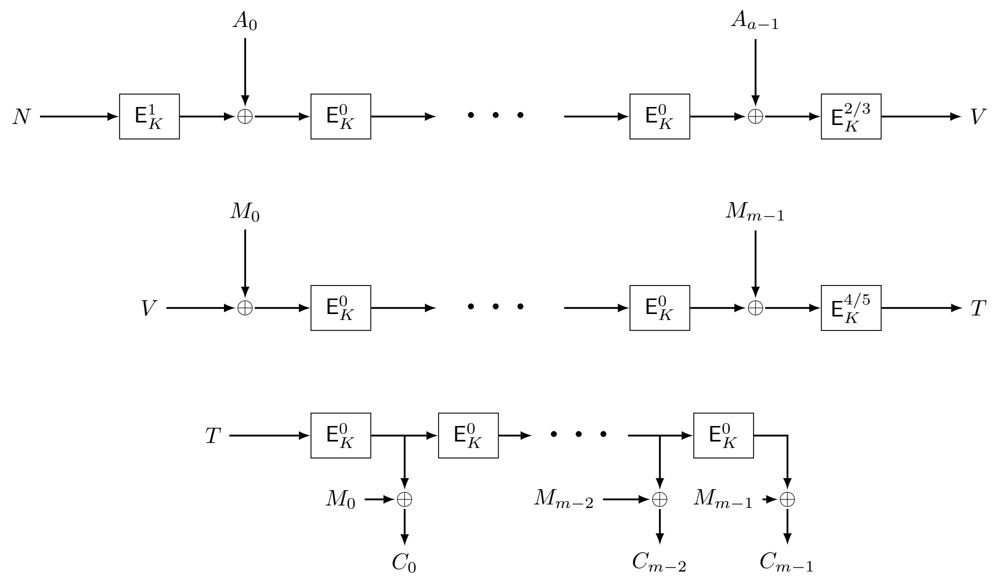

Figure 1: ESTATE with a AD blocks and m message blocks

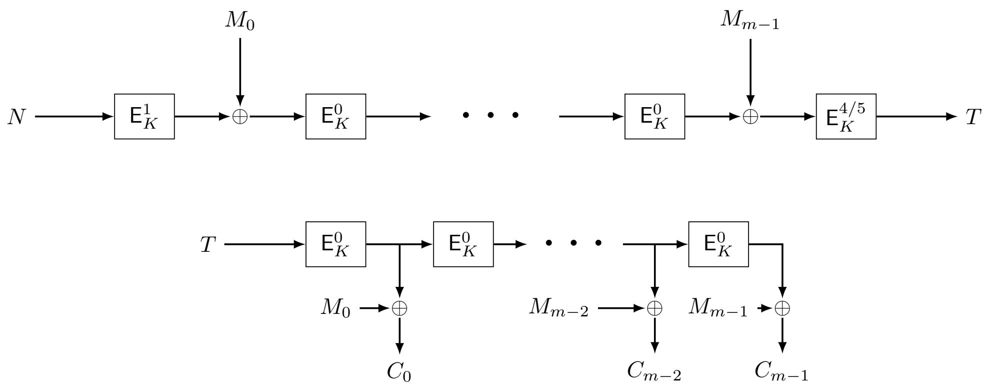

Figure 2: ESTATE with empty AD and m message blocks

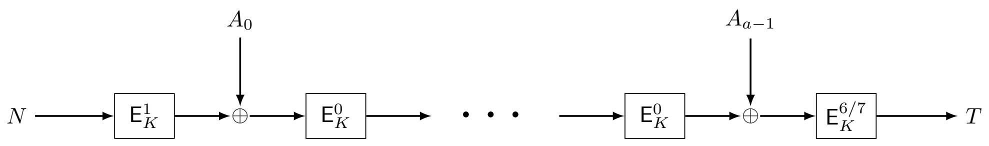

Figure 3: ESTATE with a AD blocks and empty message

1.2 sESTATE AEAD Mode

Along with ESTATE, we also define a lighter version of ESTATE, called sESTATE where we use two tweakable block ciphers: $\widetilde{\mathsf{E}}$ and a round-reduced variant of $\widetilde{\mathsf{E}}$, represented by $\widetilde{\mathsf{F}}$. The tweakable block cipher $\widetilde{\mathsf{F}}$ replaces $\widetilde{\mathsf{E}}$ in processing of non-last blocks in the MAC function. For all other tweakable block cipher calls, i.e. for processing the last block in MAC function and the full OFB processing, $\widetilde{\mathsf{E}}$ is used as usual. Further $\widetilde{\mathsf{F}}$, is always employed with tweak value 15, in order to maintain maximum distance between the 0 tweak calls to $\widetilde{\mathsf{E}}$ and calls to $\widetilde{\mathsf{F}}$. Algorithm 2 gives the algorithmic description of sESTATE.

1.3 Recommended Instantiations

We recommend the following concrete instantiations:

- ESTATE_TweAES-128: This AEAD scheme obtained by instantiating ESTATE mode of operation with $\widetilde{\mathsf{E}}$:=TweAES-128 block cipher. Here the size of the key, nonce and tag are 128 bits each. TweAES-128 is a 128-bit tweakable block cipher with 4-bit tweak and 128-bit key, proposed in [16]. As the name suggests, it is a tweakable variant of AES-128-128/128 [2] block cipher. TweAES-128 is identical to AES-128-128/128 except that we inject a tweak value at intervals of 2 rounds.

- ESTATE_TweGIFT-128: This AEAD scheme is obtained by instantiating ESTATE mode of operation with $\widetilde{\mathsf{E}}$:=TweGIFT-128 block cipher. Here the size of the key, nonce and tag are 128 bits each. TweGIFT-128 is also a 128-bit tweakable block cipher with 4-bit tweak and 128-bit key, proposed in [16]. As the name suggests, it is a tweakable variant of GIFT-128/128 [10] block cipher. TweGIFT-128 is identical to GIFT-128/128 except that we inject a tweak value at intervals of 5 rounds. We recommend ESTATE_TweGIFT-128, for hardware-oriented ultra-lightweight applications.

Algorithm 2 sESTATE Authenticated Encryption and Verified Decryption Algorithm. Here $\widetilde{\mathsf{F}}$ is a round-reduced variant of $\widetilde{\mathsf{E}}$

```
1: function sESTATE.Enc[$\widetilde{E}$,$\widetilde{F}$]$(K, N, A, M)$
2:     $T$    MAC[$\widetilde{E}, \widetilde{F}$]$(K, N, A, M)$
3:     $C$    OFB[$\widetilde{E}$]$(K, T, M)$
4:     return $(C, T)$

5: function MAC[$\widetilde{E}$,$\widetilde{F}$]$(K, N, A, M)$
6:     if $|A| = 0$ and $|M| = 0$ then
7:         return $T$    $\widetilde{E}^{8}_{K}(N)$
8:     $T$    $\widetilde{F}^{15}_{K}(N)$
9:     if $|A| > 0$ then
10:        $A_{a-1} \| \cdots \| A_0$    $A$
11:        $t$    $(|M| > 0\ ;\ |A_{a-1}| = n)\ ?\ 2 : 3 : 6 : 7$
12:        $T$    FCBC*[$\widetilde{E}, \widetilde{F}$]$(K, T, A, t)$
13:    if $|M| > 0$ then
14:        $M_{m-1} \| \cdots \| M_0$    $M$
15:        $t$    $(|M_{m-1}| = n)?\ 4 : 5$
16:        $T$    FCBC*[$\widetilde{E}, \widetilde{F}$]$(K, T, M, t)$
17:    return $T$
```

```
1: function sESTATE.DEC[$\widetilde{E}$,$\widetilde{F}$]$(K, N, A, C, T)$
2:     $M$    OFB[$\widetilde{E}$]$(K, T, C)$
3:     $T'$    MAC[$\widetilde{E}, \widetilde{F}$]$(K, N, A, M)$
4:     return $(T' = T)?\ M : \perp \leftarrow$

5: function FCBC*[$\widetilde{E}$,$\widetilde{F}$]$(K, T, D, t)$
6:     $D_{d-1} \| \cdots \| D_0$    $D$
7:     for $i = 0$ to $d - 2$ do
8:         $T$    $\widetilde{F}^{15}_{K}(T \oplus D_i)$
9:     $T$    $\widetilde{E}^{t}_{K}\big(T \oplus \mathsf{ozp}(D_{d-1})\big)$
10:    return $T$

11: function OFB[$\widetilde{E}$]$(K, T, M)$
12:    $M_{m-1} \| \cdots \| M_0$    $M$
13:    for $i = 0$ to $m - 1$ do
14:        $T$    $\widetilde{E}^{0}_{K}(T)$
15:        $C_i$    chop$(T, |M_i|) \oplus M_i$
16:    return $(C_{m-1} \| \cdots \| C_0)$
```

- sESTATE_TweAES-128-6: This AEAD scheme is obtained by instantiating sESTATE mode of operation with $\widetilde{\mathsf{E}}$:=TweAES-128, $\widetilde{\mathsf{F}}$:=TweAES-128-6 block cipher. Again, the size of the key, nonce and tag are 128 bits each. TweAES-128-6 is a reduced round variant of TweAES-128, which is composed of the first 6 rounds of TweAES-128. Notably, the last round (6-th round) includes the MixColumns operations, and the AddTweak step is called in the 2-nd and 4-th rounds. We recommend sESTATE_TweAES-128-6, for higher throughput demanding, and energy-constrained applications.

1.4 Design Rationale

We briefly describe the rationale of our proposal:

1. **Choice of the Mode.** Our basic goal is to design an ultra-lightweight mode, which is especially efficient for short messages, and secure against nonce misuses. For this, we choose SIV as base and then introduce various tweaks to make the construction single-state and inverse free, much in the same vein as in the case of SUNDAE.

2. **Use of Tweakable Block Cipher.** We use tweakable block cipher with 4-bit tweak primarily for the purpose of various domain separations such as the type of the current data (associated data or message), completeness of the final data block (partial or full), whether the associated data and/or message is empty etc. Note that, without the use of these tweaks, these domain separations would cost a few constant field multiplications and/or additional block cipher invocations, which would in turn increase the hardware footprint as well as decrease the energy efficiency and throughput for short messages.

3. **Choice of TweAES-128 and TweGIFT-128.** For both these versions, we expand the tweak in such a way that the extended tweak has very high distance and the extension can be done using only 7 bit XOR operations. We choose the tweak positions and the interval of the tweak injection with the primary goal that the tweakable versions should have similar security level as the underlying block ciphers, while minimizing the tweak injection overhead.

4. **Choice of the Tweaks.** Here we provide a detailed justification for the choice of the tweaks.

(i) Tweak for Processing Bulk Messages. We use tweak 0 for all the block ciphers used in the OFB part and all the intermediate block ciphers in the MAC function. Since TweAES-128 and TweGIFT-128 with zero tweaks are essentially AES-128 and GIFT respectively, no additional overhead is introduced in the software for longer messages due to the use of tweakable block ciphers.

(ii) Tweak for First Block Cipher Invocation. We use a separate tweak (tweak value 1) for the first block cipher invocation in the MAC function so that the adversary does not have any control over the inputs of the intermediate block ciphers. This essentially ensures the RUP security of the mode.

(iii) Tweak for Finalization. For the purpose of domain separation, we use tweak 2 and 3 (full and partial resp.) for the final AD block processing and tweak 4 and 5 (full and partial resp.) for the final plaintext block processing.

(iv) Tweak Choices for sESTATE. For sESTATE, we always use tweak 15 for the round-reduced block ciphers to maximize the distance with other tweaks, most importantly tweak 0 whose inputs and outputs are observed through OFB. In this way, we make TweAES-128-6 with tweak value 15 and TweAES-128 with tweak value 0 as much independent as possible.

1.5 ESTATE in the Light of NIST Lightweight Competition

NIST lightweight cryptography project [30] was started in 2018 recognizing the lack of efficient AE standards for lightweight applications. They mainly addressed the growing security requirements under the backdrop of the applications sensor network, distributed control systems, health care, and several others. These applications are mainly involved with resource-restricted devices communicating among themselves. There are quite a few candidates that have been submitted in the competition which uses MAC-then-Encrypt (SIV) paradigm. Here we present a comparative chart in Table 1 to study various SIV based modes submitted in the NIST competition with our proposal ESTATE.

Table 1: Comparative Study on SIV based NIST Round-1 Candidates with ESTATE. A block cipher with block size of n bits and key size of κ bits is denoted as BC-n/κ and a tweakable block cipher with n bit block, κ bit key and τ bit tweak is denoted as TBC-$n/\kappa/\tau$ (tBC-$n/\kappa/\tau$ for short tweaks)

Submission	Primitive	State size (bits)	Optimality	INT-RUP	Mult-free
ESTATE	tBC-128/128/4	260	✓	✓	✓
Limdolen	BC-128/128	384	×	×	×
SIV-Rijndael256	tBC-256/128/4	388	✓	✓	✓
SIV-TEM-PHOTON	TBC-256/128/132	516	✓	✓	✓
SUNDAE-GIFT	BC-128/128	256	×	×	×
TRIFLE	BC-128/128	384	×	×	×

The above chart depicts that considering area and energy efficiency as the light-weight metric, ESTATE has clear advantages over others since (i) it uses a state size of only 260-bits, (ii) do not use any field multiplications, (iii) achieves optimality on the number of primitive invocations, hence energy efficient and (iv) secure against INT-RUP adversaries.

2 Hardware Implementation

In this section, we describe the hardware implementation details of our cipher family ESTATE. We describe the details only for ESTATE_TweAES-128. We also provide our implementation results of all the members of the ESTATE family along with the implementation results of SUNDAE_AES-128 and SUNDAE_GIFT-128. We implement both the versions of SUNDAE by own using exactly the same interface and following the same architectural properties to have a fair comparison. In addition, we use the AES only encryption core provided in GMU Caesar Package [1] for both ESTATE_TweAES-128 and SUNDAE_AES-128. The details are given below.

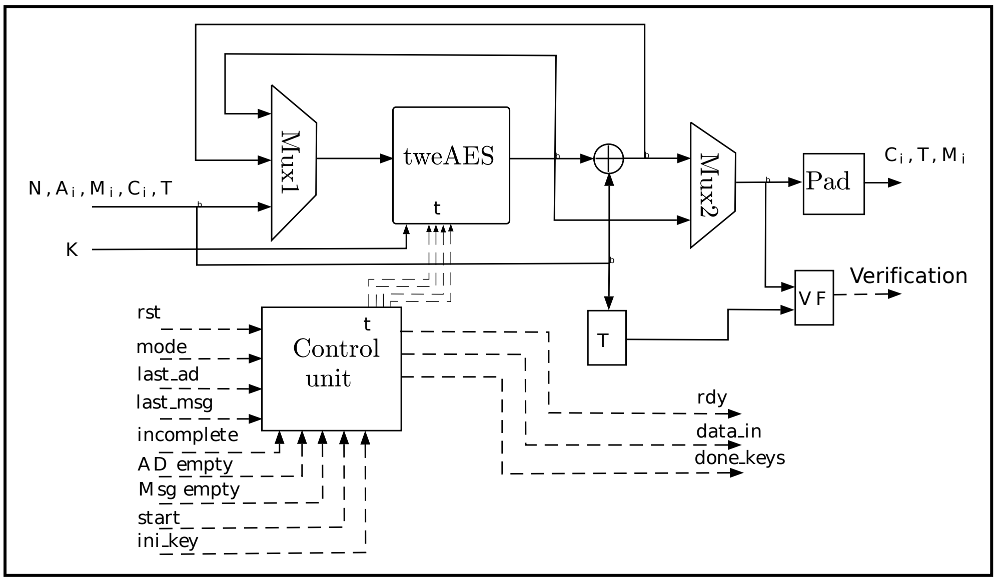

Figure 4: Hardware Architectures of ESTATE_TweAES-128

2.1 Hardware Implementation of TweAES-128

We provide a brief implementation description TweAES-128. Our implementation is based on the AES enc only core provided in [1], we just add the injection of the tweak. The architecture is shown in Figure 4. As tweak additions are not performed during all the rounds, a multiplexer is used for each bit positions where the tweak bits are injected. The overhead introduced per 1-bit inputs by these two modules consists of only 4 xors and 8 two-input multiplexers.

2.2 Hardware Architecture of ESTATE_TweAES-128

In this section, we describe the implementation of combined enc/dec architecture of ESTATE_TweAES-128. The main modules are mentioned below:

- **Registers.** Only an 128 bit register to maintain TweAES-128 state in ESTATE_TweAES-128. It is mentioned in Fig. 4.

- **Multiplexers.** Mux1 selects the input to TweAES-128 (can perform three operations: encrypt one single block in ECB mode, compute the CBC mode or generate the enc/dec stream in the OFB mode. Using Mux1, TweAES-128 gets the instruction which mode it should work. The output from TweAES-128 (direct or xored with input block) is input to Mux2 (to denote whether the architecture executes enc or dec or tag generation).

- **Pad**. This module receives as input the selected output from Mux2 and outputs either the full block for tag or partial block for message or cipher text.

- **VF**. It performs the verification process when the architecture is executed in the decryption mode, and it compares the content of the register T with the output of TweAES-128 computed from the associated data and the decrypted message.

- **Control unit.** The control unit consists of 8 states and several signals to different modules to control execution of the state changes.

2.3 Implementation Results of ESTATE and Benchmark with SUNDAE

We provide our FPGA implementation results (codes are written in VHDL and are implemented on Virtex 7 xc7vx485t in Vivado v.2018.2.2) for all the members of the ESTATE family along with SUNDAE_AES-128 and SUNDAE_GIFT-128. We use the RTL approach and use a basic round-based architecture. The detailed implementation results are depicted in Table 2.

Table 2: ESTATE and SUNDAE (combined enc/dec circuit) Implemented FPGA Results

Scheme	# Slice Registers	# LUTs	# Slices	Frequency (MHZ)	Throughput (Gbps)	Mbps/ LUT	Mbps/ Slice
TweAES-128	524	1605	559	330.52	3.85	2.40	6.90
TweGIFT-128	526	465	171	559.33	1.74	3.74	10.18
ESTATE_TweAES-128	803	1901	602	303.00	1.94	1.02	3.22
sESTATE_TweAES-128	813	1903	602	302.20	2.42	1.27	4.02
ESTATE_TweGIFT-128	796	681	263	526.00	0.84	1.23	3.20
SUNDAE_AES-128	799	1922	614	302.81	1.93	1.01	3.16
SUNDAE_GIFT-128	682	931	310	526.03	0.84	0.90	2.71

We can observe that the overhead introduced by the implementation of STATE is more significant in case of ESTATE_TweGIFT-128 since GIFT is significantly smaller than AES. The latency for TweAES-128 is 10 clock cycles configured as bulk encryption while for the reduced 6-round version it is 6 clock cycles, this is directly reflected in the throughput. Computing the throughput to process a message, ESTATE_TweAES-128 uses 20 clock cycles per block and sESTATE_TweAES-128 16. Observe that, both the versions of ESTATE are better (in hardware area) than SUNDAE. However, ESTATE_TweGIFT-128 is significantly area-efficient than SUNDAE_GIFT-128.

2.4 Short Message Processing for SUNDAE and ESTATE

Forg short message processing, we only compare between ESTATE_TweAES-128 and SUNDAE_AES-128. We can briefly mention the difference in the number of clock cycles by taking an example of one input data block (16 bytes). We make the following assumption. A possible nonce based version of SUNDAE prepends the nonce with the associated data (this assumption is also used in the NIST submitted version of SUNDAE [9]). Considering the nonce as the first block of the associated data, we assume the associated data length is always 16 bytes or one block. When we say that the message length is 16 bytes, then overall we consider one block associated data (i.e, the nonce) and one block message. In this case, SUNDAE invokes four block cipher calls, such that we need one block cipher call to encrypt the constant, one block cipher call to encrypt the nonce and two block cipher calls for the message. ESTATE avoids the block cipher call for the constant and makes three block cipher calls. In our architecture, to process a 16-byte message, ESTATE_TweAES-128 requires 31 cycles where as SUNDAE_AES-128 needs 41 clock cycles. Details with larger messages are given in Table 3 below. Note that, the throughputs for both the schemes converge to the same value with an increase in the input lengths.

2.5 Benchmarking ESTATE

We provide a benchmark of the hardware implementation results of all the members in the ESTATE family using some of the implementation listed in Athena website [6] along with the implementation results in [17–20,32] on Virtex 7. The results depict that ESTATE provides a very competitive performance. In fact, ESTATE_TweGIFT-128 is one of the best in the literature (only next to SAEB and ACORN). Note that, we directly use the AES only encryption core provided in the GMU Caesar Package [1] and we use our own implementation for TweGIFT-128.

Table 3: Throughput Comparison for Short Message Processing

	SUNDAE_AES-128						ESTATE_TweAES-128					
Message Length (bytes)	16	32	64	128	512	2048	16	32	64	128	512	2048
Clock Cycles	41	61	101	181	661	2581	31	51	91	171	651	2571
Throughput (Mbps)	945.36	1270.81	1535.04	1713.13	1876.41	1922.21	1251.10	1520.94	1704.79	1814.46	1906.43	1930.90

Table 4: Comparison on Virtex 7 [7]. Here BC denotes block cipher, SC denotes Stream cipher, (T)BC denotes (Tweakable) block cipher and BC-RF denotes the block cipher's round function,'-' means that the data is not available

Scheme	**Underlying Primitive**	**# LUTs**	**# Slices**	**Gbps**	**Mbps/ LUT**	**Mbps/ Slice**
ESTATE_TweAES-128	TBC	1901	602	1.94	1.02	3.22
sESTATE_TweAES-128	TBC	1903	602	2.42	1.27	4.02
ESTATE_TweGIFT-128	TBC (non AES)	681	263	0.84	1.23	3.20
AES-OTR [31]	BC	4263	1204	3.187	0.748	2.647
AES-OCB [29]	BC	4269	1228	3.608	0.845	2.889
AES-COPA [5]	BC	7795	2221	2.770	0.355	1.247
AES-GCM	BC	3478	949	3.837	1.103	4.043
CLOC-AES [26]	BC	3552	1087	3.252	0.478	1.561
CLOC-TWINE [26]	BC (non AES)	1552	439	0.432	0.278	0.984
SILC-AES [26]	BC	3040	910	4.365	1.436	4.796
SILC-LED [26]	BC (non AES)	1682	524	0.267	0.159	0.510
SILC-PRESENT [26]	BC (non AES)	1514	484	0.479	0.316	0.990
ELmD [22]	BC	4490	1306	4.025	0.896	3.082
JAMBU-AES [37]	BC	1595	457	1.824	1.144	3.991
JAMBU-SIMON [37]	BC (non AES)	1200	419	0.368	0.307	0.878
COFB-AES [19,19]	BC	1456	555	2.820	2.220	5.080
SAEB [32]	BC	348	–	–	–	–
AEGIS [38]	BC-RF	7504	1983	94.208	12.554	47.508
DEOXYS [27]	TBC	3234	954	1.472	0.455	2.981
Beetle[Light+] [17,18]	Sponge	608	312	2.095	3.445	6.715
Beetle[Secure+] [17,18]	Sponge	1101	512	2.993	2.718	5.846
ASCON-128 [23]	Sponge	1373	401	3.852	2.806	9.606
Ketje-Jr [11]	Sponge	1567	518	4.080	2.604	7.876
NORX [8]	Sponge	2881	857	10.328	3.585	12.051
PRIMATES-HANUMAN [3]	Sponge	1148	370	1.072	0.934	2.897
ACORN [36]	Stream cipher	499	155	3.437	6.888	22.174
TriviA-ck [13,14,21]	Stream cipher	2221	684	14.852	6.687	21.713

3 Security of ESTATE and sESTATE

It will be easier for us to argue the security of ESTATE by viewing it as an instance of the SIV paradigm [34]. So we digress a little to briefly explain the SIV paradigm. SIV is a design paradigm for constructing DAE or nonce-misuse resistant schemes. It is composed of two stages: a tag generation stage, F that computes the tag T on AD A (nonce N is considered a part of AD) and PT M; and an IV-based encryption stage, iv-enc that computes the ciphertext C on PT M using T as IV. Formally, for key space $\mathcal{L} \times \mathcal{K}$ the encryption algorithm of SIV is defined by the following mapping

$$(L, K, A, M) \mapsto \mathsf{F}(L, A, M) \,\|\, \textsf{iv-enc}\,(K, \mathsf{F}(L, A, M), M)\,,$$

for all $(L, K, A, M) \in \mathcal{L} \times \mathcal{K} \times \mathcal{A} \times \mathcal{M}$. Here, $T := \mathsf{F}(L, A, M) \in \mathcal{T}$, and $C := \textsf{iv-enc}(K, T, M) \in \mathcal{M}$. In general, the iv-enc is defined via a weak PRF G, as illustrated in Figure 5. Here we will focus on SIV with weak PRF based iv-enc. Following the security definitions of [24, 25, 33, 34], we have the following result on the DAE security of an SIV scheme $\mathfrak{A}$:

$$\mathbf{Adv}^{\mathsf{ae}}_{\mathfrak{A}}(q,\ell,\sigma) \leq \mathbf{Adv}^{\mathsf{prf}}_{\mathsf{F}}(q,\ell,\sigma) + \mathbf{Adv}^{\mathsf{wprf}}_{\mathsf{G}}(q,\ell,\sigma). \tag{1}$$

In words, to analyze the security of an SIV mode, it is sufficient to analyze PRF security of the tag generation phase, and weak PRF security of the IV-based encryption phase. Both ESTATE and sESTATE are roughly

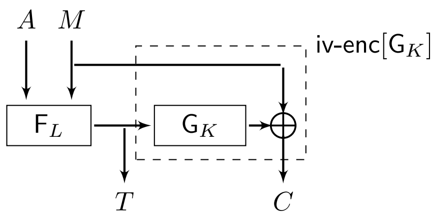

Figure 5: The SIV paradigm based on a PRF F, and a weak PRF G in the IV-based encryption phase. F denotes the tag generation phase, and the dashed rectangle, labeled with iv-enc[G_K] denotes the IV-based encryption phase based on G.

based on the SIV paradigm [34]. It is well known that SIV is a design paradigm for constructing DAE schemes, which is composed of two stages: a tag generation stage that computes the tag T on AD A and message M; and an IV-based encryption stage that computes the ciphertext C on plaintext M using T as IV.

3.1 Security of sESTATE

Coming back to sESTATE, the tag generation phase F is actually a variant of FCBC [12] and can be viewed as a instance of hash-then-PRP, which has been shown to have security in the order of $O(\sigma^2\epsilon)$, where ϵ denotes the universal bound of the hash layer. The iv-enc phase is an instance of the OFB mode with random IV, which is known to have $O(\sigma^2/2^n)$ security from folklore. A formal security proof for OFB is also available in [35]. Note that a better security bound exists for FCBC [28], but the weaker bound suffices for sESTATE, as the bound for iv-enc phase is tight. On substituting the security bounds for F and iv-enc in Eq. (1), we get the following security result on ESTATE.

Proposition 1. *The security of sESTATE is given by,*

$$\mathbf{Adv}^{\mathsf{ae}}_{\mathsf{ESTATE}[\widetilde{\mathsf{E}}]}(t,q,\ell,\sigma) \leq \mathbf{Adv}^{\mathsf{tprp}}_{\widetilde{E}}(t',\sigma) + O\left(\frac{\sigma^2}{2^n} + \sigma^2\epsilon\right), \tag{2}$$

where t, q, ℓ, σ denote the computational time, query bound, maximum query length, and the total number of tweakable block cipher calls across all encryption and decryption queries, respectively. Here, $\sigma^2\epsilon$ denotes the bound due to the PRF security of FCBC, where $\epsilon < 2^{-120}$ for 6-rounds of AES-128.

3.2 Security Analysis of ESTATE

In case of ESTATE the tweak value 0 is utilized for both tag generation and encryption. So we cannot argue the security of ESTATE directly by viewing it as an instance of the SIV paradigm [34]. But, if we can avoid the event that a collision occurs in encryption queries, among the inputs/outputs of those block cipher calls where the tweak value is 0. Since distinct tweak values are used for the first block cipher call in MAC function and the block ciphers used in OFB, and hence the adversary does not have any control over other block cipher inputs of MAC function.

Now, the above bad event occurs with probability at most $\sigma^2/2^n$. This can be easily argued as there are at most $\sigma^2/2$ pairs of inputs/outputs and for each such pair we have at most $2/2^n$ probability (assuming $\sigma < 2^{n-1}$).

Given that this event does not occur, we can simply plug in the privacy and integrity result of SIV mode to obtain the data limit of $\sigma \approx 2^{n/2}$ for ESTATE, much along the same line as sESTATE. In summary we get proposition 2 for the security of ESTATE.

Proposition 2. *The security of ESTATE is given by,*

$$\mathbf{Adv}^{\mathsf{ae}}_{\mathsf{ESTATE}[\widetilde{\mathsf{E}}]}(t,q,\ell,\sigma) \leq \mathbf{Adv}^{\mathsf{tprp}}_{\widetilde{E}}(t',\sigma) + O\left(\frac{\sigma^2}{2^n}\right), \tag{3}$$

where t, q, ℓ, σ denote the computational time, query bound, maximum query length, and the total number of tweakable block cipher calls across all encryption and decryption queries, respectively.

3.2.1 On RUP Security of ESTATE and sESTATE

We give an informal argument on the integrity under RUP (INT-RUP) security [4] of ESTATE and sESTATE. The most important observation is the fact that tweak values for the first block cipher call in tag generation and encryption phases are always distinct. So, the release of internal state information in the encryption phase gives no information regarding any internal state of tag generation phase. As a result for any forgery query the adversary has to still guess the output of a PRF, which is possible with at most $O(1/2^n)$ probability. This clearly gives an INT-RUP bound of the form $O(\sigma^2/2^n + q_d/2^n)$, where $O(\sigma^2/2^n)$ is due to the PRF security of the tag generation phase and $O(q_d/2^n)$ is due to the forgery attempt where q_d denotes the number of forgery attempts.

3.3 Security of the Recommended Instantiations

Here we summarize the security details of ESTATE_TweAES-128, sESTATE_TweAES-128-6, and ESTATE_TweGIFT-128. We provide the concrete data and time limits along with the relevant conditions allowed for the three instantiations. We list the security levels of ESTATE_TweAES-128, ESTATE_TweGIFT-128, and sESTATE_TweAES-128-6 in Table 5. Of note is the fact that we do not restrict the adversary to be nonce-respecting, and same nonce can be used for all the queries, without any degradation of the security. Further we claim integrity security even under the INT-RUP model, where the decryption algorithm releases unverified plaintext. Note that sESTATE_TweAES-128-6 allows a little bit less data and time limits then ESTATE_TweAES-128, due to the use of round-reduced variant of TweAES-128. Finally, we note that all our security claims are based on full round TweAES-128, TweGIFT-128, and the round-reduced TweAES-128-6, and we do not claim/guarantee any security for ESTATE when instantiated with other round-reduced variants of these block ciphers.

Table 5: Summary of security claims for recommended instantiations. The data and time limits indicate the amount of data and time required to make the attack advantage close to 1.

Submissions	**Privacy**		**Integrity**	
	Time	Data (in bytes)	Time	Data (in bytes)
ESTATE_TweAES-128	2^{128}	2^{64}	2^{128}	2^{64}
ESTATE_TweGIFT-128	2^{128}	2^{64}	2^{128}	2^{64}
sESTATE_TweAES-128-6	2^{112}	2^{60}	2^{112}	2^{60}

The time complexity is bounded to $\approx 2^{128}$ given the TPRP assumption on the underlying block ciphers. For sESTATE variants we further restrict the time complexity to account for the use of round-reduced variant of the block cipher.

References

[1] CAESAR Development Package, 2016. https://cryptography.gmu.edu/athena/index.php?id=download.

[2] NIST FIPS 197. Advanced Encryption Standard (AES). *Federal Information Processing Standards Publication*, 197, 2001.

[3] Elena Andreeva, Begül Bilgin, Andrey Bogdanov, Atul Luykx, Florian Mendel, Bart Mennink, Nicky Mouha, Qingju Wang, and Kan Yasuda. PRIMATEs v1.02. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round2/primatesv102.pdf.

[4] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to securely release unverified plaintext in authenticated encryption. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, pages 105–125, 2014.

[5] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Elmar Tischhauser, and Kan Yasuda. AES-COPA v.2. Submission to CAESAR, 2015. https://competitions.cr.yp.to/round2/aescopav2.pdf.

[6] ATHENa: Automated Tool for Hardware Evaluation. https://cryptography.gmu.edu/athena.

[7] Authenticated Encryption FPGA Ranking. https://cryptography.gmu.edu/athenadb/fpga_auth_cipher/rankings_view.

[8] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. NORX v3.0. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/norxv30.pdf.

[9] Subhadeep Banik, Andrey Bogdanov, Thomas Peyrin, Yu Sasaki, Siang Meng Sim1, Elmar Tischhauser, , and Yosuke Todo. SUNDAE-GIFT v1.0, 2019. https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/SUNDAE-GIFT-spec.pdf.

[10] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A small present - towards reaching the limit of lightweight encryption. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 321–345, 2017.

[11] Guido Bertoni, Michaël Peeters Joan Daemen, Gilles Van Assche, and Ronny Van Keer. Ketje v2. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/ketjev2.pdf.

[12] John Black and Phillip Rogaway. CBC macs for arbitrary-length messages: The three-key constructions. *J. Cryptology*, 18(2):111–131, 2005.

[13] Avik Chakraborti, Anupam Chattopadhyay, Muhammad Hassan, and Mridul Nandi. Trivia: A fast and secure authenticated encryption scheme. In *CHES 2015*, pages 330–353, 2015.

[14] Avik Chakraborti, Anupam Chattopadhyay, Muhammad Hassan, and Mridul Nandi. Trivia and utrivia: two fast and secure authenticated encryption schemes. *J. Cryptographic Engineering*, 8(1):29–48, 2018.

[15] Avik Chakraborti, Nilanjan Datta, Ashwin Jha, Cuauhtemoc Mancillas Lopez, Mridul Nandi, and Yu Sasaki. ESTATE. Submission to NIST Lightweight Competition, 2019. https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/estate-spec.pdf.

[16] Avik Chakraborti, Nilanjan Datta, Ashwin Jha, Cuauhtemoc Mancillas-López, Mridul Nandi, and Yu Sasaki. Elastic-tweak: A framework for short tweak tweakable block cipher. *IACR Cryptology ePrint Archive*, 2019:440, 2019.

[17] Avik Chakraborti, Nilanjan Datta, Mridul Nandi, and Kan Yasuda. Beetle family of lightweight and secure authenticated encryption ciphers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):218–241, 2018.

[18] Avik Chakraborti, Nilanjan Datta, Mridul Nandi, and Kan Yasuda. Beetle family of lightweight and secure authenticated encryption ciphers. *IACR Cryptology ePrint Archive*, 2018:805, 2018.

[19] Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, and Mridul Nandi. Blockcipher-based authenticated encryption: How small can we go? In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 277–298, 2017.

[20] Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, and Mridul Nandi. Blockcipher-based authenticated encryption: How small can we go? *IACR Cryptology ePrint Archive*, 2017:649, 2017.

[21] Avik Chakraborti and Mridul Nandi. TriviA-ck-v2. Submission to CAESAR, 2015. https://competitions.cr.yp.to/round2/triviackv2.pdf.

[22] Nilanjan Datta and Mridul Nandi. Proposal of ELmD v2.1. Submission to CAESAR, 2015. https://competitions.cr.yp.to/round2/elmdv21.pdf.

[23] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/asconv12.pdf.

[24] Shay Gueron and Yehuda Lindell. GCM-SIV: full nonce misuse-resistant authenticated encryption at under one cycle per byte. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 109–119, 2015.

[25] Tetsu Iwata and Kazuhiko Minematsu. Stronger security variants of GCM-SIV. *IACR Cryptology ePrint Archive*, 2016:853, 2016.

[26] Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, Sumio Morioka, and Eita Kobayashi. CLOC and SILC. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/clocsilcv3.pdf.

[27] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. Deoxys v1.41. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/deoxysv141.pdf.

[28] Ashwin Jha and Mridul Nandi. Revisiting structure graph and its applications to CBC-MAC and EMAC. *IACR Cryptology ePrint Archive*, 2016:161, 2016.

[29] Ted Krovetz and Phillip Rogaway. OCB(v1.1). Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/ocbv11.pdf.

[30] Kerry A. McKay, Larry Bassham, Meltem Sönmez Turan, and Nicky Mouha. Lightweight Cryptography: Round 1 candidates, 2017. https://csrc.nist.gov/Projects/Lightweight-Cryptography/Round-1-Candidates.

[31] Kazuhiko Minematsu. AES-OTR v3.1. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/aesotrv31.pdf.

[32] Yusuke Naito, Mitsuru Matsui, Takeshi Sugawara, and Daisuke Suzuki. SAEB: A lightweight blockcipher-based AEAD mode of operation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):192–217, 2018.

[33] Chanathip Namprempre, Phillip Rogaway, and Thomas Shrimpton. Reconsidering generic composition. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 257–274, 2014.

[34] Phillip Rogaway and Thomas Shrimpton. A Provable-Security Treatment of the Key-Wrap Problem. In *EUROCRYPT*, pages 373–390, 2006.

[35] Mark Wooding. New proofs for old modes. *IACR Cryptology ePrint Archive*, 2008:121, 2008.

[36] Hongjun Wu. ACORN: A Lightweight Authenticated Cipher (v3). Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/acornv3.pdf.

[37] Hongjun Wu and Tao Huang. The JAMBU Lightweight Authentication Encryption Mode (v2.1). Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/jambuv21.pdf.

[38] Hongjun Wu and Bart Preneel. AEGIS : A Fast Authenticated Encryption Algorithm (v1.1). Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/aegisv11.pdf.