ESTATE

Designers/Submitters:

Avik Chakraborti - NTT Secure Platform Laboratories, Japan
Nilanjan Datta - Indian Statistical Institute, Kolkata, India
Ashwin Jha - Indian Statistical Institute, Kolkata, India
Cuauhtemoc Mancillas Lopez - Computer Science Department, CINVESTAV-IPN, Mexico
Mridul Nandi - Indian Statistical Institute, Kolkata, India
Yu Sasaki - NTT Secure Platform Laboratories, Japan

avikchkrbrti@gmail.com, nilanjan_isi_jrf@yahoo.com, ashwin.jha1991@gmail.com, cuauhtemoc.mancillas83@gmail.com, mridul.nandi@gmail.com, sasaki.yu@lab.ntt.co.jp

March 29, 2019

Chapter 1

Introduction

Energy efficient and Single-state Tweakable block cipher based MAC-Then-Encrypt, or ESTATE in abbreviation, is a tweakable block cipher based authenticated encryption scheme that employs FCBC [7] like authentication followed by OFB [1] encryption. At a very high level, it can be viewed as the tweakable block cipher based variant of SUNDAE [4]. ESTATE is a single-state, inverse-free, RUP secure construction that does not require any field multiplications.

To instantiate ESTATE, we propose two dedicated tweakable block ciphers TweGIFT-128 and TweAES-128. As the names suggest, TweGIFT-128 is a tweakable variant of the GIFT-128 [5] block cipher, where as TweAES-128 is a tweakable variant of the AES-128 [2] block cipher. Both these tweakable block ciphers are explicitly designed for efficient processing of small tweaks of size 4 bits.

1.1 Notations

For $n \in \mathbb{N}$, we write $\{0,1\}^+$ and $\{0,1\}^n$ to denote the set of all non-empty binary[1] strings, and the set of all n-bit binary strings, respectively. We write λ to denote the empty string, and $\{0,1\}^* = \{0,1\}^+ \cup \{\lambda\}$. For $A \in \{0,1\}^*$, $|A|$ denotes the length (number of the bits) of A, where $|\lambda| = 0$ by convention. For all practical purposes, we use the little-endian format for representing binary strings, i.e. the least significant bit is the right most bit. For any non-empty binary string X, $(X_{k-1}, \ldots, X_0) \overset{n}{\leftarrow} x$ denotes the n-bit block parsing of X, where $|X_i| = n$ for $0 \leq i \leq k-2$, and $1 \leq |X_{k-1}| \leq n$. For $A, B \in \{0,1\}^*$ and $|A| = |B|$, we write $A \oplus B$ to denote the bitwise XOR of A and B. For $A, B \in \{0,1\}^*$, $A\|B$ denotes the concatenation of A and B. Note that A and B denote the most and least significant parts, respectively.

For $n, \tau, \kappa \in \mathbb{N}$, $\widetilde{\mathsf{E}}$-$n/\tau/\kappa$ denotes a tweakable block cipher family $\widetilde{\mathsf{E}}$, parametrized by the block length n, tweak length τ, and key length κ. For $K \in \{0,1\}^\kappa$, $T \in \{0,1\}^\tau$, and $M \in \{0,1\}^n$, we use $\widetilde{\mathsf{E}}_K^T(M) := \widetilde{\mathsf{E}}(K, T, M)$ to denote invocation of the encryption function of $\widetilde{\mathsf{E}}$ on input K, T, and M. We fix positive even integers n, τ, κ, and t to denote the *block size*, *tweak size*, *key size*, and *tag size*, respectively, in bits. Throughout this document, we fix $n = 128$, $\tau = 4$, and $\kappa = 128$, and $t = n$.

We sometime use the terms (*complete/full*) *blocks* for n-bit strings, and *partial blocks* for m-bit strings, where $m < n$. Throughout, we use the function ozs, defined by the mapping

$$\forall X \in \bigcup_{m=1}^{n} \{0,1\}^m, \quad X \mapsto \begin{cases} 0^{n-|X|-1}\|1\|X & \text{if } |X| < n, \\ X & \text{otherwise,} \end{cases}$$

as the padding rule to map partial blocks to complete blocks. Note that the mapping is injective over partial blocks. For any $X \in \{0,1\}^+$ and $0 \leq i \leq |X|-1$, x_i denotes the i-th bit of X. The function chop takes a string X and an integer $i \leq |X|$, and returns the least significant i bits of X, i.e. $x_{i-1} \cdots x_0$. We use the notations $X \lll i$ and $X \ggg i$ to denote i bit left and right, respectively, rotations of the bit string X.

[1] Alphabet set is $\{0,1\}$.

For some predicates $\mathtt{E}_1$ and $\mathtt{E}_2$, and possible evaluations a, b, c, d, we define the conditional operator ? ::: as follows:

$$(\mathtt{E}_1; \mathtt{E}_2) \ ? \ a : b : c : d := \begin{cases} a & \text{if } \mathtt{E}_1 \wedge \mathtt{E}_2 \\ b & \text{if } \mathtt{E}_1 \wedge \neg \mathtt{E}_2 \\ c & \text{if } \neg \mathtt{E}_1 \wedge \mathtt{E}_2 \\ d & \text{if } \neg \mathtt{E}_1 \wedge \neg \mathtt{E}_2 \end{cases}$$

The expression "$\mathtt{E}$? a : d" is the special case when $\mathtt{E}_1 \equiv \mathtt{E}_2$, i.e. it evaluates to a if $\mathtt{E}$ holds and b otherwise.

Chapter 2

Specification

In this chapter, we present the specification of ESTATE mode of operation along with the block ciphers, TweAES-128 and TweGIFT-128, used for instantiating ESTATE. 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.

2.1 ESTATE AEAD Mode

ESTATE authenticated encryption mode receives an encryption key $K \in \{0,1\}^{\kappa}$, a nonce $N \in \{0,1\}^{n}$, an associated data $A \in \{0,1\}^{*}$, and a message $M \in \{0,1\}^{*}$ as inputs, and returns a ciphertext $C \in \{0,1\}^{|M|}$, and a tag $T \in \{0,1\}^{n}$. The decryption algorithm receives a key $K \in \{0,1\}^{\kappa}$, a nonce $N \in \{0,1\}^{n}$, an associated data $A \in \{0,1\}^{*}$, a ciphertext $C \in \{0,1\}^{*}$, and a tag $T \in \{0,1\}^{n}$ as inputs, and return the plaintext $M \in \{0,1\}^{|C|}$ corresponding to C, if the tag T is valid.

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 2.1, 2.2, and 2.3.

2.1.1 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 (see section 2.3). 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.

2.1.2 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.

2.2 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

Algorithm 1 ESTATE Authenticated Encryption and Verified Decryption Algorithm.

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

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

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

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

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

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, and the corresponding pictorial description is given in Figure 2.4, 2.5, and 2.6.

2.3 TweAES-128 and TweGIFT-128 Block Ciphers

2.3.1 Specification of TweAES-128

In this section we provide the specification of the tweakable block cipher TweAES-128 and TweAES-128-6. TweAES-128 is a 128-bit tweakable block cipher with 4-bit tweak and 128-bit key. 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. The complete specification of TweAES-128 is described in Algorithm 3. Now we briefly describe the main steps of the TweAES-128 round function.

SubBytes: TweAES-128 uses the same invertible 8-bit S-box as AES-128 and applies it to each byte of the cipher state. Description of this S-box AS is given in Table 2.1.

ShiftRows: The bytes in the i-th row is cyclically shifted by i places to the left.

MixColumns: The state is multiplied by an invertible MDS matrix to achieve good diffusion. The matrix M is defined as:

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

over the field $\mathbb{F}_8$ where field multiplication is done with respect to the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$.

AddRoundKey: A 128-bit round key is extracted from the master key and XORed to the cipher state.

AddTweak: The 4-bit tweak is first expanded to an 8-bit value using a linear code and then the 8-bit value is XORed to the state at an interval of 2 rounds.

Note that, all the operations, except AddTweak, are identical to that of AES-128-128/128.

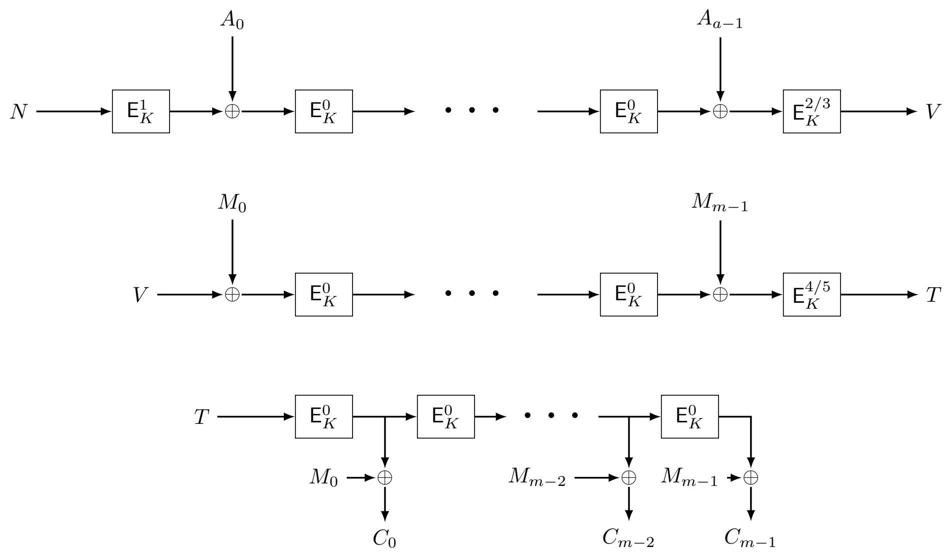

Figure 2.1: ESTATE with a AD blocks and m message blocks.

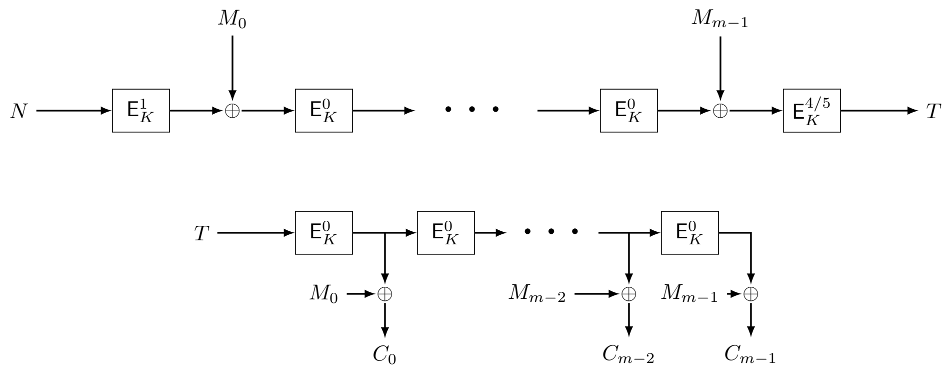

Figure 2.2: ESTATE with empty AD and m message blocks.

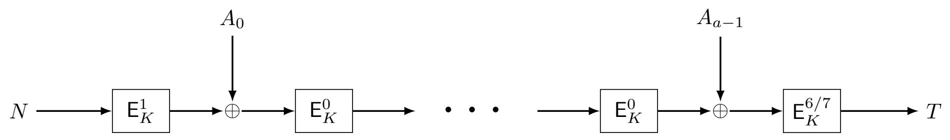

Figure 2.3: ESTATE with a AD blocks and empty message.

2.3.2 Specification of TweAES-128-6

We also define a round-reduced version of TweAES-128, called TweAES-128-6, 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. For the sake of completeness, we provide the formal specification of TweAES-128-6 in Algorithm 3.

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

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

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

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

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

2.3.3 Specification of TweGIFT-128

TweGIFT-128 is a 128-bit tweakable block cipher with 4-bit tweak and 128-bit key. As the name suggests, it is a tweakable variant of GIFT-128 [5] block cipher. TweGIFT-128 is composed of 40 rounds and each round is composed of the following operations:

SubCells: TweGIFT-128 uses the same invertible 4-bit S-box as GIFT and applies it to each nibble of the cipher state. Description of this S-box GS is given in Table 2.3.

PermBits: TweGIFT-128 also uses the same bit permutation that is used in GIFT. The permutation maps bit position i of the cipher state to bit position $\mathsf{P}(i)$, where

$$\mathsf{P}(i) = 4\lfloor i/16 \rfloor + 32\Big(\big(3\lfloor (i \mod 16)/4 \rfloor + (i \mod 4)\big) \mod 4\Big) + (i \mod 4).$$

AddRoundKey: In this step, a 64 bit round key is extracted from the master key state and added to the cipher state. This operation is also identical to that of GIFT.

AddRoundConstant: A single bit "1" and a 6 bit round constant are XORed into the cipher state at bit position 127, 23, 19, 15, 11, 7 and 3 respectively. The round constants are generated using the same 6 bit affine LFSR as SKINNY [6] and GIFT.

AddTweak: The 4-bit tweak is first expanded to a 32-bit value using a linear code and then the 32-bit value is XORed to the state at an interval of 5 rounds.

Complete specification of TweGIFT-128 is presented in Algorithm. 4.

2.4 Recommended Instantiations

We recommend the following concrete instantiations:

- ESTATE_TweAES-128: This AEAD scheme obtained by instantiating ESTATE mode of operation with TweAES-128 block cipher. Here the size of the key, nonce and tag are 128 bits each. **We recommend ESTATE_TweAES-128, as the primary version among our submissions.**

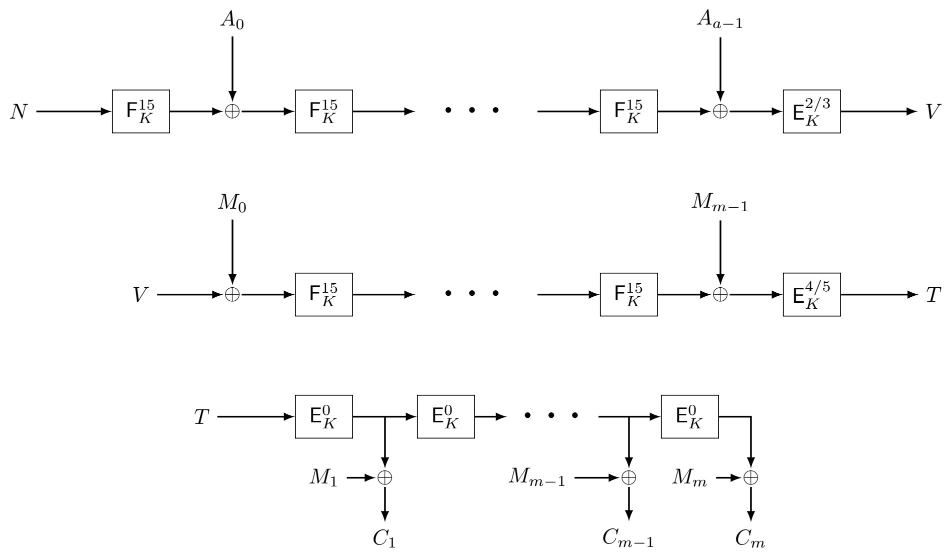

Figure 2.4: sESTATE with a AD blocks and m message blocks.

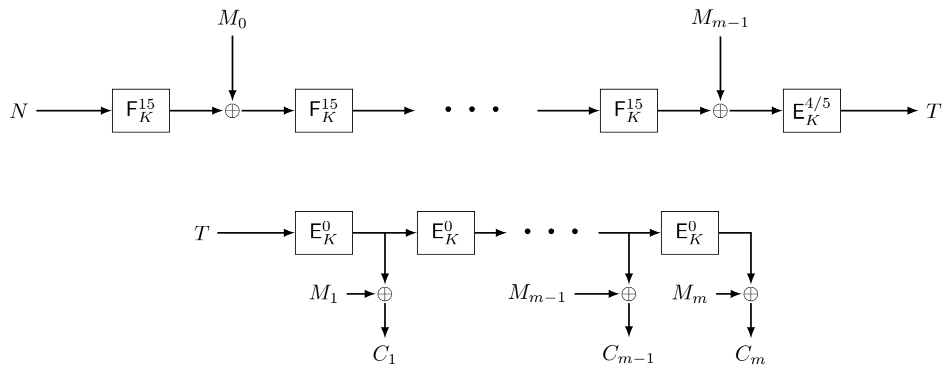

Figure 2.5: sESTATE with empty AD and m message blocks.

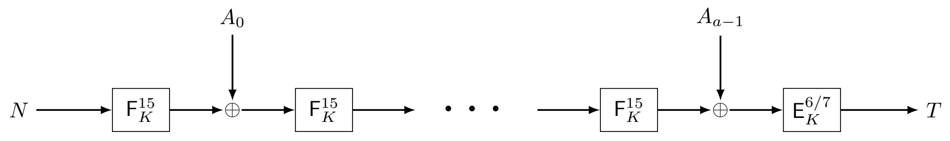

Figure 2.6: sESTATE with a AD blocks and empty message.

- ESTATE_TweGIFT-128: This AEAD scheme is obtained by instantiating ESTATE mode of operation with TweAES-128 block cipher. Here the size of the key, nonce and tag are 128 bits each. We recommend ESTATE_TweGIFT-128, for hardware-oriented ultra-lightweight applications.
- sESTATE_TweAES-128-6: This AEAD scheme is obtained by instantiating ESTATE mode of operation with TweAES-128 block cipher. Here also, the size of the key, nonce and tag are 128 bits

	00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**0A**	**0B**	**0C**	**0D**	**0E**	**0F**
00	63	7C	77	7B	F2	6B	6F	35	30	01	67	2B	FE	D7	AB	76
10	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
20	B7	FD	97	26	36	3F	F7	CC	34	A5	EF	F1	71	D8	31	15
30	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
40	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
50	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
60	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
70	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
80	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
90	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A0	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B0	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C0	BA	78	25	2e	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D0	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E0	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F0	89	A1	89	0D	BF	E6	42	68	42	99	2D	0F	B0	54	BB	16

Table 2.1: The AES-128 S-Box AS. All entries are in hexadecimal format. Row headings denote the least significant nibbles of input and column headings denote the most significant nibbles of input.

i	1	2	3	4	5	6	7	8	9	10
RCON(i)	01	02	04	08	10	20	40	80	1b	36

Table 2.2: The AES-128 RCON values in hexadecimal format.

each. We recommend sESTATE_TweAES-128-6, for higher throughput demanding, and energy-constrained applications.

i	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
GS(i)	1	A	4	C	6	F	3	9	2	D	B	7	5	0	8	E

Table 2.3: The GIFT-128 S-Box GS.

Algorithm 3 Algorithmic description of TweAES-128.

```
 1: function TweAES($K, T, M$)
 2:     $(W_{43}, \dots, W_0) \leftarrow \mathsf{KeyGen}(K)$
 3:     $X \leftarrow X \oplus (W_3, W_2, W_1, W_0)$
 4:     for $i = 1$ to 9 do
 5:         $X \leftarrow \mathsf{SubBytes}(X)$
 6:         $X \leftarrow \mathsf{ShiftRows}(X)$
 7:         $X \leftarrow \mathsf{MixColumns}(X)$
 8:         $X \leftarrow X \oplus (W_{4i+3}, W_{4i+2}, W_{4i+1}, W_{4i})$
 9:         if $i\%2 = 0$ then
10:             $X \leftarrow \mathsf{AddTweak}(X, T)$
11:     $X \leftarrow \mathsf{SubBytes}(X)$
12:     $X \leftarrow \mathsf{ShiftRows}(X)$
13:     $X \leftarrow X \oplus (W_{43}, W_{42}, W_{41}, W_{40})$
14:     return $X$

15: function SubBytes($X$)
16:     $(X_{15}, \dots, X_0) \overset{8}{\leftarrow} X$
17:     for $i = 0$ to 15 do
18:         $X_i \leftarrow \mathsf{AS}(X_i)$
19:     return $X$

20: function SubWords($X$)
21:     $(X_3, \dots, X_0) \overset{8}{\leftarrow} X$
22:     for $i = 0$ to 3 do
23:         $X_i \leftarrow \mathsf{AS}(X_i)$
24:     return $X$

25: function ShiftRows($X$)
26:     $(X_{15}, \dots, X_0) \overset{8}{\leftarrow} X$
27:     for $i = 0$ to 3 do
28:         for $j = 0$ to 3 do
29:             $Y_{4i+j} \leftarrow X_{4i+((j+i)\%4)}$
30:     return $Y$

31: function MixColumns($X$)
32:     $M \leftarrow \begin{pmatrix} 2 & 3 & 1 & 1 \\ 3 & 1 & 1 & 2 \\ 1 & 1 & 2 & 3 \\ 1 & 2 & 3 & 1 \end{pmatrix}$
33:     $Y \leftarrow M \cdot X$
34:     return $Y$
```

```
 1: function TweAES-6($K, T, X$)
 2:     $(W_{43}, \dots, W_0) \leftarrow \mathsf{KeyGen}(K, X)$
 3:     $X \leftarrow X \oplus (W_3, W_2, W_1, W_0)$
 4:     for $i = 1$ to 6 do
 5:         $X \leftarrow \mathsf{SubBytes}(X)$
 6:         $X \leftarrow \mathsf{ShiftRows}(X)$
 7:         $X \leftarrow \mathsf{MixColumns}(X)$
 8:         $X \leftarrow X \oplus (W_{4i+3}, W_{4i+2}, W_{4i+1}, W_{4i})$
 9:         if $i\%2 = 0$ and $i < 6$ then
10:             $X \leftarrow \mathsf{AddTweak}(X, T)$
11:     return $X$

12: function AddTweak($X, T$)
13:     $(X_{127}, \dots, X_0) \overset{1}{\leftarrow} X$
14:     $(T_3, \dots, T_0) \overset{1}{\leftarrow} T$
15:     $T_\oplus \leftarrow T_0 \oplus T_1 \oplus T_2 \oplus T_3$
16:     for $i = 0$ to 3 do
17:         $T_{i+4} \leftarrow T_i \oplus T_\oplus$
18:     for $i = 0$ to 7 do
19:         $X_{8i} \leftarrow X_{8i} \oplus T_i$
20:     return $X$

21: function KeyGen($K$)
22:     $(K_{15}, \dots, K_0) \overset{8}{\leftarrow} K$
23:     for $i = 0$ to 3 do
24:         $W_i \leftarrow (K_{4i+3}, K_{4i+2}, K_{4i+1}, K_{4i})$
25:     for $i = 4$ to 43 do
26:         $Y \leftarrow W_{i-1}$
27:         if $i\%4 = 0$ then
28:             $Y \leftarrow \mathsf{SubWords}(\mathsf{RotWords}(Y))$
29:             $Y \leftarrow Y \oplus \mathsf{RCON}(i/4)$
30:         $W_i \leftarrow W_{i-4} \oplus Y$
31:     return $(W_{43}, \dots, W_0)$

32: function RotWords($X$)
33:     $(X_3, \dots, X_0) \overset{8}{\leftarrow} X$
34:     $W \leftarrow (X_0, X_3, X_2, X_1)$
35:     return $W$
```

Algorithm 4 Algorithmic description of TweGIFT-128.

```
1: function TweGIFT($(K, T, X)$)
2:     $C \leftarrow 000000$
3:     for $i = 0$ to $39$ do
4:         $X \leftarrow \mathsf{SubCells}(X)$
5:         $X \leftarrow \mathsf{PermBits}(X)$
6:         $(K, X) \leftarrow \mathsf{AddRoundKey}(K, X)$
7:         $(C, X) \leftarrow \mathsf{AddRoundConstant}(C, X)$
8:         if $(i + 1)\%5 = 0$ and $i < 39$ then
9:             $X \leftarrow \mathsf{AddTweak}(X, T)$
10:    return $X$

11: function SubCells($X$)
12:     $(X_{31}, \ldots, X_0) \overset{4}{\leftarrow} X$
13:     for $i = 0$ to $31$ do
14:         $X_i \leftarrow \mathsf{GS}(X_i)$
15:     return $X$

16: function AddRoundKey($(K, X)$)
17:     $(K_7, \ldots, K_0) \overset{16}{\leftarrow} K$
18:     $(X_{127}, \ldots, X_0) \overset{1}{\leftarrow} X$
19:     $U_{31..0} \leftarrow (K_5, K_4)$
20:     $V_{31..0} \leftarrow (K_1, K_0)$
21:     for $i = 0$ to $31$ do
22:         $X_{4i+2} \leftarrow X_{4i+2} \oplus U_i$
23:         $X_{4i+1} \leftarrow X_{4i+1} \oplus V_i$
24:     $(K_7, \cdots, K_0) \leftarrow (K_1 \ggg 2, K_0 \ggg 12, K_7, \cdots, K_2)$
25:     return $(K, X)$
```

```
1: function AddTweak($(X, T)$)
2:     $(X_{127}, \ldots, X_0) \overset{1}{\leftarrow} X$
3:     $(T_3, \ldots, T_0) \overset{1}{\leftarrow} T$
4:     $T_\oplus \leftarrow T_0 \oplus T_1 \oplus T_2 \oplus T_3$
5:     for $i = 0$ to $3$ do
6:         $T_{i+4} \leftarrow T_i \oplus T_\oplus$
7:     $T_{15..8} \leftarrow T_{7..0}$
8:     $T_{23..16} \leftarrow T_{7..0}$
9:     $T_{31..24} \leftarrow T_{7..0}$
10:    for $i = 0$ to $31$ do
11:        $X_{4i} \leftarrow X_{4i} \oplus T_i$
12:    return $X$

13: function PermBits($X$)
14:     $(X_{127}, \ldots, X_0) \overset{1}{\leftarrow} X$
15:     for $i = 0$ to $127$ do
16:         $X_{\mathsf{P}(i)} \leftarrow X_i$
17:     return $X$

18: function AddRoundConstant($(C, X)$)
19:     $(C_5, \ldots, C_0) \overset{1}{\leftarrow} C$
20:     $(X_{127}, \ldots, X_0) \overset{1}{\leftarrow} X$
21:     $X_{127} \leftarrow X_{127} \oplus 1$
22:     for $i = 0$ to $5$ do
23:         $X_{4i+3} \leftarrow X_{4i+3} \oplus C_i$
24:     $(C_5, \ldots, C_0) \leftarrow (C_4, \ldots, C_0, C_5 \oplus C_4 \oplus 1)$
25:     return $(C, X)$
```

Chapter 3

Security

In this chapter, we summarize the security details of ESTATE_TweAES-128, sESTATE_TweAES-128-6, and ESTATE_TweGIFT-128. Section 3.1, gives the concrete data and time limits along with the relevant conditions allowed for the three instantiations. Section 3.2 presents analysis against generic attacks (assuming the underlying block cipher is a tweakable random permutation), i.e. the security of modes. Section 3.3 presents a brief analysis on the security of TweAES-128, TweAES-128-6, and TweGIFT-128, showing that they display close to ideal behavior under the given data and time limit.

3.1 Security Claims

Table 3.1: Summary of security claims for ESTATE_TweAES-128, ESTATE_TweGIFT-128, and sESTATE_TweAES-128-6. 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}

We list the security levels of ESTATE_TweAES-128, ESTATE_TweGIFT-128, and sESTATE_TweAES-128-6 in Table 3.1. 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. All our submissions are equally secure in nonce-misusing scenario. Further we claim both privacy and integrity security under a stronger model (see section 3.2), where the decryption algorithm releases unverified plaintext (RUP model of [3]). 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 (see section 3.3.2). Finally, we note that all our security claims are based on full round TweAES-128, TweGIFT-128, and the round-reduced TweAES-128-6, as described in section 2.3, and we do not claim/guarantee any security for ESTATE when instantiated with other round-reduced variants of these block ciphers.

3.1.1 Statement

We declare that there are no hidden weaknesses in ESTATE and sESTATE modes of operation. Further, to the best of our knowledge, public third-party analysis do not raise any security threat to the submissions, ESTATE_TweAES-128, ESTATE_TweGIFT-128, and sESTATE_TweAES-128-6, within the data and time limits prescribed in Table 3.1.

3.2 Security of ESTATE and sESTATE

ESTATE and sESTATE are identical, except for one change, sESTATE uses a round-reduced version of TweAES-128 in the tag generation phase. So from the point of view of generic attacks, similar attack strategies may apply to both ESTATE, and sESTATE. In the following discussion:

- D denotes the data complexity of the attack. This parameter quantifies the online resource requirements, and includes the total number of blocks (among all messages and associated data) processed through the underlying block cipher for a fixed master key. Note that for simplicity we also use D to denote the data complexity of forging attempts.

- T denotes the time complexity of the attack. This parameter quantifies the offline resource requirements, and includes the total time required to process the offline evaluations of the underlying block cipher. Since one call of the block cipher can be assumed to take a constant amount of time, we generally take T as the total number of offline calls to the block cipher.

Both ESTATE and sESTATE are roughly based on the SIV paradigm [9]. 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.

3.2.1 Security Analysis of sESTATE

In sESTATE, the tweak values are distinct in the tag and encryption phase. So it can be viewed as an instance of the SIV paradigm [9]. SIV requires PRF security from the tag generation and weak PRF from the encryption phase. The tag generation phase MAC, can be viewed as a instance of hash-then-PRP, which has been shown to have PRF security up to $D \approx \sqrt{\epsilon^{-1}}$ [10], where ϵ denotes the universal bound of the hash layer. From section 3.3.2, we know that TweAES-128-6 has maximum differential probability of 2^{-120}, which is also the upper bound on ϵ. The encryption phase is an instance of the OFB mode with random IV, which is known to have security up to $D \approx 2^{n/2}$. A formal security proof for OFB is also available in [11]. To summarize, sESTATE is secure upto the data limit of $D \approx 2^{60}$. Since we use the standard model PRP notion on the underlying block cipher, the time limit $T \approx 2^{\kappa}$.

3.2.2 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 [9]. 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. Thus, the above bad event occurs with probability at most $D^2/2^n$. Given that this bad event does not occur, we can now simply plug in the privacy and integrity result of SIV mode to obtain the data limit of $D \approx 2^{n/2}$ for ESTATE, much along the same line as sESTATE. Again we can directly use the PRP assumption on the underlying block cipher to get $T \approx 2^{\kappa}$.

3.2.3 On Security in RUP Model

Note that the tweak values for the first block cipher call in tag generation and encryption phases, are always distinct. So, even if the adversary gets some unverified plaintext, it cannot get any information regarding the tag generation phase. So, both ESTATE and sESTATE maintain their security in RUP model as well. Specifically, they achieve the so-called PA1 and INT-RUP security notions of [3].

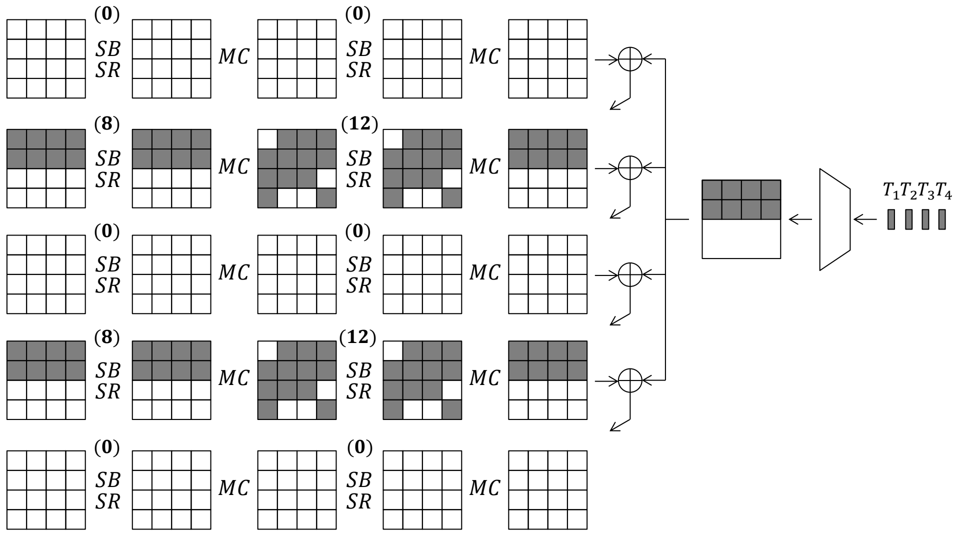

Figure 3.1: An Examples of Differential Trails with 40 Active S-boxes.

3.2.4 Validation of Security Claims

The security levels in Table 3.1 can be validated by simply substituting $n = 64$ and $\kappa = 128$, in case of ESTATE_TweAES-128 and ESTATE_TweGIFT-128. For sESTATE_TweAES-128-6 we give a conservative data and time limit in order to account for the round-reduced nature of TweAES-128-6.

3.3 Security of TweAES-128, TweAES-128-6 and TweGIFT-128

3.3.1 Security Analysis of TweAES-128

Number of Active S-boxes

In TweAES-128, the expanded tweak is injected in every 2 rounds. We evaluate the minimum number of differentially and linearly active S-boxes for the 4-round core.

When the tweak input has a non-zero difference, the expanding function ensures that at least 4 bytes are affected by the tweak difference. We modeled the problem of finding the differential trail with minimum number of active S-boxes by MILP and experimentally verified that the minimum number of active S-boxes is 40 for the entire construction. This is a tight bound. An example of the differential trails achieving 40 active S-boxes is given in Fig. 3.1.

Given that the maximum differential probability of the AES-128 S-box is 2^{-6}, the probability of the differential propagation is upper bounded by $2^{-6\times 40} = 2^{-240}$.

Reduced-Round Versions Starting from Middle Rounds

Security of full TweAES-128 is based on the fact that the heavy weight of the expanded tweak is propagated through 2 rounds in forward and backward. We call those operations "4 rounds core." We argue that the reduced-round versions of TweAES-128 in which the first or the last round is located in the middle of the 4-round core can be attacked for relatively long rounds. Owing to this unusual setting, the attacks here do not threaten the security of full TweAES-128, however we still demonstrate the attacks for better understanding of the security of TweAES-128.

7-Round Boomerang/Sandwich Attacks. The first approach is the boomerang attack or more precisely formulated version called the sandwich attack. The boomerang attack divides the cipher E into two parts E_0 and E_1 such that $E = E_1 \circ E_0$, and builds high-probability differentials for E_0 and E_1 almost independently. The attack detects a quartet of plaintext x that satisfy the non-ideal behavior shown below with probability $p^{-2}q^{-2}$, where p and q are the differential probability for $E_0 : \alpha \rightarrow \beta$ and $E_1 : \gamma \rightarrow \delta$, respectively.

$$\mathsf{Pr}\big[E^{-1}\big(E(x) \oplus \delta\big) \oplus E^{-1}\big(E(x \oplus \alpha) \oplus \delta\big) = \alpha\big] = p^{-2}q^{-2}.$$

7-rounds of TweAES-128 including four tweak injections that starts from the tweak injection are divided into E_0 and E_1 as follows.

$$E_0 := tweak - 1\mathsf{RAES\text{-}128} - 1\mathsf{RAES\text{-}128} - tweak - 1\mathsf{RAES\text{-}128},$$
$$E_1 := 1\mathsf{RAES\text{-}128} - tweak - 1\mathsf{RAES\text{-}128} - 1\mathsf{RAES\text{-}128} - tweak - 1\mathsf{RAES\text{-}128}.$$

With this configuration, the attacker can avoid building the trail over the 4-round core for both of E_0 and E_1.

The framework of the sandwich attacks show that by dividing the cipher E into three parts $E = E_1 \circ E_m \circ E_0$, the probability of the above event is calculated as $p^{-2}q^{-2}r_{qua}$, where r_{qua} is the probability for a quartet defined as

$$r_{qua} := \mathsf{Pr}\big[E_m^{-1}\big(E_m(x) \oplus \gamma\big) \oplus E_m^{-1}\big(E_m(x \oplus \beta) \oplus \gamma\big) = \beta\big].$$

We define E_m of this attack as the first S-box layer in the above E_1. The configuration and the differential trails are depicted in Fig. 3.2 The probability when E_m is a single S-box layer can be measured by using the boomerang connectivity table (BCT). The trails for E_0 and E_1 include 4 active S-boxes,

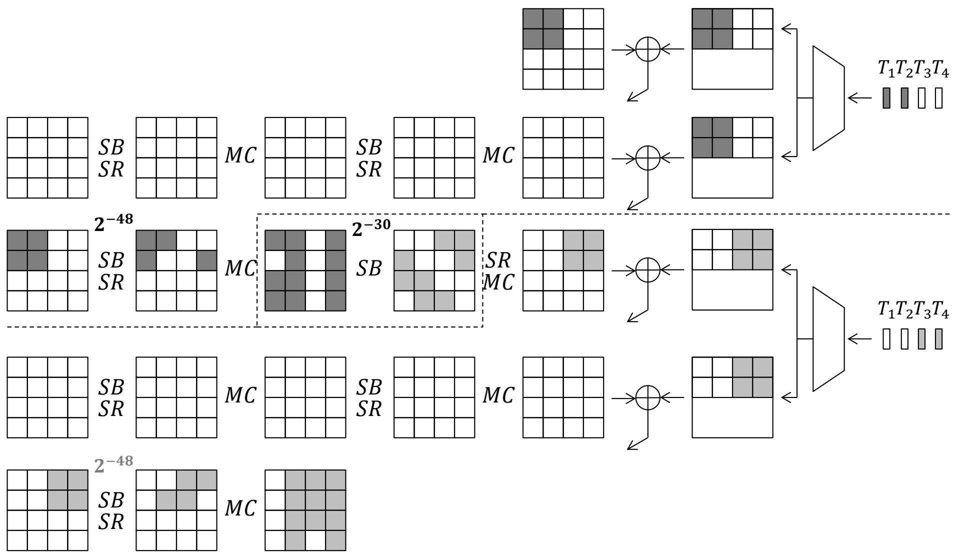

Figure 3.2: Differential Trails for Boomerang Attacks. The cells filled with black and gray represent active byte positions in E_0 and E_1, respectively.

hence both of the probability p and q are 2^{-24}. That is, $p^2q^2 = 2^{-96}$. The BCT of the AES S-box shows that the probability for each S-box in E_m is either $2^{-5.4}$, 2^{-6}, or 2^{-7} if both of the input and output differences are non-zero, and is 1 otherwise. Hence, the trail contains 5 active S-boxes with some probabilistic propagation and we assume that the probability of each S-box is 2^{-6}. Then, the probability r_{qar} is $2^{-6\times 5} = 2^{-30}$. In the end, $p^{-2}q^{-2}r_{qua} = 2^{-126}$, which would lead to a valid distinguisher for 7 rounds.

8-Round Impossible Differential Attacks against TweAES-128. Due to 2 interval rounds between tweaks, distinguishers based on impossible differential attacks can be constructed for relatively long rounds (6 rounds) by canceling the tweak difference with the state difference. The distinguisher is depicted in Fig. 3.3.

The first and last tweak differences are canceled with the state difference with probability 1. Then we have 2 blank rounds. After that, the tweak difference is injected to the state, which implies that the tweak difference must be propagated to the same tweak difference after 2 AES rounds. However, this transformation is impossible because

- 1-round propagation in forwards have 4 active bytes for the right-most column, while
- 1-round propagation in backwards have at least 2 inactive bytes in the right-most column.

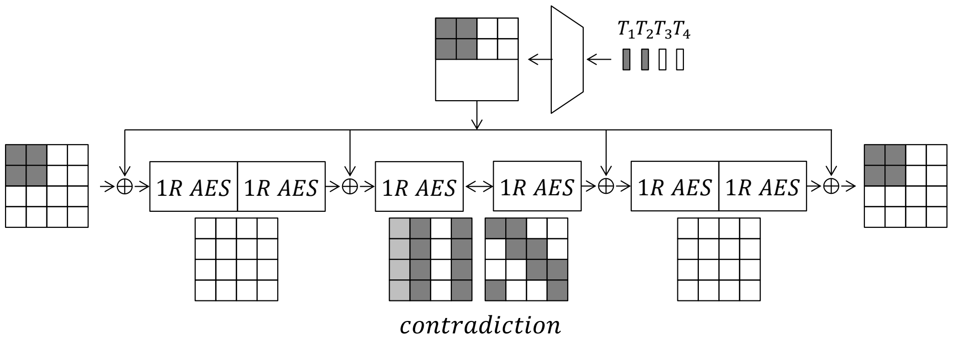

Figure 3.3: 6-round Impossible Differential Distinguisher. The bytes filled with black, white, and gray have non-zero difference, zero difference, and arbitrary difference, respectively.

For the key recovery, two rounds can be appended to the 6-round distinguisher; one is at the beginning and the other is at the end, which is illustrated in Fig. 3.4. As shown in Fig. 3.4 the trail includes 8 and

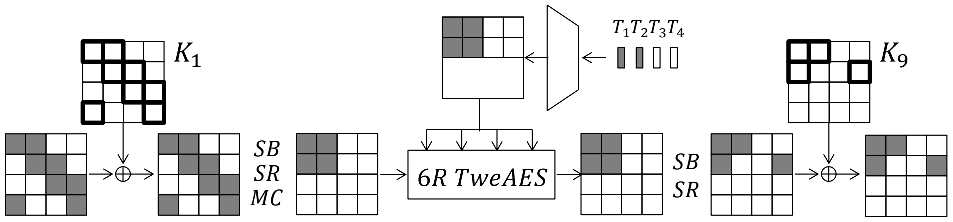

Figure 3.4: Extension to 8-round Key Recovery

4 active bytes at the input and output states. Partial computations to the middle 6-round distinguisher involve 8 bytes of subkey K_1 and 4 bytes of subkey K_9.

Recall that the tweak size is 4 bits. The attack procedure is as follows.

1. Choose all tweak values denoted by T^i where $i = 0, 1, \ldots, 2^4 - 1$.
2. For each of T^i, fix the value of inactive 8 bytes at the input, choose all 8-byte values at the active byte positions of the input state. Query those 2^{64} values to get the corresponding outputs. Those outputs are stored in the list L^i where $i = 0, 1, \ldots, 2^4 - 1$.
3. For all $\binom{2^4}{2} \approx 2^7$ pairs of L^i and L^j with $i \neq j$, find the pairs that do not have difference in 12 inactive bytes of the output state. About $2^{7+64+64-96} = 2^{39}$ pairs will be obtained.
4. For each of the obtained pairs, the tweak difference is fixed and the differences at the input and output states are also fixed. Those fix both of input and output differences of each S-box in the first round and the last round. Hence, each pair suggests a wrong key.
5. Repeat the procedure 2^{54} times from the first step by changing the inactive byte values at the input. After this step, $2^{39+54} = 2^{103}$ wrong-key candidates (including overlaps) will be obtained. The remaining key space of the involved 12 bytes becomes $2^{96} \times (1-2^{-96})^{2^{103}} \approx 2^{96} \times e^{-128} \approx 2^{-88} < 1$. Hence, the 8 bytes of K_1 and 4 bytes of K_9 will be recovered.
6. Exhaustively search the remaining 8 bytes of K_1.

The data complexity is $2^4 \times 2^{64} \times 2^{53} = 2^{121}$. The time complexity is also 2^{121} memory accesses. The memory complexity is to recored the wrong keys of the 12 bytes, which is 2^{96}.

Summary. We demonstrated two attacks against reduced-round variants that start from the middle of the 4-round core. Because security of TweAES-128 using tweak difference relies on the fact that the large-weight tweak difference will diffuse fast in the subsequent 2 rounds, those reduced-round analysis will not threaten the security of the full TweAES-128. From a different viewpoint, one can see the difficulty to extend the analysis by 1 more round from Figs. 3.2 and 3.4. The number of involved subkey bytes easily exceeds 16.

Remarks on Other Attacks

- Integral attacks collect 2^8 distinct values for a particular byte or distinct 2^{32} values for a particular diagonal. Integral attacks exploiting the tweak is difficult because the tweak will not affect all the bits in each byte, which prevents to collect 2^8 distinct values for any byte.
- Meet-in-the-middle attacks exploit the 4-round truncated differentials $1 \to 4 \to 16 \to 4 \to 1$ and focus on the fact that the number of differential characteristics satisfying this differential is at most 2^{80}. The large-weight of the expanded tweak in TweAES-128 does not allow such sparse differential trails, which makes it hard to be exploited in the meet-in-the-middle attack.

3.3.2 Security Analysis of TweAES-128-6

In TweAES-128-6, the number of rounds is reduced from TweAES-128 by considering that the attackers do not have full control over the block cipher invocation in the modes. From this background, we do not analyze the security of TweAES-128-6 as a standalone tweakable block cipher, but show that the number of active S-boxes is sufficient to prevent attacks.

As a result of running the MILP-based tool, it turned out that the differential trail achieving the minimum number of active S-boxes with some non-zero tweak difference is 20. An example of the differential trails achieving 20 active S-boxes is the first two rounds of the trail in Fig. 3.1.

Given that the maximum differential probability of the AES-128 S-box is 2^{-6}, the probability of the differential propagation is upper bounded by $2^{-6\times 20} = 2^{-120}$. Because our mode does not allow the attacker to make 2^{120} queries, it is impossible to perform the differential cryptanalysis.

3.3.3 Security of TweGIFT-128

In this expansion, the 4-bit tweak expands to 8 bits and those 8 bits are copied three times to achieve a 32-bit tweak. When the 4-bit tweak has some non-zero difference, the expanded 32-bit tweak is ensured to have at least 16 active bits, which ensures at least 16 active S-boxes in 2 rounds around the tweak injection.

Owing to the large state size and a large number of active S-boxes, it is infeasible to find the tight bound of the maximum probability of the differential characteristic for the 10-round core by using MILP. The tool so far provided that the maximum probability of the differential characteristic is upper bounded by $2^{-64.5}$. Given that the entire TweGIFT-128128 consists of 40 rounds and thus contains 4 of the 10-round cores, the upper bound of the entire construction is $2^{-64.5\times 4} = 2^{-258}$, which is sufficient to resist the attack.

Chapter 4

Features

Here we discuss the main features of ESTATE and sESTATE.

1. **Nonce-misuse Resistant:** ESTATE is a nonce-misuse resistant authenticated cipher and provides full security even with the repeatition of nonce. Alternatively said, it can be viewed as a *deterministic authenticated encryption* where the nonce is assumed to be the first block of the associated data.

2. **Single State:** ESTATE has a state size as small as the block size of the underlying cipher, and it ensures good implementation characteristics both on lightweight and high-performance platforms.

3. **Inverse-Free:** ESTATE is an inverse-free authenticated encryption algorithm. Both encryption and decryption algorithms do not require any decryption call to the underlying tweakable block cipher. This significantly reduces the overall hardware footprint in combined encryption-decryption implementations.

4. **Multiplication-free:** ESTATE does not require any field multiplications. In fact, apart from the tweakable block cipher call it requires just 128-bit XOR per block of data, which seems to be the minimum required overhead.

5. **Optimal:** ESTATE requires $(a + 2m)$ many primitive invocations to process an a block associated-data (including the nonce) and m block message. In [8], it has been shown that this is the optimal number of non-linear primitive calls required for deterministic authenticated encryption. This feature is particularly important for short messages from the perspective of energy consumption, which is directly dependent upon the number of non-linear[1] primitive calls.

6. **RUP Secure:** We separate the block cipher invocations for the OFB functions and the first tweakable block cipher input invocation by the usage of different tweaks. This essentially helps us to provide RUP security for ESTATE, making it much more robust in constraint devices. Here, we note that a related construction SUNDAE lacks this feature.

7. **Robustness:** Most of the AEAD schemes require a unique nonce value, in order to create a secret (almost) uniform random state. This helps in achieving the security requirements. But the problem with these schemes is the lack of security in absence of this secret state.[2] In contrast ESTATE mode is quite robust, as evident by point 1 and 6 above, to a lack of sufficient randomness or secret states.

[1]In general, non-linear operations consume significantly more energy as compared to linear operations

[2]A scenario possible in nonce-misusing or RUP model.

Chapter 5

Design Rationale

Now we briefly mention 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.** For ESTATE, 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. 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. 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.

Chapter 6

Hardware Implementation Results

We implemented ESTATE_TweAES-128 and ESTATE_TweGIFT-128, as well as the standalone versions of TweAES-128 and TweGIFT-128. All of them are round-based architectures (the data-path size is 128-bit). From the summary of results in Table 6.1, we can observe the overhead introduced by the extra components required for the mode of operation. In terms of LUTs, the overhead is 618 LUTs in ESTATE_TweAES-128 and 623 in ESTATE_TweGIFT-128. In terms of flip-flops, we observe that the overhead in ESTATE_TweGIFT-128 is almost two times that in ESTATE_TweAES-128. The overheads are introduced by an input multiplexer to the cipher, two 128-bit XORs to compute the FCBC* and OFB modes, one additional multiplexer to select the correct output between tag or plaintext/ciphertext, and finally the control unit that also generates the corresponding tweak values.

The throughput for ESTATE_TweGIFT-128 is more than 50% lower than ESTATE_TweAES-128. This is because the number of cycles to process one block is much more significant for TweGIFT-128 based construction, 80 cycles against 20 cycles in ESTATE_TweAES-128.

Table 6.1: Implementation results of ESTATE_TweAES-128 and ESTATE_TweGIFT-128 along with their underlying block ciphers TweAES-128 and TweGIFT-128 on *Virtex 7 FPGA*.

Primitive	LUTs	FF	Slices	Frequency (MHz)	Clock cycles	Throughput (Mbps)
TweAES-128	1617	524	574	328.27	11	3819.87
ESTATE_TweAES-128	2235	679	1110	314.71	20	2014.14
TweGIFT-128	790	408	336	597.59	41	1865.65
ESTATE_TweGIFT-128	1413	698	616	580.11	80	928.27

In terms of hardware area, we can see that ESTATE_TweAES-128 is around 40% bigger than ESTATE_TweGIFT-128. This can be explained by the nature of GIFT which was designed to be ultra-lightweight. The same behavior is observed for stand-alone implementations of the block ciphers.

ESTATE is also efficient for shorter messages, as evident from Table 6.2 which gives the throughput for message lengths between 16 bytes to 1024 bytes. From Table 6.2, it can also be observed that as the length of the message increases the throughput tends to the values presented in Table 6.1.

Table 6.2: Throughput for ESTATE_TweAES-128 and ESTATE_TweGIFT-128 for short messages, from 16 bytes to 1024 bytes

Mode	16B	32B	64B	128B	512B	1024B
ESTATE_TweAES-128	1681.04	1918.24	1965.02	1989.28	2007.87	2011.00
ESTATE_TweGIFT-128	905.54	916.72	922.41	925.29	927.45	927.82

To conclude, we can say that in terms of speed ESTATE_TweAES-128 is better, while in terms of area ESTATE_TweGIFT-128 is a better option.

It is important to note that these implementations are a first attempt at showcasing the efficiency/low-area features of ESTATE. There are many possibilities for optimization, mainly related to the optimization of the underlying cipher. For instance, take the case of TweAES-128. There are many ways to do a small footprint implementation of AES-128, which could be applied to TweAES-128 as well. For example, using a smaller data-path, say 32-bit, and T-tables instance of SBoxes. For TweGIFT-128, one can explore different degrees of serialization from bit-serial to round-based implementations. Another direction that we can explore is the implementations in constrained devices like automotive and low power FPGAs, and small, 8-bit or 16-bit micro-controllers.

Bibliography

[1] Recommendation for Block Cipher Modes of Operation: Methods and Techniques. NIST Special Publication 800-38A, 2001. National Institute of Standards and Technology.

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

[3] 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.

[4] Subhadeep Banik, Andrey Bogdanov, Atul Luykx, and Elmar Tischhauser. Sundae: Small universal deterministic authenticated encryption for the internet of things. *IACR Transactions on Symmetric Cryptology*, 2018(3):1–35, Sep. 2018.

[5] 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.

[6] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, pages 123–153, 2016.

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

[8] Avik Chakraborti, Nilanjan Datta, and Mridul Nandi. On the optimality of non-linear computations for symmetric key primitives. *J. Mathematical Cryptology*, 12(4):241–259, 2018.

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

[10] Victor Shoup. On fast and provably secure message authentication based on universal hashing. In *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, pages 313–328, 1996.

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