IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2020, No. S1, pp. 350–389. DOI:10.13154/tosc.v2020.iS1.350-389

ESTATE: A Lightweight and Low Energy Authenticated Encryption Mode

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

[1] NTT Secure Platform Laboratories, Tokyo, Japan
avikchkrbrti@gmail.com,yu.sasaki.sk@hco.ntt.co.jp
[2] Indian Statistical Institute, Kolkata, India
nilanjan_isi_jrf@yahoo.com,ashwin.jha1991@gmail.com,mridul.nandi@gmail.com
[3] Computer Science Department, Center for Research and Advanced Studies of the National Polytechnic Institute (CINVESTAV-IPN), Mexico City, Mexico
cuauhtemoc.mancillas@cinvestav.mx

Abstract. NIST has recently initiated a standardization project for efficient lightweight authenticated encryption schemes. SUNDAE, a candidate in this project, achieves optimal state size which results in low circuit overhead on top of the underlying block cipher. In addition, SUNDAE provides security in nonce-misuse scenario as well. However, in addition to the block cipher circuit, SUNDAE also requires some additional circuitry for multiplication by a primitive element. Further, it requires an additional block cipher invocation to create the starting state. In this paper, we propose a new 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 an 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, small 't' denotes short tweaks) and we instantiate it with two recently designed tBCs: TweAES and TweGIFT. We also propose a low latency variant of ESTATE, called sESTATE, that uses a round-reduced (6 rounds) variant of TweAES called TweAES-6. We provide comprehensive FPGA based hardware implementation for all the three instances. The implementation results depict that ESTATE_TweGIFT-128 (681 LUTs, 263 slices) consumes much lesser area as compared to SUNDAE_GIFT-128 (931 LUTs, 310 slices). When we moved to the AES variants, along with the area-efficiency (ESTATE_TweAES consumes 1901 LUTs, 602 slices while SUNDAE_AES-128 needs 1922 LUTs, 614 slices), we also achieve higher throughput for short messages (For 16-byte message, a throughput of 1251.10 and 945.36 Mbps for ESTATE_TweAES and SUNDAE_AES-128 respectively).

Keywords: SUNDAE · TweAES · TweGIFT · tBC · authenticated encryption · lightweight · RUP

1 Introduction

In recent years, lightweight authenticated encryption with associated data (AEAD) has seen a sudden surge in interest due to the advent of Internet of things (IoT). The present AEAD standards are not suitable in the spectrum of lightweight applications as they are designed for more general use-cases. This leads to the call for standardization process for new lightweight AE designs. The designs are mainly (tweakable) block cipher, stream cipher or permutation-based. Block cipher based designs have one particular advantage

Received: 2019-12-10, Revised: 2020-03-15, Accepted: 2020-04-01, Published: 2020-06-22

as we can have a concrete security proof in the standard model. Thus, it is attractive to design a TBC-based AEAD with a small state (low storage), fewer primitive invocations (low energy) as well as concrete security analysis. The designs mainly target the following properties.

1.1 Designing Lightweight Block Cipher Based AEAD

To design a lightweight AE mode, we need to be concerned about implementation properties as well as security properties. Implementation properties can have two directions: (1) low area implementations and (2) low energy implementations. Regarding security, it is also important to have RUP security in addition to the standard security requirements.

1.1.1 Small State Size

JAMBU [WH16] is one of the most relevant modes in terms of small state size using only $1.5n + \kappa$ bit state, with an n-bit block cipher and a κ-bit key. Here state size means a theoretical estimation of the main registers, and a block is defined as a binary string of n bits, where n is the size (in bits) of the underlying primitive. Although JAMBU has a small state size, the privacy claim has been proven to be flawed [PSWZ15]. In addition, JAMBU has rate 1/2 (ratio between the number of data blocks and the number of primitive invocations), which makes it slow (it also depends on the cost of the underlying primitive). In 2017, COFB [CIMN17a] surpassed JAMBU in terms of provable security and rate. Later, SAEB [NMSS18] optimized the state size further but with a compromise in the rate. Interestingly, all these modes are inverse-free, i.e. we do not need the decryption circuit for the underlying primitives.

1.1.2 Low Number of Primitives Calls

The energy consumption of any cryptographic scheme is directly dependent on two factors: a) energy consumption of the underlying primitive, and b) number of primitive calls. Of the two factors, the former falls under the ambit of block cipher design where several energy efficient candidates like Midori [BBI+15] and GIFT [BPP+17] are available, while the latter is accounted in the design of modes of operation. Indeed, minimizing the number of primitive calls is a common design approach for energy-efficient AEAD modes such as SUNDAE [BBLT18]. SUNDAE is primarily designed for lightweight applications, where the messages are generally short (e.g, one block). In this regard, lowering the number of primitive calls also helps in reducing the overheads for processing short messages. Although, SUNDAE minimizes the number of primitive calls to a large extent, we observe that it is still not optimal and there is scope to optimize it further.

1.1.3 Nonce Misuse and RUP Security

Nonce-misuse resistance [RS06] is a security property of AEAD when the attacker can repeat a nonce during the encryption queries. GIFT-SUNDAE is a prominent nonce-misuse resistant design. The designers mainly aim to optimize the state size, block cipher calls along with nonce-misuse resistance.

Decrypted plaintext needs to be verified before it is released. In several applications, the decrypted plaintext should be kept in a secure buffer before the verification completes. However, in some resource-constrained environments with limited buffer space, we have to release a part of the plaintext even before it is verified. This is formalized as RUP [ABL+14], where integrity is ensured even with the release of unverified plaintext. RUP is mainly useful for real-time streaming protocols (e.g. SRTP, SRTCP and SSH), Optical Transport Networks, where block-wise encryption/decryption is required and ciphertext/plaintext

are released on the fly in order to reduce the end-to-end latency and/or compensate for low memory.

1.2 SUNDAE: A Lightweight AEAD

SUNDAE is a block cipher based lightweight AEAD that aims at maximal robustness using small state size, and is particularly efficient for short messages. SUNDAE is optimized for lightweight applications by optimizing the state size using only one key, a cascade of block cipher calls and only one full block XOR along with a multiplication with a constant. The state size for SUNDAE is only n bits (excluding the key storage) using an n-bit block cipher.

There is still scope for further improvements on the structure of SUNDAE to make it more efficient for lightweight applications. In [CDN18], Chakraborti et al. showed that the optimal number of block cipher invocations required to process a data with an input string consisting of a associated data blocks and m plaintext blocks is $(a + 2m)$. However, SUNDAE requires $(a+2m+1)$ block cipher invocations. This is primarily due to the initial block cipher invocation for domain separation. The number of block cipher invocations directly relates to the energy consumption and is quite significant for processing short messages. Another possible improvement can be RUP security that SUNDAE is missing.

1.3 Our Contribution

In this paper, we propose a new highly secure and hardware efficient tBC based AEAD mode, named as ESTATE (Energy efficient and Single-state Tweakable block cipher based MAC-Then-Encrypt). The structure employs an FCBC-like MAC [BR05] followed by OFB mode [ENC01]. ESTATE is structurally close to SUNDAE, but with an additional interesting design feature of replacing the block cipher by a tBC along with a few design changes. We address the points that SUNDAE needs to adopt several internal operations to deal with domain separation, SUNDAE does not provide any provable RUP security and SUNDAE is near optimal but not optimal in the number of block cipher invocations (since it is encrypting a data type and length dependent constant during initialization). We can remove the above issues in SUNDAE by using different tweaks in the underlying tBC to (i) reduce the primitive invocation (we pre-compute a fixed tBC encrypted nonce with the unique tweak value 1 and use it all the time), (ii) provide RUP security (as we use different tweaks for the tBC used in the encryption and the first tBC call during authentication), and (iii) clean up the other domain separation related operations in SUNDAE by tweak adjustments. Overall, ESTATE has the following large set of features:

- **Optimum state size:** ESTATE has a state size as small as the underlying block cipher.
- **Multiplication-free:** ESTATE does not require any field multiplications. In fact, apart from the tBC call it requires just a 128-bit XOR per block of data, which seems to be a negligible overhead. Observe that, SUNDAE requires constant field multiplications with field elements **2** and **4** for the purpose of domain separation. In contrast, we simply use different tweaks to achieve this.
- **Optimal:** ESTATE requires $(a + 2m)$ primitive invocations to process a blocks of associated data (including the nonce) and m message blocks. In [CDN18], 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 primitive calls. SUNDAE requires a constant block encryption at the beginning primarily because the same block cipher is used in

encryption as well as authentication. We remove this extra call by using different tweaks for the tBC calls.

- **Inverse-Free:** ESTATE is inverse-free. Both encryption and decryption algorithms do not require any decryption call to the underlying tBC. This significantly reduces the overall hardware footprint in combined encryption-decryption implementations.
- **Nonce-misuse Resistant:** ESTATE is nonce-misuse resistant and provides full security even with the repetition of the nonce. Alternatively, it can be viewed as a deterministic authenticated encryption where the nonce is assumed to be the first block of the associated data.
- **RUP Secure:** We separate the block cipher invocations for the OFB functions and the first tBC input invocation by using different tweaks. This helps us to provide RUP security for ESTATE. SUNDAE lacks this feature and the designers of SUNDAE explicitly mentioned that "unverified plaintext from the decryption algorithm should not be released."
- **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 security requirements. But the problem with these schemes is the lack of security in the absence of this secret state. In contrast ESTATE is quite robust, as evident by nonce misuse resistance and RUP security, to a lack of sufficient randomness or secret states.

Next, we propose a lighter AEAD mode sESTATE, which is structurally identical to ESTATE. The only difference between sESTATE and ESTATE is that sESTATE uses a round-reduced version of the underlying tBC to compute the MAC. The tBC used in the encryption part remains the same.

Finally, we instantiate ESTATE with both TweGIFT and TweAES and sESTATE with TweAES (and its reduced version TweAES-6) as the underlying tBC. We provide complete hardware implementation details on FPGA platform along with benchmarks with the existing designs. The implementation results depict that ESTATE_TweGIFT-128 (681 LUTs, 263 slices) clearly outperforms SUNDAE_GIFT-128 (931 LUTs, 310 slices) in hardware area. In addition, ESTATE_TweAES (1901 LUTs, 602 slices) also has better area-efficiency as compared to SUNDAE_AES-128 (1922 LUTs, 614 slices) and achieves higher throughput for short messages (as example, for 16 byte message, ESTATE_TweAES has a throughput of 1251.10 Mbps while SUNDAE_AES-128 achieves 945.36 Mbps).

Remark 1. Although ESTATE's features like robustness and RUP security are lucrative in the lightweight scenario, they are also costly in terms of efficiency. In fact, a drawback for any SIV based scheme, including SUNDAE and ESTATE, is their two-pass nature which could be an implementation constraint in certain scenarios with low memory buffer. Clearly, there is a trade-off between additional security guarantees and efficiency, and we aim to maximize the efficiency after ensuring the additional security guarantees. This is evident from the optimality and multiplication-free features of ESTATE which are not true for SUNDAE.

1.4 ESTATE in Light of the NIST Lightweight Competition

NIST lightweight cryptography project [MBTM17] started in 2018 recognizing the lack of efficient AE standards for lightweight applications. They mainly addressed the growing security requirements for applications such as sensor networks, distributed control systems and health care. These applications are mainly involved with resource-restricted devices communicating among themselves. Here we present a comparative chart in Table 1 to

study five other Synthetic IV (SIV) based modes submitted to the NIST competition with our proposal ESTATE. We remark that, out of these four designs, only GIFT-SUNDAE has been selected for the second round of the NIST Lightweight Project. The above chart

Table 1: Comparative Study on SIV based NIST 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). Note that the field 'Optimality' denotes the optimality on the number of primitive invocations

Submission	Primitive	State size (bits)	Optimality	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	✓	✓	✓
GIFT-SUNDAE	BC-128/128	256	×	×	×
TRIFLE	BC-128/128	384	×	×	×

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

2 Preliminaries

2.1 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 (denoted by *data blocks*), 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 bits) of A, where $|\lambda| = 0$ by convention. For all practical purposes, we assume the least significant bit is the rightmost bit. For any non-empty binary string X, $(X_{k-1}, \ldots, X_0) \xleftarrow{n} x$ denotes the n-bit block parsing of X, where $|X_i| = n$ for $0 \le i \le k-2$, and $1 \le |X_{k-1}| \le n$. 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 left and the right 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 a key $K \in \{0,1\}^\kappa$, tweak $T \in \{0,1\}^\tau$, and a message $M \in \{0,1\}^n$, 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 sometimes 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 $0 \le i \le |X| - 1$, x_i denotes the

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

i-th bit of X. The function chop takes a string X and an integer $i \leq |X|$, and returns the rightmost 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.

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 "$\mathtt{E}\ ?\ a : b$" 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.

2.2 Authenticated Encryption

An authenticated encryption scheme offers both confidentiality, meaning that its tagged ciphertexts are indistinguishable from a string uniform at random, and integrity, meaning that its tags are unforgeable. Typically, we combine the above two functionalities of an authenticated encryption into a unified one, which is formally defined as:

Definition 1. Let $\mathfrak{A} = (\mathcal{E}, \mathcal{D}, \mathcal{V})$ be an authenticated encryption scheme. The AE security of $\mathfrak{A}$ against an adversary $\mathcal{A}$ is defined as

$$\mathbf{Adv}^{\mathsf{AE}}_{\mathfrak{A}} := |\mathsf{Pr}[\mathcal{A}^{\mathcal{E}_K, \mathcal{V}_K} = 1] - \mathsf{Pr}[\mathcal{A}^{\$, \perp} = 1]|,$$

where $\$$ is the random oracle that on input (A, M) returns (C, T) uniformly at random and $\perp$ is the oracle that on input (A, C, T), always rejects. The randomness for the first probability is defined over $K \xleftarrow{\$} \{0,1\}^k$ and also over the random coins of $\mathcal{A}$ (if any). Similarly, the randomness for the second probability is defined over the randomness of $\$$, and over the random choices of $\mathcal{A}$ (if any).

We define

$$\mathbf{Adv}^{\mathsf{AE}}_{\mathfrak{A}}(t, q_e, q_v, \sigma_e, \sigma_v) = \max_{\mathcal{A}} \mathbf{Adv}^{\mathsf{AE}}_{\mathfrak{A}}(\mathcal{A}),$$

where the maximum is considered over all adversaries with running time t, q_e encryption queries and q_v verification queries such that the total number of queried blocks are at most σ_e and σ_d, respectively.

Now we provide the extended definition of AE security in the released unverified plaintext (RUP) setting. The RUP model, called AERUP, combines RUP confidentiality (i.e. PA1) and integrity (i.e. INT-RUP) and was proposed by [CDD+19]. In this model, we have two worlds: (i) real world that is comprised of encryption, decryption and verification oracle of the AE algorithm and (ii) ideal world which is also comprised of three oracles: (a) random oracle $\$$ that on input (A, M), samples the ciphertext C of the same length uniformly at random, (b) the simulator $\mathcal{S}$ with access to the history of encryption queries, on input (A, C, T), returns the plaintext in a consistent way, and (c) reject oracle $\perp$, that on input (A, C, T) always returns $\perp$. Note that, it is sufficient to prove AERUP security as AERUP implies AE security i.e, if a scheme is AERUP secure then it is secure under conventional confidentiality and authenticity notion. Moreover, it is also secure under RUP confidentiality and authenticity notion (it is also called INT-RUP security).

Definition 2. Let $\mathfrak{A} = (\mathcal{E}, \mathcal{D}, \mathcal{V})$ be an authenticated encryption scheme. Let $\mathcal{A}$ be an adversary with access to a triplet of oracles $(\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3)$. The AERUP security of $\mathfrak{A}$ against an adversary $\mathcal{A}$ is defined as

$$\mathbf{Adv}^{\mathsf{AERUP}}_{\mathfrak{A}} = |\ \mathsf{Pr}[\mathcal{A}^{\mathcal{E}_K, \mathcal{D}_K, \mathcal{V}_K} = 1] - \mathsf{Pr}[\mathcal{A}^{\$, \mathcal{S}, \perp} = 1]\ |\,, \tag{1}$$

where the randomness is taken over $K \xleftarrow{\$} \{0,1\}^k$ in the first probability calculation and the randomness is defined over \$, $\mathcal{S}$ in the second probability calculation. However the randomness is also defined over the random coins of $\mathcal{A}$. Note that, $\mathcal{A}$ can query oracle $\mathcal{O}_2$ with input that is obtained from $\mathcal{O}_1$ as a result of some previous encryption query.

Similar to the previous definition, we define

$$\mathbf{Adv}^{\mathsf{AERUP}}_{\mathfrak{A}}(t, q_e, q_d, q_v, \sigma_e, \sigma_d, \sigma_v) = \max_{\mathcal{A}} \mathbf{Adv}^{\mathsf{AERUP}}_{\mathfrak{A}}(\mathcal{A}),$$

where the maximum is considered over all adversaries with running time t, q_e encryption queries, q_d decryption queries and q_v verification queries such that the total number of queried blocks are at most σ_e, σ_d, σ_v respectively. For brevity, we write $\sigma = \sigma_e + \sigma_d + \sigma_v$. In concrete terms, σ and t denotes the data and time complexity, respectively.

2.3 PRF, (T)PRP Security

The *TPRP-advantage* of $\mathcal{A}$ against $\widetilde{\mathsf{E}}$ is defined as

$$\mathbf{Adv}^{\mathsf{TPRP}}_{\widetilde{\mathsf{E}}}(\mathcal{A}) = |\Pr[\mathcal{A}^{\widetilde{\mathsf{E}}_K} = 1] - \Pr[\mathcal{A}^{\widetilde{\Pi}} = 1]|,$$

where $\widetilde{\Pi}$ is a tweakable random permutation uniformly distributed over the set of all tweakable permutations over tweak space $\{0,1\}^\tau$ and block space $\{0,1\}^n$. We remark that the adversary has full control over both the tweak value and input of the tweakable block cipher. We write

$$\mathbf{Adv}^{\mathsf{TPRP}}_{\widetilde{\mathsf{E}}}(t, q) = \max_{\mathcal{A}} \mathbf{Adv}^{\mathsf{TPRP}}_{\widetilde{\mathsf{E}}}(\mathcal{A}),$$

where the maximum is taken over all adversaries with running time t and at most q queries to the oracle.

The PRF advantage of distinguisher $\mathcal{A}$ against a keyed family of functions $\mathcal{F} := \{\mathcal{F}_K : \{0,1\}^m \to \{0,1\}^n\}_{K \in \{0,1\}^\kappa}$ is defined as

$$\mathbf{Adv}^{\mathsf{PRF}}_{\mathcal{F}}(\mathcal{A}) := \left|\Pr[\mathcal{A}^{\mathcal{F}_\mathsf{K}} = 1] - \Pr[\mathcal{A}^{\Gamma} = 1]\right|,$$

where Γ is a random function uniformly distributed over the set of all functions from $\{0,1\}^m$ to $\{0,1\}^n$. The PRF security of $\mathcal{F}$ is defined as

$$\mathbf{Adv}^{\mathsf{PRF}}_{\mathcal{F}}(q, t) := \max_{\mathcal{A}} \mathbf{Adv}^{\mathsf{PRF}}_{\mathcal{F}}(\mathcal{A}). \tag{2}$$

The keyed family of functions PRF is called weak PRF family, if the PRF security holds when the adversary only gets to see the output of the oracle on uniform random inputs. This is clearly a weaker notion than PRF. We denote the weak prf advantage as $\mathbf{Adv}^{\mathsf{wprf}}_{\mathsf{PRF}}(q, t)$.

2.4 Patarin's H-Coefficient Technique

We briefly discuss the H-coefficient technique of Patarin [Pat08, CS14]. Consider a computationally unbounded deterministic adaptive adversary $\mathcal{A}$ that interacts with either a real oracle $\mathcal{O}_{\text{re}}$ or an ideal oracle $\mathcal{O}_{\text{id}}$. After its interaction, $\mathcal{A}$ outputs a decision bit. The collection of all queries-responses obtained by $\mathcal{A}$ during its interaction with its oracle are summarized in a transcript τ. This transcript may contain additional information about the random oracle that is revealed to the adversary after its interaction but before it outputs its decision bit. This is without loss of generality: the adversary gains more knowledge and hence more distinguishing power.

Let X_{re} and X_{id} be the random variables that take a transcript τ induced by the real and the ideal world respectively. The probability of realizing a transcript τ in the ideal

world (i.e. $\mathsf{Pr}[X_{\text{id}} = \tau]$) is called the *ideal interpolation probability* and the probability of realizing it in the real world is called the *real interpolation probability*. A transcript τ is said to be *attainable* if the ideal interpolation probability is non-zero. We denote the set of all attainable transcripts by Θ. Following these notations, we state the main theorem of the H-coefficient technique as follows [Pat08, CS14].

Theorem 1 (H-coefficient technique)**.** *Let $\mathcal{A}$ be a fixed computationally unbounded deterministic adversary that has access to either the real oracle $\mathcal{O}_{\text{re}}$ or the ideal oracle $\mathcal{O}_{\text{id}}$. Let $\Theta = \Theta_{\text{good}} \sqcup \Theta_{\text{bad}}$ be some partition of the set of all attainable transcripts into* good *and* bad *transcripts. Suppose there exists an $\epsilon_{\text{ratio}} \geq 0$ such that for any $\tau \in \Theta_{\text{good}}$,*

$$\frac{Pr[X_{\text{re}} = \tau]}{Pr[X_{\text{id}} = \tau]} \geq 1 - \epsilon_{\text{ratio}} \,,$$

and there exists an $\epsilon_{\text{bad}} \geq 0$ such that $Pr[X_{\text{id}} \in \Theta_{\text{bad}}] \leq \epsilon_{\text{bad}}$. Then,

$$Pr[\mathcal{A}^{\mathcal{O}_{\text{re}}} \to 1] - Pr[\mathcal{A}^{\mathcal{O}_{\text{id}}} \to 1] \leq \epsilon_{\text{ratio}} + \epsilon_{\text{bad}} \,. \tag{3}$$

3 Short-Tweak Tweakable Block Ciphers

In this section, we describe two tweakable block ciphers TweAES and TweGIFT [CDJ+19] which can incorporate 4-bit short tweaks. We will instantiate our modes with these short tweak tweakable block ciphers.

3.1 Specification of TweAES

In this section we provide the specification of the tweakable block cipher TweAES and TweAES-6. TweAES 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 [FIP01] block cipher. TweAES is identical to AES-128/128 except that we inject a tweak value at intervals of 2 rounds. Now we briefly describe the main steps of the TweAES round function.

SubBytes: TweAES uses the same invertible 8-bit S-box as AES and applies it to each byte of the cipher state.

ShiftRows: The bytes in the i-th row are 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:

$$(x_1, x_2, x_3, x_4) \to (x_1, x_2, x_3, x_4, S \oplus x_1, S \oplus x_2, S \oplus x_3, S \oplus x_4), \text{ where } S = x_1 \oplus x_2 \oplus x_3 \oplus x_4.$$

and then the 8-bit value is XORed to the state at an interval of 2 rounds. To be precise, the 8-bit tweak is added to the least significant bit of each byte in top two rows of the state.

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

Specification of TweAES-6. We also define a round-reduced version of TweAES, called TweAES-6, which is composed of the first 6 rounds of TweAES. 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. A detailed description can be found in [CDJ+19].

TweAES-6 was designed to ensure security for use in our modes. This corresponds to the use of 4-round AES in various AEAD modes, for which no attack is known on the 4-round AES in proper modes under the restriction of the birthday-bound query limit. TweAES-6 was designed by following the same concept.

3.2 Specification of TweGIFT

TweGIFT 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 the GIFT-128 [BPP+17] block cipher. TweGIFT is composed of 40 rounds and each round is composed of the following operations:

SubCells: TweGIFT uses the same invertible 4-bit S-box as GIFT and applies it to each nibble of the cipher state.

PermBits: TweGIFT 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 GIFT.

AddTweak: The 4-bit tweak is first expanded to a 32-bit value:

$$(x_1, x_2, x_3, x_4) \to (X, X, X, X)\,,\ X \leftarrow (x_1, x_2, x_3, x_4, S \oplus x_1, S \oplus x_2, S \oplus x_3, S \oplus x_4),$$

where $S = x_1 \oplus x_2 \oplus x_3 \oplus x_4$. Then the 32-bit value is XORed to the state at an interval of 5 rounds. To be precise, it adds the expanded 32-bit tweak to bit positions $4i + 3$, $i = 0 \ldots 31$. A detailed description can be found in [CDJ+19].

We would like to point out that the area overhead for this tweak injection is negligible. Infact TweAES has an overhead (in LUTs) of 0.5% (and 0.7%) for combined encryption-decryption (and encryption only implementation resp), while TweGIFT-128 has an overhead of 4.04% and 4.32% resp. The details are given in Sect.6.1.

3.3 Efficient Security Evaluation for Elastic Tweak

Regarding differential and linear cryptanalysis, the lower bound of the number of active S-boxes and the upper bound of the maximum differential characteristic probability can be obtained by using various tools based on MILP and SAT, however to derive such bounds for the entire construction is often infeasible. Here, we introduce an efficient method to ensure the security against differential and linear cryptanalyses by exploiting the fact that the expanded tweak has a large weight.

Suppose that the expanded tweak is injected to the state every r rounds. Then we focus on $2r$ rounds around the tweak injection, namely a sequence of the following three operations: the r-round transformation, the tweak injection, and another r-round

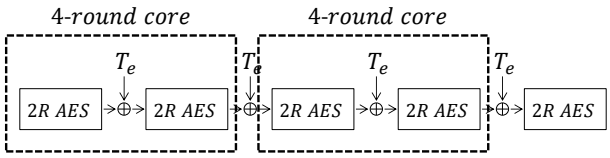

Figure 1: 4-round Core of TweAES [∗,∗,∗,2]

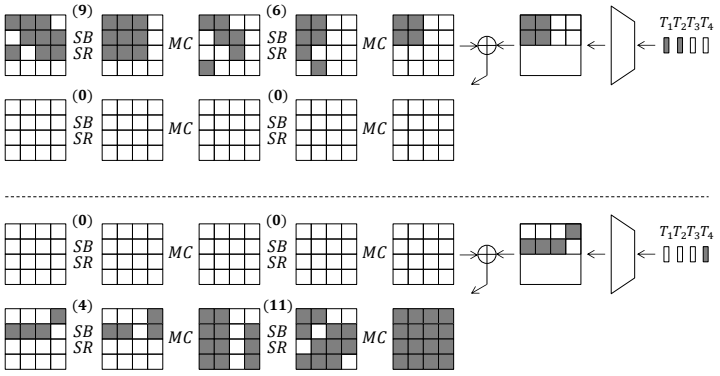

Figure 2: TweAES: Two Examples of Differential Trails with 15 Active S-boxes.

transformation. We call those operations "$2r$-*round core*," which is depicted for AES and GIFT-64 in Fig. 1. Because the entire construction includes several $2r$-round cores, security of the entire construction can be bounded by accumulating the bound for the single $2r$-round core: this depends of course on the value of r. The large weight of the expanded tweak ensures a strong security bound for the $2r$-round core, which is sufficient to ensure the security for the entire construction.

3.4 Security Analysis of TweAES

3.4.1 Resistance against Differential and Linear Cryptanalyses

We evaluate the minimum number of differentially and linearly active S-boxes for the 4-round core of TweAES. A 4-bit tweak of TweAES is divided into 4 parts denoted by T_1, T_2, T_3, T_4, where the size of each T_i is 1 bit.

When the tweak input has a non-zero difference, the expanding function ensures that at least 4 bytes are affected by the tweak difference. It is easy to check by hand that the minimum number of active S-boxes of the 4-round core under this constraint is 15. We also modeled the problem by MILP and experimentally verified that the minimum number of active S-boxes is 15. This is a tight bound and two examples of the differential trails achieving 15 active S-boxes are given in Fig. 2. Given that the maximum differential probability of the AES S-box is 2^{-6}, the probability of the differential propagation through the 4-round core with non-zero tweak difference is upper bounded by $2^{-6\times 15} = 2^{-90}$. The probability of the differential propagation of TweAES is upper bounded by $2^{-90\times 2} = 2^{-180}$ because 10 rounds of TweAES include two 4-round cores.

Thanks to the simple structure of AES, it is also possible to experimentally compute the lower bound of the number of active S-boxes of the full-round TweAES. When the tweak input has a non-zero difference, the minimum number of active S-boxes is 40 for the entire construction. Hence, the probability of the differential propagation is upper bounded by $2^{-6\times 40} = 2^{-240}$. This is a tight bound. An example of the differential trail achieving 40 active S-boxes is given in Fig. 3.

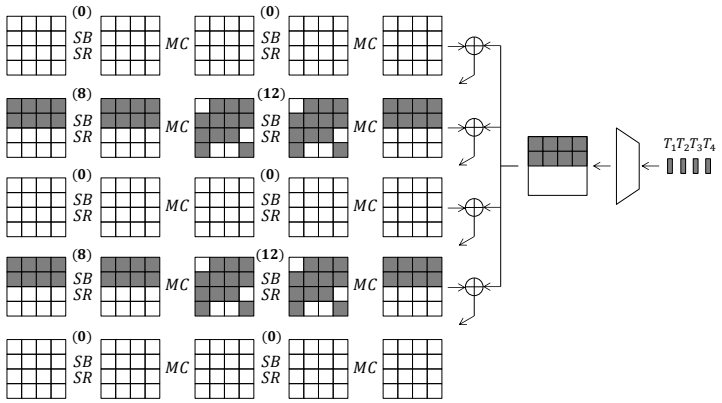

Figure 3: TweAES: An Example of the Differential Trail with 40 Active S-boxes.

The number of linearly active S-boxes can be evaluated in the same way.

3.4.2 Cryptanalysis from the First Round by Exploiting Tweak

In this section, we will show integral attacks, impossible differential attacks and truncated differential attacks against reduced-round variants that start form the initial round. The main purpose is to show the difficulty of exploiting the 4-bit tweak in the attack, thus we do not discuss the case with the fixed tweak. (When the tweak is fixed, security of TweAES is the same as the original AES. The attacks on AES can also be applied to TweAES but those do not reveal any vulnerability introduced by TweAES.)

Integral Attacks. Because the tweak starts to appear only after the second round, it is difficult to extend the integral attacks by playing with plaintexts. The most reasonable approach to exploit the tweak is to fix the plaintext and to collect all possible 2^4 tweak inputs. The propagation of the property is given in Fig. 4. Because the plaintext is fixed, the state does not change during the first two rounds. By examining 16 possible tweaks, each bit of the expanded tweak becomes zero for 8 choices and one for 8 choices. Hence, when the value before the tweak injection is c, the value after the tweak injection is either c or $c \oplus 1$ and both occur 8 times. From the similar analysis, the balanced property is preserved after 2 rounds from the tweak injection.

The key recovery starts with 16 ciphertexts. The attacker guesses the 4 bytes of the last subkey as indicated in Fig. 4. Let W_5 be $MC^{-1}(K_5)$. Then, by guessing a byte of W_5, the corresponding byte position can be partially decrypted until the beginning of round 5, and thus the attacker can check whether or not the balanced property (a sum of the byte value among 16 texts is 0) is satisfied. The probability that the balanced property is observed is 2^{-8}, hence only 1 choice of the byte-difference at W_5 will remain as a right key candidate. The analysis can be iterated for 4 bytes of W_5. In the end, for each 2^{32} choice of 4 bytes of K_6, the corresponding 4 bytes of W_5 will be fixed. Namely, 64 bits of the key space is reduced to 32 bits. By using another set of a plaintext with 16 different tweaks, the key space is reduced to 1.

The memory complexity can be reduced by first preparing two sets of 16 texts, and then the bytes of K_6 are guessed. We can apply the same analysis to all 4 different columns to determine the key without exhaustive search. Hence, the data complexity is 2^5, the computational cost is $2^5 \cdot 2^{32} \cdot 2^8 = 2^{45}$, the memory amount is negligible.

Compared to the integral attack against original AES, we can exploit two blank rounds thanks to the tweak injection in every two rounds but then the property disappears more quickly because we need to activate at least 4 byte positions. The attack on the original

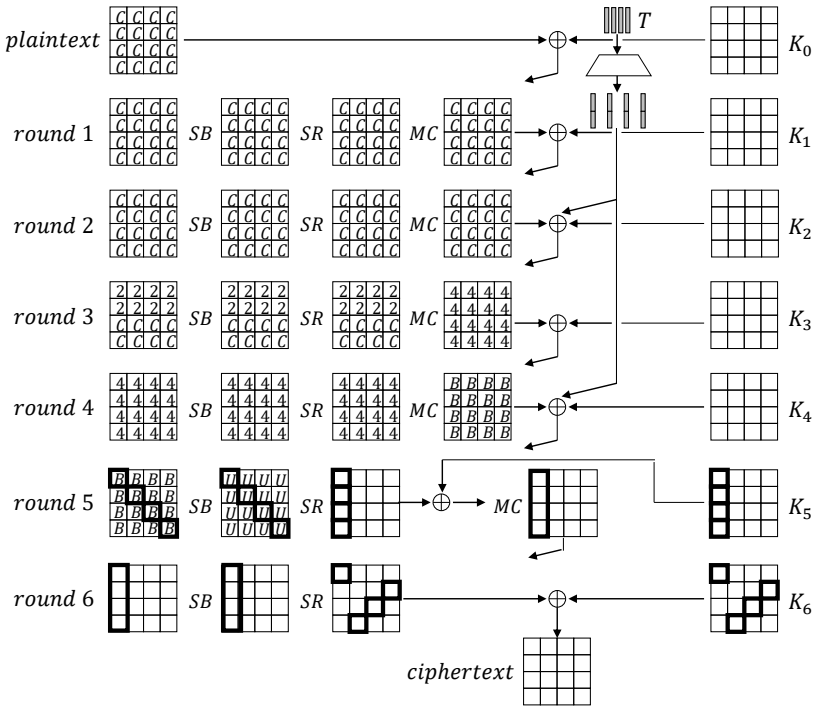

Figure 4: TweAES: Integral Distinguisher on TweAES via Tweak. '2' represents that two kinds of values appear 8 times each and '4' represents that four kinds of values appear 4 times each. By following the convention, 'B' and 'U' denote 'balanced' and 'unknown' properties, respectively.

AES appends 1 more round at the beginning of the integral distinguisher, which is difficult for TweAES via non-zero tweak because of the existence of 2 AES rounds before the first tweak injection.

Impossible Differential Attacks. With a non-zero tweak difference, the strategy to build an impossible differential is to inject it in the middle of the conventional 3.5-round impossible differential, as indicated by Fig. 5. Namely, in the middle round, the top and the bottom bytes in the left-most column are active with probability 1 in the forward direction, while those byte are inactive with probability 1 in the backward direction.

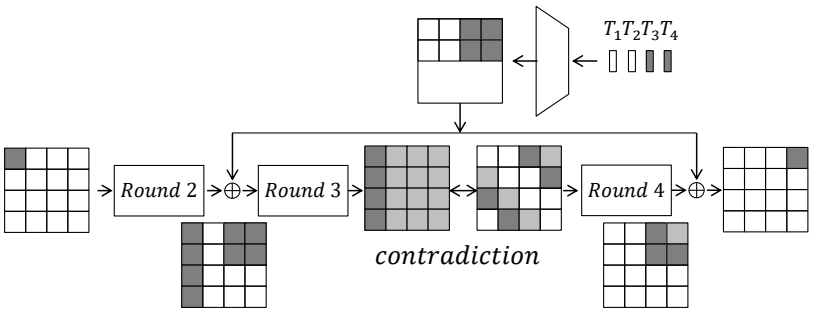

Figure 5: TweAES: 3-Round Impossible Differential with Non-Zero Tweak Difference.

For the key recovery, one round and two rounds can be appended to the beginning and the end of the 3-round impossible differential, which is illustrated in Fig. 6.

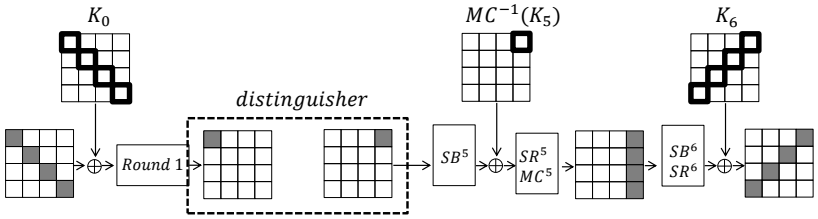

Figure 6: TweAES: 6-Round Key Recovery in Impossible Differential Attacks.

Because the tweak does not appear during the key recovery rounds, the procedure is the same as the one with the conventional 3.5-round impossible differential. To collect the data, the attacker constructs a structure, a set of 2^{32} plaintexts in which 2^{32} values are considered for active 4 bytes and the other 12 bytes are fixed. This generates $\binom{2^{32}}{2} \approx 2^{63}$ plaintext pairs. This is iterated X times by changing the value of the fixed 12 bytes of the plaintexts, which results in $X \cdot 2^{32}$ queries and $X \cdot 2^{63}$ ciphertext pairs. We only pick up the pairs that have 12 inactive bytes at the ciphertext, thus we obtain $X \cdot 2^{63}/2^{96} = X \cdot 2^{-33}$ pairs.

For each of the $X \cdot 2^{-33}$ pairs, the attacker generates the wrong values of 9 key bytes; 4 bytes of K_0, 1 byte of $MC^{-1}(K_5)$ and 4 bytes of K_6 as illustrated in Fig. 6. This can be done by choosing all possible (2^8) 1-byte differences after the first round and propagate it back to the S-box output in round 1. Then each active S-box in round 1 has fixed input and output differences, which indicates the corresponding values for those 4 S-boxes. For each difference after round 1, the attacker obtains 1 value for those 4 S-boxes on average, thus obtains 1 candidate of 4 bytes of K_0 by taking the xor with plaintext. By analyzing 2^8 differences after round 1, the attacker collects 2^8 wrong keys. Similarly, by choosing a 1-byte difference at the input of round 5 and a 4-byte difference at the input of round 6,

the attacker collects 2^{40} wrong keys for the 5 key bytes. By merging the results from two directions, the attacker obtains 2^{48} wrong keys for 9 key bytes. By iterating the analysis for $X \cdot 2^{-33}$ pairs, the attacker obtains $X \cdot 2^{15}$ wrong keys for 9 key bytes. The remaining key space for those 9 bytes can be computed as follows.

$$2^{72} \cdot \left((1-2^{-96})^{X \cdot 2^{15}}\right) = 2^{72} \cdot \left((1-2^{-96})^{2^{96} \cdot X \cdot 2^{-81}}\right) \approx 2^{72} \cdot e^{-X \cdot 2^{-81}}.$$

Considering $e^{-64} \approx 2^{-92}$, by setting $X = 2^{87}$, the remaining key space becomes less than one, thus only the right key will remain. After 4 bytes of K_0 are recovered, the remaining 12 bytes can be recovered by the exhaustive search.

The attack complexity is $2^{87+32} = 2^{119}$ queries and memory accesses to collect the pairs. $2^{87-33+48} = 2^{102}$ partial AES round operations are required to compute wrong keys. To record the detected wrong keys, it requires memory of size 2^{72}.

Truncated Differential Attacks. So far the most successful attempts can break up to 5 rounds of TweAES. There are two possible approaches. The first approach does not inject the difference from the plaintext and starts the differential propagation from the first tweak injection. The second one is to inject the difference from the plaintext and to cancel it at the first tweak injection, which makes the subsequent two rounds blank. Here we describe both approaches.

The truncated differential trail for the first approach is shown in Fig. 7. The trail

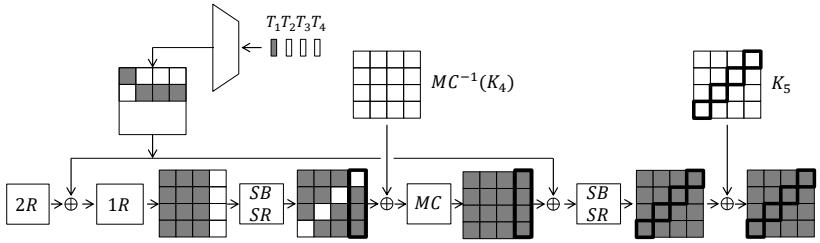

Figure 7: TweAES: 5-round Truncated Differential Attack using Tweak Difference (type 1).

can be satisfied with probability 1. After one pair of ciphertexts is obtained, the attacker analyzes the last subkey column by column. Namely, the possible number of differences before MixColumns in round 4 is 2^{24}. For each of them, the attacker can derive 1 candidate for the corresponding 4 subkey bytes of K_5, thus the key space is reduced by a factor of 2^8. The involved byte positions for 1 column are stressed in Fig. 7 by the bold line. The same analysis can be iterated by using 4 pairs of ciphertexts to reduce the key space to 1. The key for the other columns can be identified similarly. The data complexity is 2^4 paired queries, which is 2^5. Time complexity is 4 iterations of derivation of 2^{24} key candidates which is 2^{26}. The memory amount is 2^{24}.

One may wonder if it is possible to inject a difference to the plaintext and to cancel it with the first tweak addition. This is indeed possible and the key can be recovered up to 5 rounds, while it requires a much higher attack complexity. We will explain this inefficient attack to demonstrate that exploiting the plaintext to control the middle tweak injection is difficult. The truncated differential trail for the second approach is shown in Fig. 8. The trail can be satisfied with probability 2^{-128}; 2^{-64} for the first round and 2^{-64} towards the cancellation at the first tweak injection. Hence by generating 2^{128} pairs, we can expect one pair following the truncated differential trail.

The attacker makes $2^{64.5}$ encryption queries of randomly generated distinct plaintexts to pick up the pairs having 12 inactive bytes at the ciphertext in the byte positions shown

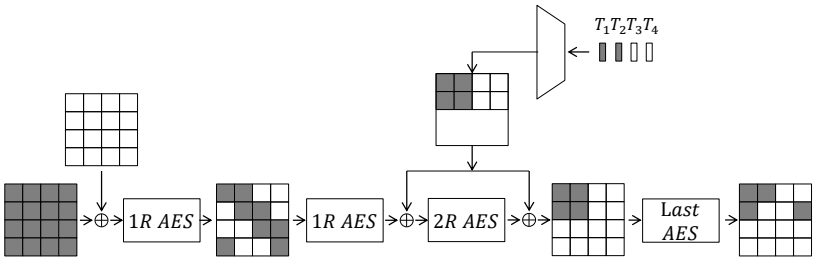

Figure 8: TweAES: 5-round Truncated Differential Attack using Tweak Difference (type 2).

in Fig. 8. Among about 2^{128} pairs, 2^{32} pairs will satisfy the 12 inactive bytes at the ciphertext and 1 pair is expected to follow the trail. For each of the 2^{32} pairs, the attacker generates 2^{64} candidate values for the first round key. Hence the 128-bit key space for the first subkey is reduce to 96 bits ($2^{32} \times 2^{64}$). By starting from $2^{66.5}$ queries to obtain 2^{132} pairs, the 128-bit key space is reduced to 1. The data complexity is $2^{66.5}$, the time complexity is 2^{98} and the memory complexity is 2^{96}.

We have tried various differential trails to attack 6 rounds of TweAES, while no attempts could successfully attack 6 rounds with a complexity significantly lower than the exhaustive key search. To find an attack on more than 5 rounds is an open problem.

Remarks on Meet-in-the-Middle Attacks. Meet-in-the-middle attacks [DS08, DFJ13] exploit the 4-round truncated differential $1 \to 4 \to 16 \to 4 \to 1$ 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 does not allow such sparse differential trails, which makes it hard to apply the meet-in-the-middle attacks.

3.4.3 Cryptanalysis Starting from the Middle of the 4-Round Core

We argue that any reduced-round version of TweAES where the first or the last round are located in the middle of any 4-round core can be attacked for relatively many rounds. Owing to this unusual setting, the attacks here do not threaten the security of full TweAES, however we still demonstrate the attacks for better understanding of the security of TweAES.

7-Round Boomerang/Sandwich Attacks. The first approach is the boomerang attack or the 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 differential for E_0 and E_1 almost independently. The attack detects a quartet of plaintexts generated by a plaintext x that satisfies the non-ideal behavior shown below with probability $p^{-2}q^{-2}$, where p and q are the differential probability for $E_0 : \alpha \to \beta$ and $E_1 : \gamma \to \delta$, 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 including four tweak injections that starts from the tweak injection are divided into E_0 and E_1 as follows.

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

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 shows 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. 9. The probability when E_m is a single S-box layer can be measured by using the boomerang connectivity table (BCT). The trails

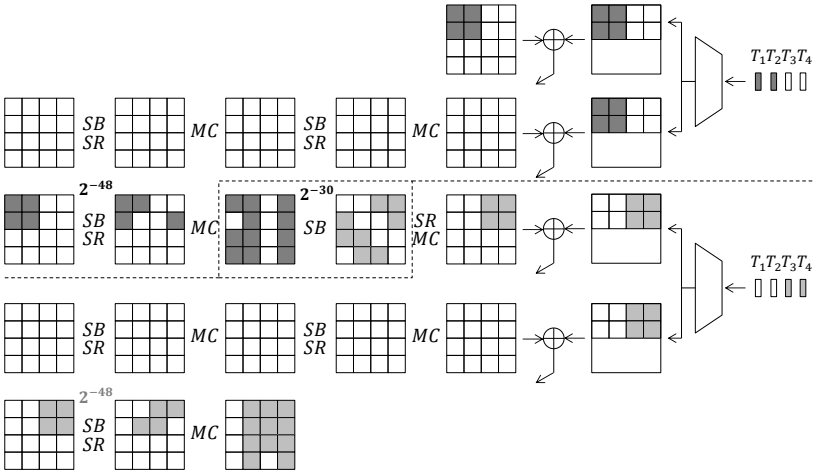

Figure 9: TweAES: Differential Trails for Boomerang Attacks. The cells filled with black and gray represent active byte positions in E_0 and E_1, respectively.

for E_0 and E_1 include 4 active S-boxes, hence the probabilities p and q are both equal to 2^{-24}. That is, $p^2q^2 = 2^{-96}$. The Boomerang Connectivity Table (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 $2^{-5.4}$. Then, the probability r_{qar} is $2^{-5.4\times5} = 2^{-27}$. In the end, $p^{-2}q^{-2}r_{qua} = 2^{-123}$, which would lead to a valid distinguisher for 7 rounds.

The data complexity is 2^{123} quartet queries, which is 2^{125}. The time complexity is 2^{125} memory accesses to the queried results. The memory amount is negligible.

8-Round Impossible Differential Attacks against TweAES. Due to 2 interval rounds between tweaks, distinguishers based on impossible differential attacks can be constructed for relatively many rounds (6 rounds) by canceling the tweak difference with the state difference. The distinguisher is depicted in Fig. 10.

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 forward direction has 4 active bytes for the right-most column, while
- 1-round propagation in backward direction has at least 2 inactive bytes in the right-most column.

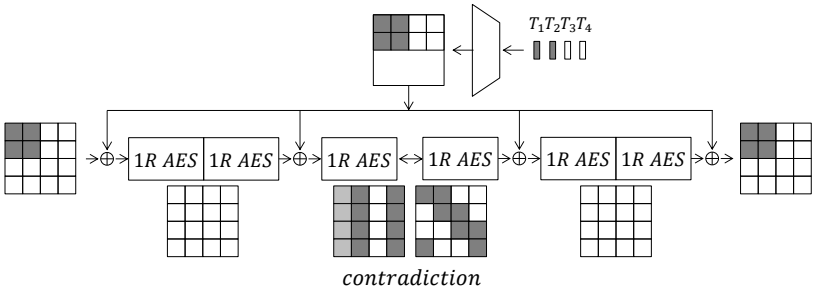

Figure 10: TweAES: 6-round Impossible Differential. The bytes filled with black, white, and gray have non-zero difference, zero difference, and arbitrary difference, respectively.

For the key recovery, two rounds can be appended to the 6 round impossible differential; one is at the beginning and the other is at the end, which is illustrated in Fig. 11. As

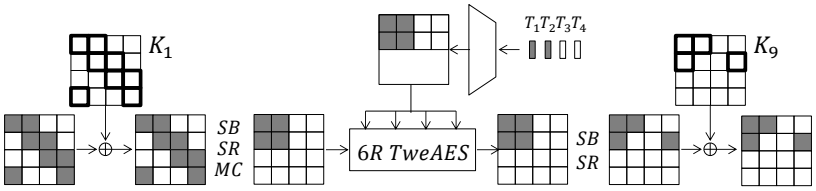

Figure 11: TweAES: Extension to 8-round Key Recovery

shown in Fig. 11 the trail includes 8 and 4 active bytes at the input and output states. Partial computations to the middle 6 rounds 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^{59} times from the first step by changing the inactive byte values at the input. After this step, $2^{39+59} = 2^{98}$ wrong-key candidates (including overlaps) will be obtained. The remaining key space of the involved 12 bytes becomes $2^{96} \times (1 - 2^{-96})^{2^{98}} \approx 2^{96} \times e^{-2} \approx 2^{90.2}$. Hence, the key space for the 8 bytes of K_1 and 4 bytes of K_9 will be reduced by a factor of $2^{5.77}$.

The data complexity is $2^4 \times 2^{64} \times 2^{59} = 2^{127}$. The time complexity is also 2^{127} memory accesses. The memory complexity is to recored the wrong keys of the 12 bytes, which is 2^{96}.

Remarks. We demonstrated two attacks against reduced-round variants that start from the middle of the 4-round core. Because the security of TweAES with non-zero tweak difference relies on the property 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. From a different viewpoint, one can see the difficulty to extend the analysis by 1 more round from Figs. 9 and 11. The number of guessed key bytes will reach 16.

3.4.4 Comparison of the Attacks on AES and TweAES Exploiting Tweak

The comparison of the number of attacked rounds and the attack complexity for the original AES and TweAES is given in Table 2. With our best effort, the number of attacked rounds for TweAES with a non-zero difference is always smaller than the attacks on AES. It is an open problem to find attacks on TweAES reaching the same number of rounds as AES.

Table 2: Comparison of the attacks on AES and TweAES exploiting tweak in Sect. 3.4.2. R, D, T and M denote the number of rounds, data complexity, time complexity and memory complexity, respectively.

Attacks Starting from the First Round

Attack	AES					TweAES			
	R	D	T	M	ref.	R	D	T	M
Integral	7	$2^{128}-2^{119}$	2^{120}	2^{64}	[FKL+00]	6	2^5	2^{45}	$negl.$
Imp. Diff.	7	$2^{106.2}$	$2^{110.2}$	$2^{90.2}$	[MDRM10]	6	2^{119}	2^{119}	2^{72}
Trunc. Diff.	6	$2^{72.8}$	2^{105}	2^{33}	[Gra19]	5	2^5	2^{26}	2^{24}

Attacks Starting from the Middle of the 4-Round Core

Boomerang	not available			7	2^{125}	2^{125}	$negl.$
Imp. Diff.	not available			8	2^{127}	2^{127}	2^{96}

3.5 Security Analysis of TweAES-6

In TweAES-6, the number of rounds is reduced from TweAES 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-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. Examples of the differential trails achieving 20 active S-boxes is the first six or the last six rounds of the trail in Fig. 3.

Given that the maximum differential probability of the AES 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 differential cryptanalysis.

Note that AEAD schemes based on the original AES often adopt 4-round AES in the mode. While 4-round AES is not fully secure as a standalone block cipher, no attack is known on the 4-round AES in proper modes under the restriction of the birthday-bound query limit. We designed TweAES-6 by following the same concept to offer a speed-up from the full-round TweAES in our modes.

3.6 Security Analysis of TweGIFT

We only consider the security of TweGIFT against attacks exploiting the tweak injection, because, without the tweak injection, the security of TweGIFT is exactly the same as the original GIFT-128.

Differential Cryptanalysis. The 4-bit tweak expands to 8 bits and those 8 bits are duplicated 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.

We modeled the differential trail search for TweGIFT with MILP under the constraints that at least 1 bit of the tweak has a difference. Owing to the large state size, the computation of the tight bound of the maximum probability of the differential characteristic is not easy even for the 10-round core. After spending 1,462,448 seconds (about 17 days), the MILP stopped and it turned out that the maximum probability of the differential characteristic for the 10-round core is 2^{-79}. Given that the entire TweGIFT-128 consists of 40 rounds and thus contains 4 of the 10-round cores, the upper bound of the entire construction is $2^{-79\times 4} = 2^{-316}$, which is sufficient to resist the attack.

Note that it is also difficult to apply the MILP-based differential trail search to the original GIFT-128 because of the large state size. The designers showed that the lower bound on the number of active S-boxes for 9 rounds of GIFT-128 is 19 [BPP+17, Table 11] and the bound is tight. The designers also evaluated the differential probability (not characteristic probability) of the trail matching the bound, which was $2^{-46.99}$. Zhu et al. [ZDY19] introduced some heuristic to search for differential trails of the reduced-round GIFT-128 with some aid of MILP. They found 12-, 14-, 18-round differential characteristics with probability $2^{-62.415}$, 2^{-85}, and 2^{-109}, respectively [ZDY19, Table 9]. By comparing those probabilities with the upper bound for the 10-round core, we believe that the best differential trail would not exploit the tweak difference, thus the tweak injection of TweAES does not introduce any vulnerability. The comparison of the bounds for the original GIFT-128 and TweGIFT is given in Table 3.

Table 3: Comparison of the Guaranteed Differential Property for GIFT-128 and TweGIFT via Non-Zero Tweak

target	rounds	evaluated object	bound type	probability	reference
GIFT-128	9	differential probability	tight bound	$2^{-46.99}$	[BPP+17]
GIFT-128	12	characteristic probability	lower bound	$2^{-62.415}$	[ZDY19]
GIFT-128	14	characteristic probability	lower bound	2^{-85}	[ZDY19]
GIFT-128	18	characteristic probability	lower bound	2^{-109}	[ZDY19]
TweGIFT	10	characteristic probability	tight bound	2^{-79}	Ours

Basically, GIFT-128 allows a sparse differential propagation. For example, the 18-round differential trail found by Zhu et al. [ZDY19] is described in Table 4.

The differential mask for the first and last rounds in Table 4 have a relatively large weight, however this is because the trail is optimized for 18 rounds. The sparse differential propagation of GIFT-128 is the ground of our belief that to have 16 active S-boxes around the tweak injection by using non-zero tweak difference strongly resists the attack.

Boomerang Attacks. If the number of attacked rounds is reduced significantly, the tweak injection actually helps an attacker to attack TweGIFT more efficiently than the original GIFT-128. An example is the boomerang attack for 10 rounds. If the attacker starts from

Table 4: 18-Round Sparse Differential Trail by Zhu et al. [ZDY19, Table 10]

Round	Input Difference								Probability
	0000	0000	7060	0000	0000	0000	0000	0000	
1	0000	0000	0000	0000	0000	0000	00a0	0000	2^{-5}
2	0000	0010	0000	0000	0000	0000	0000	0000	2^{-7}
3	0000	0000	0800	0000	0000	0000	0000	0000	2^{-10}
4	0020	0000	0010	0000	0000	0000	0000	0000	2^{-12}
5	0000	0000	0000	0000	4040	0000	2020	0000	2^{-17}
6	0000	5050	0000	0000	0000	5050	0000	0000	2^{-25}
7	0000	0000	0000	0000	0000	0000	0a00	0a00	2^{-37}
8	0000	0000	0000	0011	0000	0000	0000	0000	2^{-41}
9	0008	0000	0008	0000	0000	0000	0000	0000	2^{-57}
10	0000	0000	0000	0000	2020	0000	1010	0000	2^{-41}
11	0000	5050	0000	0000	0000	5050	0000	0000	2^{-61}
12	0000	0000	0a00	0a00	0000	0000	0000	0000	2^{-73}
13	0000	0000	0011	0000	0000	0000	0000	0000	2^{-77}
14	0090	0000	00c0	0000	0000	0000	0000	0000	2^{-83}
15	1000	0000	0080	0000	0000	0000	0000	0000	2^{-89}
16	0010	0000	0000	0000	0000	0000	8020	0000	2^{-94}
17	0000	0000	8000	0020	0000	0050	0000	0020	2^{-101}
18	0000	0100	0020	0800	0014	0404	0002	0202	2^{-109}

the zero plaintext difference with some non-zero tweak difference, the first 5 rounds do not have any difference. The tweak injection will introduce differences to multiple S-boxes, but we change the trail by following the framework of the boomerang attack. In the second trail that starts from round 6, we also choose the zero-difference to the state input, and some non-zero difference in the tweak. This also gives another 5 empty rounds. In total, we have two 5-round trails with probability 1, that easily enables attackers to attack 10 rounds plus a few more rounds by appending some key-recovery rounds. It would also be possible to extend a few more rounds at the border of the two trails by using the BCT [CHP+18].

In the original GIFT-128, the minimum number of active S-boxes for 5 rounds is 5. Hence, the 10-round boomerang trail will certainly require a non-negligible data complexity to recovery the key. The 10-round attack against TweGIFT should be much more efficient than the one against original GIFT-128.

However, because the probability of the trails is squared in the boomerang attack, it is highly unlikely that the attacker can extend the differential trail significantly. Moreover, recall that the probability of the differential characteristic is upper bounded by $2^{-72.6}$ for the 10-round core. The squared probability is $2^{-145.2}$, which is already larger than the code-book size. The boomerang attack may work efficiently for 10 and a few more rounds of TweGIFT, but given that the differential trail in Table 4 reaches 18 rounds, we do not think that the boomerang attack can be the best approach for attacking TweGIFT.

4 ESTATE: A tBC-Based Lightweight AEAD Mode

In this section, we present the formal specification of ESTATE mode of operation based on tBC. A detailed algorithmic description for the mode is given. Finally, we list the recommended instantiations. ESTATE_TweAES, sESTATE_TweAES-6 and ESTATE_TweGIFT. We use the tBCs TweAES and TweGIFT, used for listed instantiations.

4.1 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 $A \in \{0,1\}^{\leq 2^{n/2}}$, and a message $M \in \{0,1\}^{\leq 2^{n/2}}$ 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 $A \in \{0,1\}^{\leq 2^{n/2}}$, a ciphertext $C \in \{0,1\}^{\leq 2^{n/2}}$, 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 which is already larger than-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. The complete specification of ESTATE is presented in Algorithm 1. The pictorial description is given in Figure 12, 13, and 14.

4.1.1 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. Different tweak values are used to separate different cases based on the length of the associated data and the message. The tweak values are represented in 4 bits, and the corresponding integer value of the 4-bit binary representation is called the tweak value. We use the tweak value 1 while processing the first block (i.e. nonce N). All the intermediate blocks are processed with tweak 0, to minimize the overhead.

4.1.2 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

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

function MAC$[\widetilde{\mathsf{E}}](K, N, A, M)$
  if $|A| = 0$ and $|M| = 0$ **then**
    return $T \leftarrow \widetilde{\mathsf{E}}^8_K(N)$
  $T \leftarrow \widetilde{\mathsf{E}}^1_K(N)$
  if $|A| > 0$ **then**
    $A[1]\| \cdots \|A[a] \leftarrow A$
    $t \leftarrow (|M| > 0\ ;\ |A[a]| = n)\ ?\ 2 : 3 : 6 : 7$
    $T \leftarrow \mathsf{FCBC}^\star[\widetilde{\mathsf{E}}](K, T, A, t)$
  if $|M| > 0$ **then**
    $M[1]\| \cdots \|M[m] \leftarrow M$
    $t \leftarrow (|M[m]| = n)?\ 4 : 5$
    $T \leftarrow \mathsf{FCBC}^\star[\widetilde{\mathsf{E}}](K, T, M, t)$
  return T

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

function FCBC$^\star[\widetilde{\mathsf{E}}](K, T, D, t)$
  $D[1]\| \cdots \|D[d] \leftarrow D$
  for $i = 1$ **to** $d - 1$ **do**
    $T \leftarrow \widetilde{\mathsf{E}}^0_K(T \oplus D[i])$
  $T \leftarrow \widetilde{\mathsf{E}}^t_K\big(T \oplus \mathsf{ozp}(D[d])\big)$
  return T

function OFB$[\widetilde{\mathsf{E}}](K, T, M)$
  $M[1]\| \cdots \|M[m] \leftarrow M$
  for $i = 1$ **to** m **do**
    $T \leftarrow \widetilde{\mathsf{E}}^0_K(T)$
    $C[i] \leftarrow \mathsf{chop}(T, |M[i]|) \oplus M[i]$
  return $(C[1]\| \cdots \|C[m])$

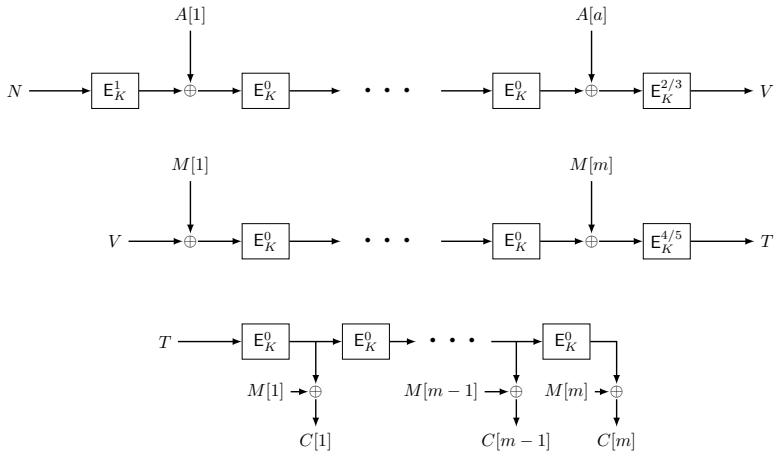

Figure 12: ESTATE with a AD blocks and m message blocks

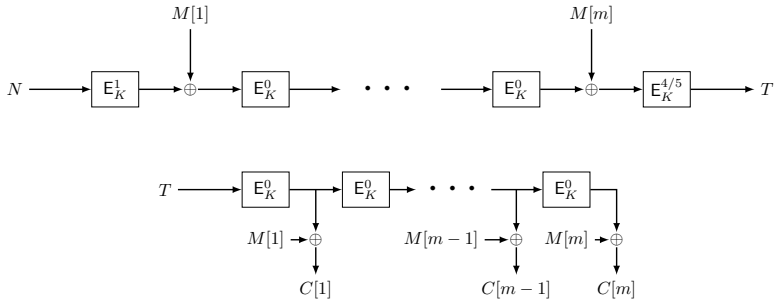

Figure 13: ESTATE with empty AD and m message blocks

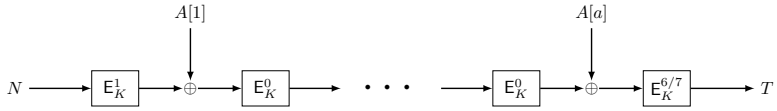

Figure 14: ESTATE with a AD blocks and empty message

4.2 sESTATE: A Lighter Variant of ESTATE

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 all the blocks in the MAC function except the last one. For all other tweakable block cipher calls, i.e. for processing the last block in the 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 the maximum distance between the 0 tweak calls to $\widetilde{\mathsf{E}}$ and calls to $\widetilde{\mathsf{F}}$.

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

function sESTATE.Enc$[\widetilde{\mathsf{E}},\widetilde{\mathsf{F}}](K, N, A, M)$
  $T \leftarrow \mathsf{MAC}[\widetilde{\mathsf{E}}, \widetilde{\mathsf{F}}](K, N, A, M)$
  $C \leftarrow \mathsf{OFB}[\widetilde{\mathsf{E}}](K, T, M)$
  return (C, T)

function MAC$[\widetilde{\mathsf{E}},\widetilde{\mathsf{F}}](K, N, A, M)$
  if $|A| = 0$ and $|M| = 0$ **then**
    return $T \leftarrow \widetilde{\mathsf{E}}^{8}_{K}(N)$
  $T \leftarrow \widetilde{\mathsf{F}}^{15}_{K}(N)$
  if $|A| > 0$ **then**
    $A[1]\| \cdots \|A[a] \leftarrow A$
    $t \leftarrow (|M| > 0 \; ; \; |A[a]| = n) \; ? \; 2 : 3 : 6 : 7$
    $T \leftarrow \mathsf{FCBC}^{\star}[\widetilde{\mathsf{E}}, \widetilde{\mathsf{F}}](K, T, A, t)$
  if $|M| > 0$ **then**
    $M[1]\| \cdots \|M[m] \leftarrow M$
    $t \leftarrow (|M[m]| = n)? \; 4 : 5$
    $T \leftarrow \mathsf{FCBC}^{\star}[\widetilde{\mathsf{E}}, \widetilde{\mathsf{F}}](K, T, M, t)$
  return T

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

function FCBC$^{\star}[\widetilde{\mathsf{E}},\widetilde{\mathsf{F}}](K, T, D, t)$
  $D[1]\| \cdots \|D[d] \leftarrow D$
  for $i = 1$ **to** $d - 1$ **do**
    $T \leftarrow \widetilde{\mathsf{F}}^{15}_{K}(T \oplus D[i])$
  $T \leftarrow \widetilde{\mathsf{E}}^{t}_{K}\left(T \oplus \mathsf{ozp}(D[d])\right)$
  return T

function OFB$[\widetilde{\mathsf{E}}](K, T, M)$
  $M[1]\| \cdots \|M[m] \leftarrow M$
  for $i = 1$ **to** m **do**
    $T \leftarrow \widetilde{\mathsf{E}}^{0}_{K}(T)$
    $C_i \leftarrow \mathsf{chop}(T, |M[i]|) \oplus M[i]$
  return $(C[1]\| \cdots \|C[m])$

4.2.1 Tweak Choices

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-6 with tweak value 15 and TweAES with tweak value 0 as much independent as possible.

4.3 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 a tweakable block cipher with 4-bit tweak primarily for the purpose of domain separation for 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. **Rationale 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 and TweGIFT with zero tweaks are AES 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 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.

4. **Rationale of the Tweak Injection Positions for TweAES.** The overall structure of TweAES is similar as KIASU-BC [JNP14], which takes a 64-bit tweak as input and adds it to the two rows of the state in every rounds. The designers of KIASU-BC pointed out that if the injection position is two columns, it immediately leads to an efficient related-key related-tweak attack. This is also the reason for the designers of KIASU-BC for not supporting a 128-bit and a 96-bit tweak. The proposed analysis is reasonable and we follow a similar analysis in the design of TweAES, i.e. to inject the 8-bit expanded tweak to the LSB of each byte in the top rows. Bit position inside the byte can be different, however we determined to inject only to the rightmost part for implementation.

   We also took into account the fact that several researchers [DEM16, TAY16, DL17, LSG+19] pointed out that many of the attack approaches on AES were extended by 1 more round when they were applied to KIASU-BC. This is mainly caused by the fact that the same tweak is injected in every round and the expanded tweak can be directly controlled by the attacker at least for one round. In TweAES, the expansion by computing the linear code makes it difficult for the attackers to control the value of the expanded tweak, and the injection in every few rounds does not allow any single-round iterative characteristic.

4.4 Recommended Instantiations

We recommend the following concrete instantiations:

- ESTATE_TweAES: This AEAD scheme obtained by instantiating the ESTATE mode of operation with $\widetilde{\mathsf{E}}$:=TweAES. Here the size of the key, nonce and tag are 128 bits each.
- ESTATE_TweGIFT: This AEAD scheme is obtained by instantiating the ESTATE mode of operation with $\widetilde{\mathsf{E}}$:= TweGIFT-128. Here the size of the key, nonce and tag are 128 bits each. We recommend ESTATE_TweGIFT, for hardware-oriented ultra-lightweight applications.
- sESTATE_TweAES-6: This AEAD scheme is obtained by instantiating the sESTATE mode of operation with $\widetilde{\mathsf{E}}$:=TweAES, $\widetilde{\mathsf{F}}$:=TweAES-6, such that $\widetilde{\mathsf{F}}$ is the 6-round version of TweAES. Again, the size of the key, nonce and tag are 128 bits each. Notably, the last round of TweAES-6 (6-th round) includes the MixColumns operations, and the tweaks are added in the 2-nd and 4-th rounds. We recommend sESTATE_TweAES-6, for high-throughput and energy-constrained applications.

5 Security of ESTATE

In this section, we prove that ESTATE is an AERUP secure authenticated encryption scheme:

Theorem 2 (AERUP security of ESTATE)**.** *Consider ESTATE authenticated encryption scheme based on the tweakable block cipher* $E : \{0,1\}^k \times \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^n$*. For any adversary* $\mathcal{A}$ *having encryption complexity* σ_e*, decryption complexity* σ_d*, and verification complexity* σ_v *(number of queried blocks in the* q_e *encryption* q_d *decryption and* q_v *verification queries, respectively), and operating in time* t*,*

$$\mathbf{Adv}^{\mathsf{AERUP}}_{\mathit{ESTATE}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{TPRP}}_{E}(\mathcal{B}) + \frac{\sigma^2}{2^n} + \frac{q_v}{2^n}\,,$$

where $\mathcal{B}$ *is some* TPRP *adversary that makes* $\sigma = \sigma_e + \sigma_d + \sigma_v$ *queries to its oracle.*

We consider an adversary $\mathcal{A}$ that has access to either $(\mathcal{E}_K, \mathcal{D}_K, \mathcal{V}_K)$ or $(\$, \mathsf{S}, \perp)$, and tries to distinguish both worlds. We first replace $E^0_K, \ldots, E^7_K$ by random permutations $P_0, \ldots, P_7$, where each $P_i \xleftarrow{\$} \mathrm{P}(n)$, at the cost of $\mathbf{Adv}^{\mathsf{TPRP}}_{E}(\mathcal{B})$ for some distinguisher $\mathcal{B}$ that makes σ queries to its oracle and operates in time $t' \approx t$. Next, we switch from $P_0, \ldots, P_7$ to random functions $R_0, \ldots, R_7$ where $R_i \xleftarrow{\$} \mathrm{F}(n)$ at the cost of $\binom{\sigma}{2}/2^n$ (applying the standard PRP-PRF switching lemma). For brevity, denote the resulting construction by $\Pi = (\mathcal{E}[R_0, \ldots, R_7], \mathcal{D}[R_0, \ldots, R_7], \mathcal{V}[R_0, \ldots, R_7])$. We have thus obtained

$$\mathbf{Adv}^{\mathsf{AERUP}}_{\mathsf{ESTATE}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{TPRP}}_{E}(\mathcal{B}) + \binom{\sigma}{2}/2^n + \mathbf{Adv}^{\mathsf{AERUP}}_{\Pi}(\mathcal{A})\,, \tag{4}$$

and our focus is on upper bounding the remaining distance $\mathbf{Adv}^{\mathsf{AERUP}}_{\Pi}(\mathcal{A})$. The theorem follows as we bound $\mathbf{Adv}^{\mathsf{AERUP}}_{\Pi}(\mathcal{A}) \leq \frac{\sigma^2}{2^n} + \frac{q_v}{2^n}$ in the following subsection.

5.1 Bounding $\mathbf{Adv}^{\mathsf{AERUP}}_{\Pi}(\mathcal{A})$

Without loss of generality, $\mathcal{A}$ is deterministic. Suppose it makes q_e encryption queries $(A^+_i, M^+_i)^{q_e}_{i=1}$ to the encryption oracle, where the block lengths of A^+_i and M^+_i are denoted by a^+_i and m^+_i, with an aggregate of total σ_e blocks, q_d decryption queries $(A^-_i, C^-_i, T^-_i)^{q_d}_{i=1}$ to the decryption oracle, where the block lengths of A^-_i and C^-_i are denoted by a^-_i and c^-_i, with an aggregate of total σ_d blocks, and q_v verification queries $(A^\star_i, C^\star_i, T^\star_i)^{q_v}_{i=1}$ to the verification oracle, where the block lengths of $A^\star_i$ and $C^\star_i$ are denoted by $a^\star_i$ and $c^\star_i$, with an aggregate of total σ_v blocks. We assume that $\mathcal{A}$ is non-trivial and non-repeating, which means that all queries are distinct and there is no $(A^\star_i, C^\star_i, T^\star_i)$ that is an answer of an earlier encryption query. By $(i, \circledcirc)$, we mean the i-th message of type $\circledcirc$, where $\circledcirc \in \{+, -, \star\}$. We use the notation $(j, \circledcirc) \prec (i, \circledast)$ to denote that j-th message of type $\circledcirc$ was queried prior to the i-th message of type $\circledast$.

Description of the Real World. The real world $\mathcal{O}_{\mathrm{re}}$ consists of the encryption oracle $\Pi.\mathcal{E}[R]$, the decryption oracle $\Pi.\mathcal{D}[R]$, and the verification oracle $\Pi.\mathcal{V}[R]$ as outlined above. After the adversary has made all its queries, the oracles release all the internal variables. The encryption and verification oracles reveal all (X, Y)'s (block cipher input-outputs corresponding to authentication part) and all (U, V)'s (block cipher input-outputs corresponding to OFB part). The decryption oracle reveals all (U, V)'s corresponding to decryption (the oracle does not verify the MAC). Note that there is some redundancy in the values, as the U's can be deduced from the values M, C, and V, but we reveal these for completeness.

Description of the Ideal World. The ideal world $\mathcal{O}_{\mathrm{id}}$ consists of three oracles $(\$, \mathsf{S}, \perp)$. The verification oracle $\perp$ simply responds with the $\perp$-sign for each input $(A^\star_i, C^\star_i, T^\star_i)$. We will elaborate on the remaining two oracles, encryption $\$$ and decryption S, in detail. For these two oracles, we maintain an initially empty table $\mathcal{L}$ to store (U, V)-tuples.

The encryption oracle \$ is a random function that for each input $(A_i^+, M_i^+) = (A_i^+[1 \dots a_i^+], M_i^+[1 \dots m_i^+])$ generates a ciphertext and tag as

$$C_i^+ = C_i^+[1 \dots m_i^+] \xleftarrow{\$} \{0,1\}^{|M_i^+|}\,,$$
$$T_i^+ \xleftarrow{\$} \{0,1\}^n\,.$$

For later purposes, \$ will *in addition* set the following internal variables, which correspond to the inputs and outputs of R that are determined by M_i^+, C_i^+, T_i^+:

$$\left(U_i^+[k]\,,\, V_i^+[k]\right) \leftarrow \begin{cases} \left(T_i^+\,, \quad M_i^+[1] \oplus C_i^+[1]\right)\,, \text{ for } k = 1\,, \\ \left(V_i^+[k-1]\,,\, M_i^+[k] \oplus C_i^+[k]\right)\,, \text{ for } k = 2, \dots, m_i^+\,. \end{cases}$$

It stores all the individual (U_i^+, V_i^+) tuples in table $\mathcal{L}$. The decryption oracle S is a simulator that we define to operate as follows on input of a query $(A_i^-, C_i^-, T_i^-) = (A_i^-[1, \dots, a_i^-], C_i^-[1, \dots, c_i^-], T_i^-)$:

- Sets $k \leftarrow 1$ and $U_i^-[1] \leftarrow T_i^-$
- While $U_i^-[k] \in \mathcal{L}$, sets $V_i^-[k] \leftarrow \mathcal{L}(U_i^-[k])$, defines $M_i^-[k] \leftarrow V_i^-[k] \oplus C_i^-[k]$ and $U_i^-[k+1] \leftarrow V_i^-[k]$ and increment k by 1.
- For $j = k$ **to** c_i^-, samples $M_i^-[j] \xleftarrow{\$} \{0,1\}^n$, sets $V_i^-[j] \leftarrow M_i^-[j] \oplus C_i^-[j]$, $U_i^-[j] \leftarrow V_i^-[j-1]$ and adds $(U_i^-[j], V_i^-[j])$ to $\mathcal{L}$.
- Finally returns $M_i^-[1 \dots c_i^-]$

Once the adversary has made all queries, we move to an *offline* phase where the adversary will be given the internal values (X, Y) and (U, V), just like in the real world. Note that the (U, V)'s have already been defined for encryption and decryption oracle. For any input query $(A_i^\star, C_i^\star, T_i^\star)$, verification oracle $\perp$ defines (U, V) in exactly the similar way as the decryption oracle defines for an input query (A_i^-, C_i^-, T_i^-) and also determines the underlying message $M_i^\star[1 \dots c_i^\star]$ which is released to the adversary. For the (X, Y)'s we use the following technique to define them. Note that we only have to focus on the encryption and verification queries; we do not bother about the (X, Y)'s for decryption queries as a decryption call does not verify the tag. For any query $(i, \circledcirc)$ with $\circledcirc \in \{+, \star\}$, we first find the query $(j, \circledast)$ which has the longest common prefix with $(i, \circledcirc)$. Let $p < \ell_i^{\circledcirc}$ be the length of the longest common prefix of $(A_i^{\circledcirc} \| M_i^{\circledcirc})$ and $(A_j^{\circledast} \| M_j^{\circledast})$. Next, we set $Y_i^{\circledcirc}[k] \leftarrow Y_j^{\circledast}[k]$ for $1 \le k \le p$, and $Y_i^{\circledcirc}[k] \xleftarrow{\$} \{0,1\}^n$, for $p+1 \le k \le \ell_i^{\circledcirc}$. Finally, we set all the $X_i^{\circledcirc}[j]$ values for $j = 1, \dots, \ell_i^{\circledcirc}$. Finally, when the sampling of internal values is over, $\mathcal{O}_{\mathrm{id}}$ returns all the internal values. These are $(X_i^+, Y_i^+) = (X_i^+[1 \dots \ell_i^+], Y_i^+[1 \dots \ell_i^+])$, $(U_i^+, V_i^+) = (U_i^+[1 \dots m_i^+], V_i^+[1 \dots m_i^+])$, for each encryption query $(A_i^+, M_i^+, C_i^+, T_i^+)$; $(U_i^-, V_i^-) = (U_i^-[1 \dots c_i^-], Y_i^-[1 \dots c_i^-])$, for each decryption query $(A_i^-, M_i^-, C_i^-, T_i^-)$, and $(X_i^\star, Y_i^\star) = (X_i^\star[1 \dots \ell_i^\star], Y_i^\star[1 \dots \ell_i^\star])\,, (U_i^\star, V_i^\star) = (U_i^\star[1 \dots m_i^\star], V_i^\star[1 \dots m_i^\star])$, for each verification query $(A_i^\star, M_i^\star, C_i^\star, T_i^\star, b_i^\star)$.

Attainable Transcripts. The overall transcript of the attack is $\tau = (\tau_e, \tau_d, \tau_v)$, where

$$\tau_e = (A_i^+, M_i^+, C_i^+, T_i^+, X_i^+, Y_i^+, U_i^+, V_i^+)_{i=1}^{q_e}\,,$$
$$\tau_d = (A_i^-, M_i^-, C_i^-, T_i^-, U_i^-, V_i^-)_{i=1}^{q_d}\,,$$
$$\tau_v = (A_i^\star, M_i^\star, C_i^\star, T_i^\star, X_i^\star, Y_i^\star, U_i^\star, V_i^\star, b_i^\star)_{i=1}^{q_v}\,.$$

A transcript $\tau = (\tau_e, \tau_d, \tau_v)$ is said to be *attainable* (with respect to $\mathcal{A}$) if the probability to realize this transcript in the ideal world $\mathcal{O}_{\mathrm{id}}$ is non-zero. Note that, particularly, for an attainable transcript τ, any verification query in τ_v satisfies $b_i^\star = \perp$. Following Sect. 2.4,

we denote by Θ the set of all attainable transcripts, and by X_{re} and X_{id} the probability distributions of transcript τ induced by the real world and ideal world, respectively.

Definition of Bad Transcripts We say that an attainable transcript τ is *bad* if one of the following events hold:

1. $\mathsf{Acc_{XX1}}$: $\exists(j,\circledast) \preceq (i,\circledcirc) : X_i^{\circledcirc}[a_i^{\circledcirc}] = X_j^{\circledast}[a_j^{\circledast}]$, where $A_i^{\circledcirc} \neq A_j^{\circledast}$.
2. $\mathsf{Acc_{XX2}}$: $\exists(j,\circledast) \preceq (i,\circledcirc) : X_i^{\circledcirc}[\ell_i^{\circledcirc}] = X_j^{\circledast}[\ell_j^{\circledast}]$.
3. $\mathsf{Acc_{XX3}}$: $\exists(j,\circledast) \preceq (i,\circledcirc), k, k'(\neq k) : X_i^{\circledcirc}[k] = X_j^{\circledast}[k']$.
4. $\mathsf{Acc_{XX4}}$: $\exists(j,\circledast) \preceq (i,\circledcirc), k \leq a_i^{\circledcirc} : X_i^{\circledcirc}[k] = X_j^{\circledast}[k]$, where $A_i^{\circledcirc}[1\dots k] \neq A_j^{\circledast}[1\dots k]$.
5. $\mathsf{Acc_{XX5}}$: $\exists(j,\circledast) \preceq (i,\circledcirc), k > a_i^{\circledcirc} : X_i^{\circledcirc}[k] = X_j^{\circledast}[k]$, where $A_i^{\circledcirc} = A_j^{\circledast}, M_i^{\circledcirc}[1\dots(k - a^{\circledcirc})] \neq M_j^{\circledast}[1\dots(k-a^{\circledcirc})]$.
6. $\mathsf{Acc_{XU}}$: $\exists(j,\circledast),(i,\circledcirc), k(\neq 1,\ell_i^{\circledcirc}), k'$ such that $U_i^{\circledcirc}[k'] = X_j^{\circledast}[k]$.
7. $\mathsf{Acc_{UU}}$: $\exists(j,\circledast) \preceq (i,\circledcirc), k, k'$ with $\big(\circledcirc = +$ or $U_i^{\circledcirc}[1] \neq U_j^{\circledast}[k-k'+1]\big)$ such that $U_i^{\circledcirc}[k'] = U_j^{\circledast}[k]$.
8. Forge: $\exists(i,\star)$ such that $Y_i^{\star}[\ell_i^{\star}] = T_i^{\star}$.

Note that, considering the real world, $\mathsf{Acc_{XX}}$ denotes the event of an accidental collision between two inputs to R in the authentication part, where we exclude trivial collisions due to common prefix. Event $\mathsf{Acc_{XU}}$ corresponds to accidental collisions between an input to R in the authentication and one in the encryption part. Event $\mathsf{Acc_{UU}}$ corresponds to accidental collisions between two inputs to R in the encryption part, where we exclude trivial collisions triggered by a decryption query for a known U-value. Event Forge corresponds to the event that for any verification query, the last block cipher output in the MAC function collides with the given tag in the verification query.

In line with the H-coefficient technique (Theorem 1), Θ_{bad} denotes the set of all attainable transcripts that are bad.

Probability of Bad Transcripts. We now bound the probability of a bad event in the ideal world.

Lemma 1. *Let X_{id} and Θ_{bad} be as defined as above. Then,*

$$\mathsf{Pr}[X_{\text{id}} \in \Theta_{\text{bad}}] \leq \binom{\sigma}{2} \cdot \frac{1}{2^n} + \frac{q_v}{2^n}\,.$$

Proof. By applying the union bound,

$$\mathsf{Pr}[X_{\text{id}} \in \Theta_{\text{bad}}] \leq \mathsf{Pr}[\mathsf{Acc_{XX}}] + \mathsf{Pr}[\mathsf{Acc_{XU}}] + \mathsf{Pr}[\mathsf{Acc_{UU}}] + \mathsf{Pr}[\mathsf{Forge}]\,,$$

and we bound the three probabilities individually. We let $\#X$ be the number of X's in the transcript and $\#U$ the number of U's.

Bounding $\mathsf{Acc_{XX}}$. For all the first four cases, the probability of each case can be bounded by $\frac{1}{2^n}$ due to the random sampling of $Y_j^{\circledast}[k-1]$. Combining all the four cases, we obtain

$$\mathsf{Pr}[\mathsf{Acc_{XX}}] \leq \binom{\#X}{2} \cdot \frac{1}{2^n}\,.$$

Bounding $\mathsf{Acc}_{\mathsf{XU}}$. The event implies $C_i^{\odot}[k'] \oplus M_i^{\odot}[k'] = Y_j^{\circledast}[k-1] \oplus A_j^{\circledast}[k]$. If $(j, \circledast) \prec (i, \odot)$, we can bound this event by $\frac{1}{2^n}$ due to the random sampling of $C_i^{\odot}[k']$ or $M_i^{\odot}[k']$ or $Y_j^{\circledast}[k-1]$. We therefore obtain

$$\mathsf{Pr}[\mathsf{Acc}_{\mathsf{XU}}] \leq (\#X \cdot \#U) \cdot \frac{1}{2^n} .$$

Bounding $\mathsf{Acc}_{\mathsf{UU}}$. We consider the following cases:
We obtain

$$\mathsf{Pr}[\mathsf{Acc}_{\mathsf{UU}}] \leq \binom{\#U}{2} \cdot \frac{1}{2^n} .$$

Bounding Forge. For a fixed verification query, the event is trivially bounded by 2^{-n} as $Y_i^{\star}[\ell^{\star}]$ is sampled uniformly at random. Summing over all possible choices of the index i, we have

$$\mathsf{Pr}[\mathsf{Forge}] \leq q_v/2^n.$$

Conclusion. We obtain that

$$\mathsf{Pr}[X_{\mathrm{id}} \in \Theta_{\mathrm{bad}}] \leq \left(\binom{\#X}{2} + (\#X \cdot \#U) + \binom{\#U}{2} \right) \cdot \frac{1}{2^n} .$$

This completes the proof, noting that

$$\binom{\#X}{2} + (\#X \cdot \#U) + \binom{\#U}{2} = \binom{\#X + \#U}{2} \leq \binom{\sigma}{2},$$

and in addition $\#U \leq \sigma$. □

Analysis of Good Transcripts. In this section we show that for a good transcript τ, realizing τ is almost as likely in the real world as in the ideal world. Formally, we prove the following lemma.

Lemma 2. *Let X_{re}, X_{id}, and Θ_{bad} be as defined as above. For any good transcript $\tau = (\tau_e, \tau_v, \tau_d) \in \Theta \backslash \Theta_{\mathrm{bad}}$,*

$$\frac{\mathsf{Pr}[X_{\mathrm{re}} = \tau]}{\mathsf{Pr}[X_{\mathrm{id}} = \tau]} = 1 .$$

Proof. Let $\tau = (\tau_e, \tau_v, \tau_d)$ be a good transcript. Let s_e be the number of distinct X values in $\mathbf{X}^{+} := (X_1^{+}, \ldots, X_{q_e}^{+})$ tuple and s_v be the number of distinct X values in $\mathbf{X}^{\star} := (X_1^{\star}, \ldots, X_{q_v}^{\star})$. Moreover, let k_i be the number of non-fresh blocks for i-th decryption query and k_i' be the number of non-fresh blocks for i-th verification query. Therefore, there are $\sigma_d' := (\sigma_d - \sum_{i=1}^{q_d} k_i)$ many $M_i^{'}$ values and $\sigma_v' := (\sigma_v - \sum_{i=1}^{q_v} k_i')$ many $M_i^{\star}$ values have been sampled. This in particular allows us to compute the ideal interpolation probability as follows: in the online phase the encryption oracle samples q_e many tag values and σ_{q_e} many cipher text blocks uniformly at random. The decryption oracle samples σ_d' many message blocks and the verification oracle samples σ_v' many message blocks. In the offline phase, the ideal oracle samples total $s_e + s_v$ many Y values. Hence,

$$\mathsf{Pr}[X_{\mathrm{id}} = \tau] = \left(\frac{1}{2^n}\right)^{q_e} \cdot \left(\frac{1}{2^n}\right)^{\sigma_e} \cdot \left(\frac{1}{2^n}\right)^{\sigma_d'} \cdot \left(\frac{1}{2^n}\right)^{\sigma_v'} \cdot \left(\frac{1}{2^n}\right)^{s_e + s_v}$$

Now, we compute the real interpolation probability for τ. Since, τ is a good transcript, $X_i^+[\ell_i]$ is fresh. Therefore, T_i^+ is uniformly distributed. Moreover, we do not have any collision in the tuple $\mathbf{U}^+ := (U_1^+, \ldots, U_{q_e}^+)$ as τ is good which gives the uniform distribution on the cipher text blocks. It is easy to see that the decryption oracle samples exactly σ'_d many message blocks and verification oracle samples exactly σ'_v many message blocks. Morever, as there are $s_e + s_v$ many distinct X values in encryption and verification query history, we have,

$$\mathsf{Pr}[X_{\mathrm{re}} = \tau] = \left(\frac{1}{2^n}\right)^{q_e} \cdot \left(\frac{1}{2^n}\right)^{\sigma_e} \cdot \left(\frac{1}{2^n}\right)^{\sigma'_d} \cdot \left(\frac{1}{2^n}\right)^{\sigma'_v} \cdot \left(\frac{1}{2^n}\right)^{s_e+s_v}$$

This gives the ratio of the real to ideal interpolation probability 1. □

Conclusion. By the H-coefficient technique of Theorem 1, we obtain for the remaining distance of (4):

$$\mathbf{Adv}_{\Pi}^{\mathsf{AERUP}}(\mathcal{A}) \leq \epsilon_{\mathrm{ratio}} + \epsilon_{\mathrm{bad}} ,$$

where $\epsilon_{\mathrm{ratio}} = 0$ given the bound of Lemma 2 and ϵ_{bad} is set to be the bound of Lemma 1.

6 Implementation

In this section, we mainly focus on hardware benchmarking of the proposed constructions. However, for the sake of completeness, a short note on the software performance of various instantiations of ESTATE and sESTATE is given towards the end of this section (see subsection 6.8).

We first describe the hardware implementation results for TweAES and TweGIFT followed by the implementation details of our cipher family ESTATE. All the members of ESTATE have the same structure. The only difference lies in the choice of the underlying primitives. Hence, it is reasonable to describe the details with respect to one of the members ESTATE_TweAES. We start with a very brief description of the implementation of TweAES. Next we describe hardware architecture details of ESTATE_TweAES. Finally, we 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. Note that, we have implemented both instantiations of SUNDAE 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 the GMU Caesar Package [GMU16] for both ESTATE_TweAES and SUNDAE_AES-128. The details are given below.

6.1 Implementation Results of TweAES and TweGIFT

Here we briefly describe the hardware implementation results for TweAES and TweGIFT. We present both the encryption/decryption (ED) version and only encryption (E) version. The VHDL code of our implementations are synthesized using the Xilinx ISE 14.7 tool in a Virtex 7 FPGA (XC7VX415TFFG1761). We have used the default options (optimized for speed) and all the S-boxes and memories to store the round keys are mapped to LUTs, and no block RAM is used. We present the results obtained from the tool after performing the place and route process.

Table 5 shows that the area-overhead (LUT counts) introduced by the tweak injection is negligible. Considering the combined encryption-decryption (ED) implementation, TweAES have an overhead (in LUTs) of 0.5%. As we move to the encryption (E) only implementation, our recommended TweAES versions have negligible area overheads of 0.7%. Note that, the reduction in the speed is also negligible.

Table 5: Implementation results for AES and TweAES on Virtex 7 FPGA. Here ED stands for both encryption and decryption circuit. E stands for encryption only circuit

BC or tBC	LUTs	FF	Slices	Frequency (MHz)	Clock cycles	Throughput (Mbps)
AES-ED	2945	533	943	297.88	11	3466.24
TweAES-ED	2960	534	1044	295.97	11	3444.01
AES-E	1605	524	559	330.52	11	3846.05
TweAES-E	1617	524	574	328.27	11	3819.87

Table 6: Implementation results for GIFT and TweGIFT on Virtex 7 FPGA

BC or tBC	LUTs	FF	Slices	Frequency (MHz)	Clock cycles	Throughput (Mbps)
GIFT-128-ED	1113	408	432	447.83	41	1398.10
TweGIFT-128-ED	1223	408	428	429.32	41	1340.31
GIFT-128-E	763	403	330	596.30	41	1861.62
TweGIFT-128-E	805	403	377	598.78	41	1869.36

Table 6 summarizes the hardware performance of TweGIFT along with the original GIFT. For the ED implementation, TweGIFT-128 has an overhead of 4.04% for a tweak size of 4. As we move to the E implementation, TweGIFT-128 has an overhead of 4.32%.

6.2 Implementation Results of ESTATE and Benchmark with SUNDAE

In this section, we present our implementation of all the members of the ESTATE family. We also implement both SUNDAE_AES-128 and SUNDAE_GIFT-128 using the same interface. The underlying combined encryption/decryption architecture of ESTATE_TweAES is given in Fig. 15. The main modules are described in Appendix A. The hardware implementations codes of ESTATE and SUNDAE members are written in VHDL and are implemented on Virtex 7 xc7vx485t (Vivado v.2018.2.2) using RTL and a basic round-based architecture. The areas are provided in terms of the number of slice registers, slice LUTs and the number of occupied slices. We also provide 8-bit implementations of ESTATE_TweAES whose sole motivation is to optimize the hardware area. We observe that it achieves a much lower hardware area than the above ones. We denote the 8-bit implementation of ESTATE_TweAES by ESTATE_TweAES (8) which achieves even a significantly lower

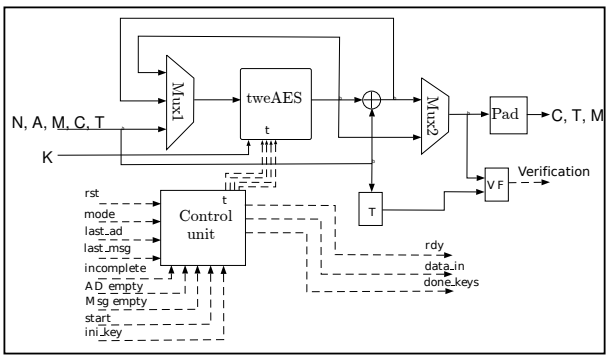

Figure 15: Hardware Architecture of ESTATE_TweAES

hardware area with a significant compromise in the throughput.

The detailed implementation results are depicted in Table 7.

Table 7: ESTATE and SUNDAE (ED Circuit) Implemented FPGA Results

Scheme	# Slice Registers	# LUTs	# Slices	Frequency (MHz)	Throughput (Gbps)	Mbps/ LUT	Mbps/ Slice
ESTATE_TweAES	803	1901	602	303.00	1.94	1.02	3.22
sESTATE_TweAES	813	1903	602	302.20	2.42	1.27	4.02
ESTATE_TweAES (8)	284	393	125	316.18	0.19	0.49	1.54
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 ESTATE is more significant in case of ESTATE_TweGIFT-128 since GIFT is significantly smaller than AES. The latency for TweAES 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 uses 20 clock cycles per block and sESTATE_TweAES uses 16. Observe that, both the versions of ESTATE are better (in hardware area) than SUNDAE. However, ESTATE_TweGIFT-128 is significantly more area-efficient than SUNDAE_GIFT-128 and the difference between ESTATE_TweAES and SUNDAE_AES-128 is minimal.

We would like to point out that the difference between ESTATE and SUNDAE (based on AES) is 1 AES encryption or 10 cycles, which is significant for short messages. For example, if we process a 16 byte message, ESTATE_TweAES achieves a throughput of 1251.10 Mbps while SUNDAE_AES-128 has a throughput of 945.36 Mbps.

6.3 Handling the 2-Pass Mode

ESTATE is a 2-pass mode and the message is processed twice for MAC and Encrypt. Very briefly, the adopted technique for handling the 2-pass mode can be storing the message in a buffer exactly similar as proposed in the GMU Lightweight interface (Sect. 2.1 in [KDT+]). To be precise, the associated data is processed first and next the message using the MAC to generate the tag. In addition, the message is stored in a buffer to be encrypted. For decryption, first the ciphertext is decrypted and the result is stored to a buffer to be authenticated. Note that, our implementation assumes arrival of the message twice while this technique needs a large buffer with size bounded by the upper bound of the input length.

6.4 Very Small Implementation of ESTATE_TweAES

We also introduce two tiny FPGA implementations of ESTATE_TweAES. The main motivation for the first implementation is to analyze the area-efficiency tradeoff for the energy efficient version ESTATE_TweAES with low area implementation. In this case, we use a 32-bit data-path AES based on the implementation introduced in [RSQL04]. This implementation uses TBOXES stored in Block RAMs, and it takes 45 clock cycles to encrypt the first block; after that, it can work in bulk mode with one encryption running for 44 clock cycles. In Table 8 we show the experimental results. The results depict that the tradeoff remains almost the same (i.e, area efficiency) on Virtex 7 with a significant decrease in the circuit area with a factor of 5 but with an increase in the throughput with almost the same factor. We can observe from Table 8, that our implementation of ESTATE_TweAES in a low power device Artix 7 *xc7a12tlcpg238-2L*, occupies almost the same resources as in Virtex 7 device but the frequency is much lower. It is interesting to

see that we can have an Deterministic AE (DAE) mode of operation using AES in just less than 130 slices. Also the overhead introduced by the mode is less than the size of AES itself.

We also provide 8-bit implementations of ESTATE_TweAES (using the same strategy of using Block RAMs). The main aim of the Block RAM based implementation is to optimize the hardware area. We observe that it achieves even a lower hardware area than the above one. However, the hardware footprint difference is not significant as the 32-bit implementation [RSQL04] is highly optimized for FPGA. In addition the frequency has been reduced significantly. We denote this 8-bit implementation of ESTATE_TweAES by ESTATE_TweAES (8). The implementation result is given in Table 8. Table 9 also depicts that the hardware area of ESTATE_TweAES (8) is comparable to that of SAEB(8) while operating in higher frequency.

Table 8: Very Small Implementations of ESTATE_TweAES in FPGA Results

Scheme	# Slice Registers	# LUTs	# Slices	Frequency (MHz)	Throughput (Mbps)	Mbps/ LUT	Mbps/ Slice
AES Artix 7	161	221	88	150.34	437.35	1.97	4.97
AES Virtex 7	165	222	89	280.29	815.39	3.67	9.16
TweAES Artix 7	190	299	102	148.5	432	1.44	4.24
TweAES Virtex 7	190	285	104	277.59	807.53	2.83	7.76
ESTATE_TweAES Artix 7	289	377	120	147.06	213.91	0.56	1.78
ESTATE_TweAES Virtex 7	289	376	124	270.27	393.12	1.05	3.17
ESTATE_TweAES (8) Artix 7	231	317	105	220.18	134.20	0.42	1.28
ESTATE_TweAES (8) Virtex 7	292	352	113	316.18	192.72	0.55	1.71

Table 9: 8-bit Implementation Results (Virtex 7) of ESTATE_TweAES and SAEB

Modules	#LUTs	#Registers	Frequency (MHz)
ESTATE_TweAES (8)	352	292	316.18
SAEB (8)	348	242	145.9

6.5 Power Consumption Results for ESTATE_TweAES

We perform a power consumption analysis on the energy efficient recommendation ESTATE_TweAES. We also perform a simulation for the two proposed architectures: one with 128-bit datapath and the other 32-bit datapath (tiny implementation of ESTATE_TweAES). We first generate 100 random pairs of AD and Message, next we perform a post-implementation simulation saving the switching activity. Finally, the saved result is used by Vivado Power Analyzer to estimate the power consumption under different operating frequencies. In Table 10 we show the results obtained from the Power Analyzer.

As we are using FPGA platform, the static power is almost constant for both the architectures implemented in Virtex 7, but the only variation is in the dynamic power, which is related to the switching activity in the design. We did the power estimation for the 32-bit data-path architecture in both Artix 7 and Virtex 7 to see the difference in power consumption. From Table 10, we observe that static power in Virtex 7 is more than four times than in Artix 7, as Artix 7 is a low power device while Virtex 7 is a high-end one. The dynamic power is a bit bigger in Virtex 7. For the 128-bit data-path architecture, we performed the power estimation only in Virtex 7, and its behavior in Artix 7 is expected to be very similar only with differences in the static power.

6.6 Benchmarking ESTATE

In Table 11, we provide a benchmark of the hardware implementation results of all the members in the ESTATE family using some of the implementation listed in the Athena

Table 10: Power consumption of the two proposed architectures for ESTATE_TweAES in FPGA

Device	# Frequency (MHz)	# Data-path size	Static Power (mW)	Dynamic Power (mW)	Total Power (mW)
Artix 7	10	32	58	2	60
	50		58	8	66
	100		58	16	74
	148.5		58	23	81
Virtex 7	10	32	242	2	244
	50		242	10	252
	100		242	20	262
	270.27		243	45	288
Virtex 7	10	128	242	3	245
	50		243	17	259
	100		243	40	283
	270.27		244	195	439

website [ATHa] along with the implementation results in [NMSS18, CIMN17a, CIMN17b, CDNY18a, CDNY18b] on Virtex 7. The results depict that ESTATE can be implemented with less hardware area keeping a competitive performance. In fact, ESTATE_TweAES with 32-bit datapath tiny implementation outperforms significantly the other designs (except SAEB). ESTATE_TweGIFT-128 is also one of the best in the literature (only next to tiny ESTATE_TweAES, SAEB and ACORN). We would like to mention that we are not claiming optimality in terms of Mbps/LUT or Mbps/slice, rather our primary focus is to have an area-efficient design with reasonable performance. Note that, we directly use the AES only encryption core provided in the GMU Caesar Package [GMU16] and our own implementation for TweGIFT-128.

6.7 Component Wise Area Calculation for AES

We show how the area is occupied by the different components for the hardware implementation of ESTATE_TweAES. We observe that the majority of the hardware area is consumed by TweAES. The distributions are described in Table 12 below. The area labeled as Logic corresponds to the circuits introduced by the non-tBC components to implement OFB and CBC modes of operations. The region labeled as registers in FF distribution corresponds to the input/output registers of the architecture.

6.8 On Software Implementation and Benchmarking

At NIST Lightweight Cryptography Workshop 2019, Renner et al. proposed a custom framework [RPM19a] for benchmarking software implementations from the NIST LwC project on embedded devices. The benchmarking framework is publicly available in an online repository [RPM19b]. This repository contains latest benchmarking results for all the second round candidates on several microcontrollers including 8-bit, 32-bit and 64-bit architectures. We refer the readers to [RPM19a, RPM19b] for further exposition on the framework, the test setup and procedures.

In Table 13, we directly reproduce the benchmarking results for ESTATE_TweAES, ESTATE_TweGIFT, and sESTATE_TweAES-6, as given in [RPM19b], for several microcontroller units. In addition, we also reproduce the benchmarking results for SUNDAE_GIFT-128 (with 128-bit nonce), the remaining SIV based round 2 candidate, to give a comparative view. For GIFT based instances of SUNDAE and ESTATE, we use the results for Rhys Weatherley's public implementations [Wea20] since they give better performance as compared to respective reference implementations given in the NIST LwC submission. For AES based ESTATE and sESTATE only reference implementation is available.

Table 11: Comparison on Virtex 7 with some of the implementation results in [ATHb]. 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 (32-bit datapath Implementation)	tBC	376	124	0.393	1.05	3.17
ESTATE_TweAES	tBC	1901	602	1.94	1.02	3.22
sESTATE_TweAES	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 [Min16]	BC	4263	1204	3.187	0.748	2.647
AES-OCB [KR16]	BC	4269	1228	3.608	0.845	2.889
AES-COPA [ABL+15]	BC	7795	2221	2.770	0.355	1.247
AES-GCM	BC	3478	949	3.837	1.103	4.043
CLOC-AES [IMG+16]	BC	3552	1087	3.252	0.478	1.561
CLOC-TWINE [IMG+16]	BC (non AES)	1552	439	0.432	0.278	0.984
SILC-AES [IMG+16]	BC	3040	910	4.365	1.436	4.796
SILC-LED [IMG+16]	BC (non AES)	1682	524	0.267	0.159	0.510
SILC-PRESENT [IMG+16]	BC (non AES)	1514	484	0.479	0.316	0.990
ELmD [DN15]	BC	4490	1306	4.025	0.896	3.082
JAMBU-AES [WH16]	BC	1595	457	1.824	1.144	3.991
JAMBU-SIMON [WH16]	BC (non AES)	1200	419	0.368	0.307	0.878
COFB-AES [CIMN17a, CIMN17a]	BC	1456	555	2.820	2.220	5.080
SAEB [NMSS18]	BC	348	–	–	–	–
AEGIS [WP16]	BC-RF	7504	1983	94.208	12.554	47.508
DEOXYS [JNP16]	TBC	3234	954	1.472	0.455	2.981
Beetle[Light+] [CDNY18a, CDNY18b]	Sponge	608	312	2.095	3.445	6.715
Beetle[Secure+] [CDNY18a, CDNY18b]	Sponge	1101	512	2.993	2.718	5.846
ASCON-128 [DEMS16]	Sponge	1373	401	3.852	2.806	9.606
Ketje-Jr [BJDAK16]	Sponge	1567	518	4.080	2.604	7.876
NORX [AJN16]	Sponge	2881	857	10.328	3.585	12.051
PRIMATES-HANUMAN [ABB+16]	Sponge	1148	370	1.072	0.934	2.897
ACORN [Wu16]	Stream cipher	499	155	3.437	6.888	22.174
TriviA-ck [CCHN15, CCHN18, CN15]	Stream cipher	2221	684	14.852	6.687	21.713

Table 12: Distribution of #LUTs (left) and #FF (right) for ESTATE_TweAES implementation

Modules	Distribution of #LUTs	Distribution of #FFs
TweAES	84.43	65.26
Control	2.52	3.49
Logic / Register	13.05	31.26

Acknowledgements. The authors would like to thank Dr. Christina Boura for his insightful comments and suggestions in preparing the final draft. We would also like to thank all the anonymous reviewers of ToSC Special Issue for their valuable comments. Avik Chakraborti, Nilanjan Datta, Ashwin Jha and Mridul Nandi are supported by the project "Study and Analysis of IoT Security" under Government of India at R.C.Bose Centre for Cryptology and Security, Indian Statistical Institute, Kolkata.

References

[ABB+16] 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.

[ABL+14] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to securely release unverified plaintext in authenticated encryption. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and*

Table 13: Summary of time and ROM usage for round 2 candidates based on ESTATE and SUNDAE. Time is in microseconds and ROM size is in bytes.

AEAD Implementation	Uno[1]		F1[2]		ESP[3]		F7[4]	
	time	ROM	time	ROM	time	ROM	time	ROM
ESTATE_TweAES	3524.86	8690	665.809	18608	154.297	221792	436.12	7644
ESTATE_TweGIFT			201.972	27220	63.784	232496	107.275	17092
sESTATE_TweAES-6	3174.96	8654	568.645	18688	126.148	221840	402.544	7708
SUNDAE-GIFT-128	5880.36	23162	205.741	22588	65.605	227520	111.01	12108

[1] Arduino Uno R3 | [2] STM32F1 "bluepill" | [3] Espressif ESP32 WROOM | [4] STM32 NUCLEO-F746ZG .

Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I, pages 105–125, 2014.

[ABL+15] 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.

[AJN16] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. NORX v3.0. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/norxv30.pdf.

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

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

[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 *Advances in Cryptology - ASIACRYPT 2015, Proceedings, Part II*, pages 411–436, 2015.

[BBLT18] 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.

[BJDAK16] 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.

[BPP+17] 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.

[BR05] John Black and Phillip Rogaway. CBC macs for arbitrary-length messages: The three-key constructions. *J. Cryptology*, 18(2):111–131, 2005.

[CCHN15] Avik Chakraborti, Anupam Chattopadhyay, Muhammad Hassan, and Mridul Nandi. Trivia: A fast and secure authenticated encryption scheme. In *CHES 2015*, pages 330–353, 2015.

[CCHN18] 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.

[CDD+19] Donghoon Chang, Nilanjan Datta, Avijit Dutta, Bart Mennink, Mridul Nandi, Somitra Sanadhya, and Ferdinand Sibleyras. Release of unverified plaintext: Tight unified model and application to anydae. *IACR Cryptology ePrint Archive*, 2019:1326, 2019.

[CDJ+19] 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.

[CDN18] 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.

[CDNY18a] 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.

[CDNY18b] 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.

[CHP+18] Carlos Cid, Tao Huang, Thomas Peyrin, Yu Sasaki, and Ling Song. Boomerang Connectivity Table: A New Cryptanalysis Tool. In *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 683–714. Springer, 2018.

[CIMN17a] 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.

[CIMN17b] 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.

[CN15] Avik Chakraborti and Mridul Nandi. TriviA-ck-v2. Submission to CAESAR, 2015. https://competitions.cr.yp.to/round2/triviackv2.pdf.

[CS14] Shan Chen and John P. Steinberger. Tight security bounds for key-alternating ciphers. 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 327–350, 2014.

[DEM16] Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. Square attack on 7-round Kiasu-BC. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *ACNS 2016*, volume 9696 of *LNCS*, pages 500–517. Springer, 2016.

[DEMS16] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/asconv12.pdf.

[DFJ13] Patrick Derbez, Pierre-Alain Fouque, and Jérémy Jean. Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting. In *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 371–387. Springer, 2013.

[DL17] Christoph Dobraunig and Eik List. Impossible-differential and boomerang cryptanalysis of round-reduced kiasu-bc. In *Topics in Cryptology - CT-RSA 2017 - San Francisco, CA, USA, February 14-17, 2017, Proceedings*, pages 207–222, 2017.

[DN15] Nilanjan Datta and Mridul Nandi. Proposal of ELmD v2.1. Submission to CAESAR, 2015. https://competitions.cr.yp.to/round2/elmdv21.pdf.

[DS08] Hüseyin Demirci and Ali Aydin Selçuk. A Meet-in-the-Middle Attack on 8-Round AES. In Kaisa Nyberg, editor, *FSE 2008*, volume 5086 of *LNCS*, pages 116–126. Springer, 2008.

[ENC01] Recommendation for Block Cipher Modes of Operation: Methods and Techniques. NIST Special Publication 800-38A, 2001. National Institute of Standards and Technology.

[FIP01] NIST FIPS. Advanced Encryption Standard (AES). *Federal Information Processing Standards Publication*, 197, 2001.

[FKL+00] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David A. Wagner, and Doug Whiting. Improved cryptanalysis of rijndael. In Bruce Schneier, editor, *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings*, volume 1978 of *Lecture Notes in Computer Science*, pages 213–230. Springer, 2000.

[GMU16] CAESAR Development Package, 2016. https://cryptography.gmu.edu/athena/index.php?id=download.

[Gra19] Lorenzo Grassi. Probabilistic mixture differential cryptanalysis on round-reduced AES. In Kenneth G. Paterson and Douglas Stebila, editors, *Selected Areas in Cryptography - SAC 2019*, volume 11959 of *LNCS*, pages 53–84. Springer, 2019.

[IMG+16] 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.

[JNP14] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. Tweaks and keys for block ciphers: The TWEAKEY framework. In *ASIACRYPT 2014*, pages 274–288, 2014.

[JNP16] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. Deoxys v1.41. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/deoxysv141.pdf.

[KDT+] Jens-Peter Kaps, William Diehl, Michael Tempelmeier, Farnoud Farahmand, Ekawat Homsirikamol, and Kris Gaj. A comprehensive framework for fair and efficient benchmarking of hardware implementations of lightweight cryptography. https://cryptography.gmu.edu/athena/LWC/LWC_HW_Benchmarking_Framework.pdf.

[KR16] Ted Krovetz and Phillip Rogaway. OCB(v1.1). Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/ocbv11.pdf.

[LSG+19] Ya Liu, Yifan Shi, Dawu Gu, Zhiqiang Zeng, Fengyu Zhao, Wei Li, Zhiqiang Liu, and Yang Bao. Improved meet-in-the-middle attacks on reduced-round Kiasu-BC and Joltik-BC. *Comput. J.*, 62(12):1761–1776, 2019.

[MBTM17] 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.

[MDRM10] Hamid Mala, Mohammad Dakhilalian, Vincent Rijmen, and Mahmoud Modarres-Hashemi. Improved impossible differential cryptanalysis of 7-round AES-128. In Guang Gong and Kishan Chand Gupta, editors, *INDOCRYPT 2010*, volume 6498 of *LNCS*, pages 282–291. Springer, 2010.

[Min16] Kazuhiko Minematsu. AES-OTR v3.1. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/aesotrv31.pdf.

[NMSS18] 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.

[Pat08] Jacques Patarin. The "coefficients h" technique. In *Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers*, pages 328–345, 2008.

[PSWZ15] Thomas Peyrin, Siang Meng Sim, Lei Wang, and Guoyan Zhang. Cryptanalysis of JAMBU. In *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, pages 264–281, 2015.

[RPM19a] Sebastian Renner, Enrico Pozzobon, and Jürgen Mottok. Benchmarking software implementations of 1st round candidates of the NIST LWC project on microcontrollers. In *NIST Lightweight Cryptography Workshop 2019*, 2019.

[RPM19b] Sebastian Renner, Enrico Pozzobon, and Jürgen Mottok. NIST LWC software performance benchmarks on microcontrollers, 2019. https://lwc.las3.de/.

[RS06] Phillip Rogaway and Thomas Shrimpton. Deterministic authenticated-encryption: A provable-security treatment of the key-wrap problem. *IACR Cryptology ePrint Archive*, 2006:221, 2006.

[RSQL04] Gaël Rouvroy, François-Xavier Standaert, Jean-Jacques Quisquater, and Jean-Didier Legat. Compact and efficient encryption/decryption module for FPGA implementation of the AES rijndael very well suited for small embedded applications. In *International Conference on Information Technology: Coding and Computing (ITCC'04), Volume 2, April 5-7, 2004, Las Vegas, Nevada, USA*, pages 583–587, 2004.

[TAY16] Mohamed Tolba, Ahmed Abdelkhalek, and Amr M. Youssef. A meet in the middle attack on reduced round kiasu-bc. *IEICE Transactions*, 99-A(10):1888–1890, 2016.

[Wea20] Rhys Weatherley. Lightweight cryptography primitives, 2020. https://github.com/rweather/lightweight-crypto.

[WH16] 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.

[WP16] 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.

[Wu16] Hongjun Wu. ACORN: A Lightweight Authenticated Cipher (v3). Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/acornv3.pdf.

[ZDY19] Baoyu Zhu, Xiaoyang Dong, and Hongbo Yu. Milp-based differential attack on round-reduced GIFT. In Mitsuru Matsui, editor, *CT-RSA 2019*, volume 11405 of *LNCS*, pages 372–390. Springer, 2019.

A Hardware Architecture Modules

- **Registers.** An 128-bit register is used in ESTATE_TweAES to maintain the TweAES state. It is evident as ESTATE is based on feedback based modes CBC and OFB and we do not require any additional information to store during the lifetime of the encryption and decryption (not the verification). During verification, it is necessary to use the nonce to decrypt in the OFB mode and we need to store the tag in the register labeled as T.

- **Multiplexers.** Mux1 selects the input to TweAES. TweAES can perform three operations: encrypt one single block in ECB mode, compute the CBC mode or generate the encryption/decryption stream in the OFB mode. Using Mux1, TweAES gets the instruction which mode it should work. The output from TweAES (direct or xored with input block) is input to Mux2 (to denote whether the architecture executes encryption or decryption 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 computed from the associated data and the decrypted message.

- **Control unit.** It provides specific signals to different modules in the architecture. To follow the ESTATE_TweAES algorithm, we implement a finite state machine shown in Fig. 16 containing the following states:

  1. Reset: This state resets all the internal variables and signals and prepares the circuit to start. The control from the Reset state goes to the Wait state.
  2. Wait: This state indicates that we should now initialize the cipher functionalities. It waits until the signal start or ini_keys change to 1.
  3. Ini_keys: This state performs the computation of the round keys for TweAES.
  4. Enc_N: During the execution of this state, the architecture performs the TBC encryption of the Nonce. When the message and associated data are empty, the output generated in this state by TweAES is given as the tag. The only change for both the cases is the value of the tweak.
  5. FCBC_AD: This state executes the CBC mode with associated data blocks as the input.
  6. FCBC_Msg: Same as FCBC_AD but here the input is the message block, the last output is the tag.

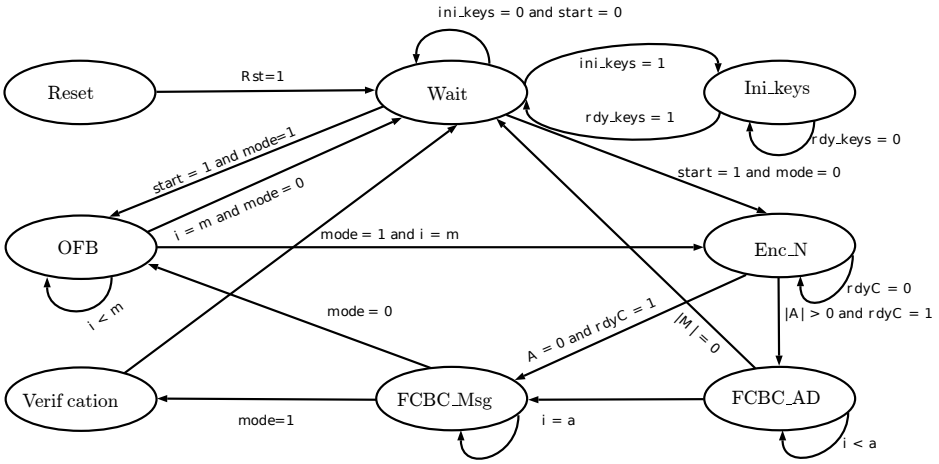

Figure 16: Finite State Machine

7. OFB: In this state, the architecture is configured to compute the encryption or decryption in the OFB mode.
8. Verification: This state just activates the output from the component VF.

It is important to note that the value for the tweak is generated inside the state machine and they are supplied to the TweAES module as shown in Figure 15. Depending on

- whether the encryption or the decryption is performed and
- whether at least one of the associated data and the message is empty,

the order of execution of the states change. The possible scenarios are shown in Table 14.

Table 14: Execution order of states for encryption/decryption and depending on the above points

Encryption	**Sequence of states**
$a > 0, m > 0$	Wait → Enc_N → FCBC_AD → FCBC_Msg → OFB → Wait
$a > 0, m = 0$	Wait → Enc_N → FCBC_AD → Wait
$a = 0, m > 0$	Wait → Enc_N → FCBC_Msg → OFB → Wait
$a = 0, m = 0$	Wait → Enc_N → Wait
Decryption	**Sequence of states**
$a > 0, m > 0$	Wait → OFB → Enc_N → FCBC_AD → FCBC_Msg → Wait
$a = 0, m > 0$	Wait → OFB → Enc_N → FCBC_Msg → Wait