Short Tweak TBC and Its Applications in Symmetric Ciphers

Avik Chakraborti[1], Nilanjan Datta[1], Ashwin Jha[2], Cuauhtemoc Mancillas-Lopez[3], Mridul Nandi[4] and Yu Sasaki[5]

[1]Institute for Advancing Intelligence, TCG CREST
avikchkrbrti@gmail.com,nilanjan.datta@tcgcrest.org
[2] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
ashwin.jha@cispa.de
[3]Indian Statistical Institute, Kolkata, India
mridul.nandi@gmail.com
[4]Computer Science Department, CINVESTAV-IPN, Mexico
cuauhtemoc.mancillas83@gmail.com
[5]NTT Secure Platform Laboratories, Tokyo, Japan
sasaki.yu@lab.ntt.co.jp

Abstract. Tweakable block cipher (TBC), a stronger notion than standard block ciphers, has wide-scale applications in symmetric-key schemes. At a high level, it provides flexibility in design and (possibly) better security bounds. In multi-keyed applications, a TBC with short tweak values can be used to replace multiple keys. However, the existing TBC construction frameworks, including Tweakey and XEX, are designed for general purpose tweak sizes. Specifically, they are not optimized for short tweaks, which might render them inefficient for certain resource constrained applications. So a dedicated paradigm to construct short-tweak TBCs (tBC) is highly desirable. In this paper, as a first contribution, we present a dedicated framework, called the Elastic-Tweak framework (ET in short), to convert any reasonably secure SPN block cipher into a secure tBC. We apply the ET framework on GIFT and AES to construct efficient tBCs, named TweGIFT and TweAES. As our second contribution, we propose a nonce misuse resistant, INT-RUP secure lightweight authenticated cipher ESTATE that uses short-tweak TBC as the underlying primitive. Finally, we show some other applications of ET-based tBCs, which are better than their block cipher counterparts in terms of key size, state size, number of block cipher calls, and short message processing. Some notable applications include, Twe-FCBC (reduces the key size of FCBC, and reduces the state size and the number of block cipher calls of CMAC), Twe-LightMAC_Plus (better rate than LightMAC_Plus), Twe-COLM (reduces the number of block cipher calls and simplifies the design of COLM).

Keywords: TBC, GIFT, AES, Tweakey, XEX

1 Introduction

Since their advent in late 1970's, block ciphers [FIP01] have become the ubiquitous building blocks in various symmetric-key based cryptographic designs, including encryption schemes [ENC01], message authentication codes (MACs) [CMA05], and authenticated encryption [CCM04]. Due to their wide-scale applicability, block ciphers are also the most well-analyzed symmetric-key primitives. As a result, the cryptographic community bestows a high degree of confidence in block cipher based designs. Block cipher structures are more or less well formalized and there are generic ways to evaluate the security of a block cipher against the classical linear [Mat93] and differential [BS90] attacks. The literature is filled with a plethora of block cipher candidates, AES [FIP01] being the most notable among them. AES is currently the NIST standard block cipher [FIP01], and it is the recommended choice for several standardized encryption, MAC and AE schemes such as CTR [ENC01], CMAC [CMA05], AES-GCM [GCM07] etc. A recent block cipher proposal, named GIFT [BPP+17] has generated a lot of interest due to its ultra-lightweight nature.

1.1 Some Issues in Block Cipher Based Designs

We would like to mention some issues in the block cipher based designs both in the design level and in the practical usage level.

For the design level, apart from the security, the designers mainly consider a trade off between the storage and the circuit complexity (in terms of the number of computations). These two points are given below.

(i) Storage is often measured by the key size, the auxiliary secret state size and the internal state size.

(ii) Circuit complexity is highly dependent on the internal module structures.

To have an efficient design, the designers always consider (1) to have optimized storage and (2) to remove avoidable modules, optimizing the circuit complexity and increasing the throughput (faster implementation with lesser operations).

The points described above are especially important in design of lightweight applications on IoT platforms. We elaborate the above mentioned design level issues in detail.

Key Size of the Designs: Several designs use more than one independent block cipher keys, which could be an issue for storage constrained applications. Some notable examples of such designs are sum of permutations, EDM [CS16], EWCDM [CS16], CLRW2 [LST12], GCM-SIV-2 [IM16], Benes construction [Pat08b]. While some of these designs have been reduced to single key variants, reducing a multi-keyed design to single-key design is, in general, a challenging problem.

Auxiliary Secret State: FCBC, a three-key MAC by Black and Rogaway [BR05], is a CBC-MAC type construction. CMAC [CMA05], the NIST recommended MAC design, reduces number of keys from three to one by using an auxiliary secret state (which is nothing but the encryption of zero block). Though CMAC is NIST recommended MAC design, it costs an extra block cipher call (compared to FCBC) and holds an additional state. This may be an issue in hardware applications, where area and energy consumption are very crucial parameters.

Simplicity of Designs: Design simplification, is a closely related topic to the single-keyed vs. multi-keyed debate. A simple design could be beneficial for real life applications, and better understanding of designs themselves. Often, the single-keyed variant of a block cipher based design is much more complex than the multi-keyed version, both in implementation and security analysis. This is due to several auxiliary functions used chiefly for domain separation. For instance CLOC and SILC [IMG+16] use several functions depending upon the associated data and message length. In contrast, the multi-keyed variants of CLOC and SILC would be much simpler.

Another point to ponder is practical usage level issues. One of the most important issues to be considered is efficient processing of short message inputs and the existing network standards are not optimized for it. Thus it is important to have designs to handle this issue. In fact the standardizing committees (e.g, NIST) are also searching for new standards and they are giving importance for efficient short message processing by the designs. This is evident from the statement published by NIST in the call for submissions for the Lightweight Cryptography Standardization Process.

"*Submitted AEAD algorithms and optional hash function algorithms should perform significantly better in constrained environments (hardware and embedded software platforms) compared to current NIST standards. They should be optimized to be efficient for short messages (e.g., as short as 8 bytes).*"

Short Message Processing: As pointed out, an essential requirement in lightweight applications is efficient short input data processing, while minimizing the memory consumption and precomputation. In use cases with tight requirements on delay and latency, the typical packet sizes are small (way less than 1 Kilobytes) as large packets occupy a link for longer duration, causing more delays to subsequent packets and increasing latency. For example, Zigbee [Zig], Bluetooth low energy and TinySec [KSW04] limit the maximum packet lengths to 127 bytes, 251 bytes and 128 bytes, respectively. Similarly, CAN FD [CAN], a well-known transmission protocol in automotive networks, allows message length up to 64 bytes. The packet sizes in EPC tag [EPC], which is an alternate to the bar code using RFID, is typically 12 bytes.

Cryptographic designs with low latency for shorter messages could be highly beneficial for such applications. As it turns out, for many designs short message performance is not that good due to some constant overhead. For instance CMAC uses one block cipher call to generate a secret state, and SUNDAE [BBLT18] uses the first call of block cipher to distinguish different possibilities of associated data and message lengths. So, to process a single block message, SUNDAE requires two block cipher calls. CLOC and SILC [IMG+16] have similar drawbacks. They cost 2 and 4 calls to process a single block message. LightMAC_Plus [LPTY16], feeds a counter-based encoded input to the block cipher, which reduces the rate.[1]

1.2 Possible Approach

The possible approach to address these problems is to design a primitive that can or helps to solve the above issues. Tweakable block cipher, a very powerful primitive, can be the best fit for this purpose.

Tweakable block cipher can actually solve most of the aforementioned issues in block ciphers quite easily. A secure TBC with distinct tweaks is actually equivalent to independently keyed instantiations of a secure block cipher. This naturally gives a TBC based single-keyed design for any block cipher based multi-keyed design. In some cases, TBCs can also avoid the extra block cipher calls. It also helps to simplify designs like CLOC and SILC.

In all these cases, we observe that a short tweak space (in most of the cases 2-bit or 4-bit tweaks) is sufficient. In other words, a short-tweak tweakable block cipher (in short we call tBC) would suffice for resolving these issues. Our aim is to describe a design, such that, by this design the attackers have degrees of freedom to attack the design only by a few bits.

1.3 Survey of Existing Designs

We do a short survey of the previous schemes to get some idea on the designs.

Tweakable Block Ciphers: The Hasty Pudding cipher [Sch98], an unsuccessful candidate for AES competition, was one of the first tweakable block ciphers.[2] Later, Liskov et al. formalized this in their foundational work on tweakable block ciphers [LRW02]. Tweakable block ciphers (TBCs) are more versatile and find a broad range of applications,

[1] No. of message blocks processed per block cipher call.
[2] It used the term "spice" for tweaks.

most notably in authenticated encryption schemes, such as OCB [KR11], COPA [ABL+15], and Deoxys [JNP16a]; and message authentication codes, such as ZMAC [IMPS17], NaT [CLS15], and ZMAC+ [LN17]. TBCs can be designed from scratch [Cro00, Sch98, FLS+10], or they can be built using existing primitives like block ciphers, and public permutations. LRW1, LRW2 [LRW02], CLRW2 [LST12], XEX [Rog04] and XHX [JLM+17] are some examples of the former category, whereas Tweakable Even-Mansour [CLS15] is an example of the latter. All the above constructions are built using generic modes and are provably secure. However, all of them use larger tweaks and may not be efficient in several of the above scenarios. Later, Tweakey framework tried to solve the performances issues with efficient instantiations and currently one of the most efficient framework to solve the above issues.

The Tweakey Framework: At Asiacrypt '14, Jean et al. presented a generic framework for TBC construction, called Tweakey [JNP14a], that considers the tweak and key inputs in a unified manner. Basically, the framework formalized the concept of tweak-dependent keys. The Tweakey framework gave a much needed impetus to the design of TBCs, with several designs like Kiasu [JNP16b], Deoxys [JNP16a], SKINNY and Mantis [BJK+16] etc. As Tweakey is conceptualized with general purpose tweak sizes in mind, it is bit difficult to optimize Tweakey for tBC. For instance, take the example of SKINNY-128. To process only 4-bit tweak, the additional register is limited but their computation modes must move from TK1 to TK2, which increases the number of rounds by 8. This in turn affects the throughput of the cipher. Although, some Tweakey-based designs, especially Kiasu-BC [JNP16b] do not need additional rounds, yet this is true in most of the existing Tweakey-based designs. We also note here that Kiasu-BC, which is based on AES, is weaker than AES by one round, as observed in several previous cryptanalytic works [DEM16, DL17, TAY16].

So, there is a need for a generic design framework for tBC, which (i) can be applied on top of a block cipher, (ii) adds minimal overheads, and (iii) is as secure as the underlying block cipher.

XE and XEX: Rogaway [Rog04], proposed two efficient ways of converting a block cipher into a tweakable block cipher, denoted by XE and XEX. These methods are widely used in various modes such as PMAC [BR02], OCB [RBB03], COPA [ABL+15], ELmD [DN15] etc. However, XE and XEX have several limitations with respect to a short tweak space, notably (i) security is limited to birthday bound (security bound degrades to the birthday bound of the security of the underlying block cipher), and (ii) precomputation and storage overhead to generate the secret state. In addition, it also requires to update the secret state for each invocation, which might add some overhead.

1.4 Our Contributions

Our contributions are manyfold. The first part describes the new generic Elastic-tweak framework which transforms a block cipher into a short tweak tweakable block cipher. The second part describes several design level applications that can improve the existing designs significantly. Finally, protocol level applications are described that can improve throughput and energy of the standard network protocols standard network protocols (protocols that process short messages such as CCMP, Bluetooth Low Energy 5.0, TLS 1.2 etc.).

1. Elastic-Tweak Framework: In this work, we address the above issues and describe a generic framework, called the Elastic-Tweak framework (ET in short), to transform a block cipher into a short tweak TBC. A short tweak can be as small as 4 bits and as large as 16 bits. This small size ensures that the tweak storage overhead is negligible. Overall, our protocol outperforms the others as it provides

(a) Negligible overheads for short tweaks,

(b) Generic conversion from BC to tBC,

(c) confidence over security evaluation as it is based on an existing block cipher,

(d) simple handling of tweaks provides advantage both in software and hardware implementations, and

(e) The Backward Compatibility feature (tBC with zero tweak functions the same as BC).

In this framework, given the block cipher, we first expand the short tweak using linear code, and then inject the expanded tweak at intervals of some fixed number of rounds, say r. Designs under this framework can be flexibly built over a secure block cipher, and are as secure as the underlying block cipher.

The ET framework distributes the effect of the tweak into the block cipher state that can generate several active bytes. In particular we choose a linear code with high branch number to expand the input tweak. This design is particularly suitable for short tweaks to ensure the security against differential cryptanalysis because the small weight of the short input always results in a large weight of the output.

Another advantage of the framework is the easiness of the security evaluation. First, for zero tweak value, the plaintext-ciphertext transformation is exactly the same as the original cipher (i.e. it has backward compatibility feature). Therefore, to evaluate the security of the new construction, we only need to consider the attacks that exploit at least one non-zero tweak. Second, the large weight of the expanded tweak ensures relatively high security only with a small number of rounds around the tweak injection. This allows a designer to focus on the security of the r-round transformation followed by the tweak injection and further followed by the r-round transformation, which is called "$2r$-round core."

We instantiate this framework with the standard and the most popular block cipher AES [FIP01] with different tweak sizes varying from 4 to 16. We also instantiate this Elastic-tweak with the GIFT [BPP+17] block cipher. We implement the instantiations both in software and hardware and find that they have negligible overheads compared to the original block ciphers.

We also present extensive security analysis of all the instantiations. In TweAES, the expanded tweak is divided into 8 parts and XORed to the top 2 rows of the state in every 2 rounds. We ensure that any non-zero tweak activates at least 15 active S-boxes for the 4-round core. We also show that by starting from the middle of the 2-round gap, 8 rounds can be attacked with impossible differential attacks. This attack, from a different viewpoint, demonstrate that attacking full rounds is difficult by exploiting tweak difference. We also discuss difficulties of applying boomerang, meet-in-the-middle, and integral attacks. Security of TweGIFT is similarly evaluated. We use MILP-based tools to evaluate its security against differential cryptanalysis.

2. Design of a concrete tBC based AEAD with nonce-misuse resistance: We describe a new highly secure and hardware efficient tweakable blockcipher (TBC) based authenticated encryption mode, dub it ESTATE (Energy efficient and Single-state Tweakable block cipher based MAC-Then-Encrypt). The structure employs MAC-then-Encrypt paradigm that employs FCBC [BR05] like MAC followed by OFB [ENC01] like encryption both with a 4-bit short tweak TBC (denoted as tBC as in line with [CDJ+19]). ESTATE is structurally close to SUNDAE, but with an additional interesting design feature of replacing the block cipher by a tBC. We address the points that SUNDAE needs to adopt several internal operations to deal with domain separations, SUNDAE does not provide any provable INT-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). However, we can resolve all these issues by using a tBC. The most interesting point is that, ESTATE does not use the tweak as the counter, rather as the

domain separator. Thus, a short tweak is sufficient. We can solve the above issues in SUNDAE by using different tweaks in the underlying tBC to (i) reduce the additional primitive invocation (we pre-compute a fixed tBC encrypted nonce with the unique tweak value 1 and use it all the time), (ii) provide INT-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. Thus ESTATE outperforms SUNDAE in various design properties. Overall, ESTATE has the following large set of features:

- **Optimum state size:** ESTATE has a state size as small as the block size of the underlying cipher, and it ensures good implementation characteristics both on lightweight and high-performance platforms.
- **Multiplication-free:** ESTATE does not require any field multiplications. In fact, apart from the tweakable block cipher call it requires just 128-bit XOR per block of data, which seems to be the minimum required overhead. Observe that, SUNDAE requires constant field multiplications (2, 4) for the purpose of domain separation. In contrast, we simply use different tweaks to achieve this.
- **Optimal:** ESTATE requires $(a + 2m)$ many primitive invocations to process an a block associated data (including the nonce) and m block message. 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 non-linear primitive calls. SUNDAE requires a constant block encryption in the beginning primarily due to the fact that same block cipher is used in encryption as well as authentication. We skip that extra call by using different tweaks for the block ciphers used in the encryption and the first block cipher call during authentication.
- **Inverse-Free:** ESTATE is an inverse-free authenticated encryption algorithm. Both encryption and decryption algorithms do not require any decryption call to the underlying tweakable block cipher. This significantly reduces the overall hardware footprint in combined encryption-decryption implementations.
- **Nonce-misuse Resistant:** ESTATE is a nonce-misuse resistant authenticated cipher and provides full security even with the repetition of the nonce. Alternatively said, it can be viewed as a deterministic authenticated encryption where the nonce is assumed to be the first block of the associated data.
- **INT-RUP Secure:** We separate the block cipher invocations for the OFB functions and the first tweakable block cipher input invocation by the usage of different tweaks. This essentially helps us to provide INT-RUP security for ESTATE and making it much more robust in constraint devices. Here, we note that the related construction SUNDAE lacks this feature and the authors 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 mode is quite robust, as evident by nonce misuse resistance and RUP security, to a lack of sufficient randomness or secret states.

A Lighter AEAD mode sESTATE: sESTATE is a lighter version of ESTATE and it is structurally identical to ESTATE. The only difference between sESTATE and ESTATE is that sESTATE uses 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 it's reduce version TweAES-6) as the underlying tBC. We provide complete hardware implementation details on FPGA platform.

3. Design Level Applications of tBC: Here we demonstrate the applicability of tBC in various constructions:

(i) **Reducing the Key Size in Multi-Keyed Modes:** The primary application of tBC is to reduce the key space of several block cipher based modes that use multiple independently sampled keys. We depict the applicability of tBC on FCBC MAC, Double Block Hash-then-Sum (DbHtS) paradigm, Sum of permutations, EDM, EWCDM, CLRW2, GCM-SIV-2 and the Benes construction.

(ii) **Efficient Processing of Short Messages:** tBC can be used to reduce the number of block cipher calls, which in turn reduces the energy consumption for short messages. We take the instance of Twe-LightMAC_Plus to demonstrate this application of tBC. Twe-LightMAC_Plus achieves a higher rate as compared to it's original counterpart LightMAC_Plus. In addition, the number of keys is reduced from 3 to 1.

(iii) **Replacement for XE and XEX.** tBC can be viewed as an efficient replacement of XE and XEX especially when we target short messages (say of size up to 1 MB). In such cases, instead of using a secret state (that we need to precompute, store and update), one can simply use tBC with the block-counters as the tweak. The applicability of this paradigm can be depicted on several MAC modes such as PMAC; encryption mode such as COPE and AEAD modes such as ELmD, COLM.

4. Protocol Level Applications of tBC: Here we demonstrate the applicability of tBC in various standard network protocols using CCM mode for authentication and encryption.

(i) **Reducing the Block Cipher Invocations in the CCM Mode:** The CCM mode uses CBC-MAC mode for MAC and CTR mode for encryption. We show that the injective padding used in the MAC (the injective padding is obtained by concatenation of the data length with the data) can be avoided without increasing the key storage using our framework. The number of block cipher calls that can be reduced is upper bounded by two and lower bounded by one. This is significant for the protocols that deal with short messages.

(ii) **List the Standard Protocols using CCM with the Data Format Description:** We list several standard network protocols that works to handle short messages and uses the CCM mode for authentication and encryption. We present the data sizes for these protocols to show that it is evident to use our proposal to make them more efficient.

1.5 Significance of the Framework in the Light of NIST Lightweight Project

Our framework is explicitly used in two first round candidates in the NIST Lightweight Project, namely (i) ESTATE and (ii) LOTUS-AEAD and LOCUS-AEAD [NIS17]. ESTATE can be viewed as a tweakable variant of SUNDAE, where the use of 4-bit tweak ensures (i) one less block cipher invocation, (ii) RUP security of the design and (iii) no constant multiplications for domain separations. In LOTUS-AEAD and LOCUS-AEAD, the short tweaks are used to especially to have simplicity in the design. Apart from these schemes, SIV-Rijndael256 and SIV-TEM-PHOTON are two round 1 submissions to NIST lightweight standardization process [NIS17], which independently used the idea of short-tweak tweakable block ciphers. We remark here that the Elastic-Tweak framework seems to be a more general approach, while their approach seems to work only for AES like ciphers.

1.6 Publications

The Elastic-tweak framework and its applications have been published in [CDJ+21]. The specification of ESTATE along with detailed implementation results have been published in [CDJ+20].

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 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 use the little-endian format for representing binary strings, i.e. the least significant bit is the right most bit. For any non-empty binary string X, $(X_{k-1}, \ldots, X_0) \xleftarrow{n} x$ denotes the n-bit block parsing of X, where $|X_i| = n$ for $0 \leq i \leq k-2$, and $1 \leq |X_{k-1}| \leq n$. For $A, B \in \{0,1\}^*$ and $|A| = |B|$, we write $A \oplus B$ to denote the bitwise XOR of A and B. For $A, B \in \{0,1\}^*$, $A \| B$ denotes the concatenation of A and B. Note that A and B denote the most and least significant parts, respectively.

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

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

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

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

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

2.2 Authenticated Encryption

An authenticated encryption scheme should offer confidentiality, meaning that its ciphertexts are computationally indistinguishable from 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

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

where \$ is the random oracle that on input (A, M) returns (C, T) uniformly at random and $\perp$ be 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

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

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 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 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

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

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 define over the random coins of $\mathcal{A}$. Note that, $\mathcal{A}$ can query to 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

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

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 q queries.

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

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

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 [Pat08a, 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, in addition, 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. $\Pr[X_{\text{id}} = \tau]$) 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 [Pat08a, 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* $\epsilon_{\text{ratio}} \geq 0$ *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 $\epsilon_{\text{bad}} \geq 0$ *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

3.1 The Elastic-Tweak Framework

In this section, we introduce the Elastic-Tweak framework (illustrated in Figure 1) on SPN based block ciphers that allows one to efficiently design tweakable block ciphers with short tweaks. As the name suggests, Elastic-Tweak refers to elastic expansion of short tweaks and we typically consider tweaks of size less than or equal to 16 bits. Using this framework, one can convert a block cipher to a short tweak tweakble block cipher denoted by tBC. We briefly recall the SPN structure on which this framework would be applied. An SPN block cipher iterates for rnd many rounds, where each round consists of three operations:

(a) SubCells (divides the state into cells and substitutes each cell by an s-bit S-box which is always non-linear),

(b) PermBits (uses a linear mixing layer over the full state to create diffusion), and

(c) AddRoundKey (add a round keys to the state).

The basic idea of the framework is to expand a small tweak (of size t) using a suitable linear code of high distance and then the expanded tweak (of size t_e) is injected (i.e. xored) to the internal block cipher state affecting a certain number of S-boxes (say, tic). We apply the same process after every gap number of rounds. An important feature of tBC is that it is implemented using very low tweak state and without any tweak schedule (only tweak expansion). In the following, we describe the linear code to expand the tweak and how to inject the tweak into the underlying block cipher state. If BC denotes the underlying SPN block cipher, we denote the tweakable block cipher as TweBC $[t, t_e, \mathsf{tic}, \mathsf{gap}]$ where $t, t_e, \mathsf{tic}, \mathsf{gap}$ are suitable parameters as described above.

3.2 Exp: Expanding the Tweak

In this section, we describe our method to expand the tweak T of t bits to an expanded tweak T_e of t_e bits. We need the parameters to satisfy the following conditions:

(a) t_e is divisible by $2t$ and tic. Let $w := t_e/\mathsf{tic}$, the underlying word size.

(b) w divides t and $w \leq s$.

The tweak expansion, called Exp, follows an "Expand then (optional) Copy" style as follows:

(i) Let $\tau := t/w$, and we view $T = (T_1, \ldots, T_\tau)$ as a $1 \times \tau$ vector of elements from $\mathbb{F}_{2^w}$. We expand T by applying a $[2\tau, \tau, \tau]$-linear code[3] over $\mathbb{F}_{2^w}$ with the generating matrix $G_{\tau \times 2\tau} = [I_\tau : I_\tau \oplus J_\tau]$, where I_τ is the identity matrix of dimension τ and J

[3] An $[n, k, d]$-linear code over a field $\mathbb{F}$ is defined by a $k \times n$ matrix G called the *generator* matrix over $\mathbb{F}$ such that for all nonzero vectors $v \in \mathbb{F}^k$, $v \cdot G$ has at least d many nonzero elements.

is the all 1 square matrix of dimension τ over $\mathbb{F}_{2^w}$. Let $T' = T \cdot G$ be the resultant code. Note that, T' can be computed as $S \oplus T_1 \| \cdots \| S \oplus T_\tau$ where $S = T_1 \oplus \cdots \oplus T_\tau$.

(ii) Finally, we compute the expanded tweak by concatenating $t_e/2t$ many copies of T' i.e.

$$T_e = T' \| \cdots \| T'.$$

Note that, T_e can be viewed as an application of $[\mathsf{tic}, \tau, \mathsf{tic}/2]$-linear code on T. The main rationale behind the choice of this expansion function is that it generates high distance codes (which is highly desired from the cryptanalysis point of view) with a low cost (only $(2\tau - 1)$ addition over $\mathbb{F}_{2^w}$ is required).

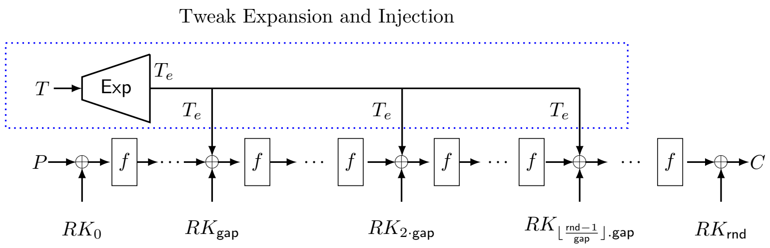


Figure 1: Elastic-Tweak Construction.

Function $\mathsf{Exp}[t_e, w](T)$

1. $\tau \leftarrow \frac{|T|}{w}$
2. $T_e \leftarrow \phi$
3. $(T_1, T_2, \ldots, T_\tau) \overset{w}{\leftarrow} T$
4. $T' \leftarrow T \| (T \oplus T \cdot J_\tau)$
5. **for** $i = 1$ **to** $t_e/2t$
6. $\quad T_e \leftarrow T_e \| T'$
7. **return** T_e

Algorithm tBC $[t_e, \mathsf{tic}, \mathsf{gap}](X, K, T)$

1. $w \leftarrow t_e/tic$
2. $T_e \leftarrow \mathsf{Exp}[t_e, w](T)$
3. **for** $i = 1$ **to** rnd
4. $\quad X \leftarrow \mathsf{SubCells}(X)$
5. $\quad X \leftarrow \mathsf{PermBits}(X)$
6. $\quad (K, X) \leftarrow \mathsf{AddRoundKey}(K, X, i)$
7. $\quad$ **if** i % gap $= 0$ and $i <$ rnd
8. $\quad\quad \mathsf{AddTweak}[\mathsf{tic}](X, T_e)$
9. **return** X

Figure 2: Function $\mathsf{Exp}(T, t_e, w)$ and tBC (X, K, T). Here, $\mathsf{AddTweak}[\mathsf{tic}](X, T_e)$ represents the xoring tweak in to the state of the block cipher.

3.3 Injecting Expanded Tweak into Round Functions

Note that the expanded tweak can be viewed as $T_{e,1} \| \cdots \| T_{e,\mathsf{tic}}$ where each $T_{e,i}$ is of size w-bits and $w \leq s$. Now we xor these tweak in addition to the round keys in tic number of S-boxes. The exact choices of S-box would be design specific so that the diffusion due to tweak difference is high.

The tweak injection is optional for each round, the tweak injection starts from round start and it is injected at an interval of gap rounds and stops at round end. To be precise, we inject tweak at the round number start, $\mathsf{start} + \mathsf{gap}$, $\mathsf{start} + 2.\mathsf{gap}$, . . . , end. To have a uniformity in the tweak injection rounds, we typically choose $\mathsf{start} = \mathsf{gap}$ and inject the tweaks at an interval of gap rounds. This implicitly sets $\mathsf{end} = \mathsf{gap}.\lfloor \frac{\mathsf{rnd}-1}{\mathsf{gap}} \rfloor$.

REQUIREMENTS FROM TweBC. We must ensure TweBC should have same security level as the underlying block cipher.

From the performance point of view, our target is to obtain the above mentioned security

"minimizing t_e *(signifies the area) and* $t_e.\lfloor \frac{rnd-1}{gap} \rfloor$ *(signifies the energy)."*

FEATURES OF TweBC.

1. Our tBC is applied to any SPN based block ciphers.
2. Due to linear expansion of tweak, tBC with zero tweak turns out to be same as the underlying block cipher (note that we keep same number of rounds as the block cipher). This feature would be useful to reduce overhead due to nonzero tweak. Later we see some applications (e.g., application on FCBC) where the nonzero tweaks is only applied to process the last block.

3.4 Tweakable GIFT and AES

In this section, we provide various instantiation of tBC built upon the two popular block ciphers GIFT and AES. We are primarily interested on tweak size 4, 8, 16, and hence considered $t \in \{4, 8, 16\}$.

3.4.1 Instantiation of tBC with 4 bit Tweak.

All the recommendations with 4-bit tweaks have extremely low overhead over the original block cipher and they can be ideal for reducing multiple keys scheme to an equivalent single key scheme instance with a minuscule loss in efficiency. Detailed description can be found in Sect. 5.

(i) TweGIFT-64[4, 16, 16, 4]. In this case the tweak is expanded from 4 bits to 16 bits and the expanded tweak is injected at bit positions $4i + 3$, for $i = 0, \ldots, 15$.

(ii) TweGIFT-128[4, 32, 32, 5]. Here we expand the 4 bit tweak to 32 bits and the expanded tweak is injected at bit positions $4i + 3$, for $i = 0, \ldots, 31$.

(iii) TweAES[4, 8, 8, 2]. Here we expand the 4 bit tweak to 8 bits and the expanded tweak is injected at the least-significant bits of each of the 8 S-Boxes in the top two rows.

3.4.2 Instantiation of tBC with 8 and 16 bit Tweak.

tBC with tweak size of 8/16-bits are ideal for replacing the length counter bits (or masking) used in many constructions. Detailed description can be found in Sect. 5.

(i) TweAES[8, 16, 8, 2]. For 8 bit tweak, we only use AES. The tweak is first extended to 16 bits and the tweak is injected at the two least-significant bits of each of the 8 S-Boxes in the top two rows.

(ii) TweGIFT-128[16, 32, 32, 4]. Here we expand the 16 bit tweak to 32 bits and the expanded tweak is injected at bit positions $4i + 3$, for $i = 0, \ldots, 31$.

(iii) TweAES[16, 32, 8, 2]. Here we expand the 16 bit tweak to 32 bits and expanded tweak is injected at the four least-significant bits of each of the 8 S-Boxes in the top two rows.

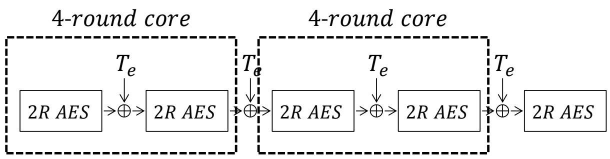


Figure 3: 4-round Core of TweAES[∗,∗,∗,2]

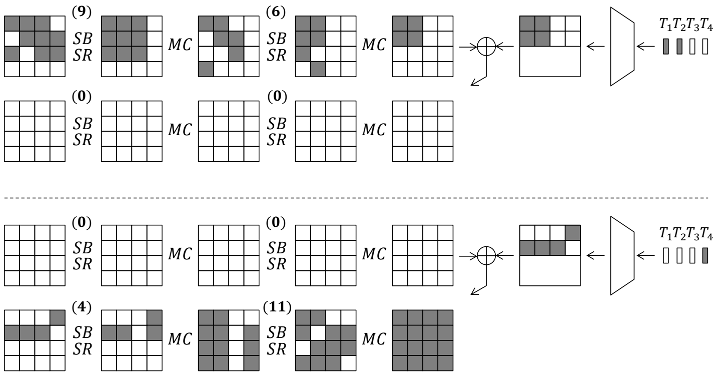


Figure 4: Two Examples of Differential Trails with 15 Active S-boxes.

3.5 Security Analysis of TweAES and TweGIFT Instances

In this section, we provide the various cryptanalysis that we performed on the TweAES and TweGIFT instances. Note that our target is single-key security, and any related-key attacks are out of our scope. The exact security bound, e.g., 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 transformation. We call those operations "$2r$*-round core*," which is depicted for AES and GIFT-64 in Fig. 22. 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. 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.

3.5.1 Security Analysis of TweAES

As explained above, we evaluate the minimum number of differentially and linearly active S-boxes for the 4-round core. The 4-bit tweaks of TweAES are 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 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 Figure 23. Given that the maximum differential probability of the

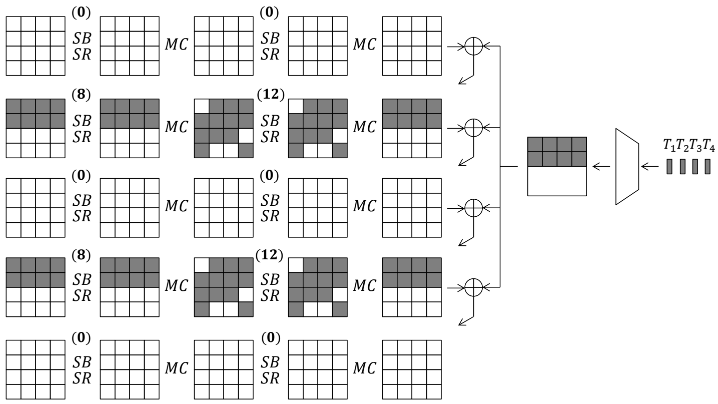


Figure 5: An Examples of Differential Trails with 40 Active S-boxes.

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 includes two 4-round cores.

For TweAES, experimentally computing the lower bound of the number of active S-boxes is also possible. When the tweak input has a non-zero difference, the minimum number of active S-boxes is 40 for the entire construction. This is a tight bound. An example of the differential trails achieving 40 active S-boxes is given in Fig. 24. The probability of the differential propagation is upper bounded by $2^{-6\times 40} = 2^{-240}$.

We argue that the reduced-round versions of TweAES in which the first or the last round is located in the middle of the 4-round core can be attacked for relatively long rounds. Owing to this unusual setting, the attacks here do not threaten the security of full TweAES, 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 more precisely formulated version called the sandwich attack. The boomerang attack divides the cipher E into two parts E_0 and E_1 such that $E = E_1 \circ E_0$, and builds high-probability differentials for E_0 and E_1 almost independently. The attack detects a quartet of plaintext x that satisfy the non-ideal behavior shown below with probability $p^{-2}q^{-2}$, where p and q are the differential probability for $E_0 : \alpha \rightarrow \beta$ and $E_1 : \gamma \rightarrow \delta$, respectively.

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

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

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

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

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

We define E_m of this attack as the first S-box layer in the above E_1. The configuration and the differential trails are depicted in Fig. 25 The probability when E_m is a single S-box layer can be measured by using the boomerang connectivity table (BCT). The trails for E_0

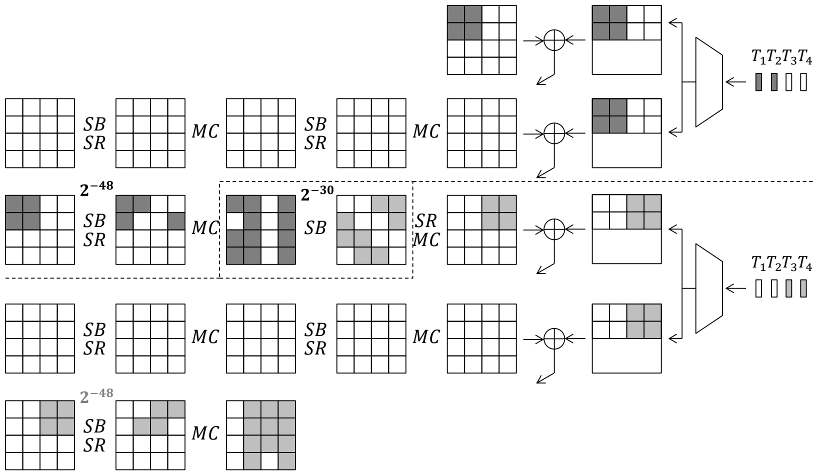


Figure 6: Differential Trails for Boomerang Attacks. The cells filled with black and gray represent active byte positions in E_0 and E_1, respectively.

and E_1 include 4 active S-boxes, hence both of the probability p and q are 2^{-24}. That is, $p^2q^2 = 2^{-96}$. The BCT of the AES S-box shows that the probability for each S-box in E_m is either $2^{-5.4}$, 2^{-6}, or 2^{-7} if both of the input and output differences are non-zero, and is 1 otherwise. Hence, the trail contains 5 active S-boxes with some probabilistic propagation and we assume that the probability of each S-box is 2^{-6}. Then, the probability r_{qar} is $2^{-6\times 5} = 2^{-30}$. In the end, $p^{-2}q^{-2}r_{qua} = 2^{-126}$, which would lead to a valid distinguisher for 7 rounds.

8-Round Impossible Differential Attacks against TweAES. Due to 2 interval rounds between tweaks, distinguishers based on impossible differential attacks can be constructed for relatively long rounds (6 rounds) by canceling the tweak difference with the state difference. The distinguisher is depicted in Fig. 26.

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

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

For the key recovery, two rounds can be appended to the 6-round distinguisher; one is at the beginning and the other is at the end, which is illustrated in Fig. 27. As shown in Fig. 27 the trail includes 8 and 4 active bytes at the input and output states. Partial computations to the middle 6-round distinguisher involve 8 bytes of subkey K_1 and 4 bytes of subkey K_9.

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

1. Choose all tweak values denoted by T^i where $i = 0, 1, \ldots, 2^4 - 1$.
2. For each of T^i, fix the value of inactive 8 bytes at the input, choose all 8-byte values at the active byte positions of the input state. Query those 2^{64} values

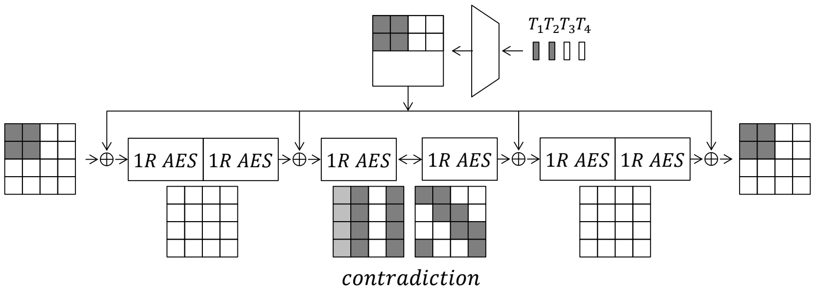


Figure 7: 6-round Impossible Differential Distinguisher. The bytes filled with black, white, and gray have non-zero difference, zero difference, and arbitrary difference, respectively.

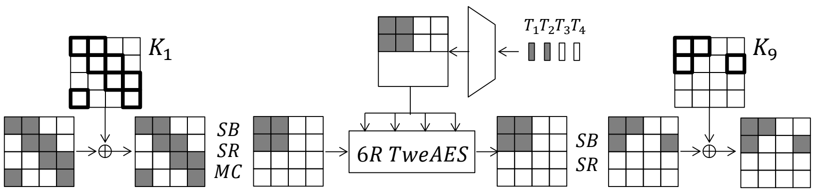


Figure 8: Extension to 8-round Key Recovery

to get the corresponding outputs. Those outputs are stored in the list L^i where $i = 0, 1, \ldots, 2^4 - 1$.

3. For all $\binom{2^4}{2} \approx 2^7$ pairs of L^i and L^j with $i \neq j$, find the pairs that do not have difference in 12 inactive bytes of the output state. About $2^{7+64+64-96} = 2^{39}$ pairs will be obtained.

4. For each of the obtained pairs, the tweak difference is fixed and the differences at the input and output states are also fixed. Those fix both of input and output differences of each S-box in the first round and the last round. Hence, each pair suggests a wrong key.

5. Repeat the procedure 2^{54} times from the first step by changing the inactive byte values at the input. After this step, $2^{39+54} = 2^{103}$ wrong-key candidates (including overlaps) will be obtained. The remaining key space of the involved 12 bytes becomes $2^{96} \times (1 - 2^{-96})^{2^{103}} \approx 2^{96} \times e^{-128} \approx 2^{-88} < 1$. Hence, the 8 bytes of K_1 and 4 bytes of K_9 will be recovered.

6. Exhaustively search the remaining 8 bytes of K_1.

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

Remarks on Other Attacks

- Integral attacks [DKR97, KW02] collect 2^8 distinct values for a particular byte or distinct 2^{32} values for a particular diagonal. Integral attacks exploiting the tweak is difficult because the tweak will not affect all the bits in each byte, which prevents to collect 2^8 distinct values for any byte.

- Meet-in-the-middle attacks [DS08, DFJ13] exploit the 4-round truncated differentials $1 \rightarrow 4 \rightarrow 16 \rightarrow 4 \rightarrow 1$ 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 be exploited in the meet-in-the-middle attack.

Summary. We demonstrated two attacks against reduced-round variants that start from the middle of the 4-round core. Because security of TweAES using tweak difference relies on the fact that the large-weight tweak difference will diffuse fast in the subsequent 2 rounds, those reduced-round analysis will not threaten the security of the full TweAES. From a different viewpoint, one can see the difficulty to extend the analysis by 1 more round from Figs. 25 and 27. The number of involved subkey bytes easily exceeds 16.

3.5.2 Cryptanalysis of TweAES with non-zero tweak from the initial round.

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 and the tweak is non-zero. The main purpose is to show the difficulty of exploiting 4 bits tweak in the attack, thus we do not discuss the case of fixing the tweak. (When tweak is zero, security is the same as the original AES, which can also be applied to TweAES but does not show any vulnerability introduced by TweAES.) The comparison of the number of attacked rounds and the attack complexity for the original AES and TweAES is given in Table 15.

Table 1: Comparison of the Attacks on AES and TweAES exploiting tweak. R, D, T and M denote the number of rounds, data complexity, time complexity and memory complexity, respectively.

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}

Integral Attacks. Because the tweak starts to appear only after the second round, to play with plaintexts is difficult to extend the integral attacks. The most reasonable approach to exploit the tweak is to set the plaintext constant and collect all possible 2^4 tweak inputs. The propagation of the property is given in Fig. 28. Because the plaintext can be 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. 28. 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

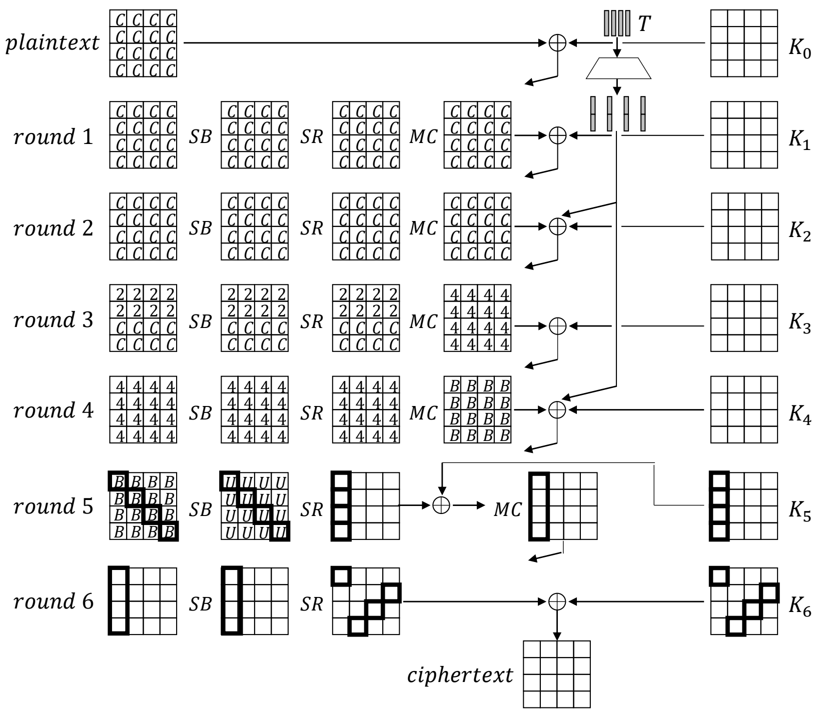


Figure 9: Integral Distinguisher on TweAES via Tweak. '2' represents that two kinds of values appear 8 times each and '4' represents that four kinds of values appear 4 times each. By following the convention, 'B' and 'U' denote 'balanced' and 'unknown' properties, respectively.

the key space is reduced to 32 bits. By using another set of a plaintext with 16 different tweaks, the key space is reduce to 1.

The memory complexity can be saved by first preparing two sets of 16 texts, and then the bytes of K_6 is 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 active at least 4 byte positions. The attack on the original 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 the 2 AES rounds before the first tweak injection.

Impossible Differential Attacks. With 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 distinguisher, as indicated by Fig. 29. Namely, the top left and the bottom left bytes are active with probability 1 in the forward direction, while those byte are inactive with probability 1 in the backward direction.

For the key recovery, one round and two rounds can be appended to the beginning and the end of the 3-round distinguisher, which is illustrated in Fig. 27.

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 distinguisher. 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

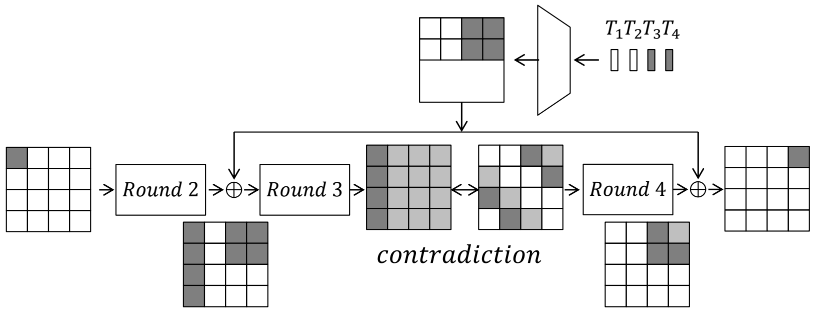


Figure 10: 3-round Impossible Differential Distinguisher using Tweak Difference.

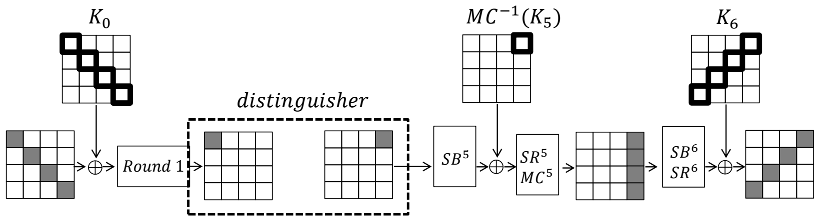


Figure 11: Extension to 6-round Key Recovery

$\binom{2^{32}}{2} \approx 2^{63}$ ciphertext pairs. This can be 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 $X \cdot 2^{-33}$ pairs, the attacker generates the wrong keys 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. 30. This can be done by choosing all possible (2^8) 1-byte difference 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 candidates. Similarly, by choosing 1-byte difference at the input of round 5 and 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 is recovered, the remaining 12 bytes can be recovered by the exhaustive search.

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

Truncated Differential Attacks. So fat 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. 31. The trail

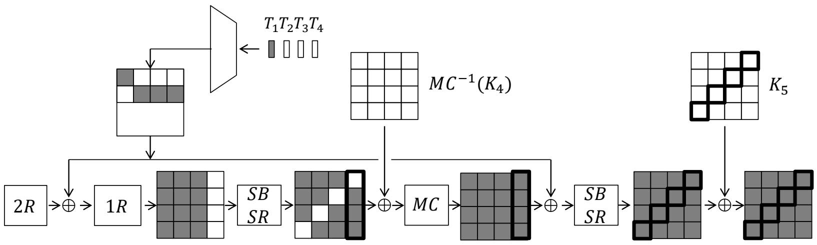


Figure 12: 5-round Truncated Differential Attack using Tweak Difference (type 1).

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 difference before MixColumns in round 4 is 2^{24}. For each of them, the attacker can derive 1 candidate of 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 is stressed in Fig. 31 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 also 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 the 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 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. 32. The trail can be satisfied with probability 2^{-128}; 2^{-64} for the first round and 2^{-64} towards

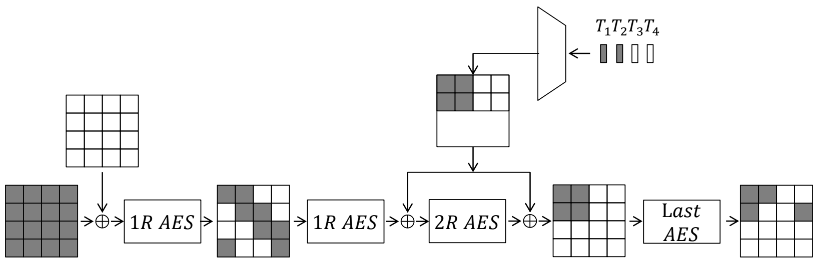


Figure 13: 5-round Truncated Differential Attack using Tweak Difference (type 2).

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 in Fig. 32. 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 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 the attack on more than 5 rounds is an open problem.

3.6 Security Analysis of TweAES-6

We also provide a round reduced version TweAES denoted by TweAES-6 (to be used in one of our applications). In TweAES-6, the number of rounds is reduced from TweAES from 10 to 6 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. 24.

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 the differential cryptanalysis.

Note that AEAD schemes based on the original AES often adopt 4-round AES in the mode, and the minimum number of the active S-boxes for 4-round AES is 25. We designed TweAES-6 to offer the similar security level as 4-round AES, and no attack is known on the 4-round AES in proper modes under the restriction of the birthday-bound query limit.

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

We modeled the differential trail search for TweGIFT with MILP under the constraints that at least 1 bit of tweak has a difference. However, owing to the large state size, it is infeasible to find the tight bound of the maximum probability of the differential characteristic even for the 10-round core. The tool so far provided that the probability of the differential characteristic is upper bounded by $2^{-72.6}$. 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 $2^{-72.6\times 4} = 2^{-300.4}$, 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 of 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 16.

Table 2: 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	upper bound	$2^{-72.6}$	Ours
TweGIFT	10	characteristic probability	lower 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 17.

Table 3: 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 differential mask for the first and last rounds in Table 17 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 is inefficient.

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 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 the active S-boxes for 5 rounds is 5. Hence, the 10-round boomerang trail will certainly require a non-negligible amount of the 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 has already been more 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 17 reaches 18 rounds, we do not think that the boomerang attack can be the best approach for attacking TweGIFT.

3.8 Hardware Performance of the TweAES and TweGIFT Instances

In this section, we provide the hardware implementation details for all our recommended TweGIFT and TweAES versions and compare their hardware overheads respective to their original counterparts GIFT and AES. We give a brief comparison on software implementation of TweAES and AES in supplementary material **??**. For each instantiations, we present both the encryption/decryption (ED) version and only encryption (E) version. The VHDL code of our implementations are synthesized using 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 rams are used. We present the results obtained from the tool after performing place and route process.

Table 4: Implementation results for AES and TweAES on Virtex 7 FPGA.

BC or tBC	LUTs	FF	Slices	Frequency (MHz)	Clock cycles	Throughput (Mbps)
AES-ED	2945	533	943	297.88	11	3466.24
TweAES-ED[4,8,8,2]	2960	534	1044	295.97	11	3444.01
TweAES-ED[8,16,8,2]	2976	534	1129	295.81	11	3442.15
TweAES-ED[16,32,8,2]	3006	534	1134	292.87	11	3407.94
AES-E	1605	524	559	330.52	11	3846.05
TweAES-E[4,8,8,2]	1617	524	574	328.27	11	3819.87
TweAES-E[8,16,8,2]	1632	524	593	325.17	11	3783.79
TweAES-E[16,32,8,2]	1659	524	592	326.56	11	3799.97

Table 18 depicts that the area-overhead (LUT counts) introduced by the tweak injection is negligeable. For Considering the combined encryption-decryption (ED) implementation, TweAES have overheads (in LUTs) of 0.5%, 1.05% and 2.07% for tweak size of 4, 8 and 16 bits respectively. As we move to the encryption (E) only implementation, our recommended TweAES versions have negligeable area overheads of 0.7%, 1.68% and 3.36% respectively. Note that, the reduction in the speed is also negligeable.

Table 19 summerizes the hardware performances of our recommended TweGIFT versions along with the original GIFT. For ED implementation, our recommended version of TweGIFT-64 has an overheads of 0.3% for 4 bit tweaks, and TweGIFT-128 has overheads of 4.04% and 9.89% for tweak size of 4 and 16 bits respectively. As we move to the E implementation, TweGIFT-64 has an overheads of 6.68% for 4 bit tweaks, and TweGIFT-128 has overheads of 4.32% and 5.5% for tweak size of 4 and 16 bits respectively.

Table 5: Implementation results for GIFT and TweGIFT on Virtex 7 FPGA.

BC or tBC	LUTs	FF	Slices	Frequency (MHz)	Clock cycles	Throughput (Mbps)
GIFT-64-ED	615	277	236	455.17	29	1004.51
TweGIFT-64-ED[4,16,16,4]	617	277	234	430.29	29	946.60
GIFT-64-E	449	275	153	596.66	29	1316.77
TweGIFT-64-E[4,16,16,4]	479	275	179	595.09	29	1313.30
GIFT-128-ED	1113	408	432	447.83	41	1398.10
TweGIFT-128-ED[4,32,32,5]	1158	408	419	416.50	41	1300.29
TweGIFT-128-ED[16,32,32,4]	1223	408	428	429.32	41	1340.31
GIFT-128-E	763	403	330	596.30	41	1861.62
TweGIFT-128-E[4,32,32,5]	796	403	332	597.59	41	1865.65
TweGIFT-128-E[16,32,32,4]	805	403	377	598.78	41	1869.36

4 ESTATE: A tBC Based Nonce-misuse Resistant AEAD

The AEAD ESTATE uses the following three instances of TweAES and TweGIFT ciphers. For the sake of simplicity we use TweAES, TweAES-6, and TweGIFT for the underlying tBCs (we optimize using the tweakable blockcipher instance names as less as possible). Precisely, the underlying tBCs are as follows.

- TweAES is the same as TweAES[4, 8, 8, 2],
- TweAES-6 is the round reduced version of TweAES, such that the number of rounds is reduced to 6 from 10, and
- TweGIFT is the same as TweGIFT-128[4, 32, 32, 5].

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

ESTATE is roughly based on the MAC-then-Encrypt paradigm. It is composed of an FCBC like MAC, we call FCBC*, and the OFB mode of encryption. ESTATE is parametrized by its underlying tweakable block cipher $\widetilde{\mathsf{E}}$-$n/\tau/\kappa$. It operates on n-bit data blocks at a time using a tweakable block cipher. Complete specification of ESTATE is presented in Algorithm 1. The pictorial description is given in Figure 14, 15, and 16.

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. The distinctness in tweaks is used to separate different cases based on the length of associated data and message. We represent a tweak value in 4 bits and the tweak value i represents the 4-bit binary representation of integer i. The processing of first block (i.e. nonce N) uses the tweak value 1. The intermediate blocks are always processed with tweak 0, to minimize the overheads

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 $(C[1]\| \cdots \|C[m])$

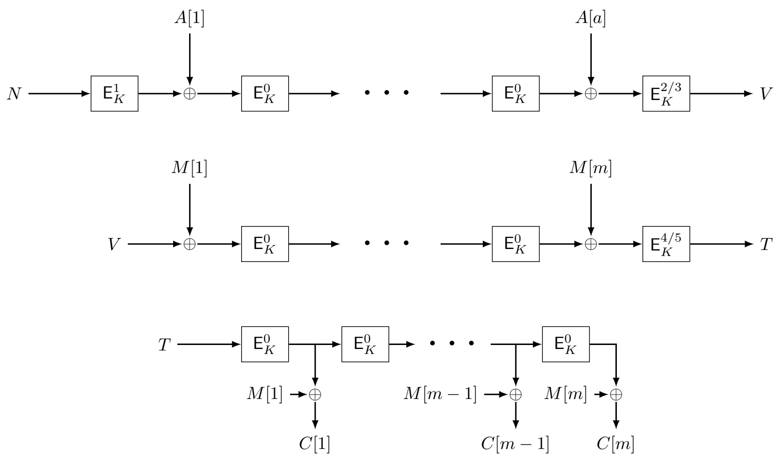


Figure 14: ESTATE with a AD blocks and m message blocks

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 non-last blocks in the MAC function. For all other tweakable block cipher calls, i.e. for processing the last block in MAC function and the full OFB processing, $\widetilde{\mathsf{E}}$ is used as usual. Further $\widetilde{\mathsf{F}}$, is always employed with tweak value 15, in order to maintain maximum distance between the 0

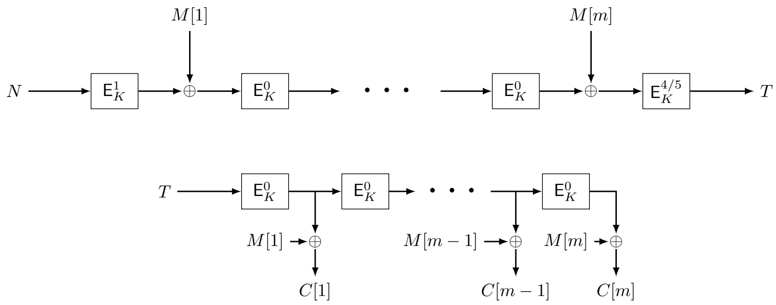


Figure 15: ESTATE with empty AD and m message blocks

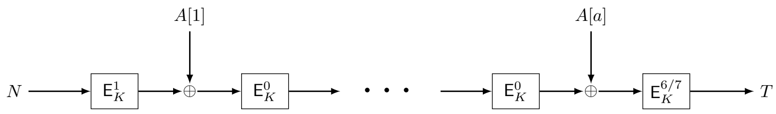


Figure 16: ESTATE with a AD blocks and empty message

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

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

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

function $\mathsf{FCBC}^{*}[\widetilde{\mathsf{E}},\widetilde{\mathsf{F}}](K, T, D, t)$
  $D[1]\| \cdots \|D[d] \leftarrow D$
  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 $\mathsf{OFB}[\widetilde{\mathsf{E}}](K, T, M)$
  $M[1]\| \cdots \|M[m] \leftarrow M$
  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

Tweak Choices for sESTATE. For sESTATE, we always use tweak 15 for the round-reduced block ciphers to maximize the distance with other tweaks, most importantly tweak 0 whose inputs and outputs are observed through OFB. In this way, we make TweAES-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 tweakable block cipher with 4-bit tweak primarily for the purpose of various domain separations such as the type of the current data (associated data or message), completeness of the final data block (partial or full), whether the associated data and/or message is empty etc. Note that, without the use of these tweaks, these domain separations would cost a few constant field multiplications and/or additional block cipher invocations, which would in turn increase the hardware footprint as well as decrease the energy efficiency and throughput for short messages.

3. **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 essentially 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 essentially ensures the RUP security of the mode.

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

4. **Rationale of the Tweak Injection Positions for TweAES.** The overall structure of TweAES is similar as KIASU-BC [JNP14b], which takes a 64-bit tweak as input and inject it to top 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 attacks. 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 the 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 LSB from the implementation reasons.

   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 a 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 ESTATE mode of operation with $\widetilde{\mathsf{E}}$:=TweAES block cipher. Here the size of the key, nonce and tag are 128 bits each.

- ESTATE_TweGIFT: This AEAD scheme is obtained by instantiating 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.

- : This AEAD scheme is obtained by instantiating 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 , for higher throughput demanding, and energy-constrained applications.

4.5 Security of ESTATE

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

Theorem 2 (AERUP security of ESTATE)**.** *Consider ESTATE authenticated encryption scheme based on 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, 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 any 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. The adversary has encryption complexity σ_e, decryption complexity σ_d, and verification complexity σ_v, with $\sigma_e + \sigma_d + \sigma_v = \sigma$, and operates in time t. As a first step, we 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 $t' \approx t$. As a second step, we switch from $P_0, \ldots, P_7$ to a random functions $R_0, \ldots, R_7$ where $R_i \xleftarrow{\$} \mathrm{F}(n)$ at the cost of $\binom{\sigma}{2}/2^n$. 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.

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

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 $(A_i^\star, C_i^\star, T_i^\star)_{i=1}^{q_v}$ to the verification oracle, where the block lengths of $A_i^\star$ and $C_i^\star$ are denoted by $a_i^\star$ and $c_i^\star$, 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 $(A_i^\star, C_i^\star, T_i^\star)$ 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 $\mathcal{O}_{\text{re}}$ 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 $\mathcal{O}_{\text{id}}$ consists of three oracles $(\$, \mathsf{S}, \perp)$. The verification oracle $\perp$ simply responds with the $\perp$-sign for each input $(A_i^\star, C_i^\star, T_i^\star)$. 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

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

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^+:

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

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 $1 \leq k \leq p$, and $Y_i^{\circledcirc}[k] \xleftarrow{\$} \{0,1\}^n$, for $p+1 \leq k \leq \ell_i^{\circledcirc}$. Finally, we set all the $X_i^{\circledcirc}[j]$ values for $j = 1, \ldots, \ell_i^{\circledcirc}$. Finally, when the sampling of internal values is over, $\mathcal{O}_{\text{id}}$ returns all the internal values. These are $(X_i^+, Y_i^+) = (X_i^+[1 \ldots \ell_i^+], Y_i^+[1 \ldots \ell_i^+]), (U_i^+, V_i^+) = (U_i^+[1 \ldots m_i^+], V_i^+[1 \ldots m_i^+])$, for each encryption query $(A_i^+, M_i^+, C_i^+, T_i^+)$; $(U_i^-, V_i^-) = (U_i^-[1 \ldots c_i^-], Y_i^-[1 \ldots c_i^-])$, for each decryption query $(A_i^-, M_i^-, C_i^-, T_i^-)$, and $(X_i^\star, Y_i^\star) = (X_i^\star[1 \ldots \ell_i^\star], Y_i^\star[1 \ldots \ell_i^\star]), (U_i^\star, V_i^\star) = (U_i^\star[1 \ldots m_i^\star], V_i^\star[1 \ldots m_i^\star])$, 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

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

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 $\mathcal{O}_{\text{id}}$ 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. $\mathsf{Acc}_{\mathsf{XX1}}$: $\exists (j, \circledast) \preceq (i, \circledcirc) : X_i^{\circledcirc}[a_i^{\circledcirc}] = X_j^{\circledast}[a_j^{\circledast}]$, where $A_i^{\circledcirc} \neq A_j^{\circledast}$.
2. $\mathsf{Acc}_{\mathsf{XX2}}$: $\exists (j, \circledast) \preceq (i, \circledcirc) : X_i^{\circledcirc}[\ell_i^{\circledcirc}] = X_j^{\circledast}[\ell_j^{\circledast}]$.
3. $\mathsf{Acc}_{\mathsf{XX3}}$: $\exists (j, \circledast) \preceq (i, \circledcirc), k, k'(\neq k) : X_i^{\circledcirc}[k] = X_j^{\circledast}[k']$.
4. $\mathsf{Acc}_{\mathsf{XX4}}$: $\exists (j, \circledast) \preceq (i, \circledcirc), k \leq a_i^{\circledcirc} : X_i^{\circledcirc}[k] = X_j^{\circledast}[k]$, where $A_i^{\circledcirc}[1 \ldots k] \neq A_j^{\circledast}[1 \ldots k]$.
5. $\mathsf{Acc}_{\mathsf{XX5}}$: $\exists (j, \circledast) \preceq (i, \circledcirc), k > a_i^{\circledcirc} : X_i^{\circledcirc}[k] = X_j^{\circledast}[k]$, where $A_i^{\circledcirc} = A_j^{\circledast}, M_i^{\circledcirc}[1 \ldots (k - a^{\circledcirc})] \neq M_j^{\circledast}[1 \ldots (k - a^{\circledcirc})]$.
6. $\mathsf{Acc}_{\mathsf{XU}}$: $\exists (j, \circledast), (i, \circledcirc), k(\neq 1, \ell_i^{\circledcirc}), k'$ such that $U_i^{\circledcirc}[k'] = X_j^{\circledast}[k]$.
7. $\mathsf{Acc}_{\mathsf{UU}}$: $\exists (j, \circledast) \preceq (i, \circledcirc), k, k'$ with $\big(\circledcirc = + \text{ or } U_i^{\circledcirc}[1] \neq U_j^{\circledast}[k - k' + 1]\big)$ 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, $\mathsf{Acc}_{\mathsf{XX}}$ 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 $\mathsf{Acc}_{\mathsf{XU}}$ corresponds to accidental collisions between an input to R in the authentication and one in the encryption part. Event $\mathsf{Acc}_{\mathsf{UU}}$ 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 $X_{\rm id}$ and $\Theta_{\rm bad}$ be as defined as above. Then,*

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

Proof. By applying the union bound,

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

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 $\mathsf{Acc}_{\mathsf{XX}}$. 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

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

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

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

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

$$\frac{\mathit{Pr}[X_{\rm re} = \tau]}{\mathit{Pr}[X_{\rm id} = \tau]} = 1\,.$$

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.

4.6 Hardware Implementation

In this section, we describe the hardware 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 implement both the instantiations of SUNDAE by own using exactly the same interface and following the same architectural properties to have a fair comparison. In addition, we use the AES only encryption core provided in GMU Caesar Package [GMU16] for both ESTATE_TweAES and SUNDAE_AES-128. The details are given below.

4.6.1 Hardware Architecture of ESTATE_TweAES

In this section, we describe the implementation of combined encryption/decryption architecture of ESTATE_TweAES. It is described in Fig. 17. The main modules are described below:

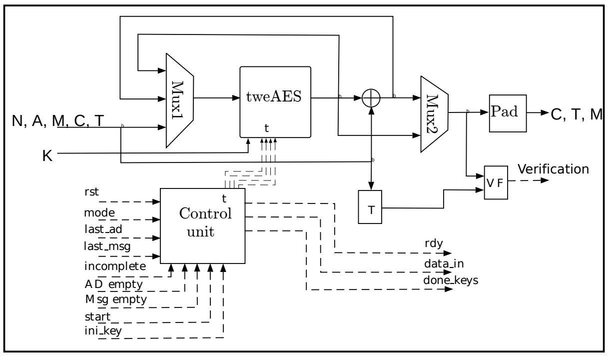


Figure 17: Hardware Architectures of ESTATE_TweAES

- **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. 18 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.

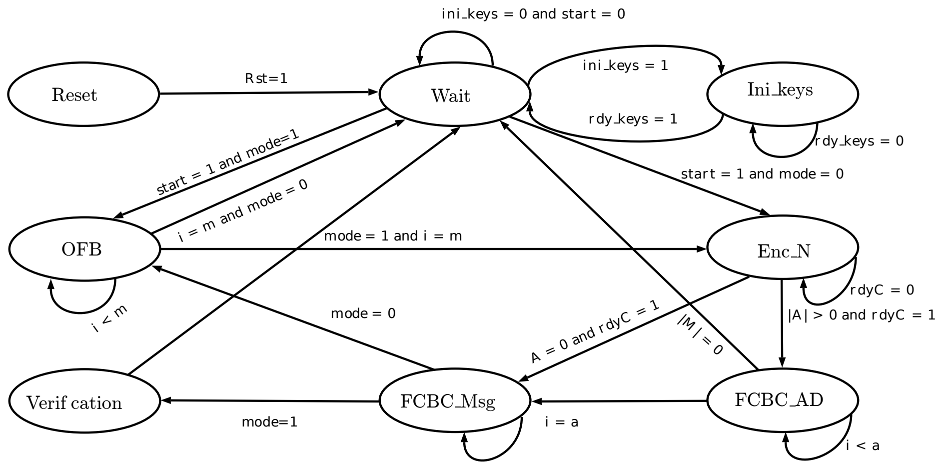


Figure 18: Finite State Machine

6. FCBC_Msg: Same as FCBC_AD but here the input is the message block, the last output is the tag.
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 17. Depending on the facts

- 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 6.

Table 6: 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

4.6.2 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 hardware implementation codes of ESTATE and SUNDAE members are written in VHDL and are implemented on Virtex 7 xc7vx485t (Vivado v.2018.2.2). We use the RTL approach and use 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. The detailed implementation results are depicted in Table 7.

Table 7: ESTATE and SUNDAE (combined enc/dec 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_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

Table 8: Throughput Comparison for Short Message Processing

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

We can observe that the overhead introduced by the implementation of STATE is more significant in case of ESTATE_TweGIFT-128 since GIFT is significantly smaller than AES. The latency for TweAES 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 area-efficient than SUNDAE_GIFT-128.

4.6.3 Short Message Processing for SUNDAE and ESTATE

Regarding short message processing, we only compare between ESTATE_TweAES and SUNDAE_AES-128. We can briefly mention the difference in the number of clock cycles by taking an example of one input data block (16 bytes). To calculate the values, we make the following assumption. A possible nonce based version of SUNDAE prepends the nonce with the associated data (this assumption is also used in the NIST submitted version of SUNDAE [BBP+19]). Hence considering the nonce as the first block of the associated data, we assume the associated data length is always 16 bytes or one block. When we say that the message length is 16 bytes, then overall we consider one block associated data (i.e, the nonce) and one block message. In this case, SUNDAE invokes four block cipher calls, such that we need one block cipher call to encrypt the constant, one block cipher call to encrypt the nonce and two block cipher calls for the message. ESTATE avoids the block cipher call for the constant and makes three block cipher calls. In our architecture, to process a 16-byte message, ESTATE_TweAES requires 31 cycles where as SUNDAE_AES-128 needs 41 clock cycles. Details with larger messages are given in Table 8 below. Note that, the throughputs for both the schemes converge to the same value with an increase in the input lengths.

4.6.4 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 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 to the message which 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.

4.6.5 Very Small Implementation of ESTATE_TweAES

We also introduce a tiny FPGA implementation of ESTATE_TweAES. The main motivation for this 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. The results depict that the tradeoff remain 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 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 smaller. It is interesting to see that we can have an 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. In Table 9 we show the experimental results.

Table 9: Very Small Implementation 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	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

4.6.6 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.

4.6.7 Benchmarking ESTATE

We provide a benchmark of the hardware implementation results of all the members in the ESTATE family using some of the implementation listed in Athena website [ATHa] along with the implementation results in [NMSS18, CIMN17a, CIMN17b, CDNY18a, CDNY18b] on Virtex 7. The results depict that ESTATE provides a very 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). Note that, we directly

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

use the AES only encryption core provided in the GMU Caesar Package [GMU16] and we use our own implementation for TweGIFT-128.

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 Fig. 19 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.

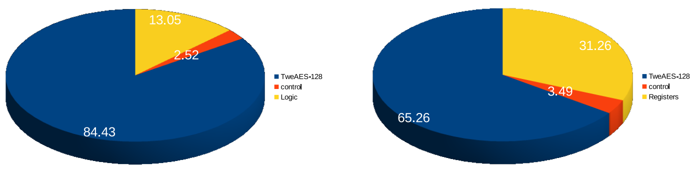


Figure 19: Distribution of #LUTs (left) and #FF (right) for ESTATE_TweAES implementation

5 Other Applications of Short Tweak tBC

Now, we present some use cases where an efficient tBC could be beneficial. Please see supplementary material A for details on security notions used here.

5.1 Reducing the Key Size in Multi-Keyed Modes of Operation

Several block cipher based modes of operation employ a block cipher with multiple independently sampled keys. In general, this is done either to boost the security, or to simplify the analysis of the overall construction. The number of keys can be naturally reduced to a single key by replacing the multi-keyed block cipher with a single keyed tBC where distinct tweaks are used to simulate independent block cipher instantiations. Proposition 1 below gives the theoretical justification for this remedy. The proof is obvious from the definitions of (tweakable) random permutation.

Proposition 1. *For some fixed $t \in \mathbb{N}$, and $k \in [2^t]$. Let* $(\Pi_1, \ldots, \Pi_k) \leftarrow\$ (\mathsf{Perm}[n])^k$ *and* $\widetilde{\Pi} \leftarrow\$ \mathsf{TPerm}[t, n]$*. Let* $\mathcal{O}_{\Pi;k}$ *and* $\mathcal{O}_{\widetilde{\Pi};k}$ *be two oracles giving bidirectional access to* $(\Pi_1, \ldots, \Pi_k)$*, and* $(\widetilde{\Pi}^1, \ldots, \widetilde{\Pi}^k)$*, respectively. Then, for all*

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 [JNP16a]	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

distinguisher $\mathcal{A}$, we have

$$\Delta_{\mathcal{A}}(\mathcal{O}_{\Pi;k};\mathcal{O}_{\widetilde{\Pi};k}) := \left|\Pr[\mathcal{A}^{\mathcal{O}_{\Pi;k}}=1]-\Pr[\mathcal{A}^{\mathcal{O}_{\widetilde{\Pi};k}}=1]\right| = 0.$$

Now, we demonstrate the utility of this idea through some examples.

5.1.1 FCBC MAC

FCBC mode is a 3-key message authentication code, by Black and Rogaway [BR05], which is defined as follows:

$$\Sigma := \mathsf{E}_{K_0}\Big(M_{m-1}\oplus \mathsf{E}_{K_0}\big(M_{m-2}\oplus \mathsf{E}_{K_0}\big(\cdots\oplus(M_2\oplus \mathsf{E}_{K_0}(M_1\oplus IV))\big)\big)\Big),$$
$$\mathsf{FCBC}[\mathsf{E}](IV,M) := \mathsf{E}_{K_t}\big(\Sigma\oplus \mathsf{ozp}(M_m)\big)\,,\ \text{where } t\leftarrow(|M_m|=n)?\ 1:2.$$

Here IV is called the initial vector, which is generally set to a fixed constant value. But one can also use a random IV or use some other way (like encrypted nonce) to generate the IV.

FCBC has not received much appreciation in its existing 3-key form, even though it offers the similar security to CMAC [IK03, CMA05, CJN22a, CJN22b]. However, we observe CMAC uses an n-bit state for the final message block masking and also uses a block cipher call to generate the mask. Keeping these in mind, we define Twe-FCBC, as follows:

$$\Sigma := \widetilde{\mathsf{E}}_K^0\Big(M_{m-1}\oplus \widetilde{\mathsf{E}}_K^0\big(M_{m-2}\oplus \widetilde{\mathsf{E}}_K^0\big(\cdots\oplus(M_2\oplus \widetilde{\mathsf{E}}_K^0(M_1\oplus IV))\big)\big)\Big),$$
$$\mathsf{Twe\text{-}FCBC}[\widetilde{\mathsf{E}}](IV,M) := \widetilde{\mathsf{E}}_K^t\big(\Sigma\oplus \mathsf{ozp}(M_m)\big)\,,$$

where $t\leftarrow(|M_m|=n)?\ 1:2$. It is clear that Twe-FCBC is a variant of FCBC, that follows the principle established in Proposition 1, and replaces the 3 block ciphers E_{K_0}, E_{K_1}, E_{K_2} with $\widetilde{\mathsf{E}}_K^0$, $\widetilde{\mathsf{E}}_K^1$ and $\widetilde{\mathsf{E}}_K^2$, respectively. Using Proposition 1 and [JN16, Theorem 3 and Remark 5], we get the PRF security for Twe-FCBC in a straightforward manner in Proposition 2.

Proposition 2. *Assuming all queries are of length $\ell \leq 2^{n/4}$, and $\sigma \leq q\ell$, we have*

$$\mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{Twe\text{-}FCBC}[\widetilde{\mathsf{E}}]}(t,q,\sigma) \leq \mathbf{Adv}^{\mathsf{TPRP}}_{\widetilde{E}}(t',\sigma) + O\left(\frac{q^2}{2^n}\right).$$

Twe-FCBC vs CMAC: Here we show two major advantages of Twe-FCBC over CMAC, which is both SP 800-38B and ISO/IEC/9797-1 standard:

(a) No need to hold an additional state for final message block masking,

(b) In addition, Twe-FCBC can also avoid the additional block cipher call used to generate the masking. Due to backward compatibility, except the last block we have used the original block cipher. So the performance overhead due to nonzero tweak only applies to the last block cipher call. This features ensures to get similar performance (or even better) for long message.

5.1.2 Double Block Hash-then-Sum:

The very basic version of Double-block Hash-then-Sum or DbHtS [DDNP18], is defined as below

$$\mathsf{DbHtS}(M) := \mathsf{E}_{K_1}(\Sigma) \oplus \mathsf{E}_{K_2}(\Theta),$$

where H is a $2n$-bit output hash function, $(\Sigma, \Theta) := \mathsf{H}_L(M)$, and L, K_1, K_2 are all sampled independently. DbHtS is a generic design paradigm that captures several popular BBB secure MACs such as 3kf9, SUM_ECBC, PMAC_Plus and LightMAC_Plus. Using a tBC, the two block cipher keys can now simply be replaced by a single tweakable block cipher key and two distinct tweaks. Formally, we define Twe-DbHtS as follows

$$\mathsf{Twe\text{-}DbHtS}(M) := \widetilde{\mathsf{E}}^1_K(\Sigma) \oplus \widetilde{\mathsf{E}}^2_K(\Theta).$$

Moreover, one can also generate the dedicated hash key using the tweakable block cipher key itself. Suppose the hash function is block cipher based, then the tBC key can be used along with a different tweak to replace the dedicated hash key. In all other cases, the hash key can be derived as $L := (\widetilde{\mathsf{E}}^0_K(0) \| \widetilde{\mathsf{E}}^0_K(1) \| \cdots \| \widetilde{\mathsf{E}}^0_K(h-1))$, where $|L| = hn$. Since $\widetilde{\mathsf{E}}^0_K(i)$'s are sampled in without replacement manner, this adds an additional factor of $\frac{h^2}{2^n}$ due to the PRP-PRF switching, which can be ignored for small h. One can easily verify that due to Proposition 1, the result on DbHtS [DDNP18, Theorem 2.(iii)] also applies to Twe-DbHtS. Formally, the security of Twe-DbHtS is given by Proposition 5.

$$\begin{aligned}\mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{Twe\text{-}DbHtS}[\mathsf{H},\widetilde{\mathsf{E}}]}(q,\ell,t) \quad &\leq \quad 2\mathbf{Adv}^{\mathsf{TPRP}}_{\widetilde{\mathsf{E}}}(2q,t') \\ &\quad\; + \mathbf{Adv}^{\mathsf{PRF}}_{C_3^*[H,\pi_0,\pi_1,\pi_2]}(q,\ell,t).\end{aligned}$$

In this way, we have one-key versions of different well known designs 3kf9, SUM_ECBC, PMAC_Plus, LightMAC_Plus etc. We note that one key version of PMAC_Plus based on solely block cipher has been proposed [DDN+17a]. However, one key version of the other designs either are not known or it can be shown to be secure up to the birthday bound.[4]

f9 vs Twe-3kf9: The 3rd Generation Partnership Project (3GPP) proposes f9 as its first MAC algorithm which provides birthday bound security. Zhang et.al proposes 3kf9 that achives beyond the birthday bound security but at the cost of 3 independent keys. We can directly use Twe-3kf9 here which provides security beyond the birthday bound with just a single key.

[4] 1kf9 is proposed in ePrint [DDN+17b], which later found to be attacked in birthday complexity [LNS18].

5.1.3 Other Designs:

Several more constructions use multiple keys to achieve better security. Some notable examples are (1) sum of two permutations (2) Encrypted Davis Meyer (EDM) [CS16] (3) Encrypted Wegman Carter Davis Meyer (EWCDM) [CS16] (4) Chained LRW2 (CLRW2) [LST12] (5) GCM-SIV-2 [IM16] and (6) The Benes Construction [Pat08b]. One can apply similar treatment as above to reduce these multi-keyed constructions to single-keyed designs with exactly same security guarantee. We provide some details on the tBC variants for (1)-(6) in the supplementary material B.

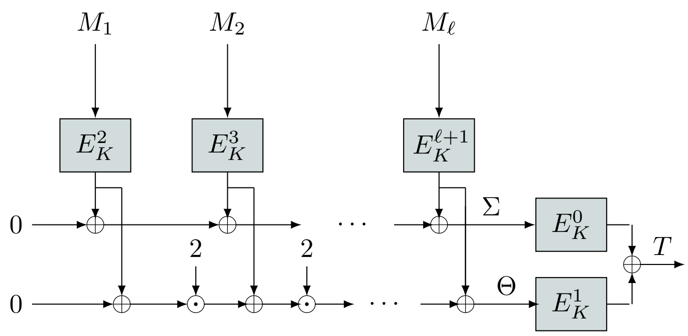
Remark: Note that, OCB like schemes use encrypted nonce as the masking value, so the above idea (i.e. removal of the masking value using tBC) is not applicable to them. Still, the advantage of using tBC in such cases is that we do not have to update the mask for each block, rather the block counter, which is used as the tweak takes care of that.

5.2 Efficient Processing for Short Messages

In energy constrained environments, reducing the number of primitive invocations is crucial, as for short messages, this reduction leads to efficient energy consumption. The tBC framework can be used to reduce the number of primitive invocations for many existing constructions such as LightMAC_Plus [Nai17].

LightMAC_Plus is a counter-based PMAC_Plus in which $\langle i\rangle_m \| M_i$ is input to the i-th keyed block cipher call, where $\langle i\rangle_m$ is the m-bit binary representation of i and M_i is the i-th message block of $n-m$ bits. The counters ensure that there is no input collision, which indirectly helps in negating the influence of ℓ. LightMAC_Plus has been shown to have $O(q^3/2^{2n})$ PRF security. However, it has two shortcomings: (i) it requires 3 keys, and (ii) it has rate $1-m/n$ which increases the number of block cipher calls. This is highly undesirable in low memory and energy constrained scenarios.

To resolve these shortcomings specifically for short to moderate length messages (slightly less than 1 Megabyte), we propose Twe-LightMAC_Plus, which can be viewed as an amalgamation of LightMAC_Plus [Nai17] and PMACx [LN17]. The key idea is to use the block counters as tweak in hash layer, while having distinct tweaks for the finalization. The pictorical description of the algorithm is given in Fig. ??. It is easy to see that Twe-LightMAC_Plus is single-keyed and it achieves rate 1. This reduces the number of block cipher calls by up to 50% for short messages, which has direct effect on reducing the energy consumption.



We claim that Twe-LightMAC_Plus is as secure as LightMAC_Plus. Formally, we have the following security result.

Proposition 3. *For* $q \le 2^{n-1}$,

$$\mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{Twe\text{-}LightMAC\_Plus}[\widetilde{\mathsf{E}}]}(t,q,\ell) \le \mathbf{Adv}^{\mathsf{TPRP}}_{\widetilde{E}}(t',q\ell) + O\left(\frac{q^3}{2^{2n}}\right).$$

The proof can be found in C. We note that similar improvements can also be applied to PMAC, PMAC_Plus.

5.3 Elastic-Tweak vs XE and XEX

The XE and XEX modes, by Rogaway [Rog04], are two reasonably efficient ways of converting a block cipher into a tweakable block cipher. These methods are widely used in various modes such as PMAC [BR02], OCB [RBB03], COPA [ABL+15], ELmD [DN15] etc. The XE scheme to generate a TBC $\widetilde{\mathsf{E}}$ from a BC E is defined as

$$\mathsf{XE}: \quad \widetilde{\mathsf{E}}_K^{i_1,\cdots,i_t}(M) := \mathsf{E}_K(\Delta \oplus M)$$

where $\Delta = \alpha_1^{i_1} \cdots \alpha_t^{i_t} \cdot L$. Here L is generally an n-bit secret state, which is generated using block cipher call.[5] It is sufficient for us to compare XE and tBC, as XEX is much similar to XE. Now one may think of using XE instead of tBC to convert multi-keyed modes to single-keyed mode, as above. But in comparison to tBC, XE lacks two important features:

(i) Degradation to Birthday Bound Security: XE (and XEX) is proved to be birthday bound secure TBC mode. This is not a big issue for birthday secure multi-keyed modes. In fact, the CMAC mode can be viewed as an example that uses the XE mode, much in the same way as Twe-FCBC uses tBC. However, if we use XE in multi-keyed applications such as DbHtS or XOR2, the security of these constructions would degrade to birthday bound. So, we cannot use XE or XEX, in a black box fashion, to instantiate the tweakable variants, without a significant degradation in the security of the modified mode. In contrast, tBC directly works on the block cipher level, and hence does not suffer from such degradation unless the block cipher is itself weak.

(ii) Additional Computational and Storage Overheads: The XE mode requires, pre-computation of the secret state L, (ii) an additional block cipher invocation to generate L, and (iii) an additional storage to store L. This cannot be neglected in constrained computation and communication environments, as mentioned earlier. On the other hand, the tBC framework incurs far less overheads. Here we provide a motivating example.

COLM vs Twe-COLM: COLM is an authenticated encryption included in the CAESAR portfolio. Here we show how we can define a tweakable variant of COLM with several advantages over COLM. Let us define Twe-COLM, which is same as COLM [ABD+] except that: (i) it does not have any masking, (ii) it uses *tBC* with a 16 bit tweak (with 13 bits used to denote the block number and 3 bits for domain separation). Twe-COLM has the following several major advantages over COLM:

(a) No need to hold an additional state for final message block masking,

(b) It avoids the additional block cipher call used to generate the masking, which is typically important for shorter messages making it energy efficient.

(c) No need for any multiplications by 2, 3, 3^2, 7, 7^2, which saves hardware area.

Similar tBC variants can be defined for PMAC [Rog04] (based on XE), COPE [ABL+15] (based on XEX) etc. much along the same line as Twe-LightMAC_Plus and COLM.

[5] Alternative constructions to define Δ can be found in [CS08, GJMN16].

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.

[ABD+] Elena Andreeva, Andrey Bogdanov, Nilanjan Datta, Atul Luykx, Bart Mennink, Mridul Nandi, Elmar Tischhauser, and Kan Yasuda. COLM v1. CAESAR Competition.

[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.

[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.

[BBP+19] Subhadeep Banik, Andrey Bogdanov, Thomas Peyrin, Yu Sasaki, Siang Meng Sim1, Elmar Tischhauser, , and Yosuke Todo. SUNDAE-GIFT v1.0, 2019. https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/SUNDAE-GIFT-spec.pdf.

[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.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In *Advances in Cryptology - CRYPTO 2016. Proceedings, Part II*, pages 123–153, 2016.

[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.

[BR02] John Black and Phillip Rogaway. A block-cipher mode of operation for parallelizable message authentication. In *Advances in Cryptology - EUROCRYPT 2002. Proceedings*, pages 384–397, 2002.

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

[BS90] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. In *Advances in Cryptology - CRYPTO '90, Proceedings*, pages 2–21, 1990.

[CAN] Can fd standards and recommendations. https://www.can-cia.org/news/cia-in-action/view/can-fd-standards-and-recommendations/2016/9/30/.

[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.

[CCM04] Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality . NIST Special Publication 800-38C, 2004. National Institute of Standards and Technology.

[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.

[CDJ+20] Avik Chakraborti, Nilanjan Datta, Ashwin Jha, Cuauhtemoc Mancillas-López, Mridul Nandi, and Yu Sasaki. ESTATE: A lightweight and low energy authenticated encryption mode. *IACR Trans. Symmetric Cryptol.*, 2020(S1):350–389, 2020.

[CDJ+21] Avik Chakraborti, Nilanjan Datta, Ashwin Jha, Cuauhtemoc Mancillas-López, Mridul Nandi, and Yu Sasaki. Elastic-tweak: A framework for short tweak tweakable block cipher. In Avishek Adhikari, Ralf Küsters, and Bart Preneel, editors, *Progress in Cryptology - INDOCRYPT 2021 - 22nd International Conference on Cryptology in India, Jaipur, India, December 12-15, 2021, Proceedings*, volume 13143 of *Lecture Notes in Computer Science*, pages 114–137. Springer, 2021.

[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.

[CJN22a] Soumya Chattopadhyay, Ashwin Jha, and Mridul Nandi. Towards tight security bounds for omac, XCBC and TMAC. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part I*, volume 13791 of *Lecture Notes in Computer Science*, pages 348–378. Springer, 2022.

[CJN22b] Soumya Chattopadhyay, Ashwin Jha, and Mridul Nandi. Towards tight security bounds for omac, XCBC and TMAC. *IACR Cryptol. ePrint Arch.*, page 1234, 2022.

[CLS15] Benoit Cogliati, Rodolphe Lampe, and Yannick Seurin. Tweaking even-mansour ciphers. In *CRYPTO 2015. Proceedings, Part I*, pages 189–208, 2015.

[CMA05] Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. NIST Special Publication 800-38B, 2005. National Institute of Standards and Technology.

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

[Cro00] Paul Crowley. Mercy: A fast large block cipher for disk sector encryption. In *Fast Software Encryption – FSE 2000. Proceedings*, pages 49–63, 2000.

[CS08] Debrup Chakraborty and Palash Sarkar. A general construction of tweakable block ciphers and different modes of operations. *IEEE Trans. Information Theory*, 54(5):1991–2006, 2008.

[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.

[CS16] Benoît Cogliati and Yannick Seurin. EWCDM: an efficient, beyond-birthday secure, nonce-misuse resistant MAC. In *CRYPTO 2016, Proceedings, Part I*, pages 121–149, 2016.

[DDN+17a] Nilanjan Datta, Avijit Dutta, Mridul Nandi, Goutam Paul, and Liting Zhang. Single key variant of pmac_plus. *IACR Trans. Symmetric Cryptol.*, 2017(4):268–305, 2017.

[DDN+17b] Nilanjan Datta, Avijit Dutta, Mridul Nandi, Goutam Paul, and Liting Zhang. Single key variant of pmac_plus. *IACR Cryptology ePrint Archive*, 2017:848, 2017.

[DDNP18] Nilanjan Datta, Avijit Dutta, Mridul Nandi, and Goutam Paul. Double-block Hash-then-Sum: A Paradigm for Constructing BBB Secure PRF. *IACR Trans. Symmetric Cryptol.*, 2018(3):36–92, 2018.

[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.

[DHT17] Wei Dai, Viet Tung Hoang, and Stefano Tessaro. Information-Theoretic Indistinguishability via the Chi-Squared Method. In *Advances in Cryptology - CRYPTO 2017. Proceedings, Part III*, pages 497–523, 2017.

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The Block Cipher Square. In Eli Biham, editor, *FSE 1997*, volume 1267 of *LNCS*, pages 149–165. Springer, 1997.

[DL17] Christoph Dobraunig and Eik List. Impossible-differential and boomerang cryptanalysis of round-reduced Kiasu-BC. In Helena Handschuh, editor, *CT-RSA 2017*, volume 10159 of *LNCS*, pages 207–222. Springer, 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.

[EPC] Electronic product code (epc) tag data standard (tds). http://www.epcglobalinc.org/standards/tds/.

[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.

[FLS+10] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker. The skein hash function family. In *Submission to NIST (round 3), 7(7.5):3*, 2010.

[GCM07] Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. NIST Special Publication 800-38D, 2007. National Institute of Standards and Technology.

[GJMN16] Robert Granger, Philipp Jovanovic, Bart Mennink, and Samuel Neves. Improved masking for tweakable blockciphers with applications to authenticated encryption. In *EUROCRYPT 2016. Proceedings, Part I*, pages 263–293, 2016.

[GL15] Shay Gueron and Yehuda Lindell. GCM-SIV: full nonce misuse-resistant authenticated encryption at under one cycle per byte. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 109–119, 2015.

[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.

[IK03] Tetsu Iwata and Kaoru Kurosawa. OMAC: One-Key CBC MAC. In *FSE*, pages 129–153, 2003.

[IM16] Tetsu Iwata and Kazuhiko Minematsu. Stronger security variants of GCM-SIV. *IACR Cryptology ePrint Archive*, 2016:853, 2016.

[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.

[IMPS17] Tetsu Iwata, Kazuhiko Minematsu, Thomas Peyrin, and Yannick Seurin. ZMAC: A Fast Tweakable Block Cipher Mode for Highly Secure Message Authentication. In *Advances in Cryptology - CRYPTO '17. Proceedings, Part III*, pages 34–65, 2017.

[JLM+17] Ashwin Jha, Eik List, Kazuhiko Minematsu, Sweta Mishra, and Mridul Nandi. XHX - A framework for optimally secure tweakable block ciphers from classical ciphers and universal hashing. *IACR Cryptology ePrint Archive*, 2017:1075, 2017.

[JN16] Ashwin Jha and Mridul Nandi. Revisiting Structure Graph and Its Applications to CBC-MAC and EMAC. *IACR Cryptology ePrint Archive*, 2016:161, 2016.

[JNP14a] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. Tweaks and keys for block ciphers: The TWEAKEY framework. In *Advances in Cryptology - ASIACRYPT 2014. Proceedings, Part II*, pages 274–288, 2014.

[JNP14b] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. Tweaks and Keys for Block Ciphers: The TWEAKEY Framework. In *ASIACRYPT 2014*, pages 274–288, 2014.

[JNP16a] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. Deoxys v1.41. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round3/deoxysv141.pdf.

[JNP16b] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. KIASU v1. Submission to CAESAR, 2016. https://competitions.cr.yp.to/round1/kiasuv1.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.

[KR11] Ted Krovetz and Phillip Rogaway. The Software Performance of Authenticated-Encryption Modes. In *FSE*, pages 306–327, 2011.

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

[KSW04] Chris Karlof, Naveen Sastry, and David Wagner. Tinysec: A link layer security architecture for wireless sensor networks. In *Proceedings of Embedded Networked Sensor Systems*, SenSys '04, pages 162–175. ACM, 2004.

[KW02] Lars R. Knudsen and David A. Wagner. Integral Cryptanalysis. In Joan Daemen and Vincent Rijmen, editors, *FSE 2002*, volume 2365 of *LNCS*, pages 112–127. Springer, 2002.

[LN17] Eik List and Mridul Nandi. ZMAC+ - an efficient variable-output-length variant of ZMAC. *IACR Trans. Symmetric Cryptol.*, 2017(4):306–325, 2017.

[LNS18] Gaëtan Leurent, Mridul Nandi, and Ferdinand Sibleyras. Generic attacks against beyond-birthday-bound macs. In *Advances in Cryptology - CRYPTO 2018. Proceedings, Part I*, pages 306–336, 2018.

[LPTY16] Atul Luykx, Bart Preneel, Elmar Tischhauser, and Kan Yasuda. A MAC Mode for Lightweight Block Ciphers. In *FSE 2016*, pages 43–59, 2016.

[LRW02] Moses Liskov, Ronald L. Rivest, and David A. Wagner. Tweakable block ciphers. In *CRYPTO 2002*, pages 31–46, 2002.

[LS13] Rodolphe Lampe and Yannick Seurin. Tweakable Blockciphers with Asymptotically Optimal Security. In *FSE 2013. Revised Selected Papers*, pages 133–151, 2013.

[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.

[LST12] Will Landecker, Thomas Shrimpton, and R. Seth Terashima. Tweakable blockciphers with beyond birthday-bound security. In *Advances in Cryptology - CRYPTO 2012. Proceedings*, pages 14–30, 2012.

[Mat93] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In *Advances in Cryptology - EUROCRYPT '93, Proceedings*, pages 386–397, 1993.

[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.

[Nai17] Yusuke Naito. Blockcipher-based macs: Beyond the birthday bound without message length. In *Advances in Cryptology - ASIACRYPT 2017. Proceedings, Part III*, pages 446–470, 2017.

[NIS17] Report on Lightweight Cryptography, 2017. http://nvlpubs.nist.gov/nistpubs/ir/2017/NIST.IR.8114.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.

[NRS14] Chanathip Namprempre, Phillip Rogaway, and Thomas Shrimpton. Reconsidering generic composition. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 257–274, 2014.

[Pat08a] 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.

[Pat08b] Jacques Patarin. A proof of security in o(2^{n}) for the benes scheme. In *Progress in Cryptology - AFRICACRYPT 2008*, pages 209–220, 2008.

[Pat13] Jacques Patarin. Security in O(2^{n}) for the Xor of Two Random Permutations - Proof with the standard H technique -. *IACR Cryptology ePrint Archive*, 2013:368, 2013.

[RBB03] Phillip Rogaway, Mihir Bellare, and John Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur.*, 6(3):365–403, 2003.

[Rog04] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, pages 16–31, 2004.

[RS06] Phillip Rogaway and Thomas Shrimpton. A Provable-Security Treatment of the Key-Wrap Problem. In *EUROCRYPT*, pages 373–390, 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.

[Sch98] Richard Schroeppel. The Hasty Pudding Cipher. 1998.

[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.

[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.

[Zig] ZigBee Alliance. http://www.zigbee.org.

Appendix

A Security Definitions

(Tweakable) Random Permutation and Random Function: For any finite set $\mathcal{X}$, $\mathsf{X} \leftarrow_{\$} \mathcal{X}$ denotes uniform and random sampling of X from $\mathcal{X}$.

We call $\Pi \leftarrow_{\$} \mathsf{Perm}(n)$ a (uniform) random permutation, and $\widetilde{\Pi} \leftarrow_{\$} \mathsf{TPerm}(\tau, n)$ a tweakable (uniform) random permutation on tweak space $\{0,1\}^{\tau}$ and block space $\{0,1\}^{n}$. Note that, $\widetilde{\Pi}^{i}$ is independent of $\widetilde{\Pi}^{j}$ for all $i \neq j \in \{0,1\}^{\tau}$. We call $\Gamma \leftarrow_{\$} \mathsf{Func}(m, n)$ a (uniform) random function from $\{0,1\}^{m}$ to $\{0,1\}^{n}$.

We say that a distinguisher is "sane" if it does not make duplicate queries, or queries whose answer is derivable from previous query responses. In this paper, we assume that the distinguisher is limited to at most q queries and t computations.

Tweakable Strong Pseudorandom Permutation (TSPRP): The TSPRP advantage of any distinguisher $\mathcal{A}$ against $\widetilde{\mathsf{E}}$ instantiated with key $\mathsf{K} \leftarrow_{\$} \{0,1\}^{\kappa}$, is defined as

$$\mathbf{Adv}^{\mathsf{tsprp}}_{\widetilde{\mathsf{E}}}(\mathcal{A}) := \left| \Pr[\mathcal{A}^{\widetilde{\mathsf{E}}_{\mathsf{K}}^{\pm}} = 1] - \Pr[\mathcal{A}^{\widetilde{\Pi}^{\pm}} = 1] \right|.$$

The TSPRP security of $\widetilde{\mathsf{E}}$, is defined as

$$\mathbf{Adv}^{\mathsf{tsprp}}_{\widetilde{\mathsf{E}}}(q, t) := \max_{\mathcal{A}} \mathbf{Adv}^{\mathsf{tsprp}}_{\widetilde{\mathsf{E}}}(\mathcal{A}). \tag{5}$$

TPRP or tweakable pseudorandom permutation and its advantage $\mathbf{Adv}^{\mathsf{TPRP}}_{\widetilde{\mathsf{E}}}(q, t)$ is defined similarly when adversary has no access of the inverse oracle.

Pseudorandom Function (PRF): The PRF advantage of distinguisher $\mathcal{A}$ against a keyed family of functions $\mathsf{PRF} := \{\mathsf{PRF}_{K} : \{0,1\}^{m} \rightarrow \{0,1\}^{n}\}_{K \in \{0,1\}^{\kappa}}$ is defined as

$$\mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{PRF}}(\mathcal{A}) := \left| \Pr_{\mathsf{K} \leftarrow_{\$} \{0,1\}^{\kappa}} [\mathcal{A}^{\mathsf{PRF}_{\mathsf{K}}} = 1] - \Pr[\mathcal{A}^{\Gamma} = 1] \right|.$$

The PRF security of PRF against $\mathbb{A}(q, t)$ is defined as

$$\mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{PRF}}(q, t) := \max_{\mathcal{A}} \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{PRF}}(\mathcal{A}). \tag{6}$$

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 $\mathbf{Adv}^{\mathsf{wprf}}_{\mathsf{PRF}}(q, t)$.

IV-BASED ENCRYPTION: An IV-Based Encryption iv-enc scheme is a tuple $\Psi := (\mathcal{K}, \mathcal{N}, \mathcal{M}, \mathcal{E}, \mathcal{D})$. Encryption algorithm $\mathcal{E}$ takes a key $K \in \mathcal{K}$ and a message $M \in \mathcal{M}$ and returns $(\mathsf{iv}, C) = \mathcal{E}(K, M)$, where $\mathsf{iv} \in \mathcal{N}$ is the initialization vector and $C \in \mathcal{M}$ is the ciphertext. Decryption algorithm $\mathcal{D}$ takes K, iv, C and returns $M = \mathcal{D}(K, \mathsf{iv}, C)$. Correctness condition says that for all $K \in \mathcal{K}$ and $M \in \mathcal{M}$ $\mathcal{D}(K, \mathcal{E}(K, M)) = M$. The Priv$ advantage [IM16, GL15, NRS14, RS06] of $\mathcal{A}$ is defined as

$$\mathbf{Adv}^{\mathsf{priv}}_{\mathsf{iv\text{-}enc}}(\mathcal{A}) := \left| \Pr_{\mathsf{K}}\left[\mathcal{A}^{\mathcal{E}_{\mathsf{K}}} = 1\right] - \Pr_{\Gamma}\left[\mathcal{A}^{\Gamma} = 1\right] \right|$$

where $\mathsf{K} \leftarrow_{\$} \mathcal{K}$ and Γ is a random function from $\mathcal{M} \to \mathcal{N} \times \mathcal{M}$. The Priv$ security of iv-enc, is defined as

$$\mathbf{Adv}^{\mathsf{priv}}_{\mathsf{iv\text{-}enc}}(q, t) := \max_{\mathcal{A}} \mathbf{Adv}^{\mathsf{priv}}_{\mathsf{iv\text{-}enc}}(\mathcal{A}). \tag{7}$$

(NONCE-BASED) AUTHENTICATED ENCRYPTION WITH ASSOCIATED DATA: A (nonce-based) authenticated encryption with associated data or NAEAD scheme $\mathfrak{A}$ consists of a key space $\mathcal{K}$, a (possibly empty) nonce space $\mathcal{N}$, a message space $\mathcal{M}$, an associated data space , and a tag space $\mathcal{T}$, along with two functions $\mathcal{E} : \mathcal{K} \times \mathcal{N} \times \times \mathcal{M} \to \mathcal{M} \times \mathcal{T}$, and $\mathcal{D} : \mathcal{K} \times \mathcal{N} \times \times \mathcal{M} \times \mathcal{T} \to \mathcal{M} \cup \{\perp\}$, with the correctness condition that for any $K \in \mathcal{K}, N \in \mathcal{N}, A \in, M \in \mathcal{M}$, we must have $\mathcal{D}(K, N, A, \mathcal{E}(M)) = M$. When the nonce space is empty, we call the AE scheme a deterministic AE or DAE scheme.

Following the security definition in [IM16, GL15, NRS14, RS06], we define the NAEAD (DAE for deterministic AE) advantage of $\mathcal{A}$ as

$$\mathbf{Adv}^{\mathsf{AE}}_{\mathfrak{A}}(\mathcal{A}) := \left| \Pr_{\mathsf{K}}\left[\mathcal{A}^{\mathcal{E}_{\mathsf{K}}, \mathcal{D}_{\mathsf{K}}} = 1\right] - \Pr_{\Gamma}\left[\mathcal{A}^{\Gamma, \perp} = 1\right] \right|,$$

where $\mathsf{K} \leftarrow_{\$} \mathcal{K}$ and Γ is a random function from $\mathcal{N} \times \times \mathcal{M} \to \mathcal{M} \times \mathcal{T}$, and $\perp$ is the reject oracle that takes (N, A, C, T) as input and returns the reject symbol $\perp$. The NAEAD/DAE security of $\mathfrak{A}$, is defined as

$$\mathbf{Adv}^{\mathsf{AE}}_{\mathfrak{A}}(q, t) := \max_{\mathcal{A}} \mathbf{Adv}^{\mathsf{AE}}_{\mathfrak{A}}(\mathcal{A}). \tag{8}$$

B Other Applications

B.1 Sum of Permutations

The sum of permutations is a popular approach of constructing an n-bit length preserving PRF. Given 2 independent instantiations, E_{K_0} and E_{K_1}, of a secure block cipher over $\{0,1\}^n$, the sum of permutations, denoted XOR2, is defined by the mapping $x \mapsto \mathsf{E}_{K_0}(x) \oplus \mathsf{E}_{K_1}(x)$.

The XOR2 construction has been proved to be n-bit secure independently by Patarin [Pat13] and Dai et al. [DHT17], though the proof by Patarin still has some unresolved gaps. There is a single key variant of XOR2, but it sacrifices one bit (i.e. defined from $\{0,1\}^{n-1}$ to $\{0,1\}^n$) for domain separation. Instead we can use a tBC to simply replace the two block cipher keys with one tBC key and two distinct tweaks. We define $\mathsf{Twe\text{-}XOR2}(x) := \widetilde{\mathsf{E}}^0_K(x) \oplus \widetilde{\mathsf{E}}^1_K(x)$. Again combining Proposition 1 with [DHT17, Theorem], we obtain

Proposition 4. *For $q \le 2^{n-4}$,*

$$\mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{Twe\text{-}XOR2}}(t, q) \le \mathbf{Adv}^{\mathsf{TPRP}}_{\widetilde{E}}(t', q) + (q/2^n)^{1.5}.$$

B.2 Tweaking Various Other Constructions

In the following list, we apply similar technique as above to several other constructions with multiple keys. The security of all the tBC-based variants is similar to the multi-key original constructions, so we skip their explicit security statements.

1. **Encrypted Davis Meyer (EDM) [CS16]:** EDM uses two keys and obtains BBB PRF security. We define the tBC-based variant as follows:

$$\mathsf{Twe\text{-}EDM}(x) := \widetilde{\mathsf{E}}^1_K(\widetilde{\mathsf{E}}^0_K(x) \oplus x).$$

2. **Encrypted Wegman Carter Davis Meyer (EWCDM) [CS16]:** EWCDM is a nonce-based BBB secure MAC that requires two block cipher keys and a hash key. The tBC-based variant of EWCDM is defined as:

$$\mathsf{Twe\text{-}EWCDM}(N, M) := \widetilde{\mathsf{E}}^2_K\big(\widetilde{\mathsf{E}}^1_K(N) \oplus N \oplus H_{\widetilde{\mathsf{E}}^0_K(0)}(M)\big).$$

3. **Chained LRW2 (CLRW2) [LST12]:** The CLRW2 construction is a TBC that achieves BBB TSPRP security using two independent block cipher keys and two independent hash keys. We define a tBC-based variant of CLRW2 as follows:

$$\mathsf{Twe\text{-}CLRW2}(M, T) := \widetilde{\mathsf{E}}^2_K\big(\widetilde{\mathsf{E}}^1_K(M \oplus h_{L_1}(T)) \oplus h_{L_1}(T) \oplus h_{L_2}(T)\big) \oplus h_{L_2}(T),$$

   where L_1 and L_2 can be derived using $\widetilde{\mathsf{E}}$ as before. It is easy to see that one can easily extend the idea to obtain single keyed CLRWr [LS13] using r distinct tweaks.

4. **GCM-SIV**-2 [IM16]. GCM-SIV-2 is an MRAE scheme with $2n/3$-bit security. However, it requires 6 independent block cipher keys along with 2 independent hash keys. We can easily make it single keyed using a tBC:

$$\begin{aligned} V_1 &:= H_{\widetilde{\mathsf{E}}^0_K(0)}(N, A, M)\,,\ V_2 := H_{\widetilde{\mathsf{E}}^0_K(1)}(N, A, M) \\ T_1 &:= \widetilde{\mathsf{E}}^1_K(V_1) \oplus \widetilde{\mathsf{E}}^2_K(V_2)\,,\ T_2 := \widetilde{\mathsf{E}}^3_K(V_1) \oplus \widetilde{\mathsf{E}}^4_K(V_2), \\ C_i &:= M_i \oplus \widetilde{\mathsf{E}}^5_K(T_1 \oplus i) \oplus \widetilde{\mathsf{E}}^6_K(T_2 \oplus i). \end{aligned}$$

   Extending the same approach, one can get a single keyed version of GCM-SIV-$r$as well.

5. **The Benes Construction [Pat08b]:** The Benes construction is a method to construct $2n$-bit length preserving PRF construction with n-bit security that uses 8 independent n bit to n bit PRFs. Formally,

$$\begin{aligned} L' &:= f_1(L) \oplus f_2(R) \\ R' &:= f_3(L) \oplus f_4(R) \\ \mathsf{Benes}(L, R) &:= (f_5(L') \oplus f_6(R'), f_7(L') \oplus f_8(R')). \end{aligned}$$

   Now these f_i functions can be constructed using sum of two permutations, however that would essentially require 16 block cipher keys. With a tBC, we can reduce the number of keys to one by instantiating $f_i := \widetilde{\mathsf{E}}^{2i}_K \oplus \widetilde{\mathsf{E}}^{2i+1}_K$ for each $i \in [8]$.

C Proof of Proposition 4.3

Proof. Twe-LightMAC_Plus is an instance of Twe-DbHtS, and hence offers similar security. The security bound of Twe-DbHtS includes a term

$$\mathbf{Adv}^{\mathsf{PRF}}_{C_3^*[H,\pi_0,\pi_1,\pi_2]}(q,\ell,t)$$

from [DDNP18]. One can verify from [DDNP18, Proof of Theorem 2.(iii)], that this term is predominantly bounded by two probabilities:

1. $\Pr[\exists$ distinct i, j, k such that $\Sigma_i = \Sigma_j, \Theta_i = \Theta_k]$.
2. $\Pr[\exists$ distinct i, j such that $\Sigma_i = \Sigma_j, \Theta_i = \Theta_j]$.

Now the hash layer of Twe-LightMAC_Plus is exactly same as the PHASHx of [LN17]. Using similar arguments as in [LN17, Proof of Theorem 1] it can be shown that 1. is upper bounded by $O(q^3/2^{2n})$, and 2. is upper bounded by $O(q^2/2^{2n})$. The result follows by combining 1 and 2. □

D Specification of GIFT

GIFT [BPP+17] is a lightweight block cipher supporting 64- and 128-bit block and 128-bit key size. The former and the latter are called GIFT64 and GIFT128, respectively. Here we introduce the specification GIFT64. Refer to the original specification for the detailed description of GIFT128.

A 64-bit plaintext P is loaded to a 64-bit state s_0. Then the state is updated by iteratively applying a round function $RF : \{0,1\}^{64} \times \{0,1\}^{32} \mapsto \{0,1\}^{64}$ 28 times as $s_i \leftarrow RF(s_{i-1}, k_{i-1})$ for $i = 1, 2, \cdots, 28$, where k_i are 28 round keys generated from a 128-bit user-specified key K by a key scheduling function $KF : \{0,1\}^{128} \mapsto (\{0,1\}^{32})^{28}$ as $(k_0, k_1, \cdots, k_{27}) \leftarrow KF(K)$. We call the computation for index i "*round i*." The last state, s_{28}, is a ciphertext C.

D.1 Round Function (RF).

Let $x_{63}, x_{62}, \cdots, x_0$ be a 64-bit state value. The round function consists of the following three operations: SubCells, PermBits, and AddRoundKey.

SubCells: It applies a 4-bit to 4-bit S-box S shown in Table 12 to 16 nibbles $x_{4i+3}, x_{4i+2}, x_{4i+1}, x_{4i}$, $\forall i = 0, 1, \cdots, 15$ in parallel.

Table 12: S-box.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S(x)$	1	a	4	c	6	f	3	9	2	d	b	7	5	0	8	e

PermBits: A bit-permutation π specified in Table 13 is applied to the 64-bit state.

AddRoundKey: This step consists of adding a round key and a round constant. A 32-bit round key k_{i-1} is extracted from the key state, it is further partitioned into two 16-bit words $k_{i-1} = U \| V = u_{15}u_{14} \cdots u_0 \| v_{15}v_{14} \cdots v_0$. For GIFT-64, U and V are XORed to x_{4i+1} and x_{4i} of the state respectively.

$$x_{4i+1} \leftarrow x_{4i+1} \oplus u_i, \quad x_{4i} \leftarrow x_{4i} \oplus v_i, \quad \forall i \in \{0, 1, \cdots, 15\}.$$

Table 13: Bit-Permutation.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi(x)$	0	17	34	51	48	1	18	35	32	49	2	19	16	33	50	3
x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$\pi(x)$	4	21	38	55	52	5	22	39	36	53	6	23	20	37	54	7
x	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$\pi(x)$	8	25	42	59	56	9	26	43	40	57	10	27	24	41	58	11
x	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$\pi(x)$	12	29	46	63	60	13	30	47	44	61	14	31	28	45	62	15

Then, a single bit '1' and a 6-bit round constant are XORed to the state at bit positions 63, 23, 19, 15, 11, 7 and 3. Round constants are generated by a simple linear feedback shift register. In our analysis, the round constants do not have any impact, thus we ignore them hereafter. The schematic diagram of the GIFT round function is shown in Fig. 20.

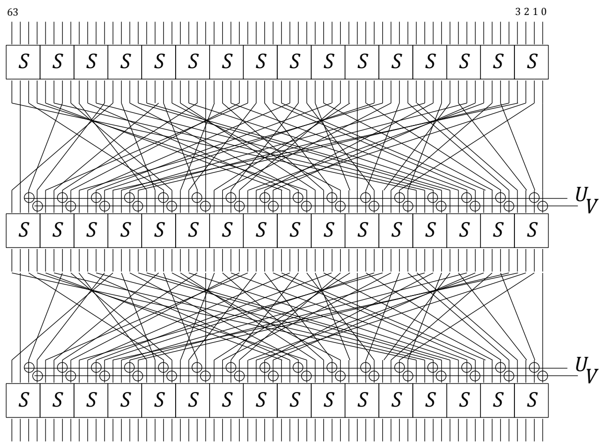


Figure 20: Schematic Diagram of Two Rounds of GIFT64.

D.2 Key Schedule Function (KF).

A 128-bit user-specified key K is loaded to a 128-bit key state that is composed of eight 16-bit words κ_7,κ_6,κ_5,κ_4,κ_3,κ_2,κ_1, and κ_0. A round key is first extracted from the key state before the key state update. For GIFT64, two 16-bit words of the key state are extracted as the round key $k_{i-1} = U \| V$,

$$U \leftarrow \kappa_1, \quad V \leftarrow \kappa_0.$$

The key state is then updated as follows,

$$\kappa_7 \| \kappa_6 \| \kappa_5 \| \cdots \| \kappa_1 \| \kappa_0 \leftarrow Rot^R(\kappa_1, 2) \| Rot^R(\kappa_0, 12) \| \kappa_7 \| \cdots \| \kappa_3 \| \kappa_2,$$

where $Rot^R(X, i)$ is an i-bit right rotation of X within a 16-bit word. The schematic diagram of the GIFT key schedule function is illustrated in Fig. 21.

D.3 Short Remarks on GIFT128.

The state size of GIFT128 is 128 bits, which consists of thirty-two 4-bit nibbles. SubCells operation apply the same S-box as GIFT64 to 32 nibbles and a 128-bit permutation

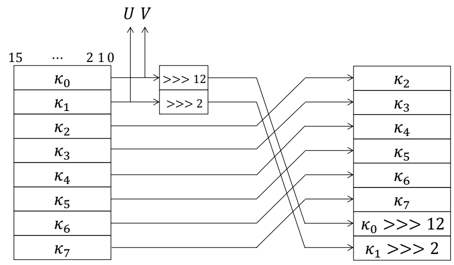


Figure 21: Schematic Diagram of Key Schedule Function of GIFT64.

is applied to the state. AddRoundKey extracts 64 bits from the key state and adds bit-position $4i+1$ and $4i+2$, where $i = 0, 1, \ldots, 31$, of the state.

E Hardware Implementation of TweGIFT

Since TweGIFT is explicitly designed for ultra-lightweight implementations, we only provide the hardware implementation results for TweGIFT.

Table 14: Implementation results for GIFT and TweGIFT on Virtex 7 FPGA.

BC or tBC	LUTs	FF	Slices	Frequency (MHz)	Clock cycles	Throughput (Mbps)
GIFT-64-ED	615	277	236	455.17	29	1004.51
TweGIFT-64-ED[4,16,16,4]	617	277	234	430.29	29	946.60
GIFT-64-E	449	275	153	596.66	29	1316.77
TweGIFT-64-E[4,16,16,4]	479	275	179	595.09	29	1313.30
GIFT-128-ED	1113	408	432	447.83	41	1398.10
TweGIFT-128-ED[4,32,32,5]	1158	408	419	416.50	41	1300.29
TweGIFT-128-ED[16,32,32,4]	1223	408	428	429.32	41	1340.31
GIFT-128-E	763	403	330	596.30	41	1861.62
TweGIFT-128-E[4,32,32,5]	796	403	332	597.59	41	1865.65
TweGIFT-128-E[16,32,32,4]	805	403	377	598.78	41	1869.36

Table 19 summerizes the hardware performances of our recommended TweGIFT versions along with the original GIFT. For ED implementation, our recommended version of TweGIFT-64 has an overheads of 0.3% for 4 bit tweaks, and TweGIFT-128 has overheads of 4.04% and 9.89% for tweak size of 4 and 16 bits respectively. As we move to the E implementation, TweGIFT-64 has an overheads of 6.68% for 4 bit tweaks, and TweGIFT-128 has overheads of 4.32% and 5.5% for tweak size of 4 and 16 bits respectively.

E.1 Security Analysis of TweAES and TweGIFT Instances

In this section, we provide the various cryptanalysis that we performed on the TweAES and TweGIFT instances. Note that our target is single-key security, and any related-key attacks are out of our scope. The exact security bound, e.g., the lower bound of the

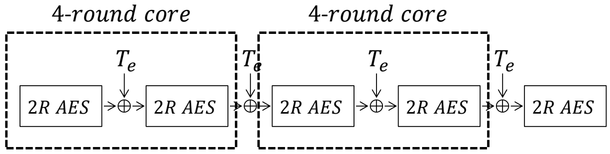


Figure 22: 4-round Core of TweAES[∗,∗,∗,2]

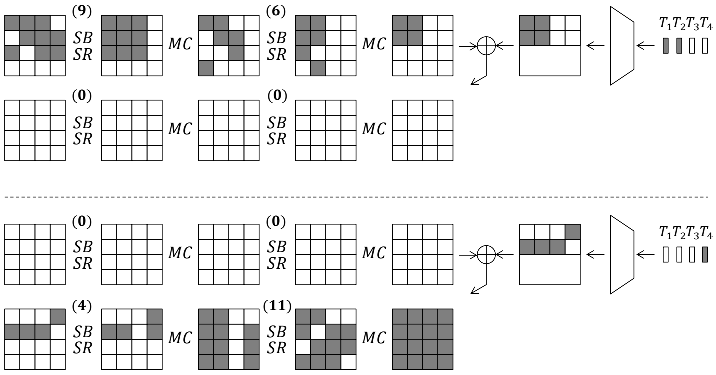


Figure 23: Two Examples of Differential Trails with 15 Active S-boxes.

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 transformation. We call those operations "$2r$-*round core*," which is depicted for AES and GIFT-64 in Fig. 22. 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. 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.

E.1.1 Security Analysis of TweAES

As explained above, we evaluate the minimum number of differentially and linearly active S-boxes for the 4-round core. The 4-bit tweaks of TweAES are 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 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 Figure 23. 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 includes two 4-round cores.

For TweAES, experimentally computing the lower bound of the number of active S-boxes

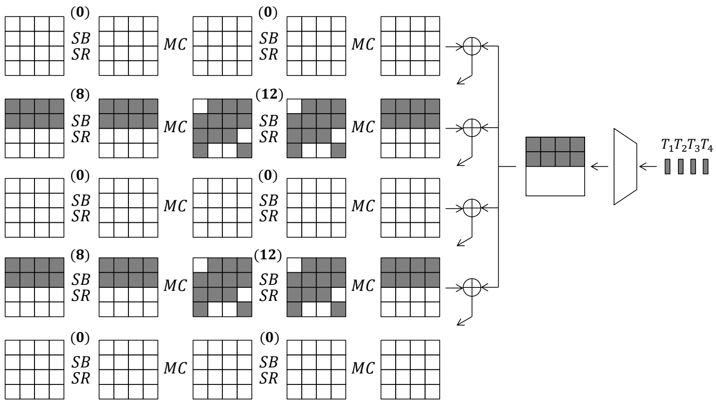


Figure 24: An Examples of Differential Trails with 40 Active S-boxes.

is also possible. When the tweak input has a non-zero difference, the minimum number of active S-boxes is 40 for the entire construction. This is a tight bound. An example of the differential trails achieving 40 active S-boxes is given in Fig. 24. The probability of the differential propagation is upper bounded by $2^{-6\times 40} = 2^{-240}$.

We argue that the reduced-round versions of TweAES in which the first or the last round is located in the middle of the 4-round core can be attacked for relatively long rounds. Owing to this unusual setting, the attacks here do not threaten the security of full TweAES, 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 more precisely formulated version called the sandwich attack. The boomerang attack divides the cipher E into two parts E_0 and E_1 such that $E = E_1 \circ E_0$, and builds high-probability differentials for E_0 and E_1 almost independently. The attack detects a quartet of plaintext x that satisfy the non-ideal behavior shown below with probability $p^{-2}q^{-2}$, where p and q are the differential probability for $E_0 : \alpha \to \beta$ 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.

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

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

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

We define E_m of this attack as the first S-box layer in the above E_1. The configuration and the differential trails are depicted in Fig. 25 The probability when E_m is a single S-box layer can be measured by using the boomerang connectivity table (BCT). The trails for E_0 and E_1 include 4 active S-boxes, hence both of the probability p and q are 2^{-24}. That is, $p^2q^2 = 2^{-96}$. The BCT of the AES S-box shows that the probability for each S-box in E_m

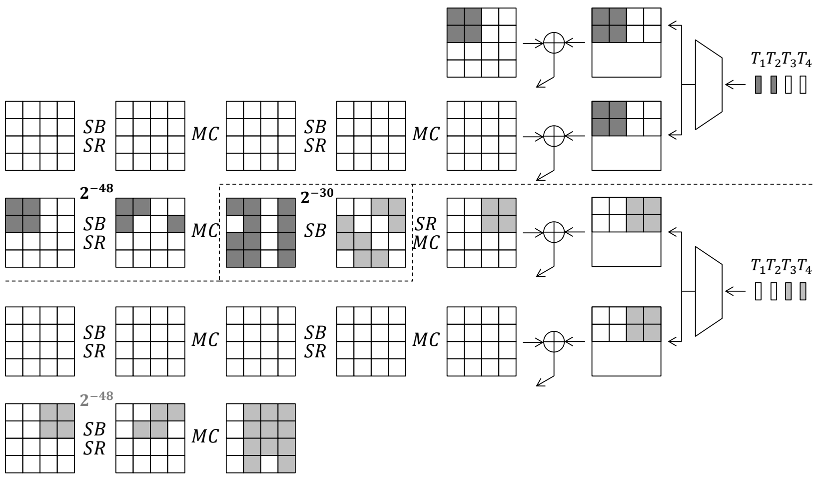


Figure 25: Differential Trails for Boomerang Attacks. The cells filled with black and gray represent active byte positions in E_0 and E_1, respectively.

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

8-Round Impossible Differential Attacks against TweAES. Due to 2 interval rounds between tweaks, distinguishers based on impossible differential attacks can be constructed for relatively long rounds (6 rounds) by canceling the tweak difference with the state difference. The distinguisher is depicted in Fig. 26.

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

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

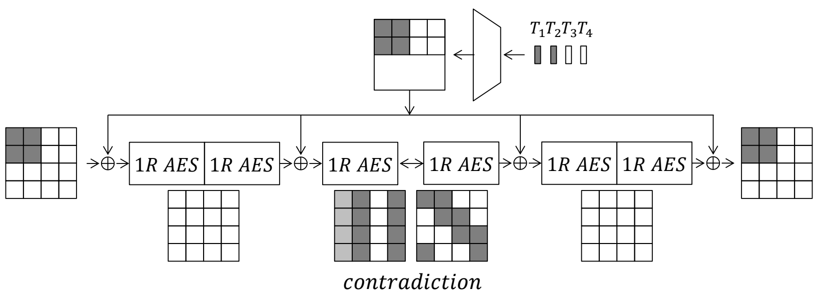


Figure 26: 6-round Impossible Differential Distinguisher. The bytes filled with black, white, and gray have non-zero difference, zero difference, and arbitrary difference, respectively.

For the key recovery, two rounds can be appended to the 6-round distinguisher; one is at the beginning and the other is at the end, which is illustrated in Fig. 27. As shown

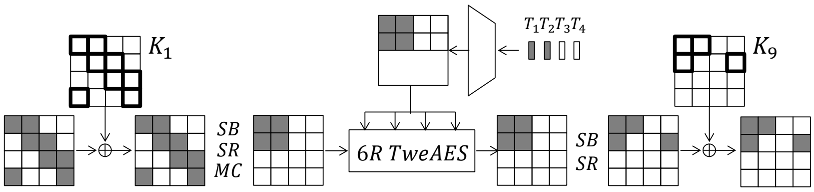


Figure 27: Extension to 8-round Key Recovery

in Fig. 27 the trail includes 8 and 4 active bytes at the input and output states. Partial computations to the middle 6-round distinguisher involve 8 bytes of subkey K_1 and 4 bytes of subkey K_9.

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

1. Choose all tweak values denoted by T^i where $i = 0, 1, \ldots, 2^4 - 1$.

2. For each of T^i, fix the value of inactive 8 bytes at the input, choose all 8-byte values at the active byte positions of the input state. Query those 2^{64} values to get the corresponding outputs. Those outputs are stored in the list L^i where $i = 0, 1, \ldots, 2^4 - 1$.

3. For all $\binom{2^4}{2} \approx 2^7$ pairs of L^i and L^j with $i \neq j$, find the pairs that do not have difference in 12 inactive bytes of the output state. About $2^{7+64+64-96} = 2^{39}$ pairs will be obtained.

4. For each of the obtained pairs, the tweak difference is fixed and the differences at the input and output states are also fixed. Those fix both of input and output differences of each S-box in the first round and the last round. Hence, each pair suggests a wrong key.

5. Repeat the procedure 2^{54} times from the first step by changing the inactive byte values at the input. After this step, $2^{39+54} = 2^{103}$ wrong-key candidates (including overlaps) will be obtained. The remaining key space of the involved 12 bytes becomes $2^{96} \times (1 - 2^{-96})^{2^{103}} \approx 2^{96} \times e^{-128} \approx 2^{-88} < 1$. Hence, the 8 bytes of K_1 and 4 bytes of K_9 will be recovered.

6. Exhaustively search the remaining 8 bytes of K_1.

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

Remarks on Other Attacks

- Integral attacks [DKR97, KW02] collect 2^8 distinct values for a particular byte or distinct 2^{32} values for a particular diagonal. Integral attacks exploiting the tweak is difficult because the tweak will not affect all the bits in each byte, which prevents to collect 2^8 distinct values for any byte.

- Meet-in-the-middle attacks [DS08, DFJ13] exploit the 4-round truncated differentials $1 \rightarrow 4 \rightarrow 16 \rightarrow 4 \rightarrow 1$ 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 be exploited in the meet-in-the-middle attack.

Summary. We demonstrated two attacks against reduced-round variants that start from the middle of the 4-round core. Because security of TweAES using tweak difference relies on the fact that the large-weight tweak difference will diffuse fast in the subsequent 2 rounds, those reduced-round analysis will not threaten the security of the full TweAES. From a different viewpoint, one can see the difficulty to extend the analysis by 1 more round from Figs. 25 and 27. The number of involved subkey bytes easily exceeds 16.

E.1.2 Cryptanalysis of TweAES with non-zero tweak from the initial round.

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 and the tweak is non-zero. The main purpose is to show the difficulty of exploiting 4 bits tweak in the attack, thus we do not discuss the case of fixing the tweak. (When tweak is zero, security is the same as the original AES, which can also be applied to TweAES but does not show any vulnerability introduced by TweAES.) The comparison of the number of attacked rounds and the attack complexity for the original AES and TweAES is given in Table 15.

Table 15: Comparison of the Attacks on AES and TweAES exploiting tweak. R, D, T and M denote the number of rounds, data complexity, time complexity and memory complexity, respectively.

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}

Integral Attacks. Because the tweak starts to appear only after the second round, to play with plaintexts is difficult to extend the integral attacks. The most reasonable approach to exploit the tweak is to set the plaintext constant and collect all possible 2^4 tweak inputs. The propagation of the property is given in Fig. 28. Because the plaintext can be 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. 28. 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 reduce to 1.

The memory complexity can be saved by first preparing two sets of 16 texts, and then the bytes of K_6 is 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

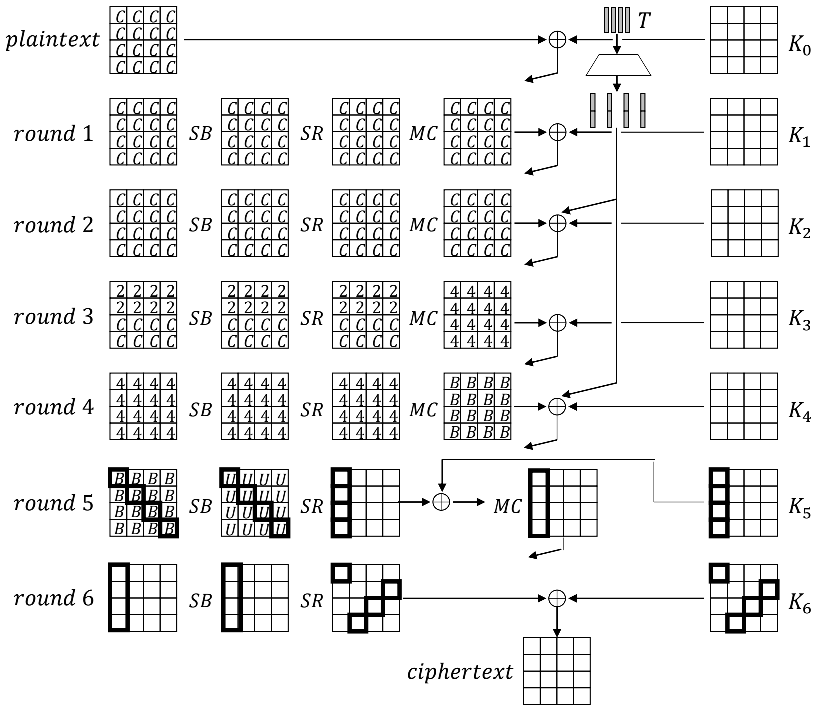


Figure 28: Integral Distinguisher on TweAES via Tweak. '2' represents that two kinds of values appear 8 times each and '4' represents that four kinds of values appear 4 times each. By following the convention, 'B' and 'U' denote 'balanced' and 'unknown' properties, respectively.

thanks to the tweak injection in every two rounds but then the property disappears more quickly because we need to active at least 4 byte positions. The attack on the original 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 the 2 AES rounds before the first tweak injection.

Impossible Differential Attacks. With 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 distinguisher, as indicated by Fig. 29. Namely, the top left and the bottom left bytes are active with probability 1 in the forward direction, while those byte are inactive with probability 1 in the backward direction.

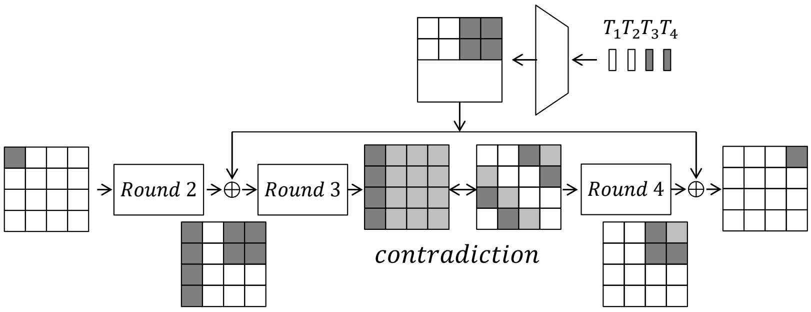


Figure 29: 3-round Impossible Differential Distinguisher using Tweak Difference.

For the key recovery, one round and two rounds can be appended to the beginning and the end of the 3-round distinguisher, which is illustrated in Fig. 27.

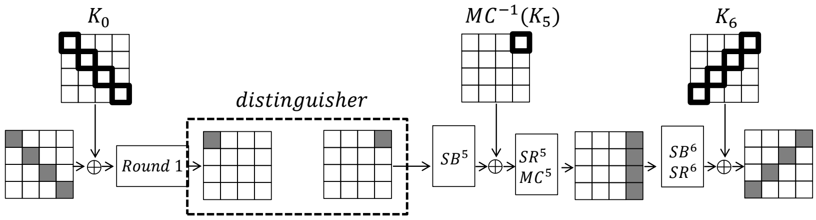


Figure 30: Extension to 6-round Key Recovery

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 distinguisher. 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 $\binom{2^{32}}{2} \approx 2^{63}$ ciphertext pairs. This can be 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 $X \cdot 2^{-33}$ pairs, the attacker generates the wrong keys 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. 30. This can be done by choosing all possible (2^8) 1-byte difference 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 candidates. Similarly, by choosing 1-byte difference at the input of round 5 and 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 is recovered, the remaining 12 bytes can be recovered by the exhaustive search.

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

Truncated Differential Attacks. So fat 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. 31. The trail 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 difference before MixColumns in round 4 is 2^{24}. For each of them, the attacker can derive 1 candidate of the corresponding 4 subkey bytes of K_5, thus the key space is reduced by a factor of

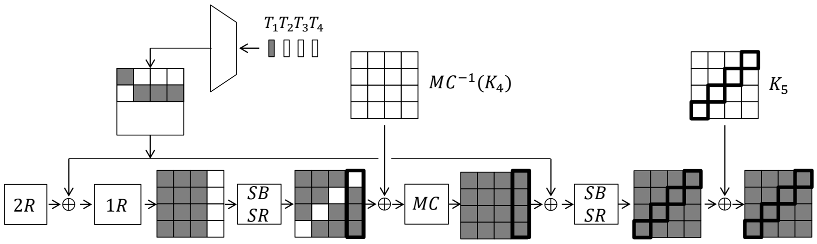


Figure 31: 5-round Truncated Differential Attack using Tweak Difference (type 1).

2^8. The involved byte positions for 1 column is stressed in Fig. 31 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 also 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 the 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 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. 32. The trail can be satisfied with probability 2^{-128}; 2^{-64} for the first round and 2^{-64} towards

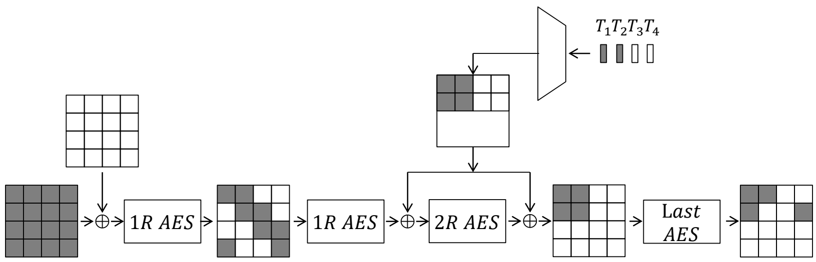


Figure 32: 5-round Truncated Differential Attack using Tweak Difference (type 2).

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 in Fig. 32. 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 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 the attack on more than 5 rounds is an open problem.

E.2 Security Analysis of TweAES-6

We also provide a round reduced version TweAES denoted by TweAES-6 (to be used in one of our applications). In TweAES-6, the number of rounds is reduced from TweAES from 10 to 6 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. 24.

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 the differential cryptanalysis.

Note that AEAD schemes based on the original AES often adopt 4-round AES in the mode, and the minimum number of the active S-boxes for 4-round AES is 25. We designed TweAES-6 to offer the similar security level as 4-round AES, and no attack is known on the 4-round AES in proper modes under the restriction of the birthday-bound query limit.

E.3 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 copied three times to achieve a 32-bit tweak. When the 4-bit tweak has some non-zero difference, the expanded 32-bit tweak is ensured to have at least 16 active bits, which ensures at least 16 active S-boxes in 2 rounds around the tweak injection.

We modeled the differential trail search for TweGIFT with MILP under the constraints that at least 1 bit of tweak has a difference. However, owing to the large state size, it is infeasible to find the tight bound of the maximum probability of the differential characteristic even for the 10-round core. The tool so far provided that the probability of the differential characteristic is upper bounded by $2^{-72.6}$. 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 $2^{-72.6\times 4} = 2^{-300.4}$, 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 of 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 16.

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 17.

The differential mask for the first and last rounds in Table 17 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 is inefficient.

Table 16: 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	upper bound	$2^{-72.6}$	Ours
TweGIFT	10	characteristic probability	lower bound	2^{-79}	Ours

Table 17: 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}

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 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 the active S-boxes for 5 rounds is 5. Hence, the 10-round boomerang trail will certainly require a non-negligible amount of the 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 has already been more 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 17 reaches 18 rounds, we do not think that the boomerang attack can be the best approach for attacking TweGIFT.

E.4 Hardware Performance of the TweAES and TweGIFT Instances

In this section, we provide the hardware implementation details for all our recommended TweGIFT and TweAES versions and compare their hardware overheads respective to their original counterparts GIFT and AES. We give a brief comparison on software implementation of TweAES and AES in supplementary material ??. For each instantiations, we present both the encryption/decryption (ED) version and only encryption (E) version. The VHDL code of our implementations are synthesized using 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 rams are used. We present the results obtained from the tool after performing place and route process.

Table 18: Implementation results for AES and TweAES on Virtex 7 FPGA.

BC or tBC	LUTs	FF	Slices	Frequency (MHz)	Clock cycles	Throughput (Mbps)
AES-ED	2945	533	943	297.88	11	3466.24
TweAES-ED[4,8,8,2]	2960	534	1044	295.97	11	3444.01
TweAES-ED[8,16,8,2]	2976	534	1129	295.81	11	3442.15
TweAES-ED[16,32,8,2]	3006	534	1134	292.87	11	3407.94
AES-E	1605	524	559	330.52	11	3846.05
TweAES-E[4,8,8,2]	1617	524	574	328.27	11	3819.87
TweAES-E[8,16,8,2]	1632	524	593	325.17	11	3783.79
TweAES-E[16,32,8,2]	1659	524	592	326.56	11	3799.97

Table 18 depicts that the area-overhead (LUT counts) introduced by the tweak injection is negligeable. For Considering the combined encryption-decryption (ED) implementation, TweAES have overheads (in LUTs) of 0.5%, 1.05% and 2.07% for tweak size of 4, 8 and 16 bits respectively. As we move to the encryption (E) only implementation, our recommended TweAES versions have negligeable area overheads of 0.7%, 1.68% and 3.36% respectively. Note that, the reduction in the speed is also negligeable.

Table 19: Implementation results for GIFT and TweGIFT on Virtex 7 FPGA.

BC or tBC	LUTs	FF	Slices	Frequency (MHz)	Clock cycles	Throughput (Mbps)
GIFT-64-ED	615	277	236	455.17	29	1004.51
TweGIFT-64-ED[4,16,16,4]	617	277	234	430.29	29	946.60
GIFT-64-E	449	275	153	596.66	29	1316.77
TweGIFT-64-E[4,16,16,4]	479	275	179	595.09	29	1313.30
GIFT-128-ED	1113	408	432	447.83	41	1398.10
TweGIFT-128-ED[4,32,32,5]	1158	408	419	416.50	41	1300.29
TweGIFT-128-ED[16,32,32,4]	1223	408	428	429.32	41	1340.31
GIFT-128-E	763	403	330	596.30	41	1861.62
TweGIFT-128-E[4,32,32,5]	796	403	332	597.59	41	1865.65
TweGIFT-128-E[16,32,32,4]	805	403	377	598.78	41	1869.36

Table 19 summerizes the hardware performances of our recommended TweGIFT versions along with the original GIFT. For ED implementation, our recommended version of TweGIFT-64 has an overheads of 0.3% for 4 bit tweaks, and TweGIFT-128 has overheads of 4.04% and 9.89% for tweak size of 4 and 16 bits respectively. As we move to the E implementation, TweGIFT-64 has an overheads of 6.68% for 4 bit tweaks, and TweGIFT-128 has overheads of 4.32% and 5.5% for tweak size of 4 and 16 bits respectively.