Elastic-Tweak: A Framework for Short Tweak Tweakable Block Cipher

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

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

Abstract. 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, 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. We present hardware and software results to show that the performance overheads for these tBCs are minimal. We perform comprehensive security analysis and observe that TweGIFT and TweAES provide sufficient security without any increase in the number of block cipher rounds when compared to GIFT and AES. We also show some concrete 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 gives better security than CMAC), Twe-LightMAC_Plus (better rate than LightMAC_Plus), Twe-SUNDAE, Twe-CLOC, and Twe-SILC (reduces the number of block cipher calls and simplifies the design of SUNDAE, CLOC and SILC).

Keywords: tweakable block cipher, GIFT, AES, TWEAKEY, XEX

1 Introduction

Since their advent in late 1970's, block ciphers [7, 59] have become the ubiquitous building blocks in various symmetric-key cryptographic algorithms, including encryption schemes [3], message authentication codes (MACs) [5], and

authenticated encryption [4]. 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 formal ways to prove the security of a block cipher against the classical linear [55] and differential [14] attacks. The literature is filled with a plethora of block cipher candidates, AES [7] being the most notable among them. AES is currently the NIST standard block cipher [7], and it is the recommended choice for several standardized encryption, MAC and AE schemes such as CTR [3], CMAC [5], AES-GCM [6] etc. A recent block cipher proposal, named GIFT [11] has generated a lot of interest due to its ultra-lightweight nature.

1.1 Some Issues in Block Cipher Based Designs

Key Size of 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 [23, 61], EDM [20], EWCDM [20], CLRW2 [50], GCM-SIV-2 [37], Benes construction [60]. 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 [16], is a CBC-MAC type construction. CMAC [5], 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. Further FCBC [44, 45] allows more number of queries per key, as compared to CMAC [58].

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 the several auxiliary functions used chiefly for domain separation. For instance CLOC and SILC [38] 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.

Short Message Processing: 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, Bluetooth low energy and TinySec [46] limit the maximum packet lengths to 127 bytes, 47 bytes and

128 bytes, respectively. Similarly, CAN FD [1], a well-known transmission protocol in automotive networks, allows message length up to 64 bytes. The packet sizes in EPC tag [2], 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 [10] 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 [38] have similar drawbacks. They cost 2 and 4 calls to process a single block message. LightMAC_Plus [54], feeds a counter-based encoded input to the block cipher, which reduces the rate.[4]

1.2 Motivation of short-tweak TBC

Tweakable Block Ciphers: The Hasty Pudding cipher [65], an unsuccessful candidate for AES competition, was one of the first tweakable block ciphers.[5] Later, Liskov et al. in formalized this in their foundational work on tweakable block ciphers [52]. Tweakable block ciphers (TBCs) are more versatile and find a broad range of applications, most notably in authenticated encryption schemes, such as OCB [48], COPA [9], and Deoxys [41]; and message authentication codes, such as ZMAC [39], NaT [19], and ZMAC+ [53]. TBCs can be designed from scratch [21, 33, 65], or they can be built using existing primitives like block ciphers, and public permutations. LRW1, LRW2 [52], CLRW2 [50], XEX [62] and XHX [43] are some examples of the former category, whereas Tweakable Even-Mansour [19] is an example of the latter.

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. For example, one can use this equivalence to define a single-keyed version of FCBC which is as secure as FCBC. This resolves the issues with CMAC. 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. An tBC is better than large tweak TBCs in two respects: (i) state size for holding tweak is small, and most importantly (ii) tBC would potentially be more efficient than large tweak TBCs.

The TWEAKEY Framework: At Asiacrypt '14, Jean et al. presented a generic framework for TBC construction, called TWEAKEY [40], that considers the

[4] No. of message blocks processed per block cipher call.

[5] It used the term "spice" for tweaks.

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 [42], Deoxys [41], SKINNY and Mantis [12] 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 [42] 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 [30, 31, 67].

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 [62], 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 [15], OCB [63], COPA [9], ELmD [27] etc. However, XE and XEX have several limitations with respect to a short tweak space, notably (i) security is limited to birthday bound, 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.

1.3 Our Contributions

Our main contributions can be divided into two parts:

1. ELASTIC-TWEAK FRAMEWORK: In this work, we address the above issues and propose a generic framework, called the Elastic-Tweak framework (ET in short), to transform a block cipher into a short tweak TBC. We consider tweaks of size less than 16 bits as "short tweak" is a tweak with size less than or equal to 16 bits and can be as small as 4 bits. This small size ensures that the tweak storage overhead is negligible. 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 several designs over two well known block ciphers AES [7] and GIFT [11] with different tweak sizes varying from 4 to 16. We implement the designs both in software and hardware and find that these tweakable versions have negligible overhead 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. Owing to the large state and complex structure of GIFT, the automatic search is infeasible for the entire 40 rounds of TweGIFT128. Our framework with the approach using the $2r$-round core enables us to derive the upper bound of the differential probability for the entire rounds.

2. Applications of tBC: Here we demonstrate the applicability of tBC in various constructions:

1. **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.
2. **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. However, this is also applicable to Twe-SUNDAE, Twe-CLOC and Twe-SILC (tBC based counterparts of SUNDAE, CLOC and SILC [38] respectively).
3. **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.

In addition to the above applications, we show that tBCs can also simplify the internal structures of various block cipher based authenticated encryption modes. For example, SUNDAE, CLOC, SILC use several auxiliary functions mainly for domain separation. We propose tBC-based variants for these, named Twe-SUNDAE, Twe-CLOC and Twe-SILC, which simplify the original designs (by cleaning up the auxiliary functions) and reduces the number of block cipher calls. These in turn help in reducing the area of hardware implementation, and significantly increasing the throughput for short messages.

2 Preliminaries

NOTATIONS: For $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, \ldots, n\}$, and $\{0,1\}^n$ denotes the set of all n-bit binary strings. We use $\{0,1\}^+$ to denote the set of all non-empty binary strings. $\perp$ denotes the empty string and $\{0,1\}^* = \{0,1\}^+ \cup \{\perp\}$. For any string $X \in \{0,1\}^n$, $|X|$ denotes the number of bits in X, and for $i \in [|X|]$, x_i denotes the i-th significant bit ($x_{|X|}$ being the most significant bit). For $X \in \{0,1\}^+$ and $n \in \mathbb{N}$, $(X)_{[\ell]} := (X_1, \ldots, X_\ell) \xleftarrow{n} X$, denotes the n-bit block parsing of X into $(X)_{[\ell]}$, where $|X_i| = n$ for $[\ell - 1]$, and $X_\ell \in [n]$. For $k \leq n \in \mathbb{N}$, and $X \in \{0,1\}^n$, $\lfloor X \rfloor_k := X_1 \ldots X_k$. The expression $a \ ? \ b \ : \ c$ evaluates to b if a is true and c otherwise.

For $n, m \in \mathbb{N}$, $\mathsf{Perm}(n)$ denotes the set of all permutations over $\{0,1\}^n$, and $\mathsf{Func}(m,n)$ denotes the set of all functions from $\{0,1\}^m$ to $\{0,1\}^n$. For $n, \kappa \in \mathbb{N}$, $\mathsf{TPerm}(\kappa, n)$ denotes the set of all families of permutations $P_k := P(k, \cdot) \in \mathsf{Perm}(n)$ indexed by $k \in \{0,1\}^\kappa$. By extending notation, we use $\mathsf{TPerm}(\kappa, \tau, n)$ to denote the set of all families of permutations $P_{k,t} \in \mathsf{Perm}(n)$, indexed by $(k, \tau) \in \{0,1\}^\kappa \times \{0,1\}^\tau$.

(TWEAKABLE) BLOCK CIPHER: A block cipher with key size κ and block size n is a family of permutations $\mathsf{E} \in \mathsf{TPerm}(\kappa, n)$. For a fixed key $k \in \{0,1\}^\kappa$, we write $\mathsf{E}_k(\cdot) = \mathsf{E}(k, \cdot)$, and its inverse is written as $\mathsf{E}_k^{-1}(\cdot)$. A tweakable block cipher with key size κ, tweak size τ, and block size n is a family of permutations $\mathsf{E} \in \mathsf{TPerm}(\kappa, \tau, n)$. For a fixed key $k \in \{0,1\}^\kappa$ and tweak $t \in \{0,1\}^\tau$, we write $\mathsf{E}_k^t(\cdot) = \mathsf{E}(k, t, \cdot)$, and its inverse is written as $\mathsf{E}_k^{-t}(\cdot)$. Throughout this paper we fix $\kappa, \tau, n \in \mathbb{N}$ as the key size, tweak size, and block size, respectively, of the given (tweakable) block cipher.

3 The Elastic-Tweak Framework

In this section, we introduce the Elastic-Tweak framework (illustrated in Figure 3.1) 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) LinLayer (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 Twe BC $[t, t_e, \mathsf{tic}, \mathsf{gap}]$ where $t, t_e, \mathsf{tic}, \mathsf{gap}$ are suitable parameters as described above.

3.1 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[6] 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 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).

[6] 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.

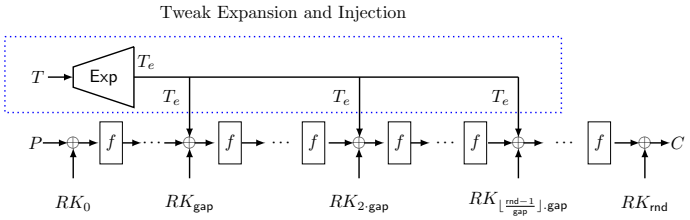

Fig. 3.1: Elastic-Tweak Construction.

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

1. $\tau \leftarrow \frac{|T|}{w}$
2. $T_e \leftarrow \phi$
3. $(T_1, T_2, \ldots, T_\tau) \xleftarrow{w} T$
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. $\quad X \leftarrow \mathsf{LinLayer}(X)$
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

Fig. 3.2: 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.

3.2 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, start + gap , start + 2.gap , . . . , end. To have a uniformity in the tweak injection rounds, we typically choose start = gap and inject the tweaks at an interval of gap rounds. This implicitly sets end = gap.$\lfloor \frac{\mathsf{rnd}-1}{\mathsf{gap}} \rfloor$.

Requirements from Twe BC. We must ensure Twe BC 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 Twe BC.

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.

3.3 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 $t \in \{4, 8, 16\}$.

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) GIFT-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) GIFT-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) AES[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.

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) AES[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) GIFT-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) AES[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.

3.4 Performance

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 C. 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 3.1: 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 3.1 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 3.2 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.

4 Security Analysis

In this section, we provide the various cryptanalysis that we had performed on TweAES and TweGIFT. Note that our target is single-key security, and any related-key attacks are out of our scope.

Table 3.2: 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

4.1 General Approach

Without exploiting the tweak, TweAES and TweGIFT offer exactly the same security as the original AES and GIFT. Hence in this paper, we focus our attention on the attacks that exploit the tweak injection.

The tweak expansion function is chosen to have a high branch number. This is very suitable for small tweaks because the small weight of the input (small tweak) ensures a large weight of the output (expanded tweak).

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 in 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 GIFT64 in Fig. 4.1. Because the entire construction includes several $2r$-round cores, security of the entire construction can be bounded by accumulating the bound for the single $2r$-round core. 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.

4.2 Security Analysis of TweAES

Number of Active S-boxes. All of three instantiations introduced in Sect. 3 inject the expanded tweak in every 2 rounds. As explained in Sect. 4.1, we evaluate the minimum number of differentially and linearly active S-boxes for the 4-round core. The 4-bit, 8-bit and 16-bit tweaks of AES[4, 8, 8, 2], AES[8, 16,

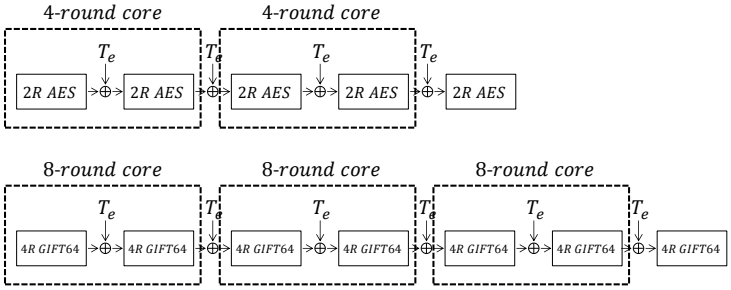

Fig. 4.1: 4-round Core of TweAES[∗,∗,∗,2] and 8-round Core of GIFT64[∗,∗,∗,4].

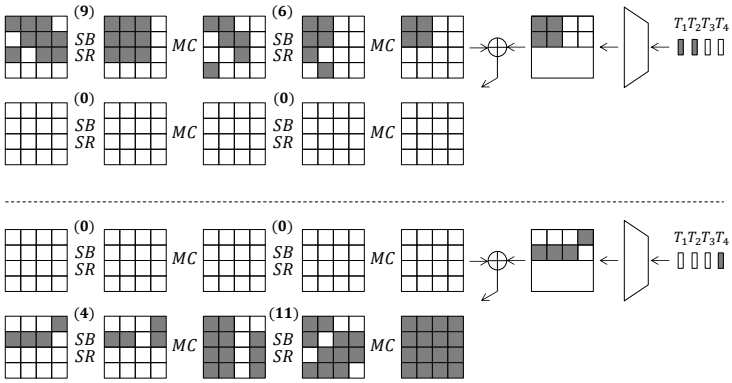

Fig. 4.2: Two Examples of Differential Trails with 15 Active S-boxes.

8, 2] and AES[16, 32, 8, 2] 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, 2-bits and 4-bits for each construction, respectively.

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. Two examples of the differential trails achieving 15 active S-boxes are given in Fig. 4.2. Both trails leave one of the 2-round transformations in the 4-round core blank.

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.

Reduced-Round Versions Starting from Middle Rounds. Security of full TweAES is based on the strong property of the 4-round core. 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 [68] or more precisely formulated version called the sandwich attack [32]. 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 two 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. 4.3 The probability when E_m is a single S-box layer can be measured by using the boomerang connectivity table (BCT) [18]. 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 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 AES[16-32-8-2]. Due to 2 interval rounds between tweaks, distinguishers based on impossible differential attacks [13] can be constructed for relatively long rounds (6 rounds) by canceling the tweak difference with the state difference. The distinguisher is depicted in Fig. 4.4.

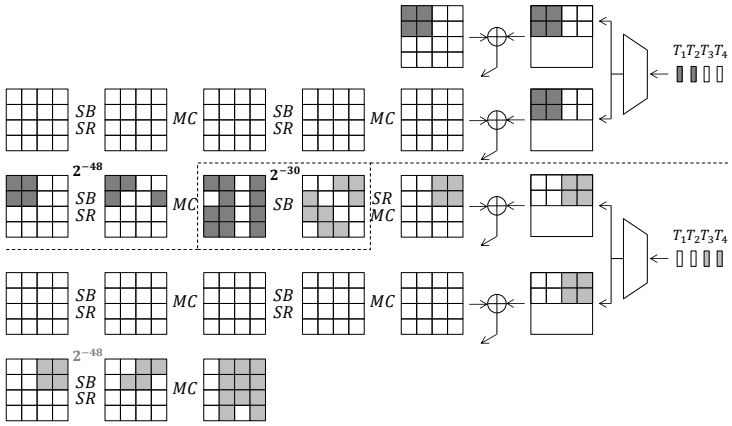

Fig. 4.3: Differential Trails for Boomerang Attacks. The cells filled with black and gray represent active byte positions in E_0 and E_1, respectively.

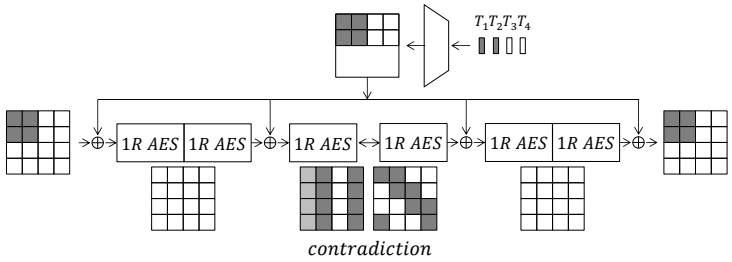

Fig. 4.4: 6-round Impossible Differential Distinguisher. The bytes filled with black, white, and gray have non-zero difference, zero difference, and arbitrary difference, respectively.

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. 4.5. As shown in Fig. 4.5 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.

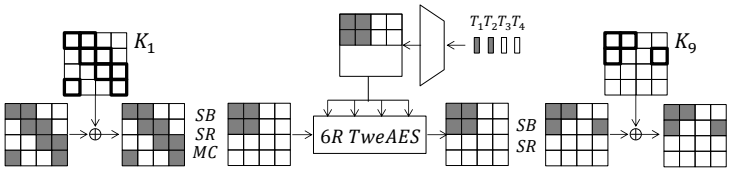

Fig. 4.5: Extension to 8-round Key Recovery

Recall that the attack target is AES[16-32-8-2], where the tweak size is 16 bits. The attack procedure is as follows.

1. Choose all tweak values denoted by T^i where $i = 0, 1, \ldots, 2^{16} - 1$.
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 $i = 0, 1, \ldots, 2^{16} - 1$.
3. For all $\binom{2^{16}}{2} \approx 2^{31}$ 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 $2^{31+64+64-96} = 2^{63}$ 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^{40} times from the first step by changing the inactive byte values at the input. After this step, $2^{63+40} = 2^{103}$ 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 $2^{16} \times 2^{64} \times 2^{40} = 2^{120}$. The time complexity is also 2^{120} memory accesses. The memory complexity is to recored the wrong keys of the 12 bytes, which is 2^{96}.

Summary. We demonstrated two attacks against reduced-round variants that start from the middle of the 4-round core. Because security of TweAES 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. 4.3 and 4.5. The number of involved subkey bytes easily exceeds 16.

Remarks on Other Attacks.

- Integral attacks [22, 47] 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.

Table 4.1: The Best Differential Trail for 8-Round Core of TweGIFT64[4,16,16,4]. The first and the second masks are for the S-box input and S-box output, respectively.

Round	Differential Mask	Probability
1	0000 0000 0000 000c 0000 0000 0000 0004	2^{-2}
2	0000 0004 0000 0000 0000 0005 0000 0000	2^{-2}
3	0000 0400 0000 0100 0000 0500 0000 0500	2^{-5}
4	0000 0101 0000 0404 0000 0909 0000 0d0d	2^{-12}
tweak difference: 0808 0808 0808 0808		
5	0000 090d 0000 090d 0000 0104 0000 0104	2^{-10}
6	0000 0004 0000 0000 0000 0505 0000 0000	2^{-6}
7	0a00 0000 0000 0000 0100 0000 0000 0000	2^{-2}
8	0000 1000 0000 0000 0000 8000 0000 0000	2^{-3}

- Meet-in-the-middle attack [28,29] exploits the 4-round truncated differentials $1 \to 4 \to 16 \to 4 \to 1$ and focuses 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.

4.3 Security Analysis of TweGIFT

In TweAES, security analysis was identical for all instantiations because the different expanded tweak sizes only result in different number of active bits in the same truncated differentials. In contrast, GIFT is a bit-oriented cipher (see supplementary material E). Different expanded tweak sizes result in different active S-boxes patterns. Hence, we need security analysis for each instantiation.

Security of TweGIFT64[4,16,16,4]. We evaluated the maximum probability of the differential trail for the 8-round core by MILP under the constraints that at least one of the tweak bits is active. The tweak expansion guarantees that 8 S-boxes are activated by the tweak difference. As a result, the maximum probability is 2^{-42}, which is smaller than 2^{-40} of the 8-round differential characteristic of original GIFT reported by Zhu et al. [70].

Note that the probability of 2^{-42} in the 8-round core is the tight bound. Namely there exists an actual differential characteristic. The detail of the characteristic is given in Table 4.1. Here, among the 4-bit tweak T_1, T_2, T_3, T_4, T_1 and T_3 are active. This makes the expanded tweak difference 0808 0808 0808 0808, which ensures that the sum of the number of active S-boxes in rounds 4 and 5 is at least 8.

Table 4.2: Upper Bound of Differential Probability of TweGIFT128[16,32,32,r]

r	1	2	4	5
Max prob of $2r$-round core	2^{-8}	$2^{-16.4}$	$2^{-26.4}$	2^{-30}
# of $2r$-round cores in 40 rounds	20	10	5	4
Upper bound for 40 rounds	2^{-160}	2^{-164}	2^{-132}	2^{-120}

As shown in Fig. 4.1, 28 rounds of TweGIFT64 contains at least three 8-round cores, which upper bounds the maximum differential probability for 28 rounds is $2^{-42\times3} = 2^{-126}$.

Security of TweGIFT128[4,32,32,5]. In this expansion, the 4-bit tweak expands to 8 bits and those 8 bits are copied three times to achieve a 32-bit tweak. When the 4-bit tweak has some non-zero difference, the expanded 32-bit tweak is ensured to have at least 16 active bits, which ensures at least 16 active S-boxes in 2 rounds around the tweak injection.

Owing to the large state size and a large number of active S-boxes, it is infeasible to find the tight bound of the maximum probability of the differential characteristic for the 10-round core by using MILP. The tool so far provided that the maximum probability of the differential characteristic is upper bounded by $2^{-64.5}$. Given that the entire TweGIFT128 consists of 40 rounds and thus contains 4 of the 10-round cores, the upper bound of the entire construction is $2^{-64.5\times4} = 2^{-258}$, which is sufficient to resist the attack.

Security of TweGIFT128[16,32,32,r]. TweGIFT128[16,32,32,r] allows the attacker to have more control over the tweak compared to TweGIFT128[4,32,32,r]. Hence the security of TweGIFT128[16,32,32,r] is strictly weaker than that of TweGIFT128[4,32,32,r].

We evaluated the maximum differential probability for the $2r$-round core of TweGIFT128[16,32,32,r] for several choices of r by using MILP, which further leads to the upper bounds for the entire construction based on the number of the $2r$-round cores. The results are shown in Table 4.2.

Considering that the number of interval rounds between tweaks should be as large as possible to save the total number of computations, we chose $r = 4$. Because the entire TweGIFT128 contains 5 of the 8-round cores, the maximum differential characteristic probability for TweGIFT128[16,32,32,4] is upper bounded by $2^{-26.4\times4} = 2^{-132}$. The strongest choice is $r = 2$ as shown in Table 4.2.

The detail of the best differential characteristics for TweGIFT128[16,32,32,2] and TweGIFT128[16,32,32,4] is given in the supplementary material F (see Table F.1). The differential characteristic for TweGIFT128[16,32,32,2] has 1 active bit in the tweak bit T_{18}. The characteristic for TweGIFT128[16,32,32,4] has two active bits in T_0 and T_4.

5 Applications of Short-Tweak Tweakable Block Ciphers

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 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.

FCBC MAC: FCBC mode is a 3-key message authentication code, by Black and Rogaway [16], which is defined as follows:

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

FCBC has not received much appreciation in its existing 3-key form, even though it offers better security, $O(q^2/2^n + q\ell^2/2^n + q^2\ell^4/2^{2n})$ in [44, 45, Theorem 3 and Remark 5], than CMAC [5, 36], $O(q^2\ell/2^n + q^2\ell^4/2^{2n})$ in [58, Theorem 4.6]. Quantitatively, the number of queries per key increases from $2^{3n/8}$ to $2^{n/2}$ for message lengths up to $2^{n/4}$ blocks. This is mainly due to presence of three keys which not only costs keys size of the algorithm but it requires to run three key scheduling algorithms. Keeping these in mind, we define Twe-FCBC, as follows:

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

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 $\widetilde{\mathsf{E}}^0_K$, $\widetilde{\mathsf{E}}^1_K$ and $\widetilde{\mathsf{E}}^2_K$, respectively. Using Proposition 1 and [44, 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*

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

Clearly, Twe-FCBC has two major advantages over CMAC- (i) no need to hold an additional state for final message block masking, (ii) security bound is free of length factor for all reasonably sized messages (close to 6 Gigabyte for a 128-bit block cipher). 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.

Double Block Hash-then-Sum: The very basic version of Double-block Hash-then-Sum or DbHtS [24], 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 PMAC_Plus, LightMAC_Plus, SUM_ECBC and 3kf9. 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 [24, Theorem 2.(iii)] also applies to Twe-DbHtS. Formally, the security of Twe-DbHtS is given by Proposition 3.

Proposition 3.

$$\mathbf{Adv}^{\mathsf{prf}}_{\mathsf{Twe\text{-}DbHtS}[\mathsf{H},\widetilde{\mathsf{E}}]}(q,\ell,t) \leq 2\mathbf{Adv}^{\mathsf{tprp}}_{\widetilde{\mathsf{E}}}(2q,t') + \mathbf{Adv}^{\mathsf{prf}}_{C_3^*[H,\pi_0,\pi_1,\pi_2]}(q,\ell,t).$$

In this way, we have one-key versions of different well known designs PMAC_Plus, LightMAC_Plus, SUM_ECBC, 3kf9 etc. We note that one key version of PMAC_Plus based on solely block cipher has been proposed [25]. 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.[7]

[7] 1kf9 is proposed in ePrint [26], which later found to be attacked in birthday complexity [51].

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) [20] (3) Encrypted Wegman Carter Davis Meyer (EWCDM) [20] (4) Chained LRW2 (CLRW2) [50] (5) GCM-SIV-2 [37] and (6) The Benes Construction [60]. 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 1. 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 [56].

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 [56] and PMACx [53]. 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. 5.1. 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 4. *For $q \leq 2^{n-1}$,*

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

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

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

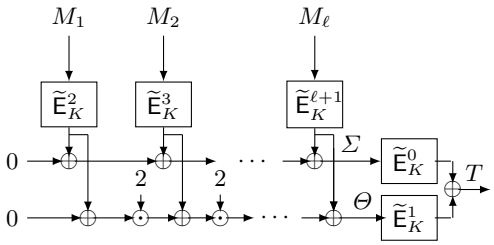

Fig. 5.1: Twe-LightMAC_Plus construction.

from [24]. One can verify from [24, 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 [53]. Using similar arguments as in [53, 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. □

We note that similar improvements can also be applied to PMAC, PMAC_Plus.

5.3 A Note on tBC's Advantages over XE and XEX

The XE and XEX modes, by Rogaway [62], 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 [15], OCB [63], COPA [9], ELmD [27] etc. The XE scheme to generate a TBC $\widetilde{\mathsf{E}}$ from a BC E is defined as

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

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.[8] 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:

1. 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

[8] Alternative constructions to define Δ can be found in [17, 34].

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.

2. Additional Computational and Storage Overheads: The XE mode requires, precomputation 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. In this respect, one can easily define simple tBC-variants of PMAC [62] (based on XE), COPE [9] (based on XEX), COLM [8] (XE like processing) etc. much along the same line as Twe-LightMAC_Plus.

6 Simplification of Authenticated Encryption Schemes

In the previous section, we showed some examples where the use of tBC leads to simpler and lighter (in terms of state size and/or number of block cipher calls) designs. In this section, we demonstrate some AE schemes that achieve a combination of these advantages. Various security definitions and notions used here, are available in supplementary material A.

6.1 Twe-SUNDAE, Twe-CLOC and Twe-SILC

We propose tBC variant for SUNDAE, CLOC and SILC, called Twe-SUNDAE, Twe-CLOC and Twe-SILC, respectively. SUNDAE is a deterministic authenticated encryption (DAE) mode that aims at maximal robustness using small state size, and it is particularly efficient for short messages. CLOC and SILC are nonce-based authentication encryption (NAEAD) modes, which aim to optimize the implementation overhead beyond the block cipher calls, the precomputation complexity, and the memory requirement. CLOC is suitable for uses in embedded processors, and SILC aims to optimize hardware implementation cost. Our choices of SUNDAE, CLOC and SILC are motivated by two factors (see subsection 6.2 below): design simplification and reduction in block cipher calls.

The three tBC variants are described in Fig. 6.1. We have made minimal changes in the original schemes. Twe-SUNDAE employs a SIV [64] approach and uses a variant of FCBC [16] combined with the OFB [3] mode encryption. CLOC and SILC employ Encrypt-then-PRF paradigm and use a variant of CFB [3] mode in its encryption part and a variant of FCBC in the authentication part.

6.2 Features of the Proposed AE Schemes

The proposed tBC-based AE schemes offer two added features over the existing block cipher based schemes.

Design Simplification: Twe-SUNDAE greatly simplifies the design of SUNDAE by avoiding field multiplications with 2, 4 etc. Twe-CLOC and Twe-SILC simplifies their respective original algorithms very efficiently. CLOC and SILC

$\mathsf{OFB}(V, M, t)$

1. $M_1 \| \cdots \| M_m \leftarrow M$
2. **for** $i = 1$ **to** m
3. $\quad V \leftarrow \widetilde{\mathscr{E}}_K^t(T)$
4. $\quad C_i \leftarrow \lfloor V \rfloor_{|M_i|} \oplus M_i$
5. **return** $(C_1 \| \cdots \| C_m)$

$\mathsf{CFB}(V, M, t)$

1. $M_1 \| \cdots \| M_m \leftarrow M$
2. $C_1 \leftarrow V \oplus M_1$
3. **for** $i = 2$ **to** m
4. $\quad C_i \leftarrow \lfloor \widetilde{\mathscr{E}}_K^t(C_{i-1}) \rfloor_{|M_i|} \oplus M_i$
5. **return** $(C_1 \| \cdots \| C_m)$

$\mathsf{ivFCBC}(T, D, t_0, t_1, t_2)$

1. $D_1 \| \cdots \| D_d \leftarrow D$
2. **for** $i = 1$ **to** $d - 1$
3. $\quad T \leftarrow \widetilde{\mathscr{E}}_K^{t_0}(T \oplus D_i)$
4. $t \leftarrow (|D_d| = n)?\ t_1 : t_2$
5. $T \leftarrow \widetilde{\mathscr{E}}_K^t\big(T \oplus \mathsf{pad}(D_d)\big)$
6. **return** T

$\mathsf{Twe\text{-}SUNDAE}_K(A, M)$

1. $T \leftarrow \mathsf{ivFCBC}(0^n, A, 0, 1, 2)$
2. $T \leftarrow \mathsf{ivFCBC}(T, M, 3, 4, 5)$
3. $C \leftarrow \mathsf{OFB}(T, M, 6)$
4. **return** (C, T)

$\mathsf{Twe\text{-}SILC}_K(N, A, M)$

1. $T \leftarrow \mathsf{ivFCBC}(0^n, N \| A, 0, 0 \| 1, \mathsf{Len} \| 1)$
2. $C \leftarrow \mathsf{CFB}(T, M, 0 \| 2)$
3. $T \leftarrow \mathsf{ivFCBC}(0, C, 1, 0 \| 3, \mathsf{Len} \| 0)$
4. **return** (C, T)

$\mathsf{Twe\text{-}CLOC}_K(N, A, M)$

1. $T \leftarrow \mathsf{ivFCBC}(T, A \| N, 0, 1, 2)$
2. $C \leftarrow \mathsf{CFB}(T, M, 3)$
3. $T \leftarrow \mathsf{ivFCBC}(0, C, 4, 5, 6)$
4. **return** (C, T)

Fig. 6.1: Encryption and algorithm of Twe-SUNDAE, Twe-SILC and Twe-CLOC. pad uses 10^* padding for Twe-SUNDAE and Twe-CLOC and 0^* padding for Twe-SILC.

require several linear functions (f, g_1, g_2, h_1, h_2 for CLOC and g for SILC) for domain separations and bit fixing operations. Twe-CLOC and Twe-SILC perform all the domain separations by using distinct tweaks, which significantly simplifies the design.

Table 6.1: Comparison between the number of (tweakable) block cipher invocations for original SUNDAE, CLOC and SILC, and their tBC counterparts. Here a, and m denote the length of associated data and plaintext, respectively.

Modes	No. of BC calls		No. of tBC calls	
	$a \neq 0$	$a = 0$	$a \neq 0$	$a = 0$
SUNDAE	$a + 2m + 1$	$2m + 1$	$a + 2m$	$2m$
CLOC	$a + 2m + 1$	$2m + 2$	$a + 2m$	$2m$
SILC	$a + 2m + 3$	$2m + 2$	$a + 2m$	$2m$

Energy Efficient for Short Inputs: Apart from the simplification of the original designs, the proposed AE schemes offer another advantage over the non-tweaked versions. They require lesser number of block cipher calls for shorter/empty AD or message processing, which essentially makes them more efficient in terms of energy consumption.

The number of block cipher invocations required to process an associated data of a blocks and message of m blocks are given in Table 6.1. As seen from the table, SUNDAE requires 2 block cipher calls to process a 1 block AD and empty message, where as Twe-SUNDAE requires only 1 block cipher call. The advantage is even bigger for Twe-SILC. While SILC requires 4 block cipher calls to process 1 block AD and empty message, Twe-SILC requires only 1 block cipher call.

6.3 Hardware Performances

In Table 6.2, we summarize the hardware performances for SUNDAE and Twe-SUNDAE instantiated with AES and TweAES$[4, 8, 8, 2]$ respectively. Implementation details can be found in supplementary material D. It is interesting to

Table 6.2: Performance of SUNDAE-AES and Twe-SUNDAE-TweAES[4 ,8 ,8 ,2] in a Virtex 7 FPGA.

Construction	LUTs	Slices	Frequency (MHz)	Message length (Bytes)	Clock Cycles	Throughput (Mbps)
SUNDAE-AES	2532	675	315.96	16	21	1925.85
				32	31	2609.22
				64	51	3171.99
				128	91	3555.42
				512	331	3909.89
				1024	651	3975.95
Twe-SUNDAE-TweAES[8,4,2,1]	2235	679	314.71	16	11	3362.08
				32	21	3836.47
				64	41	3930.04
				128	81	3978.56
				512	321	4015.74
				1024	641	4022.00

see that the simplification of the design in Twe-SUNDAE helps Twe-SUNDAE-TweAES [4, 8, 8, 2] to save an area of about 300 LUTs. This is primarily due to the reduction in the size of the multiplexers to select the input to the cipher and avoiding the field multiplications. The extra logic introduced to generate the different tweak values for Twe-SUNDAE essentially requires a 16×4 ROM memory, which is easily implemented in four LUTs. Also note that, as expected, Twe-SUNDAE achieves excellent throughput for short messages saving one block cipher call. This results depicts how tBC can help in reducing the hardware footprint of a mode by simplifying the design, and at the same time improving the throughput for small messages.

6.4 Security of the Proposed AE Schemes

Twe-SUNDAE, Twe-CLOC and Twe-SILC are in essence just the multi-key variants of SUNDAE, CLOC and SILC, respectively. So, intuitively they should be at least as secure as the original modes, and the security argument for these schemes is relatively easier than the original schemes. We show in Proposition 5 and 6 that both our intuitions are correct to a large extent. For the sake of simplicity, we refrain from giving exact bounds, and instead give the asymptotic expressions.

Security of Twe-SUNDAE: It will be easier for us to argue the security of Twe-SUNDAE by viewing it as an instance of the SIV paradigm [64]. So we digress a little to briefly explain the SIV paradigm.

THE SIV PARADIGM: SIV is a design paradigm for constructing DAE schemes. It is composed of two stages: a tag generation stage, F that computes the tag T on AD A and PT M; and an IV-based encryption stage, ivE that computes the ciphertext C on PT M using T as IV. Formally, for key space $\mathcal{L} \times \mathcal{K}$ the encryption algorithm of SIV is defined by the following mapping

$$(L, K, A, M) \mapsto \mathsf{F}(L, A, M) \,\|\, \mathsf{ivE}\,(K, \mathsf{F}(L, A, M), M)\,,$$

for all $(L, K, A, M) \in \mathcal{L} \times \mathcal{K} \times \mathcal{A} \times \mathcal{M}$. Here, $T := \mathsf{F}(L, A, M) \in \mathcal{T}$, and $C := \mathsf{ivE}(K, T, M) \in \mathcal{M}$. In general, the ivE is defined via a weak PRF G, as illustrated in Figure 6.2. Here we will focus on SIV with weak PRF based ivE. Following the security definitions of [35, 37, 57, 64], we have the following result on the DAE security of an SIV scheme $\mathfrak{A}$:

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

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

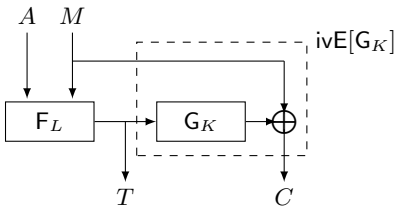

Fig. 6.2: The SIV paradigm based on a PRF F, and a weak PRF G in the IV-based encryption phase. F denotes the tag gneneration phase, and the dashed rectangle, labeled with $\mathsf{ivE}[\mathsf{G}_K]$ denotes the IV-based encryption phase based on G.

Coming back to Twe-SUNDAE, the tag generation phase F is actually a variant of FCBC (see section 5), which has been shown to have security in the order of $O(\sigma^2/2^n)$ in the original paper by Black and Rogaway [16]. The ivE phase is an instance of the OFB mode with random IV, which is known to have $O(\sigma^2/2^n)$ security from folklore. A formal security proof for OFB is also available in [69]. Note that a better security bound exists for FCBC (as noted in section 5.1), but the weaker bound suffices for Twe-SUNDAE, as the bound for ivE phase is tight. On substituting the security bounds for F and ivE in Eq. (1), we get the following security result on Twe-SUNDAE.

Proposition 5. *The security of Twe-SUNDAE is given by,*

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

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

Security of Twe-CLOC and Twe-SILC: Like SUNDAE, we first look at the abstract design paradigm behind Twe-CLOC and Twe-SILC, which is the so-called Encrypt-then-PRF, or EtPRF.

THE EtPRF PARADIGM: EtPRF [57, Construction A5] is a design paradigm to construct NAEAD schemes. It is composed of three stages (illustrated in Figure 6.3): a random IV generator, G that generates iv using the nonce N and (possibly) the AD A; an IV-based encryption phase, ivE that generates the ciphertext C using iv as the random IV; and a tag-generation phase, F that generates the tag on the input N, A, C. Formally, for key space $\mathcal{K} \times \mathcal{L}$ the encryption algorithm of EtPRF is defined by the following mapping

$$(K, L, N, A, M) \mapsto \mathsf{ivE}(K, N, A, M) \,\|\, \mathsf{F}\,(L, N, A, \mathsf{ivE}(K, N, A, M))\,,$$

for all $(L, K, N, A, M) \in \mathcal{L} \times \mathcal{K} \times \mathcal{A} \times \mathcal{M}$. Here, $C := \mathsf{ivE}(K, N, A, M) \in \mathcal{M}$, and $T := \mathsf{F}(L, N, A, C) \in \mathcal{M}$. Note that, for the sake of simplicity we subsumed the G function within the ivE phase. In [57], Namprempre et al. showed that the NAEAD security of an EtPRF scheme, $\mathfrak{A}$, given by:

$$\mathbf{Adv}^{\mathsf{ae}}_{\mathfrak{A}}(q, \ell, \sigma) \leq \mathbf{Adv}^{\mathsf{prf}}_{\mathsf{F}}(q, \ell, \sigma) + \mathbf{Adv}^{\mathsf{prf}}_{\mathsf{G}}(q, \ell, \sigma) + \mathbf{Adv}^{\mathsf{priv\$}}_{\mathsf{ivE}}(q, \ell, \sigma), \quad (3)$$

where PRIV denotes the Priv$ security (see supplementary material A).

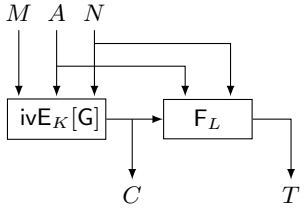

Fig. 6.3: The EtPRF paradigm based on an IV-based encryption scheme ivE for the encryption phase, and a PRF F for the tag generation phase. The $[\mathsf{G}]$ denotes that ivE internally uses G to generate the random IV.

In case of both Twe-CLOC and Twe-SILC, G and F are variants of Twe-FCBC, and hence can be shown to have $O(\sigma^2/2^n)$ PRF security [16]. ivE phase is an instance of the CFB mode with random IV, which has been shown to have $O(\sigma^2/2^n)$ security in [69]. Hence, by substituting the relevant bounds in Eq. (3), we get the following security result for Twe-CLOC and Twe-SILC.

Proposition 6. *The security of Twe-CLOC and Twe-SILC is given by:*

$$\mathbf{Adv}^{\mathsf{ae}}_{\mathit{Twe\text{-}CLOC}[\widetilde{\mathsf{E}}]}(t, q, \ell, \sigma) \leq \mathbf{Adv}^{\mathsf{tprp}}_{\widetilde{\mathsf{E}}}(t', q\ell) + O(\frac{\sigma^2}{2^n}),$$
$$\mathbf{Adv}^{\mathsf{ae}}_{\mathit{Twe\text{-}SILC}[\widetilde{\mathsf{E}}]}(t, q, \ell, \sigma) \leq \mathbf{Adv}^{\mathsf{tprp}}_{\widetilde{\mathsf{E}}}(t', q\ell) + O(\frac{\sigma^2}{2^n}).$$

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

Remark 2. The security of SUNDAE does not follow from Eq. (1), in a straightforward way, as the tag generation and encryption share the same key. Similarly the security of CLOC and SILC does not follow from Eq. (3).

7 Further Applications and Future Directions

We think that tBC can have several other applications. For instance, consider a scenario where two multiple algorithms are running on the same platform, sharing the same secret key. We could find several examples where such an arrangement could be vulnerable. For example, consider a scenario where AES-GCM and AES-CMAC are running on the same device, sharing the same secret key. Now, it is easy to see that, an adversary can trivially forge a tag for AES-CMAC using an encryption query on AES-GCM. tBC can efficiently take care of such problems by separating these algorithms using different tweak values, i.e. unique tweak values for each of these algorithms.

We have defined the Elastic-Tweak framework for SPN based block ciphers. Extending this further for ARX based constructions could be an interesting problem. Also, it would be interesting to see designs for short-tweak tweakable public permutations, which might have strong impact on the simplification of permutation based constructions such as Sponge, Beetle, Minalpher etc.

References

1. Can fd standards and recommendations. `https://www.can-cia.org/news/cia-in-action/view/can-fd-standards-and-recommendations/2016/9/30/`.
2. Electronic product code (epc) tag data standard (tds). `http://www.epcglobalinc.org/standards/tds/`.
3. Recommendation for Block Cipher Modes of Operation: Methods and Techniques. NIST Special Publication 800-38A, 2001. National Institute of Standards and Technology.
4. 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.
5. Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. NIST Special Publication 800-38B, 2005. National Institute of Standards and Technology.
6. 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.
7. NIST FIPS 197. Advanced Encryption Standard (AES). *Federal Information Processing Standards Publication*, 197, 2001.
8. Elena Andreeva, Andrey Bogdanov, Nilanjan Datta, Atul Luykx, Bart Mennink, Mridul Nandi, Elmar Tischhauser, and Kan Yasuda. COLM v1. CAESAR Competition.
9. 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`.

10. 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.
11. 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 *CHES 2017. Proceedings*, pages 321–345, 2017.
12. 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.
13. Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. *J. Cryptology*, 18(4):291–311, 2005.
14. Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. In *Advances in Cryptology - CRYPTO '90, Proceedings*, pages 2–21, 1990.
15. 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.
16. John Black and Phillip Rogaway. CBC macs for arbitrary-length messages: The three-key constructions. *J. Cryptology*, 18(2):111–131, 2005.
17. 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.
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.
19. Benoit Cogliati, Rodolphe Lampe, and Yannick Seurin. Tweaking even-mansour ciphers. In *CRYPTO 2015. Proceedings, Part I*, pages 189–208, 2015.
20. 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.
21. Paul Crowley. Mercy: A fast large block cipher for disk sector encryption. In *Fast Software Encryption – FSE 2000. Proceedings*, pages 49–63, 2000.
22. 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.
23. 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.
24. 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.
25. 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.
26. Nilanjan Datta, Avijit Dutta, Mridul Nandi, Goutam Paul, and Liting Zhang. Single key variant of pmac_plus. *IACR Cryptology ePrint Archive*, 2017:848, 2017.
27. Nilanjan Datta and Mridul Nandi. Proposal of ELmD v2.1. Submission to CAESAR, 2015. `https://competitions.cr.yp.to/round2/elmdv21.pdf`.
28. 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.

29. 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.
30. Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. Square attack on 7-round kiasu-bc. In *Applied Cryptography and Network Security - ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings*, pages 500–517, 2016.
31. Christoph Dobraunig and Eik List. Impossible-differential and boomerang cryptanalysis of round-reduced kiasu-bc. In *Topics in Cryptology - CT-RSA 2017 - San Francisco, CA, USA, February 14-17, 2017, Proceedings*, pages 207–222, 2017.
32. Orr Dunkelman, Nathan Keller, and Adi Shamir. A Practical-Time Related-Key Attack on the KASUMI Cryptosystem Used in GSM and 3G Telephony. In *CRYPTO 2010*, volume 6223 of *LNCS*, pages 393–410. Springer.
33. 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.
34. 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.
35. Shay Gueron and Yehuda Lindell. GCM-SIV: full nonce misuse-resistant authenticated encryption at under one cycle per byte. In *ACM SIGSAC Conference on Computer and Communications Security 2015. Proceedings*, pages 109–119, 2015.
36. Tetsu Iwata and Kaoru Kurosawa. OMAC: One-Key CBC MAC. In *FSE*, pages 129–153, 2003.
37. Tetsu Iwata and Kazuhiko Minematsu. Stronger security variants of GCM-SIV. *IACR Cryptology ePrint Archive*, 2016:853, 2016.
38. 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`.
39. 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.
40. 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.
41. Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. Deoxys v1.41. Submission to CAESAR, 2016. `https://competitions.cr.yp.to/round3/deoxysv141.pdf`.
42. Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. KIASU v1. Submission to CAESAR, 2016. `https://competitions.cr.yp.to/round1/kiasuv1.pdf`.
43. 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.
44. Ashwin Jha and Mridul Nandi. Revisiting structure graph and its applications to CBC-MAC and EMAC. *IACR Cryptology ePrint Archive*, 2016:161, 2016.
45. Ashwin Jha and Mridul Nandi. Revisiting structure graphs: Applications to CBC-MAC and EMAC. *J. Mathematical Cryptology*, 10(3-4):157–180, 2016.
46. 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.
47. 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.

48. Ted Krovetz and Phillip Rogaway. The Software Performance of Authenticated-Encryption Modes. In *FSE*, pages 306–327, 2011.
49. Rodolphe Lampe and Yannick Seurin. Tweakable blockciphers with asymptotically optimal security. In *FSE 2013. Revised Selected Papers*, pages 133–151, 2013.
50. 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.
51. 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.
52. Moses Liskov, Ronald L. Rivest, and David A. Wagner. Tweakable block ciphers. In *CRYPTO 2002*, pages 31–46, 2002.
53. Eik List and Mridul Nandi. ZMAC+ - an efficient variable-output-length variant of ZMAC. *IACR Trans. Symmetric Cryptol.*, 2017(4):306–325, 2017.
54. Atul Luykx, Bart Preneel, Elmar Tischhauser, and Kan Yasuda. A MAC mode for lightweight block ciphers. In *FSE 2016*, pages 43–59, 2016.
55. Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In *Advances in Cryptology - EUROCRYPT '93, Proceedings*, pages 386–397, 1993.
56. 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.
57. Chanathip Namprempre, Phillip Rogaway, and Thomas Shrimpton. Reconsidering generic composition. In *Advances in Cryptology - EUROCRYPT 2014. Proceedings*, pages 257–274, 2014.
58. Mridul Nandi. Improved security analysis for OMAC as a pseudorandom function. *J. Mathematical Cryptology*, 3(2):133–148, 2009.
59. NIST. Data Encryption Standard (AES). *FIPS Publication (Withdrawn)*, 46-3, 1999.
60. Jacques Patarin. A proof of security in $o(2^n)$ for the benes scheme. In *Progress in Cryptology - AFRICACRYPT 2008*, pages 209–220, 2008.
61. 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.
62. Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In *Advances in Cryptology - ASIACRYPT 2004. Proceedings*, pages 16–31, 2004.
63. 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.
64. Phillip Rogaway and Thomas Shrimpton. A Provable-Security Treatment of the Key-Wrap Problem. In *EUROCRYPT 2006*, pages 373–390, 2006.
65. Richard Schroeppel. The Hasty Pudding Cipher. 1998.
66. Siang Meng Sim. GIFT ciphers' reference implementation in C. 2018. [Online; accessed 10-February-2019].
67. 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.
68. David A. Wagner. The Boomerang Attack. In Lars R. Knudsen, editor, *FSE 1999*, volume 1636 of *LNCS*, pages 156–170. Springer, 1999.
69. Mark Wooding. New proofs for old modes. *IACR Cryptology ePrint Archive*, 2008:121, 2008.
70. Baoyu Zhu, Xiaoyang Dong, and Hongbo Yu. MILP-based Differential Attack on Round-reduced GIFT. *IACR Cryptology ePrint Archive*, 2018:390, 2018.

Supplementary Material

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

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

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

$$\mathbf{Adv}^{\mathsf{tsprp}}_{\widetilde{\mathsf{E}}}(q, t) := \max_{\mathcal{A}} \mathbf{Adv}^{\mathsf{tsprp}}_{\widetilde{\mathsf{E}}}(\mathcal{A}). \tag{4}$$

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 $\mathsf{F} := \{\mathsf{F}_K : \{0,1\}^{m} \to \{0,1\}^{n}\}_{K \in \{0,1\}^{\kappa}}$ is defined as

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

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

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

The keyed family of functions F 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 $\mathbf{Adv}^{\mathsf{wprf}}_{\mathsf{F}}(q, t)$.

IV-Based Encryption: An IV-Based Encryption ivE scheme is a tuple $\Psi := (\mathcal{K}, \mathcal{N}, \mathcal{M}, \mathsf{Enc}, \mathsf{Dec})$. Encryption algorithm Enc takes a key $K \in \mathcal{K}$ and a message

$M \in \mathcal{M}$ and returns $(\mathsf{iv}, C) = \mathsf{Enc}(K, M)$, where $\mathsf{iv} \in \mathcal{N}$ is the initialization vector and $C \in \mathcal{M}$ is the ciphertext. Decryption algorithm Dec takes K, iv, C and returns $M = \mathsf{Dec}(K, \mathsf{iv}, C)$. Correctness condition says that for all $K \in \mathcal{K}$ and $M \in \mathcal{M}$ $\mathsf{Dec}(K, \mathsf{Enc}(K, M)) = M$. The Priv\$ advantage [35, 37, 57, 64] of $\mathcal{A}$ is defined as

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

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 ivE, is defined as

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

(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 $\mathcal{A}$, and a tag space $\mathcal{T}$, along with two functions $\mathsf{Enc} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{M} \times \mathcal{T}$, and $\mathsf{Dec} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \times \mathcal{T} \to \mathcal{M} \cup \{\perp\}$, with the correctness condition that for any $K \in \mathcal{K}, N \in \mathcal{N}, A \in \mathcal{A}, M \in \mathcal{M}$, we must have $\mathsf{Dec}(K, N, A, \mathsf{Enc}(M)) = M$. When the nonce space is empty, we call the AE scheme a deterministic AE or DAE scheme.

Following the security definition in [35, 37, 57, 64], we define the NAEAD (DAE for deterministic AE) advantage of $\mathcal{A}$ as

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

where $\mathsf{K} \leftarrow\!\!\$\, \mathcal{K}$ and Γ is a random function from $\mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{M} \times \mathcal{T}$, 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

$$\mathbf{Adv}^{\mathsf{ae}}_{\mathfrak{A}}(q, t) := \max_{\mathcal{A}} \mathbf{Adv}^{\mathsf{ae}}_{\mathfrak{A}}(\mathcal{A}). \tag{7}$$

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 $\mathsf{XOR2}$, is defined by the mapping $x \mapsto \mathsf{E}_{K_0}(x) \oplus \mathsf{E}_{K_1}(x)$.

The $\mathsf{XOR2}$ construction has been proved to be n-bit secure independently by Patarin [61] and Dai et al. [23], though the proof by Patarin still has some unresolved gaps. There is a single key variant of $\mathsf{XOR2}$, 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 [23, Theorem], we obtain

Proposition 7. *For $q \leq 2^{n-4}$,*

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

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) [20]:** 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) [20]:** 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) [50]:** 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 [49] using r distinct tweaks.

4. **GCM-SIV-2 [37]**. 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 [60]:** 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,

$$L' := f_1(L) \oplus f_2(R)$$

$$R' := f_3(L) \oplus f_4(R)$$
$$\mathsf{Benes}(L, R) := (f_5(L') \oplus f_6(R'), f_7(L') \oplus f_8(R')).$$

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 $f_i := \widetilde{\mathsf{E}}_K^{2i} \oplus \widetilde{\mathsf{E}}_K^{2i+1}$ for each $i \in [8]$.

C Software Performance

In this section, we provide a comparison between the software implementations of different TweAES variants with their original counterpart, i.e. AES. Our reference implementations of TweAES are fully based on an AES-NI based implementation of AES, given in the white paper by Gueron. For GIFT and TweGIFT we are not presenting the exact cpb values. But, based on our experiments on the reference implementation of GIFT available in [66], we believe that TweGIFT should have at most 25% overhead over GIFT.

Platform Setup and Benchmarking Methodology: We have implemented and benchmarked all the schemes on Intel's Skylake microarchitecture. We use `gcc` compiler with the flags "`-maes -msse4 -O3`". All measurements were taken on a single core of an Intel Core i7-6500U CPU with Turbo Boost and Hyperthreading disabled. The cache warmup parameter is set at 1000 iterations. The reported performance data were obtained as the median of 101 averaged cycles per byte (cpb) of 10000 measurements each [48].

Implementation Details and Comparison between TweAES and AES: The only difference between our implementation TweAES and AES, is the inclusion of a tweak expansion function, which contributes the major part of the overhead. We initialize the extended tweak to all zeroes. The tweak expansion function is only called for non-zero tweaks. This helps in reducing the overheads, in case of zero tweak, to negligible value (the cpb values differ after 3rd decimal place) as observed from Table C.1. For non-zero tweaks there is an increase in cpb values due to the call to tweak expansion function. Still we think the performance values are tolerable for non-zero tweaks as well. Note that the cpb values for TweAES-128[8,16,8,2] are similar to TweAES-128[16,32,32,5] as the expansion functions are quite similar in the two cases. Table C.1 summarizes the encryption (first row) and decryption (second row) call cpb results for AES-128 and the three TweAES variants.

D Hardware Implementation SUNDAE and Twe-SUNDAE

As an application of our implementation of tweaked block ciphers, we have also implemented SUNDAE with AES and Twe-SUNDAE with TweAES [8, 4, 2, 1]

Table C.1: Comparison between AES-128 and the three TweAES variants. For TweAES tBCs the entry "a, b" denotes the pair of cpb values corresponding to zero tweak and non-zero tweak, respectively.

AES-128	TweAES-128[4, 8, 8, 2]	TweAES-128[8, 16, 8, 2]	TweAES-128[16, 32, 32, 5]
2.82	2.82, 3.15	2.82, 3.63	2.82, 3.68
2.82	2.82, 3.16	2.82, 3.65	2.83, 3.67

as underlying ciphers. First, we discuss on the implementations of AES and TweAES. The components used for both AES and TweAES are given below:

AES encryption round. It executes three transformations to the state; byte-substitution, shift-rows, and mix-columns, for the last round mix-column is omitted.
Round keys generator. It generates and stores the round keys taking as input the initial key K.
Add round key. It is a simple xor and computes the addition of the round key to the state.
Tweak expansion. It takes as input a t-bit tweak and expands it to t_e-bit string. This component is used only in TweAES.
Inj. It contains t_e/w multiplexers of two inputs, the size of such inputs is w. These multiplexers select between an expanded tweak value of zero; the selections depends on the number of actual round and the value it indexes in the *Tweak Scheduling Vector*. As *tweak expansion* this component is used only to instantiate TweAES.

Based on these, we provide the implementation of SUNDAE and Twe-SUNDAE. In the Figure D.1 we show the proposed architecture for SUNDAE, in the same Figure we illustrate what is changing when the tweaked cipher is used instance a usual one (This not mean that both are implemented at the same time, each of them was implemented separately). In the center of the image, there is a box which inside has a cipher that can be tweaked or not tweaked. The components outside the cipher box in Figure D.1 are used to implement SUNDAE with AES (denoted as SUNDAE-AES) or with Twe-SUNDAE with TweAES (denoted as Twe-SUNDAE-TweAES).

For both additionally to the cipher there are two multiplexers, one to select the input of the cipher and other to select the output of the architecture. The dotted lines at its inputs are only used for SUNDAE since it requires field multiplications by 2 and by 4. For SUNDAE, it is also necessary to encrypt the first block V which depends on the length of the associated data and message. On the other side, for Twe-SUNDAE the first block is 0^n and the tweak changes depending if the message or associated data are empty. To process the associated data, the xor of the input block and the output of the cipher are added and fed again to the cipher performing CBC mode of operation. For encryption/decryption of the message, the architecture is configured to perform OFB mode. For decryption it

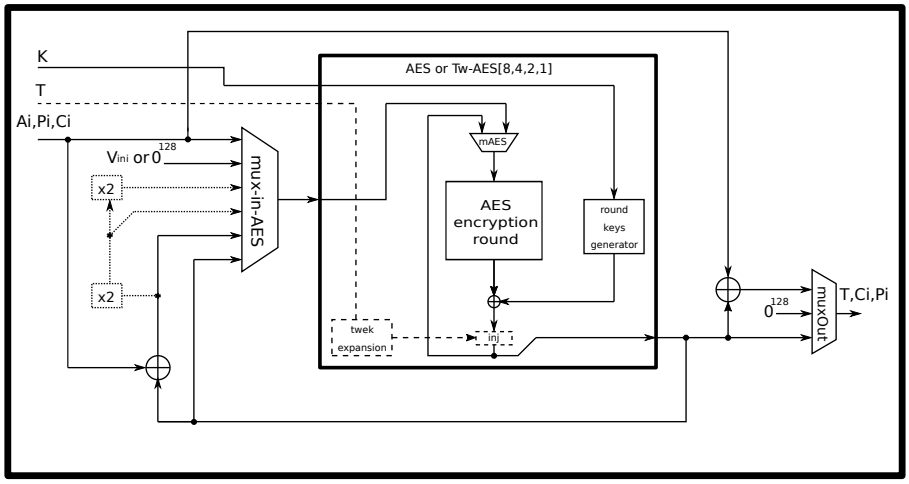

Fig. D.1: Architecture for SUNDAE using AES and TweAES. Non-continuous lines are the overhead to convert AES to TweAES, and dotted lines are the overhead of SUNDAE with usual AES in comparison when it is instantiated with TweAES.

is necessary to feed the verification Tag as IV to decrypt the message with OFB, this is done using the input port in the architecture which receives A_i, P_i, C_i. The multiplexer *muxOut* selects the output of the architecture between the output of OFB mode (ciphertext/plaintext) or directly from the cipher (Tag), when a non-valid data is in the output of the cipher, *muxOut* outputs zero.

From Figure D.1 we can observe that the use of TweAES saves two inputs to the cipher doing the multiplexer smaller, and also avoid the field multiplications. These savings are reflected in the area utilized by the implementations reported.

E Specification of GIFT

GIFT [11] 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.

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 E.1 to 16 nibbles $x_{4i+3}, x_{4i+2}, x_{4i+1}, x_{4i}$, $\forall i = 0, 1, \cdots, 15$ in parallel.

Table E.1: 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 E.2 is applied to the 64-bit state.

Table E.2: 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

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.

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

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. E.1.

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. E.2.

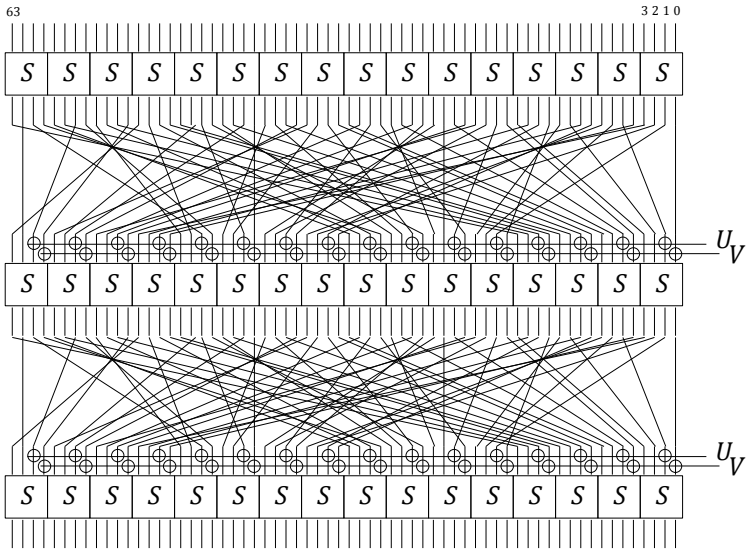

Fig. E.1: Schematic Diagram of Two Rounds of GIFT64.

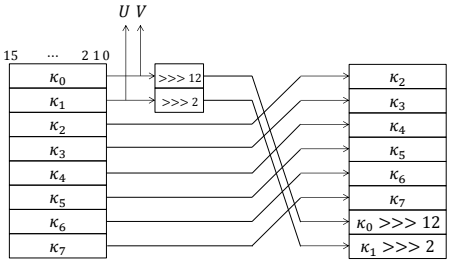

Fig. E.2: Schematic Diagram of Key Schedule Function of GIFT64.

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 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.

F Best Differential Characteristics for $2r$-Round Core of TweGIFT128

Table F.1: The Best Differential Trail for 4-Round Core of TweGIFT128[16,32,32,2] and 8-Round Core of TweGIFT128[16,32,32,4].

Round	Differential Mask	Probability
1	0000 0000 0000 4000 0000 0000 0000 0000 0000 0000 0000 5000 0000 0000 0000 0000	2^{-2}
2	0004 0000 0000 0000 0001 0000 0000 0000 0009 0000 0000 0000 000c 0000 0000 0000	2^{-6}
Δ tweak: 8000 8000 0000 8000 0000 0000 8000 0000		
3	0000 0000 0000 c000 0000 0000 9000 0000 0000 0000 0000 4000 0000 0000 8000 0000	2^{-5}
4	0004 0000 0000 0000 0000 0000 0000 0080 0007 0000 0000 0000 0000 0000 0000 0030	$2^{-3.4}$

Round	Differential Mask	Probability
1	0000 0000 0000 0000 0000 0000 0600 0000 0000 0000 0000 0000 0000 0000 0300 0000	$2^{-1.4}$
2	0000 0020 0000 0010 0000 0000 0000 0000 0000 0060 0000 0060 0000 0000 0000 0000	2^{-5}
3	0000 0000 0000 0000 0404 0000 0202 0000 0000 0000 0000 0000 0505 0000 0505 0000	2^{-8}
4	0000 0000 0000 5050 0000 0000 0000 5050 0000 0000 0000 8080 0000 0000 0000 8080	2^{-12}
Δ tweak: 0000 0000 0008 0008 0000 0000 0008 0008		
5	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000	1
6	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000	1
7	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000	1
8	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000	1