Tweaks and Keys for Block Ciphers: the TWEAKEY Framework

Jérémy Jean, Ivica Nikolić, and Thomas Peyrin

Division of Mathematical Sciences, School of Physical and Mathematical Science,
Nanyang Technological University, Singapore

{JJean,INikolic,Thomas.Peyrin}@ntu.edu.sg

Abstract. We propose the TWEAKEY framework with goal to unify the design of tweakable block ciphers and of block ciphers resistant to related-key attacks. Our framework is simple, extends the key-alternating construction, and allows to build a primitive with arbitrary tweak and key sizes, given the public round permutation (for instance, the AES round). Increasing the sizes renders the security analysis very difficult and thus we identify a subclass of TWEAKEY, that we name STK, which solves the size issue by the use of finite field multiplications on low hamming weight constants. We give very efficient instances of STK, in particular, a 128-bit tweak/key/state block cipher Deoxys-BC that is the first AES-based ad-hoc tweakable block cipher. At the same time, Deoxys-BC could be seen as a secure alternative to AES-256, which is known to be insecure in the related-key model. As another member of the TWEAKEY framework, we describe Kiasu-BC, which is a very simple and even more efficient tweakable variation of AES-128 when the tweak size is limited to 64 bits.
In addition to being efficient, our proposals, compared to the previous schemes that use AES as a black box, offer security beyond the birthday bound. Deoxys-BC and Kiasu-BC represent interesting pluggable primitives for authenticated encryption schemes, for instance, ΘCB3 instantiated with Kiasu-BC runs at about 0.75 c/B on Intel Haswell. Our work can also be seen as advances on the topic of secure key schedule design for AES-like ciphers, describing several proposals in this direction.

Key words: tweak, block cipher, key schedule, AES, authenticated encryption.

1 Introduction

Block ciphers are among the most scrutinized cryptographic primitives, used in many constructions as basic secure bricks that ensure data encryption and/or authenticity. In the last few decades, a lot of research has been conducted on this topic, and it is believed that building a secure and efficient block cipher is now a well-understood problem. In particular, designs that allowed to prove their security against classical differential or linear attacks have been a very important step forward, and have been incorporated in the current main worldwide standard AES-128 [47]. This topic is mature and the community has recently been focusing on other directions, such as the possibility to build ciphers dedicated to very constrained environments [11, 14, 30].

The security of the block ciphers, both Feistel and Substitution-Permutation networks, has been well studied when the key is fixed and secret, however, when the attacker is allowed to ask for encryption or decryption with different (and related) keys the situation becomes more complicated. In the past, many published ciphers have been broken in this so-called *related-key model* [5, 6] and it has even been demonstrated that the Advanced Encryption Standard (AES) has flaws in this model [7, 8]. It is known how to design a cryptographically good permutation composed of several iterated rounds, but when it comes to keying this permutation with subkeys generated by the key schedule, it is hard to ensure that the overall construction remains secure. Most key schedule constructions are ad-hoc, in the sense that the designers came up with a key schedule that is quite different from the internal permutation of the cipher, in a hope that no meaningful structure is created by the interaction of the two components. This is the case of PRESENT [11] or AES [47] ciphers, where the key schedule is purposely made different from the round function. Some key schedules can be very weak but fast and lightweight (like in LED [30], where many rounds are required to ensure security against related-key attacks), while some can be very strong but slow (like in the internal cipher of the WHIRLPOOL hash function [3]). In order to partially ease this task of deriving a good schedule, some automatic tools analyzing the resistance of the ciphers against simple related-key differential attacks have been developed [9, 28, 43].

The Hasty Pudding cipher [51], proposed to the AES competition organized by the NIST, permitted the user to insert an additional input to the classical key and plaintext pair, called *spice* by the designers

of this cipher. This extra input T, later renamed as *tweak*, was supposed to be completely public and to randomize the instance of the block cipher: to different values of T correspond different and independent families of permutations E_K. This feature was formalized in 2002 by Liskov et al. [40,41], who showed that tweakable block ciphers are valuable building blocks if retweaking (changing the tweak value) is less costly than changing its secret key. Tweakable block ciphers (see MERCY [16], for example) found many different utilizations in cryptography, such as disk encryption where each block is ciphered with the same key, but the block index is used as tweak value.

Simple constructions of a tweakable block cipher $E_K(T, P)$ based on a block cipher $E_K(P)$, like XORing the tweak into the key input and/or message input, are not satisfactory. For example, only XORing the tweak into the key input would result in an undesirable property that $E_K(T, P) = E_{K \oplus X}(T \oplus X, P)$. Liskov et al. propose instead to use universal hash families for that purpose. The XE and XEX constructions [50] (and the follow-up standard XTS [25]) are based on finite field multiplications in $GF(2^n)$, and present the particularity of being efficient if sequential tweaks are used. Nonetheless, even with such feature, these scheme might not be really efficient as the cipher execution is not negligible compared to a finite field multiplication in $GF(2^n)$ (for example when AES is the internal block cipher and the scheme implementation uses AES-NI instructions). More importantly, *these methods ensure only security up to the birthday-bound (relative to the block cipher size)*. This can be a problem as the main block cipher standards only have 64- or 128-bit block size. Minematsu [45] partially overcomes this limitation by proving beyond birthday-bound security for his design, but at the expense of a very reduced efficiency. The same observation applies to more recent beyond-birthday constructions such as [38,53]. Overall, *none of the state-of-the-art block-cipher-based schemes provide both efficiency and beyond birthday-bound security.*

Ad-hoc constructions would be a solution, with the obvious drawback that security proofs regarding the construction would be very hard to obtain. So far, this direction has seen a surprisingly low number of proposals. The NIST SHA-3 competition for hash functions triggered a few, like SKEIN [27] (with its ad-hoc internal tweakable block cipher Threefish) and BLAKE2 [2]. It is interesting to note that both are Addition-Rotation-XOR (ARX) functions and thus offer less possibility of proofs with regard to classical differential-linear attacks. As of today, it remains an open problem to design an ad-hoc AES-like tweakable block cipher, which in fact would be very valuable for authenticated encryption as AES-NI instruction sets guarantee extremely fast software implementations. Such a primitive would enable very efficient authenticated encryption with beyond birthday-bound security and proof regarding the mode of operation.

Liskov et al. proposed to separate the roles of the secret key (which provides uncertainty to the adversary) from that of the tweak (which provides independent variability) – interestingly, almost all tweakable block cipher proposals (except Threefish) follow this rule. This might be seen as counter intuitive as it is required the tweak input to be somehow more efficient than the key input, but at the same time the security requirement on the tweak seem somehow stronger than on the key, since the attacker can fully control the former (even though tweak-recovery attacks are irrelevant). We argue in this article that, in practice, when one designs a block cipher these two inputs should be considered almost the same, as incorporating a tweak and a secret key shares in fact a lot of common ground, especially for the large family of key-alternating ciphers.

Our contributions. In this article, we bring together key schedule design and tweak input handling for block ciphers in a common framework that we call TWEAKEY (Section 2). The idea is to provide a simple framework to design a tweakable block cipher with any key and any tweak sizes. Our construction is very simple and can be seen as a natural extension of key-alternating ciphers: a subtweakey (i.e. a value obtained from the key and the tweak inputs) is incorporated into the internal state at every round of the iterative cipher. One advantage of such a framework is that one can obtain a tweakable single-key block cipher or a double-key length block cipher with the very same primitive.

Not all instances of TWEAKEY are secure and, in particular, the case where the key and tweak material is treated exactly the same way does not lead to a secure cipher. However, handling the key and the tweak material the same way would be attractive in terms of performance, implementation, but more importantly it would greatly simplify the security analysis, which is currently the main difficulty

designers have to face when constructing an ad-hoc tweakable block cipher. Indeed, the main challenge is to evaluate the appropriate number of rounds required to make the cipher secure – when the tweak size t and key size k are too large this problem becomes infeasible. We propose a solution in Section 3 and we give a subclass of TWEAKEY for AES-like ciphers, named STK (for Superposition TWEAKEY), where the key and the tweak materials are treated almost the same way – the small difference between the linear key and the tweak schedules is sufficient to remove the aforementioned weakness. Due to the structure of STK, the security analysis is rendered much easier, and the number of rounds can be kept small. The STK construction leads to promising performances: we show in Section 4 a complete 128-bit tweak 128-bit key 128-bit block cipher proposal Deoxys-BC based on the AES round function, but faster and more lightweight than other tentatives to build a tweakable block cipher from AES-128. When used in ΘCB3 [37] authenticated encryption, Deoxys-BC runs at about 1.3 c/B on the latest Intel processors. This has to be compared to OCB3, which runs at 0.7-0.88 c/B when instantiated with AES-128, but only ensures birthday-bound security. Alternatively, Deoxys-BC could be a replacement for AES-256, which has related-key issues as shown in [8]. The STK construction offers a very lightweight tweakey schedule (only composed of a substitution of bits), that even allows the key to be hardwired in hardware implementations. For constrained environments, one may use the 64-bit tweak 64/128-bit key 64-bit block cipher Joltik-BC, based on a lightweight AES-like round function, for which we estimate that one would require about 1500 GE for a serial ASIC implementation.

We study as well in Section 5 the problem of tweaking AES without altering the key schedule. Using STK with the AES-128 key schedule does not lead to an efficient and easy-to-analyze cipher, but we show that another very simple construction within the TWEAKEY framework solves the problem for the cases when the tweak is limited to 64 bits. This 64-bit tweak 128-bit key 128-bit block cipher that we name Kiasu-BC is very efficient compared to existing methods that derive a tweakable block cipher from AES-128, making it an attractive choice for authenticated encryption modes built upon such primitives. Used in ΘCB3 [37] authenticated encryption for example, it has about the same speed as OCB3 while ensuring full 128-bit security. Moreover, we estimate that Kiasu-BC can be implemented with as low as 3000 GE for a serial ASIC implementation.

We emphasize that the security of all our proposals has been scrutinized with regard to state-of-the-art cryptanalysis techniques, with a particular focus on key-schedule specific attacks such as meet-in-the-middle and related-key attacks.

2 The TWEAKEY framework

In this section, we introduce the TWEAKEY construction framework that allows to add a tweak of (almost) any length to a key-alternating block cipher and/or to extend the key space of the block cipher to (almost) any size. In some sense, one can view the TWEAKEY framework as a simple generalization of key-alternating ciphers, offering more flexibility with regards to tweak and/or key sizes. Similarly to key-alternating ciphers, we emphasize that not all TWEAKEY instances are secure. We give in later sections natural instances of TWEAKEY that lead to secure ciphers.

2.1 Key-alternating ciphers

A symmetric primitive like a block cipher E is usually built upon a smaller building block f that is iterated a certain number of times – we refer to such a function f as a *round function*. Usually f is cryptographically weak, but its iterations bring security to E. The number r of iterations heavily depends on the targeted security of E, the structure of f, its differential properties, its algebraic degree, etc. In general, the function f takes two inputs: the first is the state, while the second is a round-dependent parameter called *round key* or *subkey*. The round keys are obtained by the expansion of a master secret K with an expansion (key schedule) algorithm: $K \rightarrow (K_0, \ldots, K_r)$. Formally, for a non-negative $i < r$, we write $f(s_i, K_i) = s_{i+1}$ the function that transforms the state s_i in one round into the state s_{i+1}, with the use of the round key K_i. Initially, the state s_0 is set to the plaintext value P, and state s_r at the output of the r-th round is the ciphertext C .

As a subclass of iterated block ciphers, we consider further the particular case of *key-alternating block ciphers*, which specify how the round keys are used (see Figure 1). The concept has been initially

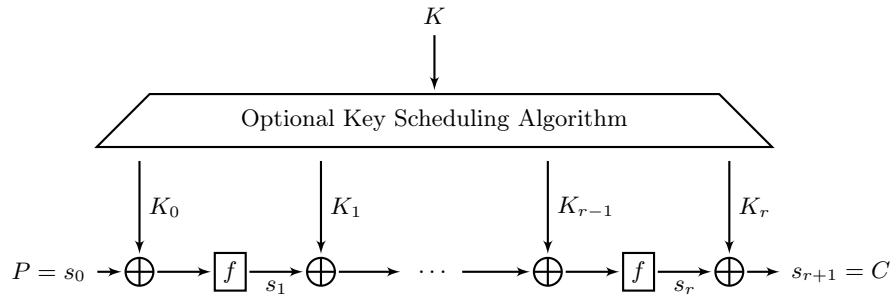
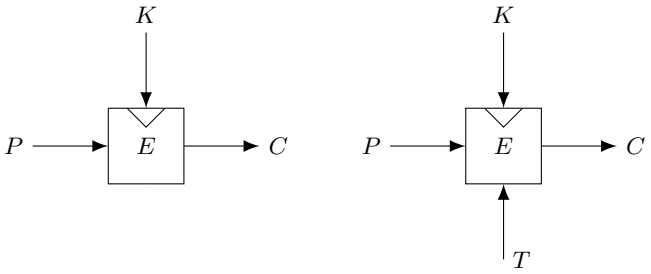


Figure 1: Key-alternating cipher: the function f is applied r times, surrounded by subkey mixing operations.

introduced by Daemen in [17,19] and has later been reused in many block cipher designs, e.g. [11,18,30]. Specifically, we say that E is a key-alternating cipher when the general form $f(s_i, K_i) = s_{i+1}$ for $i < r$ becomes $f(s_i \oplus K_i) = s_{i+1}$, where the current state s_i and the incoming round key K_i are XORed prior to the application of the round function f. Moreover, a final round key K_r is added after the r applications of f to produce the ciphertext. The soundness of such a construction has been theoretically studied recently in [1,12].

2.2 Tweakable block ciphers

The concept of tweakable block ciphers goes back to the Hasty Pudding cipher [51], and has later been formalized by Liskov, Rivest and Wagner in [40,41], where they suggest to move the randomization of symmetric primitives brought by the high-level operations of the modes directly at the block-cipher level. The signature of standard block ciphers can be described as $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ where an n-bit plaintext P is transformed into an n-bit ciphertext $C = E(K, M)$ using a k-bit key K. On top on these inputs, tweakable block ciphers introduce an additional t-bit parameter T called tweak (see Figure 2). The signature for a tweakable block cipher therefore becomes $E : \{0,1\}^k \times \{0,1\}^t \times \{0,1\}^n \to \{0,1\}^n$, the ciphertext $C = E(K, T, P)$ where the tweak T does not need to be secret and thus can be placed in the public domain. Similarly to a regular block cipher where $E(K, \cdot)$ is a permutation for all $K \in \{0,1\}^k$, a tweakable block cipher preserves this behavior as $E(K, T, \cdot)$ is a permutation for all $(K, T) \in \{0,1\}^k \times \{0,1\}^t$.



(a) Regular Block Cipher. **(b)** Tweakable Block Cipher.

Figure 2: Types of ciphers

Usually, the security notion expected from a tweakable block cipher is to be indistinguishable from a tweakable random permutation (a family of independent random permutations parameterized by T). It is important to note that the security model considers that the attacker has full control over both the message and the tweak inputs.

Adversarial model. Besides the classical single-key attack model, a typical model for block ciphers is the related-key model, where the adversary can ask for encryption/decryption of plaintext/ciphertext with a key related to the original one. In this article, we only consider the relation between the keys and tweaks to be the classical XOR difference, and refer to [20] for more details on this so-called *key access scheme*. Similarly to the related-key model, the related-tweak model denotes a situation where

the adversary can ask for encryption/decryption of plaintext/ciphertext with a tweak related to the original one, while the key remains the original one. Continuing further, we can also combine these two models and consider the related-key related-tweak adversarial model. Moreover, instead of related-key or related-tweak model, one can consider open-key and/or open-tweak models, where the adversary has full control over the key/tweak. This model is reasonable to consider as in practice an active adversary might have a full control over the tweak. For the key, this model might be interesting when the block cipher is used in a hash function setting, where message blocks are usually inserted in the key input of the inner block cipher of the compression function. Since in this article we do not always separate key and tweak input, we sometimes denote *related-tweakey* or *open-tweakey* to refer to related-key related-tweak or open-key open-tweak model, respectively.

2.3 The TWEAKEY construction

In theory, for a tweakable block cipher the distinction between the tweak input and the key input is clear: the former is public and can be fully controlled by the attacker, while the later is secret. This might indicate that in practice the tweak input must be handled more carefully then the key input, since the attacker is given more power[1]. However, from the point of view of applications, what is intrinsically required for a tweakable block cipher is that computing consecutive cipher calls with different random tweak values should be very efficient, while not necessarily required for the key input. This tends to indicate that, in the contrary, the tweak input should not use more computations than the key input.

This contradiction regarding the proportion of computations between the tweak and key inputs should make tweakable block cipher designers handle both inputs almost equivalently (we note that this is the case for example in Threefish [27]). Moving in this direction, we introduce the TWEAKEY framework, that tries to bridge the gap between key and tweak inputs by providing a unified vision. This framework can be seen as a direct extension of the key-alternating cipher construction. As of today, building a tweakable block cipher with a key-alternating approach has never been considered, but we note that Goldenberg et al. [29] studied how to insert a tweak input inside a Luby-Rackoff cipher from a theoretical point of view.

The term *tweakey* refers to an input that can be both tweak or key material, *without distinction*. Using our framework, the obvious advantage is that one can leverage the work already done on key schedule design in order to build proper tweak schedule, or tweakey schedule more generally.

The TWEAKEY construction is a framework to build a n-bit tweakable block cipher with t-bit tweak and k-bit key. It consists of two states: the n-bit internal state s and the $(t+k)$-bit tweakey state tk, and we denote respectively as s_i and tk_i their values throughout the rounds. The state s_0 is initialized with the plaintext P (or ciphertext C for decryption), and tk_0 is initialized with the tweak and key material. Then, the cipher is composed of r successive rounds each composed of three steps:

- a subtweakey extraction function g from the tweakey state, and incorporation of this subtweakey to the internal state (for ease of description, we consider that the subtweakey incorporation is done with a simple XOR, but this can be trivially extended to other operations),
- an internal state update permutation f,
- a tweakey state update function h.

This can be summarized as: $s_{i+1} = f(s_i \oplus g(tk_i))$ followed by $tk_{i+1} = h(tk_i)$. At the end, the last subtweakey is incorporated to the last internal state and $s_r \oplus g(tk_r)$ represents the ciphertext C (or plaintext P for decryption). The subtweakeys are usually of size n bits, but they might be smaller. The framework is depicted in Figure 3.

Increasing the amount of tweak or key material obviously renders the task of the designer much more complex in terms of security analysis. To separate these situations, we denote TK-p the class of tweakable block ciphers when one handles $p \times n$ of tweakey material. For example, a simple single-key

[1] One may argue that key recovery attacks are not to be considered for the tweak input, which makes the tweak and the key inputs fundamentally different. However, from a designer perspective, it seems easier to protect against key-recovery attacks, than against a known-key distinguisher. For example, for most ciphers, more rounds can be attacked in the open-key model than in the related-key model.

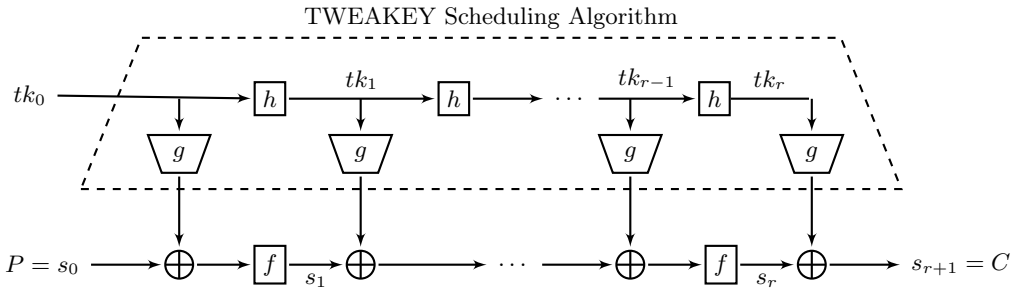


Figure 3: The TWEAKEY framework.

cipher would fit in TK-1, while an n-bit key, n-bit tweak block cipher (or for a double-key cipher with no tweak input) would fit in TK-2. By extension, a public permutation would fit in TK-0. The tweakey material can be any amount of key and/or tweak. A tweak-only cipher can be an interesting primitive as well, for example when building a compression function (the members of the MD-SHA hash function family would actually fit in our framework, the subtweakey having smaller size than n).

We emphasize that TWEAKEY is only a framework and, as such, will not guarantee a secure cipher. It is up to the designer to ensure picking a proper TWEAKEY instance. The functions f, g and h must be chosen along with the number of rounds r such that no known attack can apply on the resulting primitives. More precisely, this must be true for any choice of the tweak/key size tradeoff inside the tweakey input. A natural way to achieve this while keeping the same f, g and h would be to set the number of rounds as the maximal number of required rounds over all the possible tweak/key size tradeoffs. By known attacks, we refer in particular to classical differential/linear attacks, even in related-tweakey or open-tweakey model, or meet-in-the-middle techniques. Moreover, the key schedule is often used to break inherent symmetries from the internal state update function and to break round similarities (for example in the case of AES), hence this has to be taken in account as well.

Cipher instances separation. Since the tweak and key material are not made distinct in our framework, one might argue that since the tweakable block cipher is always the same whatever is the amount of key or tweak inputs, there are some obvious relations between these different versions. If the designer would prefer to avoid these properties, this can be easily and securely done for example by encoding the various cipher versions on a few bits of the tweakey state (with two distinct key/tweak sizes versions, one tweak bit would then have to be booked for that matter). Nevertheless, in the rest of this article, we do not consider related-cipher attacks [54].

3 The STK construction

3.1 Motivation

The TWEAKEY framework unifies the tweak and key input for a tweakable block cipher, but does not provide real instantiation of this construction, i.e. which functions f, g and h (and number of rounds r) one should choose. For instance, a trivial example resulting in a non-secure primitive consists in choosing the identity function for the update function h (i.e. the key schedule of LED [30]), and defining g as the XOR of all n-bit tweakey words. In such a case, regardless of the choice of the function f, the construction would not be secure as cancellations of the tweakey words would lead to outputs of g consisting of zero bits.

One of the main causes for the low number of ad-hoc tweakable block ciphers is the fact that adding a tweak input makes the security analysis much harder. Building a block cipher secure in the related-key model is already not an easy task, and by incorporating an additional tweak or a double key, the task becomes even more difficult. In the case of AES, there exists tools [9,28] to analyze the best differential characteristics in the related-key model, but they mainly work for TK-1. As soon as we switch to bigger keys or add tweak inputs, like TK-2 or TK-3, the searches might become infeasible, unless very good characteristics exist to speed up the search with branching cuts, which would mean that the cipher is insecure.

One research direction that we follow in this article consists in finding a construction within the TWEAKEY framework that simplifies this analysis. A potential and natural solution would be that all $p = (t + k)/n$ n-bit words of tweakey are handled the same way (i.e. the function h is symmetric with regards to the p n-bit words of the tweakey state of TK-p), and that g simply XORs all these n-bit tweakey state words to the internal state. The security analysis is simplified as any analysis independently performed on one of the n-bit words of tweakey will hold for the other words as well (and thus the tools working for TK-1 could now do the analysis even for TK-p with $p > 1$). The problem is to understand what happens when all words are considered together as their interaction might cause potential weaknesses (e.g. if we insert differences in all the tweakey words). For example, assume we would like to build an AES-like cipher with double key: this would fit in TK-2 as $k = 2n$ and $t = 0$. If the two n-bit tweakey words were treated equivalently, we could use the differential characteristic search tools to assess the security of the primitive with regards to classical differential attacks, and then use this information to pick an appropriate number of rounds. However, there is an obvious weakness if we strictly follow this strategy: starting with the two tweakey words equal would lead to zero being XORed to the internal state every round, since their value would always cancel each other in the XOR. Using constants to separate the two words would work, but only if the h function is strongly non-linear, which is something we would like to avoid for efficiency reasons. In fact, we would like to push even further the efficiency incentive and only consider nibble-wise substitutions for the h function.

In the remaining of the section, we propose a simple solution to overcome this issue for AES-like ciphers. The basic idea is to minimize possible differences cancellations between tweakey words by using small field multiplications. Following this mechanism still allows to apply the existing differential characteristic search tools, while avoiding the trivial characteristic in the tweakey scheduling algorithm.

3.2 The STK (Superposition TWEAKEY) construction

The STK construction is a subclass of the TWEAKEY framework for AES-like ciphers defined over a finite field $GF(2^c)$. Recall that $p = (t + k)/n$ denotes the number of n-bit words in the tweakey state composed of t-bit tweak and k-bit key. Assuming that the AES-like S-Box operates on c bits (thus we have n/c nibbles in a n-bit word), the STK construction further specifies the f, g and h functions as follows (also see Figure 4):

- the function g simply XORs all the p n-bit words of the tweakey state to the internal state (AddRoundTweakey, denoted ART), and then XORs a round-dependent constant C_i,
- the function h first applies the same nibble position substitution function h' to each of the p n-bit words of the tweakey state, and then multiply each c-bit cell of the j-th n-bit word by a nonzero coefficient α_j in the finite field $GF(2^c)$ (with $\alpha_i \neq \alpha_j$ for all $1 \leq i \neq j \leq p$)
- the function f is an AES-like round.

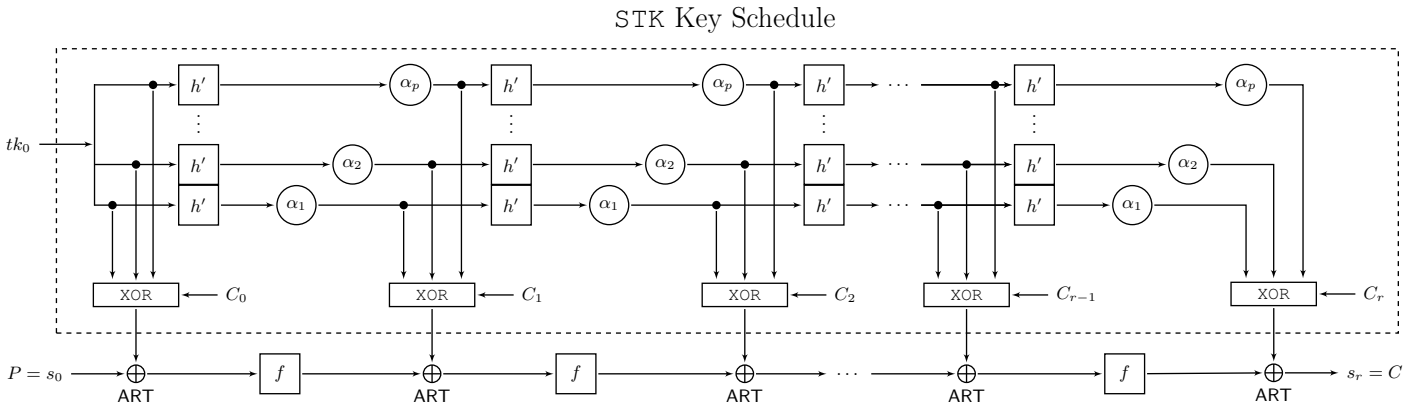


Figure 4: The STK construction: example with TK-p.

3.3 Rationale behind the STK construction

Most automated differential analysis tools for AES-like ciphers (e.g., [9,23,28]) use truncated differential representation to make feasible the search for differential characteristics. In the truncated difference representation [36], the exact value of a difference in a nibble is not specified. Rather, only the presence (active nibble) or absence (inactive nibble) of a difference is kept track of. In the STK construction, the different subtweakey words will have precisely the same truncated representation of the difference if the input tweakey words have the same difference. The reason behind this is that they all apply the same functions g and h, which are completely independent of the tweakey word considered. This feature already significantly simplifies the analysis for the designer, since a simple TK-1 differential analysis (already known to be possible with the current tools) will ensure the security for all situations in which only a single tweakey word contains a difference. Having all the tweakey words treated almost equivalently is therefore very helpful for the designer.

The issue, however, is to understand what happens when differences are placed in several tweakey words at the same place (in the same nibbles). In particular, the difficulty lies in the cancellations that might happen in the nibbles at the output of g (recall that g will XOR all the subtweakey word to the state). These cancellations are the reason why having exactly the same update function for all tweakey words leads to a design that is not secure. The trick we use is to apply a nibble-wise multiplication with a distinct coefficient α_j for all tweakey words. This prevents the large number of cancellation of differences in a particular nibble position at the output of g. To explain this, first observe that as we apply the very same nibble position substitution function h' to each of the p tweakey words, the relative position of the nibble between the tweakey words is always the same (i.e. two nibbles at the same position inside their tweakey word will always keep that property). Thus, we can divide the tweakey nibbles into n/c fully independent subgroups (according to the nibble position in the n-bit tweakey words), and to each of these subgroups will correspond one and only one nibble at the output of g at every round. More precisely, in each subgroup, we have p input nibbles $\mathbf{x} = [x_1, \ldots, x_p]$ (one in each tweakey word) and $r+1$ output nibbles $\mathbf{y} = [y_0, \ldots, y_r]$ (since we have to generate $r+1$ sub-tweakeys). Our STK construction ensures that whenever a non-null difference is inserted in the input nibbles of the subgroup, there will always be at least $r+1-p$ active output nibbles. These output nibbles $\mathbf{y}$ can be expressed in terms of $\mathbf{x}$ by using a right-matrix multiplication $\mathbf{y} = \mathbf{x} \times \mathbf{V}$ with the following $p \times (r+1)$ Vandermonde matrix:

$$\mathbf{V} = \left(\alpha_i^j\right)_{i,j} = \begin{pmatrix} \alpha_1^0 & \alpha_1^1 & \ldots & \alpha_1^r \\ \alpha_2^0 & \alpha_2^1 & \ldots & \alpha_2^r \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_p^0 & \alpha_p^1 & \ldots & \alpha_p^r \end{pmatrix},$$

In order to minimize the number of nonzero elements in $\mathbf{y}$ for $\mathbf{x} \neq 0$, we need to ensure that all the columns in $\mathbf{V}$ are linearly independent. This is true as long as the α_i coefficients, $1 \leq i \leq p$ are pairwise distinct. Using for example the specific distinct coefficients $\alpha_j = j \in GF(2^c)$, $1 \leq j < 2^c$, in TK-p, $1 \leq p \leq c-1$, then at most p elements of $\mathbf{y}$ can be zero for $\mathbf{x} \neq 0$, which is the property that we targeted.

To summarize, when we deal with differences in several tweakey words (which is supposedly very hard to analyze due to the important number of nibbles), the study of the STK construction is again the same as for a classical TK-1 analysis, except that at most p active output nibbles can be erased in each subgroup. This extra constraint in the search is rather easy to include in the existing analysis tools [9,28] and this is precisely why we believe the STK construction to be interesting. It has been created with this criteria in mind, so as to ease a systematic cryptographic analysis by existing tools, rather than only relying on ad-hoc constructions, which are *de facto* more difficult to evaluate.

As a side note, the constants C_i in the STK construction prevent obvious issues regarding symmetries in the internal state for an AES-like cipher, as the RCON constants do for the original AES key scheduling algorithm. The choice of these constants are left at the discretion of the designers, but one could recommend for instance to use the AES RCON constants, based of the exponentiation of 2 in $GF(2^c)$, or the exponentiation of any other primitive element in that field.

The nibble positions permutation h' is also left at the discretion of the designers, but it must be carefully chosen so as to provide the best resistance against classical differential/linear attacks. This will permit the designers to safely choose an appropriate number of rounds r. This number will of course strongly depend on the amount of tweakey material, since more tweakey material makes it harder for the designer to create a secure tweakable block cipher. Our analysis tools indicate that using identity function instead of h' would lead to designs that require a great number of rounds. Therefore, we recommend h' to be a nibble positions permutation so as to prevent the existence of very good differential characteristics, but yet remaining a very efficient function to compute.

As a proof of concept, we describe in the next section real instances of the STK construction.

3.4 Performances

The performance of the STK construction is very high due to the simple transformations used in the schedules – all of them are linear and lightweight. The cost of the nibble position permutation h' is very low, however, the choice of the coefficients α_j might have a significant impact on the performances. For optimal efficiency, one should typically use $\alpha_1 = 1$ and $\alpha_2 = 2$ in the case of TK-2. For larger instances, TK-p with $p > 2$, one could use powers of 2 as coefficients α_j in order to maintain high efficiency in the computations of the coefficients multiplications. In most of the applications, the tweak is changed more frequently than the key. For instance, in a number of authenticated encryption schemes, the key is the same across different calls to the tweakable cipher, while the tweak is different in each call. Thus, it is reasonable to make the tweak schedule more efficient than the key schedule. Therefore, the tweak schedule should use the most efficiently implementable coefficients α_j ($\alpha_1 = 1$ would be the first choice). However, for some particular use-cases, it can be better to assign coefficient $\alpha_1 = 1$ to a key input. Indeed, for hardware implementations, it might be very valuable in certain scenarios to hard-wire the key in order to greatly reduce the area required (this is a feature of several lightweight ciphers). Yet, this would be possible only if the key input is not modified during the execution of the entire cipher and this is ensured only if $\alpha_1 = 1$ is assigned to this key input. The efficiency of the STK constructions can best be measured in term of key/tweak agility, i.e. how well the construction behaves when the key and/or the tweak are frequently changed. Due to the very low number of transformations, and all being completely linear, this construction has obviously one of the simplest possible schedules.

4 STK proposals

The STK construction can easily be instantiated by defining the required functions, and can result in tweakable block ciphers of any size. In the sequel, we give examples of such instantiations.

4.1 Deoxys-BC and Joltik-BC

We present two concrete tweakable block ciphers based on the STK construction: Deoxys-BC and Joltik-BC. The former processes plaintext blocks of $n = 128$ bits and uses the unkeyed AES round function as the function f. The later processes plaintext blocks of $n = 64$ bits and uses a lightweight AES-like function based on 4-bit nibbles as the function f. Both ciphers come with two versions: the first belongs to TK-2 (two n-bit tweakey words, thus Deoxys-BC-256 and Joltik-BC-128), while the second to TK-3 (three n-bit tweakey words, thus Deoxys-BC-384 and Joltik-BC-192). Although we give only two instances per cipher, due to the TWEAKEY framework, the amount of key/tweak for these ciphers can be even larger as long as it fits in the total tweakey state.

To describe fully the two ciphers, we only need to define the key/tweak schedule transformations, i.e. the nibble permutation h' and the the coefficients α_i used in h, and the constants C_i used in the g function[2], as well as the round function f and the number of rounds r. The nibble permutation h' is the same for Deoxys-BC and Joltik-BC, and also for their TK-2 and TK-3 versions. Namely, if we

[2] The function g for all instances of the STK construction is always defined as an XOR all the n-bit tweakey words and the constants C_i to the internal state, thus requires no additional description.

regard the internal state as a 4×4 matrix of nibbles, the permutation h' is defined as:

$$\begin{pmatrix} 0 & 4 & 8 & 12 \\ 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \end{pmatrix} \overset{h'}{\longmapsto} \begin{pmatrix} 1 & 5 & 9 & 13 \\ 6 & 10 & 14 & 2 \\ 11 & 15 & 3 & 7 \\ 12 & 0 & 4 & 8 \end{pmatrix}.$$

Concerning the coefficients α_i, for efficiency reasons we choose $(1, 2)$ for TK-2, and $(1, 2, 4)$ for TK-3 (see Figure 5). The multiplication for Deoxys-BC is done in $GF(2^8)$ defined by the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$, while for Joltik-BC in $GF(2^4)$ defined by the irreducible polynomial $x^4 + x + 1$. The constants C_i are given in Appendix A and Appendix B.

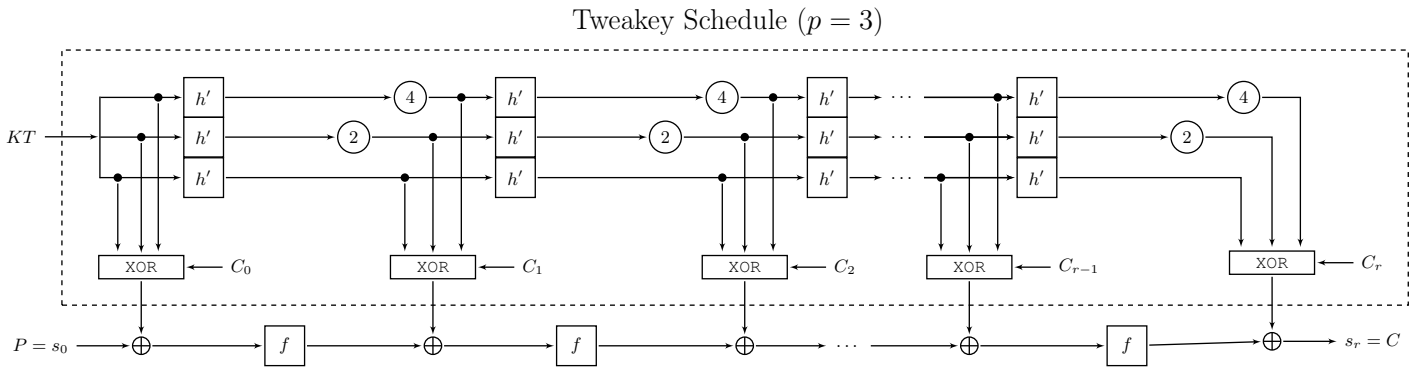


Figure 5: Instantiation of the TWEAKEY framework for Deoxys-BC and Joltik-BC.

The f function of Deoxys-BC is exactly the unkeyed AES round while the number of rounds r for Deoxys-BC-256 is 14 and for Deoxys-BC-384 is 16. The f function of Joltik-BC is an unkeyed lightweight AES round like transformation (acts on 4-bit nibbles rather than 8-bit bytes) that has the standard three operations: SubNibbles, ShiftRows, and MixNibbles. A full description of the operations is given in the Appendix B. The number r of rounds for Joltik-BC-128 is 24 while for Joltik-BC-192 is 32 rounds.

4.2 Security analysis

The security of our STK proposals in the single-key model reduces to the analysis of the underlying round function f. Since the round functions follow the wide-trail strategy, it is easy to see that our ciphers have high resistance against classical linear and differential attacks. The resistance against attacks that exploit symmetries (rotational, internal differential, slide attacks) is achieved with the use of round constants C_i. The real threat to the STK proposals comes from attacks that exploit the tweakey schedule, i.e. related-key/tweak (related tweakey) differential attacks and meet-in-the-middle attacks. Let us take a closer look to these attacks and show how we achieve the resistance.

Related-tweakey differential characteristics. One of the major design goals of the STK construction is to ease the task of finding the best related-tweakey differential characteristics. Note that finding such characteristics is not trivial, however, it can be done with the known tools [9, 10, 26]. Our choice of the nibble permutation h' in Deoxys-BC and Joltik-BC has been made after examining several options for h', testing the resistance of the ciphers against related-tweakey attacks (with the tools) when instantiated with the chosen h', and selecting the one that provides the highest resistance. The number of rounds has been chosen to provide comfortable security margin. The current h' guarantees that in Deoxys-BC-256 and Deoxys-BC-384, there are no related-tweakey characteristics on more than 10 and 12 rounds, respectively, that have a probability higher than 2^{-128}. Similarly, for Joltik-BC-128 and Joltik-BC-192, it guarantees that there are no such characteristics on more than 16 and 24 rounds respectively, with a probability higher than 2^{-64}. We stress that these are only upper bounds

on the number of rounds, and the actual bounds might be lower. Consequently, the security margin against related-tweakey differential attacks is at least 4 rounds in Deoxys-BC, and at least 8 rounds in Joltik-BC.

Meet-in-the-middle attacks. To assess the resistance of the proposed schemes against meet-in-the-middle attacks, we focus on the more recent advances done in this context. In particular, we target the new meet-in-the-middle strategy on AES devised by Dunkelman, Keller and Shamir in [24], which has later been improved by Derbez, Fouque and Jean in [22, 23], Derbez and Fouque in [21] and Li, Jia and Wang in [39]. This technique uses the key schedule equations to perform an advanced differential meet-in-the-middle cryptanalysis of reduced-round variants of AES-like ciphers. In the original articles targeting AES, the best attacks can reach 7 rounds for AES-128, and 9 rounds for both AES-192 and AES-256. The basic idea of this class of attacks starts in an offline phase by constructing a large table for the middle rounds tabulating all possible values for a particular function. Then, it continues in the online phase by performing structured encryptions queries to the oracle and tries to obtained a message that lies in the precomputed table. This table inherently depends on the key schedule as it links the values of the inner cipher state to key-byte values. The more independent bytes in all the subkeys, the less the attack would work as the table would require a larger amount of memory to be stored.

In regards to our proposals, this cryptanalysis technique which focuses on a particular class of meet-in-the-middle attacks makes sense to investigate as we change the key schedule of the original AES, and notably by replacing it with a fully linear byte-wise permutation. Indeed, simplifying the schedule that way would intuitively make the attack more powerful and reach more rounds.

We study here the particular cases of TK-1, TK-2 and TK-3, without distinction on the particular proposal, as they share the exact same design and the same attacks would apply to both of them. Our preliminary results show that the maximal number of rounds an advanced meet-in-the-middle attack can reach for TK-1, TK-2 and TK-3 are 7, 10 and 11 rounds, respectively. We have evaluated this results using the generalized technique presented in [21], which uses more linear relations than [23] to perform the meet-in-the-middle. This allows more tradeoffs and therefore introduces slightly better complexities. In terms of security, our proposals Deoxys-BC and Joltik-BC in regard to advanced meet-in-the-middle attack therefore offer 4 rounds of security margin for Deoxys-BC-256, 5 for Deoxys-BC-384, 14 for Joltik-BC-128 and 21 for Joltik-BC-192. We note that the AES offers only 3 rounds of security margin as 7 rounds of AES-128 (out of 10) can be attacked by this technique, as well as 9 rounds of AES-192 (out of 12).

4.3 Software and hardware performances

The chosen round functions (and the nibble sizes), suggest that Deoxys-BC is software oriented, while Joltik-BC is hardware (and lightweight) oriented design. We now have a closer look at how well these designs perform on the corresponding platforms.

As Deoxys-BC is based on the AES, it allows very efficient software implementation on the processors that support AES-NI, especially when used in a mode of operation that offers parallelization. For constructions that use tweakable ciphers, the actual overhead of Deoxys-BC compared to AES-128 comes from the increased number of rounds and from the tweak schedule. However, for constructions that use the tweak as extended key, Deoxys-BC-256 outperforms AES-256 as both have the same number of rounds, but Deoxys-BC-256 has far lighter key schedule. The permutation h' can be implemented as single processor instruction pshufb. Furthermore, the coefficients α_i have a hamming weight of 1, thus the field multiplication can be implemented with a few SIMD instructions as well. Note, for Deoxys-BC-256, $\alpha_2 = 2$, thus the multiplication is in the key schedule only and therefore the tweak schedule is composed only of byte permutations, and the overhead per round is only one pshufb and one XOR – arguably it is one of the most efficient tweak schedules in our framework.

The precise benchmarks we have obtained after implementing Deoxys-BC on the latest Intel Haswell processor are as follows. On longer inputs and modes based on parallelizable block cipher calls (such as ΘCB3), Deoxys-BC-256 runs at around 1.3 cycles per byte, while Deoxys-BC-384 at around 1.55 cycles per byte. This is to be compared to AES in OCB3 mode, which runs at around 0.70 - 0.88 cycles per byte (but has only birthday bound security). We would like to point out that

Deoxys-BC-256, used as a block cipher (with 256-bit key and no tweak), has much higher key agility than AES-256 due to the simpler key schedule. Furthermore, other key schedules variants proposed for AES (see [15, 44, 48]), have even lower efficiency than the original AES key schedule, thus our variant is the fastest known when testing for key agility.

Despite the fact that Joltik-BC is hardware-oriented, it will perform rather well on software platforms, using recent bitsliced implementations of AES-like lightweight ciphers [4, 42]. Concerning hardware implementations, we briefly explain why we believe Deoxys-BC and especially Joltik-BC would be potentially lightweight primitives and the logic behind our ASIC implementation estimations. The starting point for Deoxys-BC is the best ASIC lightweight implementation [46] of AES, that requires only 2400GE (out of which 70% comes from the memory to store the key and the internal state). Since Deoxys-BC-256 is exactly the AES except the key and tweak layers, it is safe to estimate that the overhead will be due to storing the 128-bit tweak value and XORing it to the internal state (and in addition an extra 128-bit key for Deoxys-BC-384). In particular, the key schedule in Deoxys-BC is more lightweight than the AES one, since it requires only wiring and a few simple multiplications in $GF(2^8)$. Therefore, we expect an overhead of about $128 \times (4.67 + 2.67) = 940$ GE for Deoxys-BC-256, by counting 2.67 GE per XOR (which might sometimes be optimized to 2.33 GE) and 4.67 GE for single-bit input flip-flop to store the tweak. We would need approximately twice this amount in the case of Deoxys-BC-384. Therefore, we estimate that the entire Deoxys-BC-256 can be implemented with around 3400 GE, and Deoxys-BC-384 with around 4400 GE.

Since Joltik-BC is also a 64-bit lightweight AES-like cipher, our starting point is the best ASIC lightweight implementation [30] of LED-128, that only requires 1265 GE (from which 77% comes from the memory to store the key and the internal state). The main differences of Joltik-BC compared to LED-128 is that the S-Box in Joltik-BC is a bit more compact than in LED, but on the other hand the diffusion matrix can not be computed serially as it is the case for LED. This later point should imply 3 nibbles (12 bits) of extra memory for the computation, which we estimate to $12 \times 6 = 72$ GE (counting 6 GE for a two-bit input flip-flop). The coefficients used in the Joltik-BC matrix are very lightweight (all coefficients in a row can be implemented with only 6 XORs [35] thanks to the careful choice of the coefficients and the finite field), so this should not have much impact compared to LED. Both LED-128 and Joltik-BC-128 have the same tweakey size, and both use a very light tweakey schedule. The permutation for the tweakey in Joltik-BC-128 can be implemented with bit wiring so this should have no impact on the area figures. The main difference would be that 128 bits are XORed from the tweakey state to the internal state each round, while in the case of LED it is only 64 bits. Therefore, we have to add an extra $64 \times (2.67) = 171$ GE by counting 2.67 GE per XOR. We omit the multiplication by 2 in $GF(2^4)/0x13$ since it can be implemented with only shifts and a single XOR. Overall, we expect an overhead of 243 GE for Joltik-BC-128 compared to LED-128. This would lead us to about 1500 GE to implement Joltik-BC-128. The reasoning for Joltik-BC-192 is exactly the same, except that we have an extra $64 \times (2.67) = 171$ GE to compute the extra tweakey XOR in the internal state. Moreover, an additional memory of 64 bits of tweakey material is required, which adds another $64 \times (4.67) = 298$ (counting 4.67 GE for a single-bit input flip-flop). Finally, we omit the multiplication by 4 in $GF(2^4)/0x13$ since it can be implemented with only shifts and two XORs. Overall, we expect Joltik-BC-192 to fit in about 2000 GE.

5 Kiasu-BC: tweaking AES-128 for 64-bit tweak

In this section, we focus on the particular case of tweaking the AES-128 block cipher, and we provide Kiasu-BC, a very simple yet efficient solution when the tweak size equals 64 bits.

Since AES is already a well-established standard, we would like to be able to tweak AES-128 directly, that is without redefining another cipher like we did for the key schedule in Deoxys-BC (even though it is very close to AES). It is tempting to use directly the STK-like construction with AES-128, by applying the AES-128 key schedule to both the tweak and the key, and avoiding too many difference cancellations with the α_i multiplication trick. However, the AES-128 key schedule is a bad choice for two reasons: efficiency and security analysis. The former comes from the nonlinearity of the key schedule. It is quite slow even on the latest Intel processor, thus using it as a tweak schedule, which will be executed in each call to the cipher, would make the whole construction slow. The later

is due to the complexity of the schedule – it makes the analysis very hard to perform even with the available tools. Moreover, for legacy implementations it would be valuable to get exactly the AES-128 cipher when the tweak input is null. We emphasize that this is not the case for Deoxys-BC since the permutation-based key schedule is different from the one in AES-128. Also, since the tweak value is supposed to change more often than the key value, we would like to avoid a strong and slow function to handle the tweak input.

Therefore, we go back to the TWEAKEY framework and propose a very simple solution when the tweak size is limited to 64 bits: in every round, the tweak value T is XORed to the top two rows of the internal state (see Figure 6). More precisely, Kiasu-BC is an ad-hoc family of tweakable block ciphers, that has a 128-bit internal state and 64-bit tweakey state (i.e. in TK-1.5). It is exactly the AES cipher, except that the tweak value is XORed to the two top rows of the internal state at every round after the addition of the subkeys (after the AddRoundKey operation). On Figure 7, we represent the ith round of Kiasu-BC, where we first XOR a subkey k_i (AK), we then XOR the 64-bit tweak T (AT), and then continue with the three AES usual transformations: SubBytes (SB), ShiftRows (SR) and MixColumns (MC).

$$T = \begin{array}{|c|c|c|c|}\hline T_0 & T_2 & T_4 & T_6 \\ \hline T_1 & T_3 & T_5 & T_7 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array}$$

Figure 6: Tweak in Kiasu-BC: the 64-bit values of the tweak $T = T_0||T_1 \ldots||T_7$ are placed on the top two rows of the AES internal state.

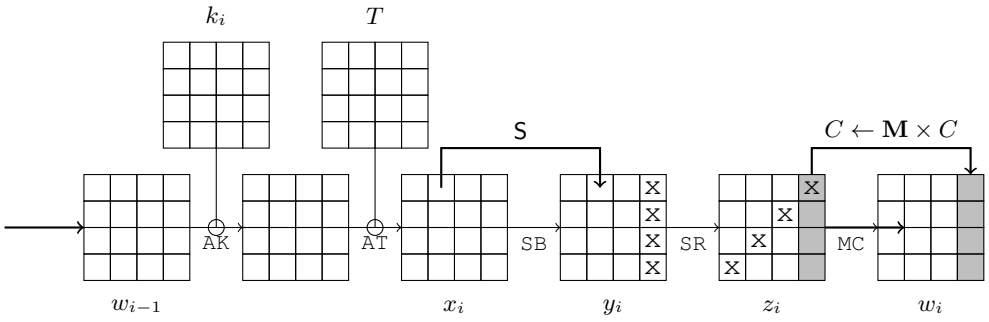


Figure 7: One round of Kiasu-BC: the round-dependent subkey k_i is first XORed to the internal state (AK), then the constant tweak T is (AT), and the three AES operations are applied (SB, SR, MC).

5.1 Security analysis

We have conducted to Kiasu-BC the same analysis as for Deoxys-BC and Joltik-BC. Note that the position of the state bytes where the tweak is XORed and the tweak size (64 bits) is not chosen randomly, but rather obtained after a careful security analysis, in particular after running the tools for search of related-tweakey differential characteristics and analyzing their outputs. For instance, XORing two columns (instead of rows) would immediately lead to an insecure variant. Furthermore, increasing the tweak size to 96 bits (by XORing it to three rows) also leads to a variant that has related-tweakey characteristics with probability higher than 2^{-128}. The current choice in Kiasu-BC assures that no such high probability characteristics exist on more than 6 rounds and no boomerang characteristics on more than 7 rounds.

As for meet-in-the-middle attacks, the exact same strategy applies to Kiasu-BC as the one on AES-128 done in [22, 23]. Indeed, the tweak value being public and known to an adversary, he can mount the same strategy to get a key-recovery attack on at most 7 rounds of the cipher, with the same complexities as for AES-128.

We stress that the above security analysis applies only to the case when the key is 128 bits and the tweak is 64 bits. However, Kiasu-BC is part of the TWEAKEY framework and as such it should allow variable size of key and tweak (as long as their accumulative size is 192 bits). However, we do not recommend to use Kiasu-BC with a key size larger than 128 bits. In the event that such a primitive is required, i.e. to make Kiasu-BC secure regardless of the choice of the tweak/key size tradeoff, we note that for larger keys the design should have 12 rounds. Indeed, when Kiasu-BC is used as 192-bit key cipher with no tweak, the meet-in-the-middle attacks can penetrate up to 9 rounds as now the attacker can afford 2^{192} complexity (same as AES-192). In such a case, the cipher will still have comfortable security margin of 3 rounds. We have decided to keep the cipher with 10 rounds for obvious efficiency reasons, AES backward compatibility and because 10 rounds are sufficient when the key space is limited to 128 bits.

5.2 Performances

Similarly to Deoxys-BC, as Kiasu-BC is based on the AES, it allows very efficient software implementation on the processors that support AES-NI. Moreover, it is trivial to modify an arbitrary AES implementation to produce Kiasu-BC. The actual overhead compared to AES only comes from the tweak schedule. However, this translates only into a single additional XOR of the tweak per round (in terms of AES-NI, if one AES round is implemented as `state=aesenc(state,k[i])`, then in Kiasu-BC one round can be implemented as `state=aesenc(state,xor(k[i],tweak))`). Thus, the proposed tweak schedule in Kiasu-BC is the simplest and most efficient possible candidate in the TWEAKEY framework.

The actual implementation results vary depending on the type of the AES implementation (table-based, bitsliced, or AES-NI) and on the type of the mode (serial or parallel). In serial modes, Kiasu-BC runs at virtually the same speed as AES, while in parallel only 5%-15% slower – for instance, OCB on Intel Haswell runs at 0.70 c/B, while Kiasu-BC at 0.75 c/B (but again, it provides full 128-bit security, unlike the birthday security of OCB).

Concerning hardware implementations, we briefly explain why we believe Kiasu-BC would be a potentially lightweight primitive and the logic behind our ASIC implementation estimation. As for Deoxys-BC, our starting point is the best ASIC lightweight implementation [46] of AES, that only requires 2400GE. Since Kiasu-BC is exactly the AES, plus only the tweak layer, it is pretty straightforward to estimate that the overhead will be due to storing the 64-bit tweak value and XORing it to the internal state. This should require around $64 \times (4.67 + 2.67) = 470$ GE, by counting 2.67 GE per XOR (which might sometimes be optimized to 2.33 GE) and 4.67 GE for single-bit input flip-flop to store the tweak. Therefore, we estimate that the entire Kiasu-BC can be implemented with around 2900 GE.

6 Conclusion

We have introduced the TWEAKEY framework, which helps designers to build a secure tweakable block cipher by bringing together key schedule design and tweak input. Inside this framework, we have identified a new type of construction, named STK, that is simple and generic and which provides efficient schemes, as shown by the two STK instances Deoxys-BC and Joltik-BC. We have also shown how to directly tweak the AES-128 block cipher, with the very simple and extremely efficient Kiasu-BC tweakable block cipher. Our three proposals Deoxys-BC, Joltik-BC and Kiasu-BC are the base of the three CAESAR authenticated encryption competition candidates Deoxys [31], Joltik [32] and Kiasu [33], respectively.

We believe this work opens many questions and future works. First, it would be interesting to prove the soundness of our framework and the STK construction. Namely, can we generalize the recent proofs done on key-alternating ciphers? Secondly, we believe that better nibble positions permutation h' might exist for the STK construction, compared the one we used in Deoxys-BC and Joltik-BC. The search space is quite large, thus a smart method in order to prune bad candidates is necessary, as well as very optimized search tools. This problem is actually even more complex, since a best permutation for TK-i might not necessarily be the best for TK-j with $i \neq j$. Then, a very valuable advance would be to find

a way to tweak directly the AES-128 (keeping the original key schedule) with a 128-bit tweak, since Kiasu-BC only handles 64-bit tweak. Our searches led us to the conclusion that this seems quite hard to achieve. Finally, the problem of designing a simple, secure and efficient key schedule for AES-like ciphers remains an open problem. Is it possible to find an efficient key schedule that could lead to simple human-readable proofs on the minimal number of active S-Boxes in a differential characteristic in the related-tweakey model?

Acknowledgements. The authors would like to thank Tetsu Iwata and Gaëtan Leurent for very fruitful discussions on tweakable block ciphers. This work is supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06).

References

1. Andreeva, E., Bogdanov, A., Dodis, Y., Mennink, B., Steinberger, J.P.: On the Indifferentiability of Key-Alternating Ciphers. [13] 531–550
2. Aumasson, J.P., Neves, S., Wilcox-OÕHearn, Z., Winnerlein, C.: BLAKE2: simpler, smaller, fast as MD5. In: Applied Cryptography and Network Security, Springer (2013) 119–135
3. Barreto, P.S.L.M., Rijmen, V.: The WHIRLPOOL Hashing Function. Submitted to NESSIE, September 2000 (2000)
4. Benadjila, R., Guo, J., Lomné, V., Peyrin, T.: Implementing Lightweight Block Ciphers on x86 Architectures. appeared in SAC 2013.
5. Biham, E.: New Types of Cryptoanalytic Attacks Using related Keys (Extended Abstract). In Helleseth, T., ed.: EUROCRYPT'93. Volume 765 of LNCS., Springer (May 1993) 398–409
6. Biham, E., Dunkelman, O., Keller, N.: A Unified Approach to Related-Key Attacks. In Nyberg, K., ed.: FSE 2008. Volume 5086 of LNCS., Springer (February 2008) 73–96
7. Biryukov, A., Khovratovich, D.: Related-Key Cryptanalysis of the Full AES-192 and AES-256. In Matsui, M., ed.: ASIACRYPT. Volume 5912 of Lecture Notes in Computer Science., Springer (2009) 1–18
8. Biryukov, A., Khovratovich, D., Nikolic, I.: Distinguisher and Related-Key Attack on the Full AES-256. In Halevi, S., ed.: CRYPTO 2009. Volume 5677 of LNCS., Springer (August 2009) 231–249
9. Biryukov, A., Nikolic, I.: Automatic Search for Related-Key Differential Characteristics in Byte-Oriented Block Ciphers: Application to AES, Camellia, Khazad and Others. In Gilbert, H., ed.: EUROCRYPT 2010. Volume 6110 of LNCS., Springer (May 2010) 322–344
10. Biryukov, A., Nikolic, I.: Search for Related-Key Differential Characteristics in DES-Like Ciphers. [34] 18–34
11. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In Paillier, P., Verbauwhede, I., eds.: CHES 2007. Volume 4727 of LNCS., Springer (September 2007) 450–466
12. Bogdanov, A., Knudsen, L.R., Leander, G., Standaert, F.X., Steinberger, J.P., Tischhauser, E.: Key-Alternating Ciphers in a Provable Setting: Encryption Using a Small Number of Public Permutations - (Extended Abstract). In Pointcheval, D., Johansson, T., eds.: EUROCRYPT 2012. Volume 7237 of LNCS., Springer (April 2012) 45–62
13. Canetti, R., Garay, J.A., eds.: CRYPTO 2013, Part I. In Canetti, R., Garay, J.A., eds.: CRYPTO 2013, Part I. Volume 8042 of LNCS., Springer (August 2013)
14. Cannière, C.D., Dunkelman, O., Knezevic, M.: KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In Clavier, C., Gaj, K., eds.: CHES 2009. Volume 5747 of LNCS., Springer (September 2009) 272–288
15. Choy, J., Zhang, A., Khoo, K., Henricksen, M., Poschmann, A.: AES Variants Secure against Related-Key Differential and Boomerang Attacks. In Ardagna, C.A., Zhou, J., eds.: WISTP. Volume 6633 of Lecture Notes in Computer Science., Springer (2011) 191–207
16. Crowley, P.: Mercy: A Fast Large Block Cipher for Disk Sector Encryption. In Schneier, B., ed.: FSE 2000. Volume 1978 of LNCS., Springer (April 2000) 49–63
17. Daemen, J., Govaerts, R., Vandewalle, J.: Correlation Matrices. [49] 275–285
18. Daemen, J., Knudsen, L.R., Rijmen, V.: The Block Cipher Square. In Biham, E., ed.: FSE'97. Volume 1267 of LNCS., Springer (January 1997) 149–165
19. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer (2002)
20. Daemen, J., Rijmen, V.: On the related-key attacks against AES. Proceedings of the Romanian Academy, Series A **13**(4) (2012) 395–400
21. Derbez, P., Fouque, P.A.: Exhausting Demirci-Selcuk Meet-in-the-Middle Attacks against Reduced-Round AES. In: FSE. Lecture Notes in Computer Science (2013) *to appear.*
22. Derbez, P., Fouque, P.A., Jean, J.: Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting (extended version). Cryptology ePrint Archive, Report 2012/477 (2012)
23. Derbez, P., Fouque, P.A., Jean, J.: Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting. In Johansson, T., Nguyen, P.Q., eds.: EUROCRYPT 2013. Volume 7881 of LNCS., Springer (May 2013) 371–387
24. Dunkelman, O., Keller, N., Shamir, A.: Improved Single-Key Attacks on 8-Round AES-192 and AES-256. In Abe, M., ed.: ASIACRYPT 2010. Volume 6477 of LNCS., Springer (December 2010) 158–176

25. Dworkin, Morris J.: SP 800-38E. Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices (2010)
26. Emami, S., Ling, S., Nikolić, I., Pieprzyk, J., Wang, H.: The resistance of PRESENT-80 against related-key differential attacks. Cryptography and Communications (2013) 1–17
27. Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The SKEIN Hask Function Family. (2009)
28. Fouque, P.A., Jean, J., Peyrin, T.: Structural Evaluation of AES and Chosen-Key Distinguisher of 9-Round AES-128. [13] 183–203
29. Goldenberg, D., Hohenberger, S., Liskov, M., Schwartz, E.C., Seyalioglu, H.: On Tweaking Luby-Rackoff Blockciphers. In Kurosawa, K., ed.: ASIACRYPT 2007. Volume 4833 of LNCS., Springer (December 2007) 342–356
30. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.J.B.: The LED Block Cipher. In Preneel, B., Takagi, T., eds.: CHES 2011. Volume 6917 of LNCS., Springer (September / October 2011) 326–341
31. Jean, J., Nikolić, I., Peyrin, T.: Deoxys v1.1 (2014) Submission to the CAESAR competition, http://www1.spms.ntu.edu.sg/~syllab/Deoxys.
32. Jean, J., Nikolić, I., Peyrin, T.: Joltik v1.1 (2014) Submission to the CAESAR competition, http://www1.spms.ntu.edu.sg/~syllab/Joltik.
33. Jean, J., Nikolić, I., Peyrin, T.: Kiasu v1.1 (2014) Submission to the CAESAR competition, http://www1.spms.ntu.edu.sg/~syllab/Kiasu.
34. Joux, A., ed.: FSE 2011. In Joux, A., ed.: FSE 2011. Volume 6733 of LNCS., Springer (February 2011)
35. Khoongming Khoo and Frédérique Oggier and Thomas Peyrin and Siang Meng Sim: Lightweight MDS Involution Matrices (2014) *Article in preparation.*
36. Knudsen, L.R.: Truncated and Higher Order Differentials. [49] 196–211
37. Krovetz, T., Rogaway, P.: The Software Performance of Authenticated-Encryption Modes. [34] 306–327
38. Landecker, W., Shrimpton, T., Terashima, R.S.: Tweakable Blockciphers with Beyond Birthday-Bound Security. In Safavi-Naini, R., Canetti, R., eds.: CRYPTO 2012. Volume 7417 of LNCS., Springer (August 2012) 14–30
39. Li, L., Jia, K., Wang, X.: Improved Single-Key Attacks on 9-Round AES-192/256. In: FSE 2014, Springer (2014)
40. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable Block Ciphers. In Yung, M., ed.: CRYPTO 2002. Volume 2442 of LNCS., Springer (August 2002) 31–46
41. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable Block Ciphers. Journal of Cryptology **24**(3) (July 2011) 588–613
42. Matsuda, S., Moriai, S.: Lightweight Cryptography for the Cloud: Exploit the Power of Bitslice Implementation. In Prouff, E., Schaumont, P., eds.: CHES 2012. Volume 7428 of LNCS., Springer (September 2012) 408–425
43. Matsui, M.: On Correlation Between the Order of S-boxes and the Strength of DES. In Santis, A.D., ed.: EUROCRYPT'94. Volume 950 of LNCS., Springer (May 1994) 366–375
44. May, L., Henricksen, M., Millan, W., Carter, G., Dawson, E.: Strengthening the Key Schedule of the AES. In Batten, L.M., Seberry, J., eds.: ACISP. Volume 2384 of Lecture Notes in Computer Science., Springer (2002) 226–240
45. Minematsu, K.: Beyond-Birthday-Bound Security Based on Tweakable Block Cipher. In Dunkelman, O., ed.: FSE 2009. Volume 5665 of LNCS., Springer (February 2009) 308–326
46. Moradi, A., Poschmann, A., Ling, S., Paar, C., Wang, H.: Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In Paterson, K.G., ed.: EUROCRYPT. Volume 6632 of Lecture Notes in Computer Science., Springer (2011) 69–88
47. National Institute of Standards and Technology (NIST): Advanced Encryption Standard (AES). FIPS PUB 197, U.S. Department of Commerce (November 2001)
48. Nikolic, I.: Tweaking AES. In Biryukov, A., Gong, G., Stinson, D.R., eds.: Selected Areas in Cryptography. Volume 6544 of Lecture Notes in Computer Science., Springer (2010) 198–210
49. Preneel, B., ed.: FSE'94. In Preneel, B., ed.: FSE'94. Volume 1008 of LNCS., Springer (December 1994)
50. Rogaway, P.: Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In Lee, P.J., ed.: ASIACRYPT 2004. Volume 3329 of LNCS., Springer (December 2004) 16–31
51. Schroeppel, R.: The Hasty Pudding Cipher (1998)
52. Shibutani, K., Isobe, T., Hiwatari, H., Mitsuda, A., Akishita, T., Shirai, T.: Piccolo: An Ultra-Lightweight Blockcipher. In Preneel, B., Takagi, T., eds.: CHES. Volume 6917 of Lecture Notes in Computer Science., Springer (2011) 342–357
53. Shrimpton, T., Terashima, R.S.: A Modular Framework for Building Variable-Input-Length Tweakable Ciphers. In Sako, K., Sarkar, P., eds.: ASIACRYPT 2013, Part I. Volume 8269 of LNCS., Springer (December 2013) 405–423
54. Wu, H.: Related-Cipher Attacks. In Deng, R.H., Qing, S., Bao, F., Zhou, J., eds.: ICICS 02. Volume 2513 of LNCS., Springer (December 2002) 447–455

A The constants for Deoxys-BC

The constants C_i used in Deoxys-BC are defined as:

$$C_i = \begin{pmatrix} 1 & \texttt{RCON}[i] & 0 & 0 \\ 2 & \texttt{RCON}[i] & 0 & 0 \\ 4 & \texttt{RCON}[i] & 0 & 0 \\ 8 & \texttt{RCON}[i] & 0 & 0 \end{pmatrix}$$

where $\texttt{RCON}[i], i = 0, \ldots, 16$ denotes the i-th key schedule constants of the AES and are defined as RCON = {2f, 5e, bc, 63, c6, 97, 35, 6a, d4, b3, 7d, fa, ef, c5, 91, 39, 72}.

B The round transformations and the constans for Joltik-BC

The f function of Joltik-BC is an unkeyed AES-like round and as such it has the following three transformations applied to the internal state in the order specified below:

- SubNibbles – Apply the 4-bit S-Box $\mathcal{S}$ defined below to the 16 nibbles of the internal state,
- ShiftRows – Rotate the 4-nibble i-th row left by $\rho[i]$ positions, where $\rho = (0, 1, 2, 3)$.
- MixNibbles – Multiply the internal state by the 4×4 constant MDS matrix $\mathbf{M}$ defined below.

The 4-bit S-Box $\mathcal{S}$ used in Joltik-BC is the one selected for the Piccolo block cipher [52], and is exhaustively defined by the following table:

$$[14\ 4\ 11\ 2\ 3\ 8\ 0\ 9\ 1\ 10\ 7\ 15\ 6\ 12\ 5\ 13]$$

The MDS matrix $\mathbf{M}$ we use in Joltik-BC is non-circulant, MDS and involutory (computations are done in $GF(16)$ defined by the irreducible polynomial $x^4 + x + 1$):

$$\mathbf{M} = \begin{pmatrix} 1 & 4 & 9 & 13 \\ 4 & 1 & 13 & 9 \\ 9 & 13 & 1 & 4 \\ 13 & 9 & 4 & 1 \end{pmatrix} = \mathbf{M}^{-1}.$$

We refer to [35] for more details on this class of matrices.

Finally, the constants C_i are similar to the constants used in LED cipher [30] and are defined as:

$$C_i = \begin{pmatrix} 0 & (rc_5||rc_4||rc_3) & 0 & 0 \\ 1 & (rc_2||rc_1||rc_0) & 0 & 0 \\ 2 & (rc_5||rc_4||rc_3) & 0 & 0 \\ 3 & (rc_2||rc_1||rc_0) & 0 & 0 \end{pmatrix}$$

where the values of $(rc_5, rc_4, rc_3, rc_2, rc_1, rc_0)$ are given in Tbl. 1.

Rounds	**Constants**
1-12	01, 03, 07, 0F, 1F, 3E, 3D, 3B, 37, 2F, 1E, 3C
13-24	39, 33, 27, 0E, 1D, 3A, 35, 2B, 16, 2C, 18, 30
25-32	21, 02, 05, 0B, 17, 2E, 1C, 38, 31

Table 1: Constants (given in hexadecimal) used in the internal block cipher of Joltik-BC. The constants are given in terms of bytes, and rc_0 represents the less significant bit.