Forkcipher: a New Primitive for Authenticated Encryption of Very Short Messages

Elena Andreeva[1], Virginie Lallemand[2], Antoon Purnal[1], Reza Reyhanitabar[3], Arnab Roy[4], and Damian Vizár[5]

[1] imec-COSIC, KU Leuven, Belgium
[2] Université de Lorraine, CNRS, Inria, LORIA, France
[3] (now with) TE Connectivity, Germany
[4] University of Bristol, UK
[5] CSEM, Switzerland.
`elena.andreeva@esat.kuleuven.be`, `antoon.purnal@esat.kuleuven.be`, `virginie.lallemand@loria.fr`, `reza.reyhanitabar@te.com`, `arnab.roy@bristol.ac.uk`, `damian.vizar@csem.ch`

Abstract. Highly efficient encryption and authentication of *short* messages is an essential requirement for enabling security in constrained scenarios such as the CAN FD in automotive systems (max. message size 64 bytes), massive IoT, critical communication domains of 5G, and Narrowband IoT, to mention a few. In addition, one of the NIST lightweight cryptography project requirements is that AEAD schemes shall be "optimized to be efficient for short messages (e.g., as short as 8 bytes)".

In this work we introduce and formalize a novel primitive in symmetric cryptography called *forkcipher*. A forkcipher is a keyed primitive expanding a fixed-lenght input to a fixed-length output. We define its security as indistinguishability under a chosen ciphertext attack (for n-bit inputs to $2n$-bit outputs). We give a generic construction validation via the new *iterate-fork-iterate* design paradigm.

We then propose ForkSkinny as a concrete forkcipher instance with a public tweak and based on `SKINNY`: a tweakable lightweight cipher following the TWEAKEY framework. We conduct extensive cryptanalysis of ForkSkinny against classical and structure-specific attacks.

We demonstrate the applicability of forkciphers by designing three new provably-secure nonce-based AEAD modes which offer performance and security tradeoffs and are optimized for efficiency of very short messages. Considering a reference block size of 16 bytes, and ignoring possible hardware optimizations, our new AEAD schemes beat the best `SKINNY`-based AEAD modes. More generally, we show forkciphers are suited for lightweight applications dealing with predominantly short messages, while at the same time allowing handling arbitrary messages sizes.

Furthermore, our hardware implementation results show that when we exploit the inherent parallelism of ForkSkinny we achieve the best performance when directly compared with the most efficient mode instantiated with `SKINNY`.

Keywords: Authenticated encryption, new primitive, forkcipher, ForkSkinny, lightweight cryptography, short messages.

1 Introduction

Authenticated encryption (AE) aims at achieving the two fundamental security goals of symmetric-key cryptography: confidentiality (privacy) and integrity (together with authentication). Historically, these two goals were achieved by the generic composition of an encryption scheme (for confidentiality) and a message authentication code (MAC) [23]. For instance, *old* versions of major security protocols such as TLS, SSH and IPsec included variants of generic composition, namely MAC-then-Encrypt, Encrypt-and-MAC and Encrypt-then-MAC schemes, respectively. But it turned out that this approach is neither the most efficient (as it needs processing the whole message twice) nor the most robust to security and implementation issues [22, 43, 44]; rather it is easy for practitioners to get it wrong even when using the best known method among the three, i.e. Encrypt-then-MAC, following standards [41].

The notion of AE as a primitive in its own right—integrating encryption and authentication by exposing a single abstract interface— was put forth by Bellare and Rogaway [25] and independently by Katz and Yung [34] in 2000. It was further enhanced by Rogaway [46] to authenticated encryption with associated data (AEAD). Being able to process associated data (AD) is now a default requirement for any authenticated encryption scheme; hence we use AE and AEAD interchangeably. After nearly two decades of research and standardization activities, recently fostered by the CAESAR competition (2014–2018) [26], we now have a rich set of *general-purpose* AEAD schemes, some already standardized (e.g. GCM and CCM) and some expected to be adopted by new applications and standards (e.g. the CAESAR finalists Ascon [30], ACORN [53], AEGIS-128 [55], OCB [36], COLM [9], Deoxys II [32], and MORUS [54]).

This progress may lead to the belief that the AEAD problem is "solved". However, as evidenced by the ECRYPT-CSA report in 2017 [14], several critical ongoing "Challenges in Authenticated Encryption" still need research efforts stretching years into the future. Thus, it is interesting to investigate to what extent CAESAR has resulted in solutions to these problems.

Our Target Challenge. Among the four categories of challenges—security, interface, performance, mistakes and malice—reported by the ECRYPT-CSA [14], we aim at delving into the performance regarding authenticated encryption of *very short messages*. General-purpose AEAD schemes are usually optimized for handling (moderately) long messages, and often incur some initialization and/or finalization cost that is amortized when the message is long. To quote the ECRYPT-CSA report: "The performance target is wrong $\cdots$ Another increasingly common scenario is that an authenticated cipher is applied to many small messages $\cdots$ The challenge here is to minimize overhead."

Therefore, designing efficient AEAD for short messages is an important objective as also evidenced by NIST's first call for submissions (May 14, 2018) for lightweight cryptography [42], where it is stressed as a *design requirement* that lightweight AEAD submissions shall be "optimized to be efficient for short messages (e.g., as short as 8 bytes)".

Plenty of Use Cases. The need for high-performance and low-latency processing of short messages is identified as an essential requirement in a multitude of security and safety critical use cases in various domains. Examples are Secure On board Communication (SecOC) in automotive systems [6], handling of short data bursts in critical communication and massive IoT domains of 5G [1], and Narrowband IoT (NB-IoT) [2, 5] systems. For example, the new CAN FD standard (ISO 11898-1) for vehicle bus technology [3,4], which is expected to be implemented in most cars by 2020, allows for a payload up to 64 bytes. In NB-IoT standards [2, 5] the maximum transport block size (TBS) is 680 bits in downlink and 1000 bits in uplink (the minimum TBS size is 16 bits in both cases). Low energy protocols also come with stringent requirements on the maximum packet size: the Bluetooth, SigFox, LoraWan and ZigBee protocols allow for maximum sizes of 47, 12, 51-255 (51 bytes for slowest data rate, 255 for the fastest), and 84 bytes packet sizes, respectively. In use cases with tight requirements on delay and latency, the typical packet sizes should be small as large packets occupy a link for more time, causing more delays to subsequent packets and increasing latency. Furthermore, in applications such as smart parking lots the data to be sent is just one bit ("free" or "occupied"), so a minimum allowed TBS size of 2 bytes (16 bits) would suit the application. Even more, most medical implant devices, such as pacemakers, permit the exchange of messages of length at most 16 bytes between the device programmer and the device.

Our Goal. Our main objective is to construct secure, modular (provably secure) AEAD schemes that excel in efficiency over previous modular AEAD constructions at processing very short inputs, while also being able to process longer inputs, albeit somewhat less efficiently. We insist that our AEAD schemes ought to be able to securely process inputs of arbitrary lengths to be fairly comparable to other general-purpose (long message centric) schemes, and to be qualified as a full-fledged variable-input-length AEAD scheme according to the requirements in NIST's call for lightweight cryptography primitives.
Towards this goal, we take an approach that can be seen as a parallel to the shift from generic composition to dedicated AEAD designs, but on the level of the primitive. We rethink the way a low level fixed-input-length (FIL) primitive is designed, and how variable-input-length (VIL) AEAD schemes are constructed from such a new primitive.

The Gap between the Primitives and AEAD. Our first observation is that there is a large gap between the high level security goal to be achieved by the VIL AEAD schemes and the security properties that the underlying FIL primitives can provide. Modular AEAD designs typically confine the AE security to the mode of operation only; the lower-level primitives, such as (tweakable) block ciphers, cryptographic permutations and compression functions, are never meant to possess any AE-like features, and in particular they are never expanding as needed to ensure ciphertext integrity in AEAD. Hence, a VIL AEAD scheme Π designed as a mode of operation for an FIL primitive F plays two roles: not only does it extend the domain of the FIL primitive but it also transforms and boosts the security property of the primitive to match the AEAD security notion. A natural question then arises, whether by explicitly decoupling these two AEAD roles we can have more efficient designs and more transparent security proofs.

The first, most obvious approach to resolving the latter question is to remove the security gap between the mode and its primitive altogether, i.e., to start from a FIL primitive F which itself is a secure FIL AEAD. This way a VIL AEAD mode will only have one role: a property-preserving domain extender for the primitive F. Property-preserving domain extension is a well-studied and popular design paradigm for other primitives such as hash functions [11, 24, 45].

Informally speaking, the best possible security that a FIL AEAD scheme with a *fixed* ciphertext expansion (stretch) can achieve is to be indistinguishable from a *tweakable random injective* function, i.e., to be a tweakable pseudorandom injection (PRI) [31, 48]. But starting directly with a FIL tweakable PRI, we did not achieve a desirable solution in our quest for the most *efficient AEAD design for short messages.*[6] It seems that, interestingly, narrowing the security gap between the mode and its primitive, but not removing the gap entirely, is what helps us achieve our ultimate goal.

Contribution 1: Forkcipher – a New Symmetric Primitive. We introduce a novel primitive—a tweakable **forkcipher**—that yields efficient AEAD designs for short messages. A tweakable forkcipher is *nearly*, but not exactly, a FIL AE primitive; "nearly" because it produces *expanded* ciphertexts with a non-trivial redundancy, and not exactly because it has no integrity-checking mechanisms.[7] When keyed and tweaked, we show how a forkcipher maps an n-bit input block M to an output C of $2n$ bits. Intuitively, this is equivalent to evaluating *two independent* tweakable permutations on M but with an *amortized computational cost* (see Figure 1 for an illustration of the forkcipher's high-level structure). We give a strict formalization of the security of such a forkcipher. Our new notion of *pseudorandom tweakable forked permutation* captures the game of indistinguishability of a n-bit to $2n$-bits forkcipher from a pair of random permutations in the context of chosen ciphertext attacks.

Contribution 2: Instantiating a Forkcipher. We give an efficient instance of the tweakable forkcipher and name it ForkSkinny. It is based on the lightweight tweakable block cipher `SKINNY` [18]. Building ForkSkinny on an existing block cipher enables us to rely on the cryptanalysis results behind `SKINNY` [12, 13, 49, 51, 56, 57], and in addition, helps us provide systematic analysis for the necessary forkcipher alterations. We also inherit the cipher's efficiency features and obtain a natural and consistent metric for comparison of the forkcipher performance with that of its underlying block cipher.

`SKINNY` comes with multiple optimization tradeoffs in area, throughput, power, efficiency and software performance in lightweight applications. Additionally, `SKINNY` also provides a number of choices for its block size and tweak size which

[6] See the discussion section in full version [10].

[7] We demonstrate that when used in a minimalistic mode of operation, a secure tweakable forkcipher yields a miniature FIL AEAD scheme which achieves tweakable PRI security.

we incorporate naturally into ForkSkinny. We have performed cryptanalyses of ForkSkinny against differential, linear, algebraic, impossible differential, MITM, integral attacks and boomerang attacks. We have taken the security analysis of ForkAES [17] into account to ensure that the same type of attacks is not possible against ForkSkinny.

To obtain ForkSkinny, we apply our newly proposed *iterate-fork-iterate*(IFI) paradigm: when encrypting a block M of n bits with a secret key and a tweak (public), we first transform M into M' using r_{init} SKINNY rounds together with the tweakey schedule. Then, we fork the encryption process by applying two parallel paths (left and right) each comprising r SKINNY rounds. Along left path the state of the cipher is processed using tweakey schedule of SKINNY, thus producing the same ciphertext as SKINNY. Along the right path the state is processed with a tweakey schedule which differs from that of the left path at each round. The IFI design strategy also provides a scope of parallelizing the implementation of the design. The IFI paradigm is conceptually easy, and supports the transference of security and performance results based on the underlying tweakable cipher. We also provide arguments for the generic security of the IFI construction paradigm assuming that the building blocks are behaving as secure pseudorandom permutations. Our generic result is indicative of the forkcipher structural soundness (but does not directly imply security, because a real forkcipher is never built from a secure pseudorandom permutation). While a forkcipher inherits some of the side-channel security features of its underlying structure, the fully-fledged side-channel security of forkciphers is out of the scope of this paper.

Contribution 3: New AEAD Modes. In our work we follow the well-established modular AE design approach for arbitrary long data in the provable security framework. There is no general consensus in the cryptographic community if AEAD schemes can claim higher merits for being modular and provably secure or not. For instance, 3 out of 7 CAESAR [26] finalists, namely ACORN, AEGIS and MORUS are monolithic designs and do not follow the provable security paradigms. Nonetheless, we trust and follow in the modular and provable security methodology for its well-known security benefits [20, 47]. Moreover, the class of provably secure AEAD designs includes all currently standardized AEAD schemes, as well as the majority of CAESAR finalists. We also emphasize that, by defining the forkcipher as a new fully-fledged primitive and building modes on top in a provable way, we clearly differentiate ourselves from the "prove-then-prune" design approaches.

Regarding the state of the art in AE designs, it appears that aiming for a *provably secure* AEAD mode that achieves the best performance for *both* long and short message scenarios is an ambitious goal. Instead, we design high-performance AEAD modes for very short inputs *whilst* maintaining the functionality and security for long ones. All our three modes, PAEF, SAEF and RPAEF can be further implemented very efficiently when instantiated with ForkSkinny.

Our first scheme **PAEF** (Parallel AEAD from a forkcipher) makes ℓ calls to a forkcipher to process a message of ℓ blocks. PAEF is fully parallelizable and thus can leverage parallel computation. We prove its *optimal* security: n bit confidentiality and n-bit authenticity (for an n-bit block input).

Our second scheme **RPAEF** (Reduced Parallel AEAD from a forkcipher) is also fully parallelizable, but in contrast to PAEF only uses the left forkcipher path for the first $(\ell - 1)$ blocks, and the full (left and right) forkcipher evaluation for the final block (first block for the single block-message). When instantiated with ForkSkinny, RPAEF computes the equivalent of $(\ell - 1)$ calls to SKINNY and 1 call to ForkSkinny. This general mode optimization, as compared to PAEF, comes at the cost of restrictive use of large tweaks (as large as 256 bits) and increased HW area footprint. Similarly to PAEF, we prove that RPAEF achieves optimal quantitative security.

Our third scheme **SAEF** (Sequential AEAD from a forkcipher) encrypts each block "on-the-fly" in a sequential manner (and hence is not parallelizable). SAEF lends itself well to low-overhead implementations (as it does not store the nonce and the block counter) but its security is birthday-bounded in the block size ($n/2$-bit confidentiality and authenticity for n-bit block).

Contribution 4: Hardware Performance and Comparisons. PAEF and SAEF need an equivalent of about 1 and 1.6 SKINNY evaluation per block of AD and message, respectively (both encryption and decryption). RPAEF reduces further the computational cost for all but the last message blocks to an equivalent of 1 SKINNY evaluation. When compared directly with block cipher modes instantiated with SKINNY with a fixed tweak (to facilitate the comparison), such as the standardized GCM [40], CCM [52], and OCB [37], we outperform those significantly for predominantly short data sizes of up to four blocks. We achieve a performance gain in the range of $(10 - 50)\%$ for data ranging from 4 blocks down to 1 block, respectively. The additional overhead for all block-cipher-based modes is incurred by at least two additional cipher calls: one for subkey/mask generation and one for tag computation.

We provide a hardware comparison (in Section 7, Table 10) of our three modes (with different ForkSkinny variants) with Sk-AEAD. The Sk-AEAD is the tweakable cipher mode TAE [38], which is same as ΘCB [37], instantiated with Skinny-AEAD M1/M2, M5/M6 [19]. We compare on the bases of block size, nonce, and tag sizes variants. Based on the round-based implementations all of our three modes perform faster (in terms of cycles) for short data (up to 3 blocks) with about the same area. RPAEF beats its competitor for *all* message sizes at the cost of a area increase of about 20% (for only one of its variants). We further *optimize* the performances by exploiting the in-built parallelism (//) in the ForkSkinny primitive and obtain superior performance results. Namely, for messages up to three 128-bit blocks, the speed-up of PAEF and SAEF (both parallel (//)) ranges from 25% to 50%, where the advantage is largest for the single-block messages. Most importantly, the RPEAF, PAEF, and SAEF (//) instances result in fewer cycles than the ΘCB variants *for all* message sizes at a small cost in area increase. However, the relative advantage of the latter instances is more explicit for short messages; as it diminishes asymptotically with the message blocks. For message sizes up to 8 bytes, which is emphasized by

NIST [42], the Paef-ForkSkinny-64-192 instances are more than 58% faster with also a considerably smaller implementation size.

Related work. An AE design which bears similarities with our forkcipher idea is Manticore [8] (the CS scheme). They use the middle state of a block cipher to evaluate a polynomial hash function for authentication purposes. Yet, for a single block, Manticore needs 2 calls to the block cipher (compared to ≈1.6 SKINNY calls in ForkSkinny), thus failing to realize optimal efficiency for very short messages. The CS design, which has been shown insecure [50] (and fixed with an extra block cipher call), necessitates a direct cryptanalysis on the level of an AE scheme, which is a much more daunting task than dedicated cryptanalysis of a compact primitive. In [15], Avanzi proposes a somewhat similar design approach which splits an intermediate state to process them seperately. More concretely, it uses a nonce addition either prior to the encryption or in the middle of the encryption rounds, specifically at the splitting phase. Yet, the fundamental difference with our design is that we use a different framework (TWEAKEY [33]) which considers the nonce and key together and injects a transformation of those *throughout* the forkcipher rounds. Moreover, it seems impossible to describe the latter designs ([8], [15]) as neither primitives nor modes with clearly defined security goals, whereas our approach aims the opposite.

It is worth mentioning that the recent permutation based construction Farfalle [27] also has superficially similar design structure. For example, in Farfalle with a fixed input length message it is possible to produce two or more fixed length outputs. However, the design strategy of ForkSkinny and Farfalle are different in two aspects: 1. ForkSkinny follows an iterative design strategy (with round keys, round constants etc.), while Farfalle is a permutation based design, and 2. ForkSkinny has an explicit tweak input which is processed using the tweakey framework.

2 Preliminaries

All strings are binary strings. The set of all strings of length n (for a positive integer n) is denoted $\{0,1\}^n$. We let $\{0,1\}^{\leq n} = \bigcup_{i=0}^{n}\{0,1\}^n$. We denote by $\mathrm{Perm}(n)$ the set of all permutations of $\{0,1\}^n$. We denote by $\mathrm{Func}(m,n)$ the set of all functions with domain $\{0,1\}^m$ and range $\{0,1\}^n$, and we let $\mathrm{Inj}(m)n \subset \mathrm{Func}(m)n$ denote the set of all injective functions with the same signature.

For a string X of ℓ bits, we let $X[i]$ denote the i^{th} bit of X for $i = 0,\ldots,\ell-1$ (starting from the left) and $X[i\ldots j] = X[i]\|X[i+1]\|\ldots\|X[j]$ for $0 \leq i < j < \ell$. We let $\mathsf{left}_\ell(X) = X[0\ldots(\ell-1)]$ denote the ℓ leftmost bits of X and $\mathsf{right}_r(X) = X[(|X|-r)\ldots(|X|-1)]$ the r rightmost bits of X, such that $X = \mathsf{left}_\chi(X)\|\mathsf{right}_{|X|-\chi}(X)$ for any $0 \leq \chi \leq |X|$. Given a (possibly implicit) positive integer n and an $X \in \{0,1\}^*$, we let denote $X\|10^{n-(|X| \bmod n)-1}$ for simplicity. Given an implicit block length n, we let $\mathsf{pad10}(X) = X\|10^*$ return X if $|X| \equiv 0 \pmod n$ and $X\|10^*$ otherwise.

Given a string X and an integer n, we let $X_1,\ldots,X_x,X_* \xleftarrow{n} X$ denote partitioning X into n-bit blocks, such that $|X_i| = n$ for $i = 1,\ldots,x$, $0 \leq |X_*| \leq n$

and $X = X_1\|\dots\|X_x\|X_*$, so $x = \max(0, \lfloor X/n \rfloor - 1)$. We let $|X|_n = \lceil X/n \rceil$. We let $(M', M_*) = \mathsf{msplit}_n(M)$ (as in message split) denote a splitting of a string $M \in \{0,1\}^*$ into two parts $M'\|M_* = M$, such that $|M_*| \equiv |M| \pmod n$ and $0 \leq |M_*| \leq n$, where $|M_*| = 0$ if and only if $|M| = 0$. We let $(C', C_*, T) = \mathsf{csplit}_n(C)$ (as in ciphertext split) denote splitting a string C of at least n bits into three parts $C'\|C_*\|T = C$, such that $|C_*| = n$, $|T| \equiv |C| \pmod n$, and $0 \leq |T| \leq n$, where $|T| = 0$ if and only if $|C| = n$. Finally, we let $C'_1, \dots, C'_m, C_*, T \leftarrow \mathsf{csplit\text{-}b}_n(C)$ (as in csplit to blocks) denote the result of $\mathsf{csplit}_n(C)$ followed by partitioning of C' into $|C'|_n$ blocks of n bits, such that $C' = C'_1\|\dots\|C'_m$.

The symbol $\perp$ denotes an error signal, or an undefined value. We denote by $X \leftarrow\!\!\$\ \mathcal{X}$ sampling an element X from a finite set $\mathcal{X}$ following the uniform distribution.

3 Forkcipher

We formalize the syntax and security goals of a *forkcipher*. Informally, a forkcipher is a symmetric primitive that takes as input a fixed-length block M of n bits with a secret key K and possibly a public tweak T, and expands it to an output block of fixed length greater than n bits.

In this article we formalize and instantiate the forkcipher as a tweakable keyed function which maps an n-bit input M to a $2n$-bit output block $C_0\|C_1$. We additionally require that the input M is computable from either of the two output blocks C_0 or C_1. Also, given one half of the output C_0, the other half C_1 should be *reconstructible* from it, and vice versa. These are the basic properties imposed in the syntax of our n-bit to $2n$-bit forkcipher.

When used with a random key, the *ideal* forkcipher implements a *pair* of independent random permutations π_0 and π_1 for every tweak T, namely $C_0 = \pi_0(M)$ and $C_1 = \pi_1(M)$. We define a secure forkcipher to be computationally indistiguishable from such an idealized object - a tweak-indexed collection of *pairs* of random permutations.

A trivial forkcipher. It may be clear at this point that the security notion towards which we are headed can be achieved with two instances of a secure tweakable block cipher that are used in parallel. One could thus instantiate a forkcipher by a secure tweakable block cipher used with two independent keys (or a tweak-space separation mechanism).

The main novelty in a forkcipher is that it provides the same security as a pair of tweakable block ciphers at a reduced cost. Yet this reduction of complexity has nothing to do with the security goals and syntax; these only model the kind of object a forkcipher inevitably is, and which security properties it aspires to achieve.

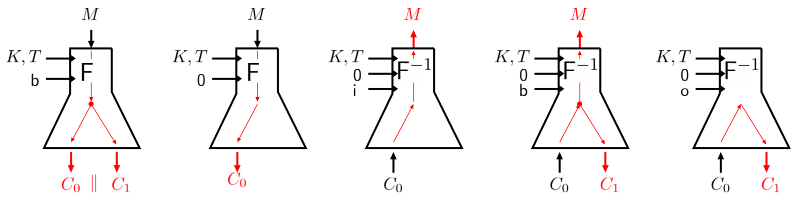

Fig. 1: Forkcipher encryption (two leftmost): the output selector s outputs **b**oth output blocks C_0, C_1 if $s = \mathsf{b}$, the "left" ciphertext block C_0 if $s = 0$ (if $s = \mathsf{b}$ then C_1). Forkcipher decryption (three rightmost): the first indicator $b = 0$ denotes the left ciphertext block is input ($b = 1$ when right). The second output selector $s = \mathsf{i}$ when the ciphertext is **i**nverted to block M (middle); $s = \mathsf{b}$ when **b**oth blocks M, C' are output; and $s = \mathsf{o}$ when the **o**ther ciphertext block C' is output.

3.1 Syntax

A forkcipher is a pair of deterministic algorithms, the encryption[8] algorithm:

$$\mathsf{F} : \{0,1\}^k \times \mathcal{T} \times \{0,1\}^n \times \{0,1,\mathsf{b}\} \to \{0,1\}^n \cup \{0,1\}^n \times \{0,1\}^n$$

and the inversion algorithm:

$$\mathsf{F}^{-1}\{0,1\}^k \times \mathcal{T} \times \{0,1\}^n \times \{0,1\} \times \{\mathsf{i},\mathsf{o},\mathsf{b}\} \to \{0,1\}^n \cup \{0,1\}^n \times \{0,1\}^n.$$

The encryption algorithm takes a key K, a tweak $\mathsf{T} \in \mathcal{T}$, a plaintext block M and an output selector s, and outputs the "left" n-bit ciphertext block C_0 if $s = 0$, the "right" n-bit ciphertext block C_1 if $s = 1$, and a **b**oth blocks C_0, C_1 if $s = \mathsf{b}$. We write $\mathsf{F}(K, \mathsf{T}, M, s) = \mathsf{F}_K(\mathsf{T}, M, s) = \mathsf{F}_K^{\mathsf{T}}(M, s) = \mathsf{F}_K^{\mathsf{T},s}(M)$ interchangeably. The decryption algorithm takes a key K, a tweak T, a ciphertext block C (left/right half of output block), an indicator b of whether this is the left or the right ciphertext block and an output selector s, and outputs the plaintext (or **i**nverse) block M if $s = \mathsf{i}$, the **o**ther ciphertext block C' if $s = \mathsf{o}$, and **b**oth blocks M, C' if $s = \mathsf{b}$. We write $\mathsf{F}^{-1}(K, \mathsf{T}, M, b, s) = \mathsf{F}^{-1}{}_K(\mathsf{T}, M, b, s) = \mathsf{F}^{-1}{}_K^{\mathsf{T}}(M, b, s) = \mathsf{F}_K^{\mathsf{T},b,s}(M)$ interchangeably. We call k, n and $\mathcal{T}$ the keysize, blocksize and tweak space of F, respectively.

A tweakable forkcipher F meets the *correctness condition*, if for every $K \in \{0,1\}^k, \mathsf{T} \in \mathcal{T}, M \in \{0,1\}^n$ and $\beta \in \{0,1\}$ all of the following conditions are met:

1. $\mathsf{F}^{-1}(K, \mathsf{T}, \mathsf{F}(K, \mathsf{T}, M, \beta), \beta, \mathsf{i}) = M$
2. $\mathsf{F}^{-1}(K, \mathsf{T}, \mathsf{F}(K, \mathsf{T}, M, \beta), \beta, \mathsf{o}) = \mathsf{F}(K, \mathsf{T}, M, \beta \oplus 1)$
3. $(\mathsf{F}(K, \mathsf{T}, M, 0), \mathsf{F}(K, \mathsf{T}, M, 1)) = \mathsf{F}(K, \mathsf{T}, M, \mathsf{b})$
4. $\big(\mathsf{F}^{-1}(K, \mathsf{T}, C, \beta, \mathsf{i}), \mathsf{F}^{-1}(K, \mathsf{T}, C, \beta, \mathsf{o})\big) = \mathsf{F}^{-1}(K, \mathsf{T}, C, \beta, \mathsf{b})$

In other words, for each pair of key and tweak, the forkcipher applies two independent permutations to the input to produce the two output blocks. We focus on a specific form of $\mathcal{T}$ only: when $\mathcal{T} = \{0,1\}^t$ for some positive t.

[8] We again conflate the label for the primitive with the label of the encryption algorithm.

The formalization we just gave faithfully models how a forkcipher is used to realize its full potential. As explained in the discussion section of the full version [10], the most suitable FIL expanding cipher to construct modes of operation is a forkcipher, which implements two parallel tweakable permutations. Such a primitive can be formalized with a simpler syntax and equivalent functionality, such as by fixing the selector to b in both the algorithms (one could discard an unneeded output block). Yet, such a syntax would not align well with the way a forkcipher is used (for example in Section 6): our syntax of choice allows the user of a forkcipher to precisely select what gets computed, to do so more efficiently when both output blocks are needed, and without wasting computations if only one output block is required. This will become clear upon inspection of ForkSkinny in Section 4.

3.2 Security Definition

We define the security of forkciphers by indistiguishability from the closest, most natural idealized version of the primitive, a pseudorandom tweakable forked permutation, with the help of security games in Figure 2. A forked permutation is a pair of oracles, that make use of two permutations, s.t. the two permutations are always used with the same preimage, no matter if the query is made in the forward or the backward direction.

An adversary $\mathcal{A}$ that aims at breaking a tweakable forkcipher F plays the games **prtfp-real** and **prtfp-ideal**. We define the advantage of $\mathcal{A}$ at distinguishing F from a pair of random tweakable permutations in a *chosen ciphertext attack* as

$$\mathbf{Adv}_{\mathsf{F}}^{\mathrm{prtfp}}(\mathcal{A}) = \Pr[\mathcal{A}^{\mathbf{prtfp\text{-}real}_{\mathsf{F}}} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathbf{prtfp\text{-}ideal}_{\mathsf{F}}} \Rightarrow 1].$$

3.3 Iterate-Fork-Iterate

One approach to build a forkcipher from an existing iterated tweakable cipher is by applying our novel *iterate-fork-iterate*(IFI) paradigm. Following the IFI, in encryption a fixed length message block M is transformed via a fixed number of rounds or *iterations* of a tweakable cipher to M'. Then, M' is *forked* and two copies of the internal state are created, which are *iterated* to produce C_0 and C_1. Two of the main objectives of designing forkcipher in the IFI paradigm are (partial) transference of security results and maintaining forkcipher security without increasing the original cipher key size. In order to rule out that the IFI design succumbs to *generic* attacks (i.e., attacks that treat the primitive as a blackbox), we carry out a provable generic analysis. This result indicates structural soundness in the sense that no additional exploitable weakness are introduced, but does not directly imply security of IFI forkciphers, because a real forkcipher never uses a number of rounds in the partial iteration that is a secure pseudorandom permutation.

```
Game prtfp-real_F
$K \leftarrow_\$ \{0,1\}^k$
$b \leftarrow \mathcal{A}^{\text{Enc},\text{Dec}}$
return $b$

Oracle Enc$(\mathsf{T}, M, s)$
return $\mathsf{F}(K, \mathsf{T}, M, s)$

Oracle Dec$(\mathsf{T}, C, \beta, s)$
return $\mathsf{F}^{-1}(K, \mathsf{T}, C, \beta, s)$
```

```
Game prtfp-ideal_F
for $\mathsf{T} \in \mathcal{T}$ do $\pi_{\mathsf{T},0}, \pi_{\mathsf{T},1} \leftarrow_\$ \text{Perm}(n)$
$b \leftarrow \mathcal{A}^{\text{Enc},\text{Dec}}$
return $b$

Oracle Enc$(\mathsf{T}, M, s)$
if $s = 0$ then return $\pi_{\mathsf{T},0}(M)$
if $s = 1$ then return $\pi_{\mathsf{T},1}(M)$
if $s = \mathsf{b}$ then return $\pi_{\mathsf{T},0}(M)$, $\pi_{\mathsf{T},1}(M)$

Oracle Dec$(\mathsf{T}, C, \beta, s)$
if $s = \mathsf{i}$ then return $\pi^{-1}_{\mathsf{T},\beta}(C)$
if $s = \mathsf{o}$ then return $\pi_{\mathsf{T},(\beta\oplus 1)}(\pi^{-1}_{\mathsf{T},\beta}(C))$
if $s = \mathsf{b}$ then return $\pi^{-1}_{\mathsf{T},\beta}(C)$, $\pi_{\mathsf{T},(\beta\oplus 1)}(\pi^{-1}_{\mathsf{T},\beta}(C))$
```

Fig. 2: Games **prtfp-real** and **prtfp-ideal** defining the security of a (strong) forkcipher.

IFI Generic Validation. We show that a IFI forkcipher is a structurally sound construction as long as the three components: three tweak-indexed collections of permutations are ideal tweak permutations in the full version of the paper. Fix the block length n and the tweak length t. Formally, for three tweakable random permutations p, p_0, p_1 (i.e. $p = (p\mathsf{T} \leftarrow_\$ \text{Perm}(n))_{\mathsf{T}\in\{0,1\}^t}$ is a collection of independent uniform elements of Perm(n) indexed by the elements of $\mathsf{T} \in \{0,1\}^t$, and similar applies for p_0 and p_1), the forkcipher $\mathsf{F} = \text{IFI}[p, p_0, p_1]$ is defined by $\mathsf{F}^{\mathsf{T},\mathsf{b}}(M) = p_{\mathsf{T},0}(p_{\mathsf{T}}(M)), p_{\mathsf{T},1}(p_{\mathsf{T}}(M))$, and by ${\mathsf{F}^{-1}}^{\mathsf{T},b,\mathsf{b}}(C) = p^{-1}_{\mathsf{T}}(p^{-1}_{\mathsf{T},b}(C)), p_{\mathsf{T},b\oplus 1}(p^{-1}_{\mathsf{T},b}(C))$ (the rest follows from the correctness). We note that the three tweakable random permutations act as a key for IFI$[p, p_0, p_1]$ and we omit them for the sake of simplicity. In the full version [10], we prove the indistinguishability of the IFI construction from a single *forked* random permutation in the information-theoretic setting.

Our IFI instantiation. IFI is motivated by the most popular design strategy for block cipher design - *iterative* or round-based structure where the round functions are typically identical, up to round keys and constants. In forkcipher, after an initial number of rounds r_{init} two copies of the internal state are processed with different tweakeys. The number of rounds after the forking step, r_0 (left) and r_1 (right), are determined from the cryptanalytic assurances of the IFI block cipher instantiation. The block cipher round functions instantiate the forkcipher round functions (both before and after forking), again up to constants and round key addition. The single (secret) key SK security of both (left and right) forward $\mathsf{F}^{\mathsf{T},0}$, $\mathsf{F}^{\mathsf{T},1}$ and inverse ${\mathsf{F}^{-1}}^{\mathsf{T},0,\mathsf{i}}$ (resp. ${\mathsf{F}^{-1}}^{\mathsf{T},1,\mathsf{i}}$) forkcipher transformations, and the related-key (RK) security of $\mathsf{F}^{\mathsf{T},1}$ follow easily from the underlying security of the block cipher. We further perform the SK and RK analysis for $\mathsf{F}^{\mathsf{T},0}$ and the reconstruction $\mathsf{F}^{-1}{}^{\mathsf{T},0,\mathsf{o}}$ (resp. $\mathsf{F}^{-1}{}^{\mathsf{T},1,\mathsf{o}}$) transformations.

In our instantiation, $r_0 = r_1$ as a direct consequence of the IFI design approach. Suppose, in the SK model $\mathsf{F}^{\mathsf{T},0}$ is secure using $r_{\mathsf{init}} + r_0$ number of rounds. Such $\mathsf{F}^{\mathsf{T},0}$ can be instantiated using any existing (secure) off-the-shelf tweakable block cipher, which is the approach taken here. Then, having $r_{\mathsf{init}} + r_1$ rounds, where $r_1 < r_0$, for $\mathsf{F}^{\mathsf{T},1}$ will obviously weaken the security of the forkcipher. This is true, assuming that we apply the same round function in both forking branches. In this article we choose a tweakable SPN-based block cipher to construct a forkcipher.

4 ForkSkinny

We design the forkcipher ForkSkinny using the recently published lightweight tweakable block cipher SKINNY [18]. As detailed in Table 1, we propose several instances, with various block and tweakey sizes, in order to fit the different use cases. For simplifying the notation, in the rest of this section we will denote the transformations $C_b \leftarrow \mathsf{ForkSkinny}_K^{\mathsf{T},b}(M)$ as $\mathsf{ForkSkinny}_b$, where $b = 0$ or 1 and the corresponding inverse transformations $\mathsf{ForkSkinny}^{-1}{}_K^{\mathsf{T},b,\mathsf{i}}$ as $\mathsf{ForkSkinny}_b^{-1}$.

4.1 Specification

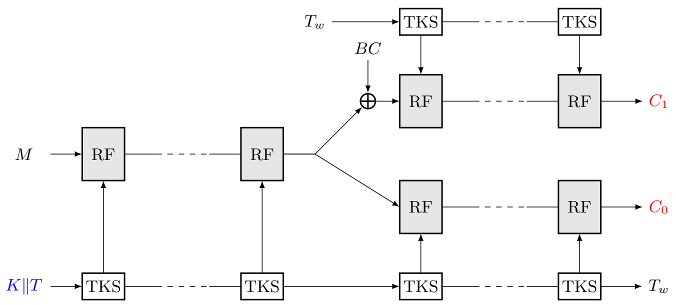

Fig. 3: ForkSkinny encryption with selector $s = \mathsf{b}$. A plaintext M, a key K and a tweak T (blue) are used to compute a ciphertext $C = C_0 \| C_1$ (red) of twice the size of the plaintext. RF is a single round function of SKINNY (with modified round constant), TKS is round tweakey update function [18], and BC is a branch constant that we introduce.

Overall Structure. We illustrate our design in Fig. 3 for ForkSkinny-128-192. This version takes a 128-bit plaintext M, a 64-bit tweak T and a 128-bit secret key K as input, and outputs two 128-bit ciphertext blocks C_0 and C_1 (i.e., $\mathsf{ForkSkinny}(K, T, M, \mathsf{b}) = C_0, C_1$). The first $r_{\mathsf{init}} = 21$ rounds of ForkSkinny are almost identical to the one of SKINNY and only differ in the value of the constant added to the internal state. After that, the encryption is *forked*, which means that two copies of the internal state are further modified with different sets of tweakeys. For reasons that we detail below, a constant denoted by BC (Branch Constant) is added to the internal state used to compute C_1, right after forking. Then, $\mathsf{ForkSkinny}_0$ iterates $r_0 = 27$ rounds and $\mathsf{ForkSkinny}_1$ iterates $r_1 = 27$ rounds. As illustrated in Figure 3, after forking, the tweakeys for the round functions of $\mathsf{ForkSkinny}_0$ are computed from the tweakey state obtained after r_{init} rounds, while the tweakeys for the round functions of $\mathsf{ForkSkinny}_1$ are derived from the tweakey state at the end of $r_{\mathsf{init}} + r_0$ rounds (denoted by T_w). Figure 4 details the ForkSkinny construction, where $\mathsf{Enc\text{-}Skinny}_r(\cdot, \cdot)$ denotes the `SKINNY` encryption using r round functions taking as input a plaintext or state together with a tweakey. Similarly, $\mathsf{Dec\text{-}Skinny}_r(\cdot, \cdot)$ denotes the corresponding decryption algorithm using r rounds.

```
function ForkSkinnyEnc($M, K, T, s$)
    $tk \leftarrow K||T$
    $L \leftarrow \mathsf{Enc\text{-}Skinny}_{r_{\mathsf{init}}}(M, tk)$
    if $s = 0$ or $s = \mathsf{b}$ then
        $C_0 \leftarrow \mathsf{Enc\text{-}Skinny}_{r_0}(L, \mathsf{TKS}_{r_{\mathsf{init}}}(tk))$
    end if
    if $s = 1$ or $s = \mathsf{b}$ then
        $tk' \leftarrow \mathsf{TKS}_{r_{\mathsf{init}}+r_0}(tk)$
        $C_1 \leftarrow \mathsf{Enc\text{-}Skinny}_{r_1}(L \oplus BC, tk')$
    end if
    if $s = 0$ return $C_0$
    if $s = 1$ return $C_1$
    if $s = \mathsf{b}$ return $C_0, C_1$
end function
```

```
function ForkSkinnyDec($C, K, T, b, s$)
    $tk \leftarrow K||T$
    $tk' \leftarrow \mathsf{TKS}_{r_{\mathsf{init}}}(tk)$
    if $b = 0$ then
        $L \leftarrow \mathsf{Dec\text{-}Skinny}_{r_0}(C, tk')$
    else if $b = 1$ then
        $tk'' \leftarrow \mathsf{TKS}_{r_0}(tk')$
        $L \leftarrow \mathsf{Dec\text{-}Skinny}_{r_1}(C_b, tk'') \oplus BC$
    end if
    if $s = \mathsf{i}$ or $s = \mathsf{b}$ then
        $M \leftarrow \mathsf{Dec\text{-}Skinny}_{r_{\mathsf{init}}}(L, tk)$
    end if
    if $s = \mathsf{o}$ or $s = \mathsf{b}$ then
        if $b = 0$ then $tk' \leftarrow \mathsf{TKS}_{r_0}(tk')$
        $C' \leftarrow \mathsf{Enc\text{-}Skinny}_{r_{b\oplus 1}}(L, tk')$
    end if
    if $s = \mathsf{i}$ return $M$
    if $s = \mathsf{o}$ return $C'$
    if $s = \mathsf{b}$ return $M, C'$
end function
```

Fig. 4: ForkSkinny encryption and decryption algorithms. Here TKS denotes the round tweakey scheduling function of `SKINNY`. TKS_r depicts r rounds of TKS.

Round function. As stated previously, the round function used in ForkSkinny is derived from the one of `SKINNY` and can be described as:

$$\mathcal{R}_i = \texttt{Mixcolumn} \circ \texttt{Addconstant} \circ \texttt{Addroundtweakey} \circ \texttt{Shiftrow} \circ \texttt{Subcell}$$

where `Subcell`, `Shiftrow` and `Mixcolumn` are identical to the ones of `SKINNY`. The `Addroundtweakey` function and the tweakey schedule are also left unchanged. Note that in ForkSkinny more tweakeys than in `SKINNY` are produced since we have $r_{\mathsf{init}} + r_0 + r_1$ rounds. To keep the content short, we leave the details of these operations to full version [10] of this article.

The only change we made in the round function of ForkSkinny stands in the `AddConstants` step. Instead of using 6 bit round constants (generated with an LFSR), we use 7 bit ones. This change was required to avoid adding the same round constant to different rounds, as 6 bit round constants only provides 64 different values while some of our instances require a number of iterations higher than that. These 7 bit round constants may be chosen randomly and fixed. In our implementation we use an affine 7 bit LFSR to generate the round constant. The update function is defined as:

$$(rc_6||rc_5||\dots||rc_0) \to (rc_5||rc_4||\dots||rc_0||rc_6 \oplus rc_5 \oplus 1)$$

The 7 bits are initialized to 0 and updated before using in the round function. The bits from the LFSR are used exactly the same way as in Skinny. The 4×4 array

$$\begin{pmatrix} c_0 & 0 & 0 & 0 \\ c_1 & 0 & 0 & 0 \\ c_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

is constructed depending on the size of the internal state, where $c_2 = \mathtt{0x2}$ and

$$(c_0, c_1) = (rc_3||rc_2||rc_1||rc_0, 0||rc_6||rc_5||rc_4) \text{ when each cell is 4 bits}$$
$$(c_0, c_1) = (0||0||0||0||rc_3||rc_2||rc_1||rc_0, 0||0||0||0||0||rc_6||rc_5||rc_4) \text{ when each cell is 8 bits.}$$

Branch Constant. We introduce constants to be added right after the forking point. When each cell is made of 4 bits we add BC_4, and when each cell is a byte we add BC_8, where:

$$BC_4 = \begin{pmatrix} \mathtt{1} & \mathtt{2} & \mathtt{4} & \mathtt{9} \\ \mathtt{3} & \mathtt{6} & \mathtt{d} & \mathtt{a} \\ \mathtt{5} & \mathtt{b} & \mathtt{7} & \mathtt{f} \\ \mathtt{e} & \mathtt{c} & \mathtt{8} & \mathtt{1} \end{pmatrix} \qquad BC_8 = \begin{pmatrix} \mathtt{01} & \mathtt{02} & \mathtt{04} & \mathtt{08} \\ \mathtt{10} & \mathtt{20} & \mathtt{41} & \mathtt{82} \\ \mathtt{05} & \mathtt{0a} & \mathtt{14} & \mathtt{28} \\ \mathtt{51} & \mathtt{a2} & \mathtt{44} & \mathtt{88} \end{pmatrix}.$$

This addition is made right after forking, to the right branch leading to C_1. Note that these constants are generated by clocking LFSRs, given by: $(x_3||x_2||x_1||x_0) \to (x_2||x_1||x_0||x_3 \oplus x_2)$, and initialised with $x_0 = 1$, $x_1 = x_2 = x_3 = 0$ for BC_4, and with the LFSR $(x_7||x_6||x_5||x_4||x_3||x_2||x_1||x_0) \to (x_6||x_5||x_4||x_3||x_2||x_1||x_0||x_7 \oplus x_5)$, again initialised with $x_0 = 1$ and all the other bits equal to 0 for BC_8.

This introduction is necessary to avoid that two `SubCells` steps cancel each others when looking at the sequence of operations relating C_0 and C_1 in the reconstruction scenario.

Variants. Other sets of parameters can be chosen. We propose some variants in Table 1. Note that their exact number of rounds (that are the parameters $r_0 = r_1$ and r_{init}), were determined from the security analysis of the cipher, detailed below.

4.2 Design Rationale

Using SKINNY. A forkcipher in IFI paradigm can be instantiated in various ways. We build our forkcipher design reusing the iterative structure of the SPN-based lightweight tweakable block cipher SKINNY. SPNs are very well-researched and allow to apply existing cryptanalysis techniques to the security analysis of our forkcipher. A large number of cryptanalytic results [12, 13, 49, 51, 56, 57] have further been published against round reduced SKINNY showing that the full version of the cipher have comfortable security margins. Unlike other lightweight block ciphers, such as Midori [16], PRINCE [29], the SKINNY design is constructed following the TWEAKEY framework, and in addition supports a number of choices for the tweak size; an important aspect for the choice of SKINNY for our design. SKINNY is good for lightweight applications on both hardware and software platforms. We also assume that the target application platform does not have AES instruction set available, hence avoiding AES based instantiation.

ForkSkinny Components. In ForkSkinny we have introduced features which aim to serve the forkcipher construction characteristics and security requirements. The 7 bit LFSR introduced in Addconstant avoids the repetition of round constants that could have possibly lead to *slide attack*-like cryptanalyses. The Branch Constant added after forking ensures that in the reconstruction scenario the two non-linear layers positionned around the forking point do not cancel each other. Finally, the required round tweakeys are computed by extending the key schedule of SKINNY by the necessary number of rounds. We chose this particular way of computing the extra tweakeys due to its simplicity, ability to maximally reuse components of SKINNY, and because it was among the most conservative options security-wise.

5 Security Analysis

For most attacks (for instance differential and linear cryptanalysis), the results devised on SKINNY give sufficient arguments to show the resistance of ForkSkinny. First, the series of operations leading M to C_0 correspond exactly to one encryption with SKINNY (up to the round constants) so the existing results transfer

Primitive	block	tweak	tweakey	r_{init}	r_0	r_1
ForkSkinny-64-192	64	64	192	17	23	23
ForkSkinny-128-192	128	64	192	21	27	27
ForkSkinny-128-256	128	128	256	21	27	27
ForkSkinny-128-288	128	160	288	25	31	31
ForkSkinny-128-384	128	256	384	25	31	31

Table 1: The ForkSkinny primitives with their internal parameters for round numbers r_{init}, r_0 and r_1 and their corresponding external parameters of block and tweakey sizes (in bits) for fixed 128 bit keys.

easily in this case. Then, when looking at the relation between M and C_1 we have a version of SKINNY with different round constants and a different tweak after r_{init} rounds. One way to give security arguments here is to look at what happens in the first r_{init} rounds and independently, in the next r_1 rounds to have a (pessimistic) estimation (for instance of the number of active Sboxes). A similar technique can be applied to study the reconstruction path. In both cases, the very large security margins[9] of SKINNY imply that ForkSkinny appears out of reach of the attacks we considered.

Our full security analysis is detailed in the full version [10] of this article. It covers truncated, impossible differential, boomerang, meet-in-the-middle, integral and algebraic attacks. We particularly stress that the boomerang type attack which was shown against ForkAES [17], is not applicable to ForkSkinny. This is due to two reasons: first, the number of rounds after the forking step protects against such attacks by making the boomerang path of very low probability. Second, the branch constant introduced in the right branch protects against such attacks by making the state of two branches different immediately after forking. Note that the attack in [17] against (9 out of 10 rounds) ForkAES in fact uses the property for which there is no difference between the states after forking.

We detail below our analysis of differential and linear attacks.

5.1 Detail of the Evaluation of Differential and Linear Attacks

Arguments in favor of the resistance of ForkSkinny to differential [28] and linear [39] cryptanalysis can easily be deduced from the available analysis on SKINNY. First, we refer to the bounds on the number of active Sboxes provided in the SKINNY specification document (recalled in the full version [10]). These bounds were later refined, and for instance Abdelkhalek et al. [7] showed that in the single key scenario there are no differential characteristics of probability higher than 2^{-128} for 14 rounds or more of SKINNY-128.

The previous results transfer to the case where we look at a trail covering the path from the input message up to C_0. Due to the change in the tweakey schedule we expect different bounds in the related-tweakey for the path from the input message up to C_1. A rough estimate of the minimal number of active Sboxes on this trail can be obtained by summing the bound on r_{init} rounds and the bound on r_1 rounds. For instance for ForkSkinny-128-192 (in TK2 model), 21 rounds activate at least 59 Sboxes. If we consider that the branch starting from the forking point is independent and can start from any internal state difference and tweakey difference (this is the very pessimistic case), only 8 rounds after forking are necessary to go below the characteristic probability of 2^{-128}.

The last case that needs to be evaluated is the reconstruction path scenario. An estimate can be computed following the same idea as before: the number of active Sboxes can be upper bounded by the bound obtained by summing the one for r_0 rounds and the one for r_1 rounds. If we consider that $r_0 = r_1$ as for our concrete instances, we obtain that 16 rounds are required to get more than 64

[9] At the time of writing, the best attacks on SKINNY cover at most 55% of the cipher.

active Sboxes. For ForkSkinny-128-192, 30 rounds are required to get more than 64 active Sboxes.

With respect to the parameters we chose, these (optimistic for the attacker) evaluations make us believe that differential attacks pose no threat to our proposal.

Similar arguments lead to the same conclusion for linear attacks. Also, we refer to the FSE 2017 paper [35] by Kranz et al. that looks at the linear hull of a tweakable block cipher and shows that the addition of a tweak does not introduce new linear characteristics, so that no additional precaution should be taken in comparison to a key-only cipher.

6 Tweakable Forkcipher Modes

We demonstrate the applicability of forkciphers by designing provably secure AEAD modes of operation for a tweakable forkcipher. Our AEAD schemes are designed such that (1) they are able to process strings of *arbitrary length* but (2) they are most efficient for data whose total number of blocks (in AD and message) is small, e.g. below four.

We define three forkcipher, nonce-based AEAD modes of operation: PAEF, SAEF and RPAEF. The first mode is fully parallelizable and (quantitatively) optimally secure in the nonce respecting model. The second mode SAEF sequentially encrypts "on-the-fly", has birthday-bounded security, and lends itself to low-overhead implementations. The third mode RPAEF is derived from the first one; it only uses both output blocks of a forkcipher in the final call, allowing to further reduce computational cost even for longer messages. The improved efficiency comes at the price of an n-bit larger tweak, and thus increased HW area footprint.

A small AE primitive. While a secure forkcipher does not directly capture integrity, we show in Section 6.9 that a secure forkcipher can be used as an AEAD scheme with fixed length messages and AD in the natural way, provably delivering strong AE security guarantees.

6.1 Syntax and Security of AEAD

Our modes following the AEAD syntax proposed by Rogaway [46]. A nonce-based AEAD scheme is a triplet $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. The key space $\mathcal{K}$ is a finite set. The deterministic encryption algorithm $\mathcal{E} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{C}$ maps a secret key K, a nonce N, an associated data A and a message M to a ciphertext $C = \mathcal{E}(K, N, A, M)$. The nonce, AD and message domains are all subsets of $\{0,1\}^*$. The deterministic decryption algorithm $\mathcal{D} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{C} \to \mathcal{M} \cup \{\perp\}$ takes a tuple (K, N, A, C) and either returns a mesage $M \in \mathcal{M}$, or a distinguished symbol $\perp$ to indicate an authentication error.

We require that for every $M \in \mathcal{M}$, we have $\{0,1\}^{|M|} \subseteq \mathcal{M}$ (i.e. for any integer m, either all or no strings of length m belong to $\mathcal{M}$) and that for all

$K, N, A, M \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M}$ we have $|\mathcal{E}(K, N, A, M)| = |M| + \tau$ for some non-negative integer τ called the stretch of Π. For correctness of Π, we require that for all $K, N, A, M \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M}$ we have $M = \mathcal{D}(K, N, A, \mathcal{E}(K, N, A, M))$. We let $\mathcal{E}_K(N, A, M) = \mathcal{E}(K, N, A, M)$ and $\mathcal{D}_K(N, A, M) = \mathcal{D}(K, N, A, M)$.

We follow Rogaway's two-requirement definition of AE security. A chosen plaintext attack of an adversary $\mathcal{A}$ against the confidentiality of a nonce-based AE scheme Π is captured with the help of the security games **priv-real** and **priv-real**. In both games, the adversary can make arbitrary chosen plaintext queries to a blackbox encryption oracle, such that each query must have a unique nonce, and such that the queries are replied with the scheme Π using a random secret key (real), or with independent uniform strings of the same length (ideal). The goal of $\mathcal{A}$ is to distinguish the two games. We define the advantage of $\mathcal{A}$ in breaking the confidentiality of Π as $\mathbf{Adv}_{\Pi}^{\mathbf{priv}}(\mathcal{A}) = \Pr[\mathcal{A}^{\mathbf{priv\text{-}real}_{\Pi}} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathbf{priv\text{-}ideal}_{\Pi}} \Rightarrow 1]$.

A chosen ciphertext attack against the integrity of Π is captured with the game **auth**, in which an adversary can make nonce-respecting chosen plaintext and arbitrary chosen ciphertext queries to a black-box instance of Π with the goal of finding a forgery: a tuple that decrypts correctly but is not trivially knwn from the encryption queries. We define the advantage of $\mathcal{A}$ in breaking the integrity of Π as $\mathbf{Adv}_{\Pi}^{\mathbf{priv}}(\mathcal{A}) = \Pr[\mathcal{A}^{\mathbf{auth}_{\Pi}} \text{forges}]$ where "$\mathcal{A}$ forges" denotes a decryption query that returns a value $\neq \bot$. (For convenience, the games are included in full version of this article.)

6.2 Parallel AE from a Forkcipher

The nonce-based AEAD scheme PAEF ("Parallel AE from a Forkcipher") is parameterized by a forkcipher F (Section 3) with $\mathcal{T} = \{0,1\}^t$ for a positive t. It is further parameterized by a nonce length $0 < \nu \leq t - 4$. An instance $\mathrm{PAEF}[\mathsf{F}, \nu] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ has $\mathcal{K} = \{0,1\}^k$ and the encryption (Figure 6) and decryption algorithms are defined in Figure 5. Its nonce space is $\mathcal{N} = \{0,1\}^\nu$, and its message and AD space are respectively $\mathcal{M} = \{0,1\}^{\leq n \cdot (2^{(t-\nu-3)}-1)}$, and $\mathcal{A} = \{0,1\}^{\leq n \cdot (2^{(t-\nu-3)}-1)}$ (i.e., AD and message can have at most $2^{(t-\nu-3)} - 1$ blocks). The ciphertext expansion of $\mathrm{PAEF}[\mathsf{F}, \nu]$ is n bits.

In an encryption query, AD and message are partitioned into blocks of n bits. Each block is processed with one call to F using a tweak that is composed of: 1) the nonce; 2) a three-bit flag $f_0 \| f_1 \| f_2$; 3) a $(t - \nu - 3)$-bit encoding of the block index (unique for both AD and message). The nonce-length is a parameter that allows to make a trade-off between the maximal message length and maximal number of queries with the same key. The bit $f_0 = 1$ iff the final block of message is being processed, $f_1 = 1$ iff a block of message is being processed, and $f_2 = 1$ iff the final block of the current input (depending on f_1) is processed and the block is incomplete. The ciphertext blocks are the "left" output blocks of F applied to message blocks, and the right "right" output blocks are xor-summed with the AD output blocks, and the result xored to the final ciphertext block.

The decryption proceeds similarly as the encryption, except that "right" output blocks of the message blocks are reconstructed from ciphertext blocks (using the reconstruction algorithm) to recompute the tag, which is then checked.

6.3 Security of PAEF

We state the formal claim about the nonce-based AE security of PAEF in Theorem 1.

Theorem 1. *Let F be a tweakable forkcipher with $\mathcal{T} = \{0,1\}^t$, and let $0 < \nu \leq t-4$. Then for any nonce-respecting adversary $\mathcal{A}$ whose queries lie in the proper domains of the encryption and decryption algorithms and who makes at most q_v decryption queries, we have*

$$\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{PAEF}[\mathsf{F},\nu]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{B}) \quad \text{and} \quad \mathbf{Adv}^{\mathbf{auth}}_{\mathrm{PAEF}[\mathsf{F},\nu]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{prfp}}_{\mathsf{F}}(\mathcal{C}) + \frac{q_v \cdot 2^n}{(2^n-1)^2}$$

for some adversaries $\mathcal{B}$ and $\mathcal{C}$ who make at most twice as many queries in total as is the total number of blocks in all encryption, respectively all encryption and decryption queries made by $\mathcal{A}$, and who run in time given by the running time of $\mathcal{A}$ plus an overhead that is linear in the total number of blocks in all $\mathcal{A}$'s queries.

Proof (sketch). The full proof appears in the full version of the paper [10]. For both confidentiality and authenticity, we first replace F with a pair of independent random tweakable permutations. Using a standard hybrid argument we have that $\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{PAEF}[\mathsf{F},\nu]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{B}) + \mathbf{Adv}^{\mathbf{priv}}_{\mathrm{PAEF}[(\pi_0,\pi_1),\nu]}(\mathcal{A})$, and also that $\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{PAEF}[\mathsf{F},\nu]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{C}) + \mathbf{Adv}^{\mathbf{priv}}_{\mathrm{PAEF}[(\pi_0,\pi_1),\nu]}(\mathcal{A})$.

For confidentiality, it is easy to see that in a nonce-respecting attack, every ciphertext block, and each tag is processed using a unique tweak-permutation combination, and all are uniformly distributed. Thus $\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{PAEF}[(\pi_0,\pi_1),\nu]}(\mathcal{A}) = 0$. For authenticity, we analyse the probability of forgery for an adversary $\mathcal{A}'$ that makes a single decryption query against $\mathrm{PAEF}[(\pi_0,\pi_1),\nu]$ by the means of a case analysis, and then use a result of Bellare et al. [21] to obtain $\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{PAEF}[(\pi_0,\pi_1),\nu]}(\mathcal{A}) \leq q_v \cdot \mathbf{Adv}^{\mathbf{auth}}_{\mathrm{PAEF}[(\pi_0,\pi_1),\nu]}(\mathcal{A}')$.

6.4 Sequential AE from a Forkcipher

SAEF (as in "Sequential AE from a Forkcipher," pronounce as "safe") is a nonce-based AEAD scheme parameterized by a tweakable forkcipher F (as defined in Section 3) with $\mathcal{T} = \{0,1\}^t$ for a positive $t \leq n$. An instance $\mathrm{SAEF}[\mathsf{F}] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ has a key space $\mathcal{K} = \{0,1\}^k$, nonce space $\mathcal{N} = \{0,1\}^{t-4}$, and the AD and message spaces are both $\{0,1\}^*$ (although the maximal AD/message length influences the security). The ciphertext expansion of $\mathrm{SAEF}[\mathsf{F}]$ is n bits. The encryption and decryption algorithms are defined in Figure 7 and the encryption algorithm is illustrated in Figure 8.

In an encryption query, first AD and then message are processed in blocks of n bits. Each block is processed with exactly one call to F, using a tweak

```
function $\mathcal{E}(K, N, A, M)$
    $A_1, \ldots, A_a, A_* \overset{n}{\leftarrow} A$
    $M_1, \ldots, M_m, M_* \overset{n}{\leftarrow} M$
    $S \leftarrow 0^n$; $c \leftarrow (t - \nu - 3)$
    for $i \leftarrow 1$ to $a$ do
        ◇$\mathsf{T} \leftarrow N\|000\|\langle i \rangle_c$
        ○$\mathsf{T} \leftarrow N\|000\|\langle i \rangle_c\|0^n$
        $S \leftarrow S \oplus \mathsf{F}_K^{\mathsf{T},0}(A_i)$
    end for
    if $|A_*| = n$ then
        ◇$\mathsf{T} \leftarrow N\|001\|\langle a+1 \rangle_c$
        ○$\mathsf{T} \leftarrow N\|001\|\langle a+1 \rangle_c\|0^n$
        $S \leftarrow S \oplus \mathsf{F}_K^{\mathsf{T},0}(A_*)$
    else if $|A_*| > 0$ or $|M| = 0$ then
        ◇$\mathsf{T} \leftarrow N\|011\|\langle a+1 \rangle_c$
        ○$\mathsf{T} \leftarrow N\|011\|\langle a+1 \rangle_c\|0^n$
        $S \leftarrow S \oplus \mathsf{F}_K^{\mathsf{T},0}(A_*\|10^*)$
    end if                    ▷ Do nothing if $A = \varepsilon, M \neq \varepsilon$
    for $i \leftarrow 1$ to $m$ do
        ◇$\mathsf{T} \leftarrow N\|100\|\langle i \rangle_c$
        ◇$C_i, S' \leftarrow \mathsf{F}_K^{\mathsf{T},\mathsf{b}}(M_i)$
        ◇$S \leftarrow S \oplus S'$
        ○$\mathsf{T} \leftarrow N\|100\|\langle i \rangle_c\|0^n$
        ○$C_i \leftarrow \mathsf{F}_K^{\mathsf{T},0}(M_i)$
        ○$S \leftarrow S \oplus M_i$
    end for
    if $|M_*| = n$ then
        ◇$\mathsf{T} \leftarrow N\|101\|\langle m+1 \rangle_c$
        ○$\mathsf{T} \leftarrow N\|101\|\langle m+1 \rangle_c\|S$
    else if $|M_*| > 0$ then
        ◇$\mathsf{T} \leftarrow N\|111\|\langle m+1 \rangle_c$
        ○$\mathsf{T} \leftarrow N\|111\|\langle m+1 \rangle_c\|S$
    else
        return $S$
    end if
    $C_*, T \leftarrow \mathsf{F}_K^{\mathsf{T},\mathsf{b}}(\mathsf{pad10}(M_*))$
    ◇$C_* \leftarrow C_* \oplus S$
    return $C_1\|\ldots\|C_m\|C_*\|\mathsf{left}_{|M_*|}(T)$
end function
```

```
function $\mathcal{D}(K, N, A, C)$
    $A_1, \ldots, A_a, A_* \overset{n}{\leftarrow} A$
    $C_1, \ldots, C_m, C_*, T \leftarrow \mathsf{csplit\text{-}b}_n(C)$
    $S \leftarrow 0^n$; $c \leftarrow (t - \nu - 3)$
    for $i \leftarrow 1$ to $a$ do
        ◇$\mathsf{T} \leftarrow N\|000\|\langle i \rangle_c$
        ○$\mathsf{T} \leftarrow N\|000\|\langle i \rangle_c\|0^n$
        $S \leftarrow S \oplus \mathsf{F}_K^{\mathsf{T},0}(A_i)$
    end for
    if $|A_*| = n$ then
        ◇$\mathsf{T} \leftarrow N\|001\|\langle a+1 \rangle_c$
        ○$\mathsf{T} \leftarrow N\|001\|\langle a+1 \rangle_c\|0^n$
        $S \leftarrow S \oplus \mathsf{F}_K^{\mathsf{T},0}(A_*)$
    else if $|A_*| > 0$ or $|T| = 0$ then
        ◇$\mathsf{T} \leftarrow N\|011\|\langle a+1 \rangle_c$
        ○$\mathsf{T} \leftarrow N\|011\|\langle a+1 \rangle_c\|0^n$
        $S \leftarrow S \oplus \mathsf{F}_K^{\mathsf{T},0}(A_*\|10^*)$
    end if                    ▷ Do nothing if $A = \varepsilon, M \neq \varepsilon$
    for $i \leftarrow 1$ to $m$ do
        ◇$\mathsf{T} \leftarrow N\|100\|\langle i \rangle_c$
        ◇$M_i, S' \leftarrow \mathsf{F^{-1}}_K^{\mathsf{T},0,\mathsf{b}}(C_i)$
        ◇$S \leftarrow S \oplus S'$
        ○$\mathsf{T} \leftarrow N\|100\|\langle i \rangle_c\|0^n$
        ○$M_i \leftarrow \mathsf{F^{-1}}_K^{\mathsf{T},0,\mathsf{i}}(C_i)$
        ○$S \leftarrow S \oplus M_i$
    end for
    if $|T| = n$ then
        ◇$\mathsf{T} \leftarrow N\|101|\langle m+1 \rangle_c$
        ○$\mathsf{T} \leftarrow N\|101|\langle m+1 \rangle_c\|S$
    else if $|T| > 0$ then
        ◇$\mathsf{T} \leftarrow N\|111\|\langle m+1 \rangle_c$
        ○$\mathsf{T} \leftarrow N\|111\|\langle m+1 \rangle_c\|S$
    else
        if $C_* \neq S$ then return $\perp$
        return $\varepsilon$
    end if
    ◇$C_* \leftarrow C_* \oplus S$
    $M_*, T' \leftarrow \mathsf{F^{-1}}_K^{\mathsf{T},0,\mathsf{b}}(C_* \oplus S)$
    $T' \leftarrow \mathsf{left}_{|T|}(T')$; $P \leftarrow \mathsf{right}_{n-|T|}(M_*)$
    if $T' \neq T$ return $\perp$
    if $P \neq \mathsf{left}_{n-|T|}(10^{n-1})$ return $\perp$
    return $M_1\|\ldots\|M_m\|\mathsf{left}_{|T|}(M_*)$
end function
```

Fig. 5: The PAEF$[\mathsf{F}, \nu]$ (unmarked lines and ◇-marked lines) and the RPAEF$[\mathsf{F}, \nu]$ (unmarked lines and ○-marked lines) AEAD schemes. Here $\langle i \rangle_\ell$ is the cannonical encoding of an integer i as an ℓ-bit string.

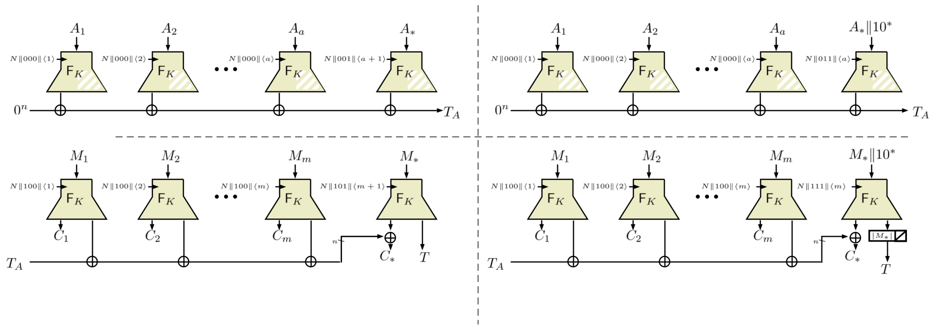
A_1
A_2
A_a
A_*
$N\|000\|\langle 1\rangle$
$N\|000\|\langle 2\rangle$
$N\|000\|\langle a\rangle$
$N\|001\|\langle a+1\rangle$
F_K
0^n
T_A
$A_*\|10^*$
$N\|011\|\langle a\rangle$
M_1
M_2
M_m
M_*
$N\|100\|\langle 1\rangle$
$N\|100\|\langle 2\rangle$
$N\|100\|\langle m\rangle$
$N\|101\|\langle m+1\rangle$
C_1
C_2
C_m
C_*
T
n
$M_*\|10^*$
$N\|111\|\langle m\rangle$
$|M_*|$

Fig. 6: The encryption algorithm of PAEF[F] mode. The picture illustrates the processing of AD when length of AD is a multiple of n (**top left**) and when the length of AD is not a multiple of n (**top right**), and the processing of the message when length of the message is a multiple of n (**bottom left**) and when the length of message is not a multiple of n (**bottom right**). The white hatching denotes that an output block is not computed.

that is composed of: (1) the nonce followed by a 1-bit in the initial F call, and the string $0^{\tau-3}$ otherwise, (2) three-bit flag f. The binary flag f takes different values for processing of different types of blocks in the encryption algorithm. The values $f = \{000, 010, 011, 110, 111, 001, 100, 101\}$ indicate the processing of respectively: non-final AD block; final complete AD block; final incomplete AD block; final complete AD block to produce tag; final incomplete AD block to produce tag; non-final message block; final complete message block; and final incomplete message block.

One output block of every F call is used as a whitening mask for the following F call, masking either the input (in AD processing) or both the input and the output (in message processing) of this subsequent call. The initial F call in the query is unmasked. The tag is the last "right" output of F produced in the query. The decryption proceeds similarly to the encryption, except that the plaintext blocks and the right-hand outputs of F in the message processing part are computed with the inverse F algorithm.

6.5 Security of SAEF

We state the formal claim about the nonce-based AE security of SAEF in Theorem 2.

Theorem 2. *Let F be a tweakable forkcipher with $\mathcal{T} = \{0,1\}^\tau$. Then for any nonce-respecting adversary $\mathcal{A}$ whose makes at most q encryption queries, at most q_v decryption queries such that the total number of forkcipher calls induced by all the queries is at most σ, with $\sigma \leq 2^n/2$, we have*

$$\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{SAEF}[\mathsf{F}]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{B}) + 2 \cdot \frac{(\sigma-q)^2}{2^n},$$

$$\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{SAEF}[\mathsf{F}]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{C}) + \frac{(\sigma-q+1)^2}{2^n} + \frac{\sigma(\sigma-q)}{2^n} + \frac{q_v(q+2)}{2^n}$$

```
function $\mathcal{E}(K, N, A, M)$
    $A_1, \dots, A_a, A_* \xleftarrow{n} A$
    $M_1, \dots, M_m, M_* \xleftarrow{n} M$
    $\mathsf{noM} \leftarrow 0$
    if $|M| = 0$ then $\mathsf{noM} \leftarrow 1$
    $\Delta \leftarrow 0^n$; $\mathsf{T} \leftarrow N \| 0^{\tau-4-\nu} \| 1$
    for $i \leftarrow 1$ to $a$ do
        $\mathsf{T} \leftarrow \mathsf{T} \| 000$
        $\Delta \leftarrow \mathsf{F}_K^{\mathsf{T},0}(A_i \oplus \Delta)$
        $\mathsf{T} \leftarrow 0^{\tau-3}$
    end for
    if $|A_*| = n$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| \mathsf{noM} \| 10$
        $\Delta \leftarrow \mathsf{F}_K^{\mathsf{T},0}(A_* \oplus \Delta)$
        $\mathsf{T} \leftarrow 0^{\tau-3}$
    else if $|A_*| > 0$ or $|M| = 0$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| \mathsf{noM} \| 11$
        $\Delta \leftarrow \mathsf{F}_K^{\mathsf{T},0}((A_* \| 10^*) \oplus \Delta)$
        $\mathsf{T} \leftarrow 0^{\tau-3}$
    end if                    ▷ Do nothing if $A = \varepsilon, M \neq \varepsilon$
    for $i \leftarrow 1$ to $m$ do
        $\mathsf{T} \leftarrow \mathsf{T} \| 001$
        $C_i, \Delta \leftarrow \mathsf{F}_K^{\mathsf{T},\mathsf{b}}(M_i \oplus \Delta) \oplus (\Delta, 0^n)$
        $\mathsf{T} \leftarrow 0^{\tau-3}$
    end for
    if $|M_*| = n$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| 100$
    else if $|M_*| > 0$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| 101$
    else
        return $\Delta$
    end if
    $C_*, T \leftarrow \mathsf{F}_K^{\mathsf{T},\mathsf{b}}(\mathsf{pad10}(M_*) \oplus \Delta) \oplus (\Delta \| 0^n)$
    return $C_1 \| \dots \| C_m \| C_* \| \mathsf{left}_{|M_*|}(T)$
end function
```

```
function $\mathcal{D}(K, N, A, C)$
    $A_1, \dots, A_a, A_* \xleftarrow{n} A$
    $C_1, \dots, C_m, C_*, T \leftarrow \mathsf{csplit\text{-}b}_n C$
    $\mathsf{noM} \leftarrow 0$
    if $|C| = n$ then $\mathsf{noM} \leftarrow 1$
    $\Delta \leftarrow 0^n$; $\mathsf{T} \leftarrow N \| 0^{\tau-4-\nu} \| 1$
    for $i \leftarrow 1$ to $a$ do
        $\mathsf{T} \leftarrow \mathsf{T} \| 000$
        $\Delta \leftarrow \mathsf{F}_K^{\mathsf{T},0}(A_i \oplus \Delta)$
        $\mathsf{T} \leftarrow 0^{\tau-3}$
    end for
    if $|A_*| = n$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| \mathsf{noM} \| 10$
        $\Delta \leftarrow \mathsf{F}_K^{\mathsf{T},0}(A_* \oplus \Delta)$
        $\mathsf{T} \leftarrow 0^{\tau-3}$
    else if $|A_*| > 0$ or $|T| = 0$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| \mathsf{noM} \| 11$
        $\Delta \leftarrow \mathsf{F}_K^{\mathsf{T},0}((A_* \| 10^*) \oplus \Delta)$
        $\mathsf{T} \leftarrow 0^{\tau-3}$
    end if                    ▷ Do nothing if $A = \varepsilon, M \neq \varepsilon$
    for $i \leftarrow 1$ to $m$ do
        $\mathsf{T} \leftarrow \mathsf{T} \| 001$
        $M_i, \Delta \leftarrow \mathsf{F^{-1}}_K^{\mathsf{T},0,\mathsf{b}}(C_i \oplus \Delta) \oplus (\Delta, 0^n)$
        $\mathsf{T} \leftarrow 0^{\tau-3}$
    end for
    if $|T| = n$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| 100$
    else if $|T| > 0$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| 101$
    else
        if $C_* \neq \Delta$ then return $\perp$
        return $\varepsilon$
    end if
    $M_*, T' \leftarrow \mathsf{F^{-1}}_K^{\mathsf{T},0,\mathsf{b}}(C_* \oplus \Delta) \oplus (\Delta, 0^n)$
    $T' \leftarrow \mathsf{left}_{|T|}(T')$; $P \leftarrow \mathsf{right}_{n-|T|}(M_*)$
    if $T' \neq T$ return $\perp$
    if $P \neq \mathsf{left}_{n-|T|}(10^{n-1})$ return $\perp$
    return $M_1 \| \dots \| M_m \| \mathsf{left}_{|T|}(M_*)$
end function
```

Fig. 7: The SAEF[F] AEAD scheme.

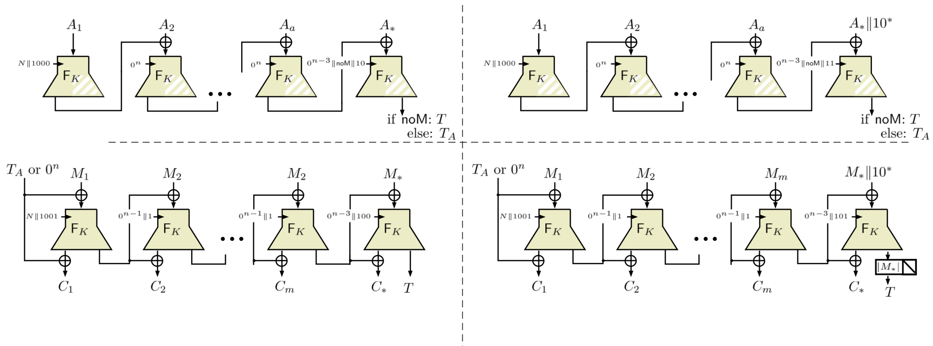

Fig. 8: The encryption algorithm of SAEF[F] mode. The bit $\mathsf{noM} = 1$ iff $|M| = 0$. The picture illustrates the processing of AD when length of AD is a multiple of n (**top left**) and when the length of AD is not a multiple of n (**top right**), and the processing of the message when length of the message is a multiple of n (**bottom left**) and when the length of message is not a multiple of n (**bottom right**). The white hatching denotes that an output block is not computed.

for some adversaries $\mathcal{B}$ and $\mathcal{C}$ who make at most 2σ queries, and who run in time given by the running time of $\mathcal{A}$ plus $\gamma \cdot \sigma$ for some constant γ.

Proof (sketch). The full proof appears in the full version of the paper [10]. As with PAEF, we first replace F with a pair of independent random tweakable permutations, resulting in a similar birthday gap.

For confidentiality, we further replace tweakable permutations by random "tweakable" functions, further increasing the bound by $2 \cdot (\sigma - q)^2/2^{n+1}$ due to an RP-RF switch. Unless there is a non-trivial collision of inputs to f_0 and f_1, confidentiality of SAEF$[(f_0, f_1), \nu]$ is perfect. With such a collision appearing with a probability no greater than $2 \cdot (\sigma - q)^2/2^{n+1}$, we obtain the bound.

In the proof of integrity, we replace certain random permutations (indexed by a specific subset of tweaks) of the underlying tweakable permutations by tweakable functions with the same signature, increasing the bound by $(\sigma - q + 1)^2/2^{n+1}$ due to an RP-RF switch. We then define a variant of the **auth** game (call it **auth′**), which prevents $\mathcal{A}$ to win if an primitive input collision occurs in any of the encryption queries. The transition to the new game increases the bound by $\sigma(\sigma - q)/2^n$. Finally, (using the result of Bellare as for PAEF), we bound the probability of a successful forgery in **auth′** with help of a case analysis by $2 \cdot q_v/(2^n - 1)$.

6.6 Reduced Parallel AE from a Forkcipher

The nonce-based AEAD scheme RPAEF ("Reduced Parallel AE from a Forkcipher") is a derivative of PAEF that only uses the left output block of the underlying forkcipher for most of the message blocks. This allows for *reducing* the computational cost if the unevaluated fork can be switched off (as in ForkSkinny) at the expense of increasing the required tweak size. It is parameterized by a forkcipher F (Section 3) with $\mathcal{T} = \{0,1\}^t$ for a positive $t \geq n+5$. It is further parameterized by a nonce length $0 < \nu \leq t - (n+4)$. An instance RPAEF$[\mathsf{F}, \nu] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ has $\mathcal{K} = \{0,1\}^k$ and the encryption (Figure 9) and decryption algorithms are defined in Figure 5. Its nonce space is $\mathcal{N} = \{0,1\}^\nu$, and its message and AD space are respectively $\mathcal{M} = \{0,1\}^{\leq n \cdot (2^{(t-(n+\nu+3))}-1)}$, and $\mathcal{A} = \{0,1\}^{\leq n \cdot (2^{(t-(n+\nu+3))}-1)}$ (i.e. AD and message can have at most $2^{(t-(n+\nu+3))} - 1$ blocks). The ciphertext expansion of PAEF$[\mathsf{F}, \nu]$ is n bits.

In an encryption query, AD and message are processed in blocks of n bits. Each block is processed with one call to F using a tweak in which the first t bits are the same as in PAEF and the remaining n bits are either equal to a "checksum" of of all AD blocks and all-but-last message blocks, or to n zero bits (all other F calls). For all message blocks except the last one, only the left output block of F is used. The decryption proceeds similarly as the encryption, except that putative message blocks are reconstructed from ciphertext blocks to recompute the "checksum".

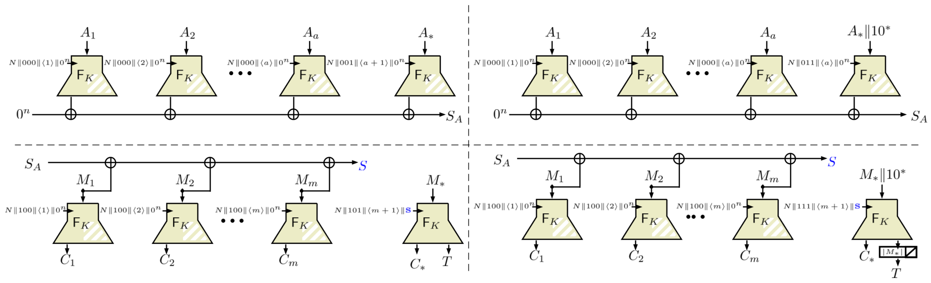
A_1
A_2
A_a
A_*
$A_*\|10^*$
$N\|000\|\langle 1\rangle\|0^n$
$N\|000\|\langle 2\rangle\|0^n$
$N\|000\|\langle a\rangle\|0^n$
$N\|001\|\langle a+1\rangle\|0^n$
$N\|011\|\langle a\rangle\|0^n$
F_K
0^n
S_A
S
M_1
M_2
M_m
M_*
$M_*\|10^*$
$N\|100\|\langle 1\rangle\|0^n$
$N\|100\|\langle 2\rangle\|0^n$
$N\|100\|\langle m\rangle\|0^n$
$N\|101\|\langle m+1\rangle\|S$
$N\|111\|\langle m+1\rangle\|S$
C_1
C_2
C_m
C_*
T
$|M_*|$

Fig. 9: The encryption algorithm of RPAEF[F] mode. The picture illustrates the processing of AD when length of AD is a multiple of n (**top left**) and when the length of AD is not a multiple of n (**top right**), and the processing of the message when length of the message is a multiple of n (**bottom left**) and when the length of message is not a multiple of n (**bottom right**). The white hatching denotes that an output block is not computed.

6.7 Security of RPAEF

Theorem 3. *Let F be a tweakable forkcipher with $\mathcal{T} = \{0,1\}^t$ and $t \geq n+5$, and let $0 < \nu \leq t-4$. Then for any nonce-respecting adversary $\mathcal{A}$ whose queries lie in the proper domains of the encryption and decryption algorithms and who makes at most q_v decryption queries, we have*

$$\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{PAEF}[\mathsf{F},\nu]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{B}) \quad \text{and} \quad \mathbf{Adv}^{\mathbf{auth}}_{\mathrm{PAEF}[\mathsf{F},\nu]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{prfp}}_{\mathsf{F}}(\mathcal{C}) + \frac{2 \cdot q_v}{(2^n-1)}$$

for some adversaries $\mathcal{B}$ and $\mathcal{C}$ who make at most twice as many queries in total as is the total number of blocks in all encryption, respectively all encryption and decryption queries made by $\mathcal{A}$, and who run in time given by the running time of $\mathcal{A}$ plus an overhead that is linear in the total number of blocks in all $\mathcal{A}$'s queries.

Proof (sketch). The full proof appears in the full version of the paper [10]. For both confidentiality and authenticity, we first replace F with a pair of independent random tweakable permutations, similarly as for PAEF.

For confidentiality, it is easy to see that, exactly as with PAEF, in a nonce-respecting attack every ciphertext block and all tags are uniformly distributed. We have $\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{PAEF}[(\pi_0,\pi_1),\nu]}(\mathcal{A}) = 0$.

For authenticity, we combine a case analysis and the same result of Bellare et al. [21] as used for PAEF to obtain the bound.

6.8 Aggressive RPAEF instance.

We remark that when instantiated with ForkSkinny-128-384 (smaller tweakey would not make sense due to RPAEF's tweak size requirements), one of the three 128-bit tweakey schedule registers is effectively unused for all but last message blocks (it holds the the 0^n tweak component). Based on this observation, we consider a further, more aggressive optimization of RPAEF, which consists in *lowering the numbers of applied rounds* to those from ForkSkinny-128-256 for all but last message blocks, and for all AD blocks. A thorough analysis of this aggressive variant of ForkSkinny with a number of rounds adjusted to the effective tweak size is left as an open question. We do note, however that every tweak will only ever be used with a fixed number of rounds.

6.9 Deterministic MiniAE

In the introduction, we stated that a forkcipher is nearly, but not exactly, an AE primitive: we clarify this statement in the full version of the paper [10]. In short: it is easy to see that the syntax and security goals of a forkcipher, as proposed in Section 3, capture neither AE functionality nor AE security goals. Yet, *constructing* a secure PRI (with the same signature) from the forkcipher is trivial: just set $\mathcal{E}(K,N,A,M) = \mathsf{F}_K^{N\|A,\mathsf{b}}(M)$ and $\mathcal{D}(K,N,A,C\|T) = \mathsf{F}^{-1}{}_K^{N\|A,0,\mathsf{i}}(C)$ iff $T = \mathsf{F}^{-1}{}_K^{N\|A,0,\mathsf{o}}(C)$. We prove that when used in this minimalistic "mode" of operation, a secure forkcipher yields a miniature AE scheme for fixed-size messages, which achieves PRI security [48].

7 Hardware Performance

Due to the independent branching of the data flow after the forking point, ForkSkinny comes with inherent data-level parallelism that does not exist in normal (tweakable) blockciphers like SKINNY. We illustrate how round-based hardware implementations amplify the performance boost of our forkcipher modes, well beyond the reduction of blockcipher rounds as argued in Section 1. We give a preliminary hardware implementation of all ForkSkinny variants in our three modes of operation, and compare the results with Skinny-Aead [19] as the most fairly comparable TBC mode of operation based on SKINNY.

Implementations. Figure 10 presents hardware synthesis results (ASIC) for open cell library NANGATE45NM in typical operating conditions. Messages as small as 8 bytes (64 bits) are considered separately, for which we select M6 as the most suitable SKINNY-AEAD family member. For processing 128-bit blocks, concrete instances are partitioned based matching tweakey lengths. The hardware area is partly based on synthesis results (i.e. the primitive) and partly estimated (i.e. the mode). For details on implementation assumptions, area estimation methodology and synthesis flow, please refer to TO DO: Full version.

For SKINNY-AEAD, we resynthesize the publicly available SKINNY round-based encryption implementations[10]. The ForkSkinny implementations are a modification thereof, with a second state register, branch constant logic and extended round constant. We then go on to obtain parallel ForkSkinny implementations, denoted (//), by adding an extra copy of the round function to compute both branches simultaneously. We also implement the aggressive variant of RPAEF with tuned-down number of SKINNY rounds (see Section 6.8).

Results Interpretation. When implementations exploit the available primitive-level parallelism, the forkcipher performance boost is substantial. For instance, for messages up to three 128-bit blocks, the speed-up of PAEF and SAEF (both parallel (//)) ranges from 25% to 50%, where the advantage is largest for the single-block messages. RPAEF shows similar numbers, with a $5\% - 22\%$ speed-up for the "aggressive" version. Most notably, for parallel instances (//) the forkcipher invocations are essentially equally fast as block cipher invocations, which results in fewer cycles than SKINNY-AEAD *for all* message sizes. However, this advantage diminishes asymptotically with the message size (cf. the *general* column). For message sizes up to 8 bytes, emphasized by NIST [42], the PAEF-FORKSKINNY-64-192 instances are more than 58% faster (40 vs. 96 cycles) at a considerably smaller implementation size. SAEF has the disadvantage of being a serial mode but it has the smallest area (no block counter and nonce in tweak).

8 Conclusion

The idea of forkcipher opens up numerous interesting open question and research directions. For a detailed discussion we refer to the full version of this article [10].

Ackowledgements Elena Andreeva was supported in part by the Research Council KU Leuven C1 on Security and Privacy for Cyber-Physical Systems and the Internet of Things with contract number C16/15/058 and by the Research Council KU Leuven, C16/18/004, through the EIT Health RAMSES project, through the IF/C1 on New Block Cipher Structures, and through the NIST project. In addition, this work was supported by the European Commission through the Horizon 2020 research and innovation programme under grant agreement H2020-DS-2014-653497 PANORAMIX and through the grant H2020-DS-SC7-2016-740507 Eunity. The work is supported in part by funding

[10] Available at https://sites.google.com/site/skinnycipher/implementation

Implementation (round-based)	Area [GE]	$f_{\max}$ [MHz]	Nb. cycles for encrypting $(a+m)$ **64-bit** blocks						
			$a=0$			$a=1$			General
			$m=1$	$m=2$	$m=3$	$m=0$	$m=1$	$m=2$	
Sk-Aead M6	6288	1075	96	96	144	48	96	**96**	$48(\lceil \frac{a}{2} \rceil + \lceil \frac{m}{2} \rceil + 1)$
Paef-64-192	**4205**	**1265**	**63**	126	189	**40**	103	166	$40(a+1.575m)$
Paef-64-192 (//)	**4811**	**1265**	**40**	**80**	**120**	**40**	**80**	120	$40(a+m)$

Implementation (round-based)	Area [GE]	$f_{\max}$ [MHz]	Nb. cycles for encrypting $(a+m)$ **128-bit** blocks						
			$a=0$			$a=1$			General $(m \geq 1)$
			$m=1$	$m=2$	$m=3$	$m=0$	$m=1$	$m=2$	
Sk-Aead M5	6778	1075	96	144	192	96	144	192	$48(a+m+1)$
Paef-128-256	7189	1053	**75**	150	225	**48**	**123**	198	$48(a+1.562m)$
Paef-128-256 (//)	8023	1042	**48**	**96**	**144**	**48**	**96**	**144**	$\boldsymbol{48(a+m)}$
Saef-128-256 (//)	7064	1042	**48**	**96**	**144**	**48**	**96**	**144**	$\boldsymbol{48(a+m)}$
Rpaef (aggr.)	8203	1052	**87**	**135**	**183**	**48**	**135**	**183**	$\boldsymbol{48(a+m)+39}$
Sk-Aead M1-2	8210	1000	112	168	224	112	168	224	$56(a+m+1)$
Paef-128-288	**7989**	971	**87**	174	261	**56**	**143**	230	$56(a+1.553m)$
Paef-128-288 (//)	9308	962	**56**	**112**	**168**	**56**	**112**	**168**	$\boldsymbol{56(a+m)}$
Rpaef (cons.)	**8178**	**1052**	**87**	**143**	**199**	**56**	**143**	**199**	$\boldsymbol{56(a+m)+31}$

Fig. 10: Synthesis results and cycles for encrypting a blocks associated data and m blocks message. Superior performance w.r.t. the baseline (Sk-Aead [19]) is indicated **in bold**. The area is a partly synthesized and partly estimated. Rpaef (conservative) is RPAEF instantiated with ForkSkinny-128-384, and Rpaef (aggressive) is described in Section 6.8.

from imec of the Flemish Government. Antoon Purnal is supported by the Horizon 2020 research and innovation programme under Cathedral ERC Advanced Grant 695305. Reza Reyhanitabar's work on this project was initiated when he was with KU Leuven and supported by an EU H2020-MSCA-IF fellowship under grant ID 708815, continued and submitted when he was with Elektrobit Automotive GmbH, and revised while he is now with TE Connectivity. Arnab Roy is supported by the EPSRC grant No. EPSRC EP/N011635/1.

References

1. 3GPP TS 22.261: Service requirements for next generation new services and markets. `https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3107`
2. 3GPP TS 36.213: Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures. `https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2427`
3. CAN FD Standards and Recommendations. `https://www.can-cia.org/news/cia-in-action/view/can-fd-standards-and-recommendations/2016/9/30/`
4. ISO 11898-1:2015: Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling. `https://www.iso.org/standard/63648.html`

5. NB-IoT: Enabling New Business Opportunities. `http://www.huawei.com/minisite/iot/img/nb_iot_whitepaper_en.pdf`
6. Specification of Secure Onboard Communication. `https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_SecureOnboardCommunication.pdf`
7. Abdelkhalek, A., Sasaki, Y., Todo, Y., Tolba, M., Youssef, A.M.: MILP modeling for (large) s-boxes to optimize probability of differential characteristics. IACR Trans. Symmetric Cryptol. **2017**(4), 99–129 (2017)
8. Anderson, E., Beaver, C.L., Draelos, T., Schroeppel, R., Torgerson, M.: Manticore: Encryption with joint cipher-state authentication. In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) ACISP 2004. LNCS, vol. 3108, pp. 440–453. Springer (2004)
9. Andreeva, E., Bogdanov, A., Datta, N., Luykx, A., Mennink, B., Nandi, M., Tischhauser, E., Yasuda, K.: COLM v1 (2014), `"https://competitions.cr.yp.to/round3/colmv1.pdf"`
10. Andreeva, E., Lallemand, V., Purnal, A., Reyhanitabar, R., Roy, A., Vizar, D.: Forkcipher: a new primitive for authenticated encryption of very short messages. Cryptology ePrint Archive, Report 2019/1004 (2019), `https://eprint.iacr.org/2019/1004`
11. Andreeva, E., Neven, G., Preneel, B., Shrimpton, T.: Seven-Property-Preserving Iterated Hashing: ROX. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 130–146. Springer (2007)
12. Ankele, R., Banik, S., Chakraborti, A., List, E., Mendel, F., Sim, S.M., Wang, G.: Related-key impossible-differential attack on reduced-round skinny. In: Gollmann, D., Miyaji, A., Kikuchi, H. (eds.) ACNS 2017. LNCS, vol. 10355, pp. 208–228. Springer (2017)
13. Ankele, R., Kölbl, S.: Mind the gap - A closer look at the security of block ciphers against differential cryptanalysis. In: Cid, C., Jr., M.J.J. (eds.) SAC 2018. LNCS, vol. 11349, pp. 163–190. Springer (2018)
14. Aumasson, J.P., Babbage, S., Bernstein, D.J., Cid, C., Daemen, J., Gaj, O.D.K., Gueron, S., Junod, P., Langley, A., McGrew, D., Paterson, K., Preneel, B., Rechberger, C., Rijmen, V., Robshaw, M., Sarkar, P., Schaumont, P., Shamir, A., Verbauwhede, I.: CHAE: Challenges in Authenticated Encryption. ECRYPT-CSA D1.1, Revision 1.05, 1 March 2017
15. Avanzi, R.: Method and apparatus to encrypt plaintext data. US patent 9294266B2 (2013), `https://patents.google.com/patent/US9294266B2/`
16. Banik, S., Bogdanov, A., Isobe, T., Shibutani, K., Hiwatari, H., Akishita, T., Regazzoni, F.: Midori: A block cipher for low energy. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015, Part II. LNCS, vol. 9453, pp. 411–436. Springer (2015)
17. Banik, S., Bossert, J., Jana, A., List, E., Lucks, S., Meier, W., Rahman, M., Saha, D., Sasaki, Y.: Cryptanalysis of forkaes. Cryptology ePrint Archive, Report 2019/289 (2019), `https://eprint.iacr.org/2019/289`
18. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The skinny family of block ciphers and its low-latency variant mantis. In: CRYPTO 2016. pp. 123–153. Springer-Verlag, Berlin, Heidelberg (2016)
19. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: Skinny-aead and skinny-hash. NIST LWC Candidate. (2019)
20. Bellare, M.: Practice-oriented provable-security. In: Okamoto, E., Davida, G.I., Mambo, M. (eds.) ISW '97, Tatsunokuchi. LNCS, vol. 1396, pp. 221–231. Springer (1998)

21. Bellare, M., Goldreich, O., Mityagin, A.: The Power of Verification Queries in Message Authentication and Authenticated Encryption. IACR Cryptology ePrint Archive **2004**, 309 (2004)
22. Bellare, M., Kohno, T., Namprempre, C.: Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the encode-then-encrypt-and-mac paradigm. ACM Trans. Inf. Syst. Secur. **7**(2), 206–241 (2004)
23. Bellare, M., Namprempre, C.: Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 531–545. Springer (2000)
24. Bellare, M., Ristenpart, T.: Multi-property-preserving hash domain extension and the EMD transform. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 299–314. Springer (2006)
25. Bellare, M., Rogaway, P.: Encode-Then-Encipher Encryption: How to Exploit Nonces or Redundancy in Plaintexts for Efficient Cryptography. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 317–330. Springer (2000)
26. Bernstein, D.J.: Cryptographic competitions: CAESAR. `http://competitions.cr.yp.to`
27. Bertoni, G., Daemen, J., Hoffert, S., Peeters, M., Van Assche, G., Van Keer, R.: Farfalle: parallel permutation-based cryptography. IACR Transactions on Symmetric Cryptology **2017** (2017), `https://tosc.iacr.org/index.php/ToSC/article/view/855`
28. Biham, E., Shamir, A.: Differential cryptanalysis of des-like cryptosystems. J. Cryptology **4**(1), 3–72 (1991)
29. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçin, T.: PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 208–225. Springer (2012)
30. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: ASCON v1.2 (2014), `"https://competitions.cr.yp.to/round3/asconv12.pdf"`
31. Hoang, V.T., Krovetz, T., Rogaway, P.: Robust Authenticated-Encryption AEZ and the Problem That It Solves. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 15–44. Springer (2015)
32. Jean, J., Nikolić, I., Peyrin, T., Seurin, Y.: Deoxys v1.41 v1 (2016), `"https://competitions.cr.yp.to/round3/deoxysv141.pdf"`
33. Jean, J., Nikolic, I., Peyrin, T.: Tweaks and Keys for Block Ciphers: The TWEAKEY Framework. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 274–288. Springer (2014)
34. Katz, J., Yung, M.: Unforgeable encryption and chosen ciphertext secure modes of operation. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 284–299. Springer (2001)
35. Kranz, T., Leander, G., Wiemer, F.: Linear cryptanalysis: Key schedules and tweakable block ciphers. IACR Trans. Symmetric Cryptol. **2017**(1), 474–505 (2017)
36. Krovetz, T., Rogaway, P.: OCB v1.1 (2014), `"https://competitions.cr.yp.to/round3/ocbv11.pdf"`
37. Krovetz, T., Rogaway, P.: The Software Performance of Authenticated-Encryption Modes. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 306–327. Springer (2011)
38. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. In: CRYPTO 2002. pp. 31–46 (2002)
39. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT '93. LNCS, vol. 765, pp. 386–397. Springer (1993)

40. McGrew, D.A., Viega, J.: The security and performance of the galois/counter mode (GCM) of operation. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT 2004. LNCS, vol. 3348, pp. 343–355. Springer (2004)
41. Namprempre, C., Rogaway, P., Shrimpton, T.: Reconsidering Generic Composition. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 257–274. Springer (2014)
42. NIST: DRAFT Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process. `https://csrc.nist.gov/Projects/Lightweight-Cryptography` (2018)
43. Paterson, K.G., Yau, A.K.L.: Cryptography in theory and practice: The case of encryption in ipsec. IACR Cryptology ePrint Archive **2005**, 416 (2005), `http://eprint.iacr.org/2005/416`
44. Paterson, K.G., Yau, A.K.L.: Cryptography in Theory and Practice: The Case of Encryption in IPsec. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 12–29. Springer (2006)
45. Reyhanitabar, M.R., Susilo, W., Mu, Y.: Analysis of property-preservation capabilities of the ROX and esh hash domain extenders. In: Boyd, C., Nieto, J.M.G. (eds.) ACISP 2009. LNCS, vol. 5594, pp. 153–170. Springer (2009)
46. Rogaway, P.: Authenticated-Encryption with Associated-Data. In: ACM CCS 2002. pp. 98–107 (2002)
47. Rogaway, P.: Practice-oriented provable security and the social construction of cryptography. IEEE Security & Privacy **14**(6), 10–17 (2016)
48. Rogaway, P., Shrimpton, T.: A Provable-Security Treatment of the Key-Wrap Problem. In: EUROCRYPT 2006. pp. 373–390 (2006)
49. Sadeghi, S., Mohammadi, T., Bagheri, N.: Cryptanalysis of reduced round SKINNY block cipher. IACR Trans. Symmetric Cryptol. **2018**(3), 124–162 (2018)
50. Sui, H., Wu, W., Zhang, L., Wang, P.: Attacking and fixing the CS mode. In: Qing, S., Zhou, J., Liu, D. (eds.) ICICS 2013. LNCS, vol. 8233, pp. 318–330. Springer (2013)
51. Tolba, M., Abdelkhalek, A., Youssef, A.M.: Impossible differential cryptanalysis of reduced-round SKINNY. In: Joye, M., Nitaj, A. (eds.) AFRICACRYPT 2017. LNCS, vol. 10239, pp. 117–134 (2017)
52. Whiting, D., Housley, R., Ferguson, N.: Counter with CBC-MAC (CCM). IETF RFC 3610 (Informational) (Sep 2003), `http://www.ietf.org/rfc/rfc3610.txt`
53. Wu, H.: ACORN v3 (2014), `"https://competitions.cr.yp.to/round3/acornv3.pdf"`
54. Wu, H., Huang, T.: MORUS v2 (2014), `"https://competitions.cr.yp.to/round3/morusv2.pdf"`
55. Wu, H., Preneel, B.: AEGIS v1.1 (2014), `"https://competitions.cr.yp.to/round3/aegisv11.pdf"`
56. Zhang, P., Zhang, W.: Differential cryptanalysis on block cipher skinny with MILP program. Security and Communication Networks **2018**, 3780407:1–3780407:11 (2018)
57. Zhang, W., Rijmen, V.: Division cryptanalysis of block ciphers with a binary diffusion layer. IET Information Security **13**(2), 87–95 (2019)