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

Elena Andreeva, Virginie Lallemand, Antoon Purnal, Reza Reyhanitabar, Arnab Roy, Damian Vizár

▶ To cite this version:

Elena Andreeva, Virginie Lallemand, Antoon Purnal, Reza Reyhanitabar, Arnab Roy, et al.. Forkcipher: A New Primitive for Authenticated Encryption of Very Short Messages. ASIACRYPT 2019 - 25th Annual International Conference on the Theory and Application of Cryptology and Information Security, Dec 2019, Kobe, Japan. pp.153-182, 10.1007/978-3-030-34621-8_6 . hal-02388234

HAL Id: hal-02388234
https://hal.inria.fr/hal-02388234

Submitted on 30 Dec 2020

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

Extended version*

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, F-54000 Nancy, 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 a *forkcipher*. A forkcipher is a keyed function expanding a fixed-length input to a fixed-length output. We define its security as indistinguishability under chosen ciphertext attack. 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 block cipher constructed using 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 the SKINNY block cipher.

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) [22]. 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

* This is an extended version of the article with the same title accepted at Asiacrypt 2019.

the most robust to security and implementation issues [21,50,51]; 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 [48].
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 [24] and independently by Katz and Yung [38] in 2000. It was further enhanced by Rogaway [53] 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 [31], ACORN [64], AEGIS-128 [66], OCB [41], COLM [9], Deoxys II [35], and MORUS [65]).
This progress may lead to the belief that the AEAD problem is "solved". However, as evidenced by the ECRYPT-CSA report in 2017 [13], 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 [13], 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 ··· Another increasingly common scenario is that an authenticated cipher is applied to many small messages ··· 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 [49], 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 [10, 23, 52].

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) [32,55]. 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 [17]. Building ForkSkinny on an existing block cipher enables us to rely on the cryptanalyses result behind SKINNY [11,12,56,61,67,68], 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 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 [16] into account to ensure that the same type of attacks is not possible against ForkSkinny.

[6] See Section 8 for a brief discussion.

[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.

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 [19, 54]. 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 [46], CCM [63], and OCB [42], 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 [43], which is same as ΘCB [42], instantiated with SKINNY-AEAD M1/M2, M5/M6 [18]. 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 [49], 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 [59] (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 [14], 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 [37]) 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], [14]) 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 $X = X_1 \| \ldots \| X_x \| X_*$, 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 $C'_1, \ldots, C'_m, C_*, T \leftarrow \mathsf{csplit\text{-}b}_n(C)$ (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 $C' = C'_1 \| \ldots \| C'_m$.

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.

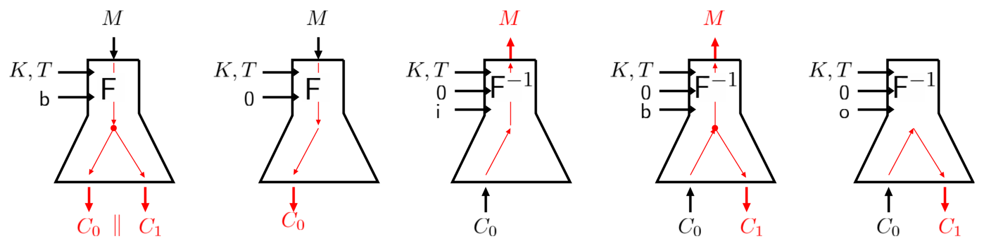

Fig. 1: Forkcipher encryption (two leftmost): the output selector s outputs **b**oth output blocks C_0, C_1 if $s = \mathsf{b}$, the "left" ciphertext block C_0 if $s = 0$ (if $s = \mathsf{b}$ then C_1). Forkcipher decryption (three rightmost): the first indicator $b = 0$ denotes the left ciphertext 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

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

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.

The formalization we just gave faithfully models how a forkcipher is used to realize its full potential. As explained in Section 8, 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 prove 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 Appendix A[9]. 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 $\text{Perm}(n)$ 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 $\mathsf{F}^{-1\mathsf{T},b,\mathsf{b}}(C) = p_{\mathsf{T}}^{-1}(p^{-1}_{\mathsf{T},b}(C)), p_{\mathsf{T},b\oplus 1}(p^{-1}_{\mathsf{T},b}(C))$ (the rest follows from the correctness). We note that the three tweakable random permutations act as a key for $\text{IFI}[p, p_0, p_1]$ and we omit them for the sake of simplicity.

In Appendix A, Theorem 1 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 ForkSkinny forkcipher using the recently published lightweight tweakable block cipher SKINNY [17]. 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 $\mathsf{ForkSkinny}_b^{-1}$.

[9] We refer to all the materials in Appendix provided as supplementary material to the submission.

4.1 Specification

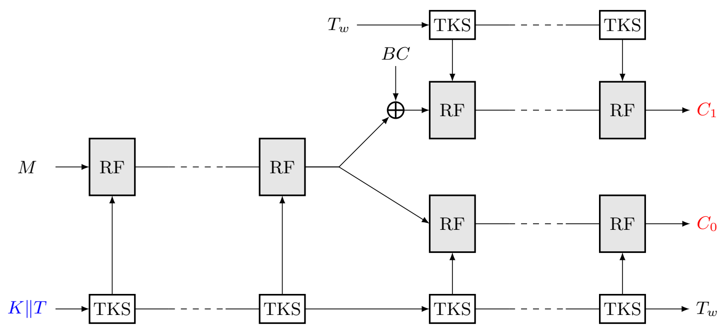

Fig. 3: ForkSkinny encryption with selector $s = \mathsf{b}$. A plaintext M, a key K and a tweak T (blue) are used to compute a ciphertext $C = C_0\|C_1$ (red) of twice the size of the plaintext. RF is a single round function of SKINNY and TKS is round tweakey update function [17]. and BC is a branch 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_{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.

```
1: function ForkSkinnyEnc($M, K, T, s$)
2:     $tk \leftarrow K\|T$
3:     $L \leftarrow \mathsf{Enc\text{-}Skinny}_{r_{\mathsf{init}}}(M, tk)$
4:     if $s = 0$ or $s = \mathsf{b}$ then
5:         $C_0 \leftarrow \mathsf{Enc\text{-}Skinny}_{r_0}(L, \mathsf{TKS}_{r_{\mathsf{init}}}(tk))$
6:     end if
7:     if $s = 1$ or $s = \mathsf{b}$ then
8:         $tk' \leftarrow \mathsf{TKS}_{r_{\mathsf{init}}+r_0}(tk)$
9:         $C_1 \leftarrow \mathsf{Enc\text{-}Skinny}_{r_1}(L \oplus BC, tk')$
10:    end if
11:    if $s = 0$ return $C_0$
12:    if $s = 1$ return $C_1$
13:    if $s = \mathsf{b}$ return $C_0, C_1$
14: end function
```

```
1: function ForkSkinnyDec($C, K, T, b, s$)
2:     $tk \leftarrow K\|T$
3:     $tk' \leftarrow \mathsf{TKS}_{r_{\mathsf{init}}}(tk)$
4:     if $b = 0$ then
5:         $L \leftarrow \mathsf{Dec\text{-}Skinny}_{r_0}(C, tk')$
6:     else if $b = 1$ then
7:         $tk'' \leftarrow \mathsf{TKS}_{r_0}(tk')$
8:         $L \leftarrow \mathsf{Dec\text{-}Skinny}_{r_1}(C_b, tk'') \oplus BC$
9:     end if
10:    if $s = \mathsf{i}$ or $s = \mathsf{b}$ then
11:        $M \leftarrow \mathsf{Dec\text{-}Skinny}_{r_{\mathsf{init}}}(L, tk)$
12:    end if
13:    if $s = \mathsf{o}$ or $s = \mathsf{b}$ then
14:        if $b = 0$ then $tk' \leftarrow \mathsf{TKS}_{r_0}(tk')$
15:        $C' \leftarrow \mathsf{Enc\text{-}Skinny}_{r_{b\oplus 1}}(L, tk')$
16:    end if
17:    if $s = \mathsf{i}$ return $M$
18:    if $s = \mathsf{o}$ return $C'$
19:    if $s = \mathsf{b}$ return $M, C'$
20: 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 tweakeyschedule are also left unchanged, but more tweakeys than in `SKINNY` are produced given that we have $r_{\mathsf{init}} + r_0 + r_1$ rounds. To keep the paper short, we leave the details of these operations to Appendix F.

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 in order to avoid that the same round constant is added 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:

$$(rc_6||rc_5||\ldots||rc_0) \rightarrow (rc_5||rc_4||\ldots||rc_0||rc_6 \oplus rc_5 \oplus 1)$$

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 $c_2 = \texttt{0x2}$ 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:

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

This addition is made right after forking, to the right branch leading to C_1. To save memory and since there are no constraints on the values used these constants are generated by clocking LFSRs, given by: $(x_3||x_2||x_1||x_0) \rightarrow (x_2||x_1||x_0||x_3 \oplus x_2)$, and initialised with $x_0 = 1$, $x_1 = x_2 = x_3 = 0$ for BC_4, and with the LFSR $(x_7||x_6||x_5||x_4||x_3||x_2||x_1||x_0) \rightarrow (x_6||x_5||x_4||x_3||x_2||x_1||x_0||x_7 \oplus x_5)$, 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 [11,12,56,61,67,68] have further been published on round reduced `SKINNY` showing that the full version of the cipher has comfortable security margins. Unlike other lightweight block ciphers such as Midori [15] and

PRINCE [30], 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 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 ones 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[10] of SKINNY imply that ForkSkinny appears out of reach of the attacks we considered.

Our full security analysis is detailed in Appendix G. 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 [16], 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 [16] against (9 out of 10 rounds) ForkAES in fact uses the property that 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 [29] and linear [45] 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 Table 6 in

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

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.

Appendix G). 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 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 [40] 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 [53]. 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 Appendix H)

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 $\text{PAEF}[\mathsf{F},\nu] = (\mathcal{K},\mathcal{E},\mathcal{D})$ 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 $\text{PAEF}[\mathsf{F},\nu]$ 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*

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

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 Appendix B. For both confidentiality and authenticity, we first replace F with a pair of independent random tweakable permutations π_0, π_1, i.e. $\pi_0 = (\pi_{\mathsf{T},0} \leftarrow\$ \text{Perm}(n))_{\mathsf{T}\in\{0,1\}^t}$ is a collection of independent uniform elements of $\text{Perm}(n)$ indexed by the elements of $\mathsf{T} \in \{0,1\}^t$ (and similarly $\pi_1 = (\pi_{\mathsf{T},1} \leftarrow\$ \text{Perm}(n))_{\mathsf{T}\in\{0,1\}^t}$). We let

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

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

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

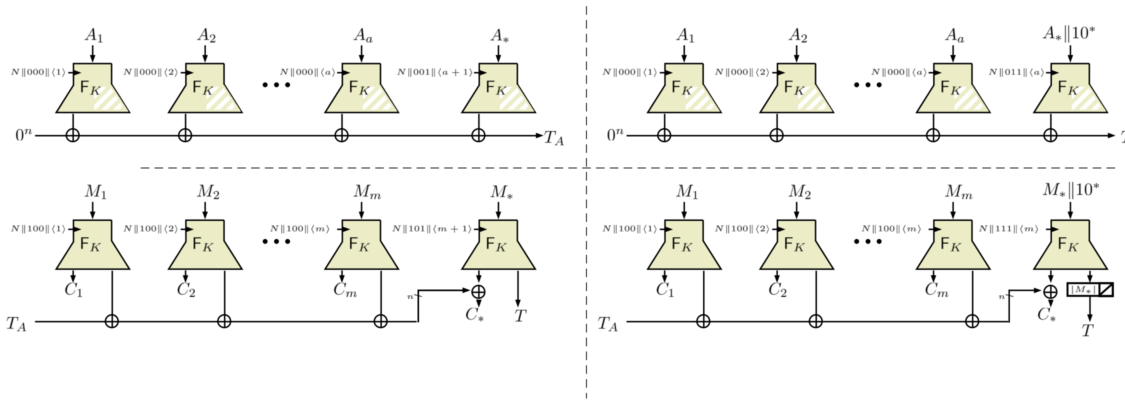

Fig. 6: The encryption algorithm of PAEF[F] mode. The picture illustrates the processing of AD when length of AD is a multiple of n (**top left**) and when the length of AD is not a multiple of n (**top right**), and the processing of the message when length of the message is a multiple of n (**bottom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

PAEF$[(\pi_0,\pi_1),\nu]$ denote the PAEF mode that uses π_0,π_1 instead of F. Using a standard hybrid argument we have that $\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{PAEF}[\mathsf{F},\nu]}(\mathcal{A}) \le \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{B}) + \mathbf{Adv}^{\mathbf{priv}}_{\mathrm{PAEF}[(\pi_0,\pi_1),\nu]}(\mathcal{A})$, and also that $\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{PAEF}[\mathsf{F},\nu]}(\mathcal{A}) \le \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{C}) + \mathbf{Adv}^{\mathbf{priv}}_{\mathrm{PAEF}[(\pi_0,\pi_1),\nu]}(\mathcal{A})$.

For confidentiality, it is easy to see that in a nonce-respecting attack, every ciphertext block, and all tags are processed using a unique tweak-permutation combination, and thus 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 PAEF$[(\pi_0,\pi_1),\nu]$ and then use a result of Bellare et al. [20] to obtain $\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{PAEF}[(\pi_0,\pi_1),\nu]}(\mathcal{A}) \le q_v \cdot \mathbf{Adv}^{\mathbf{auth}}_{\mathrm{PAEF}[(\pi_0,\pi_1),\nu]}(\mathcal{A}')$.

We analyze $\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{PAEF}[(\pi_0,\pi_1),\nu]}(\mathcal{A}')$ by a case analysis of all $\mathcal{A}'$ possible forgery attempts. The forgery succeeds with the highest probability $2^n/(2^n-1)^2$ if the tweak used to process the final block and the tag has been used before (the decryption query has the same nonce, same number of blocks etc. as some previous enc. query) and the final message block is incomplete (this corresponds to guessing n out of $2n$ bits of two images under two random permutations for which there was exactly one other image sampled before). Thus, $\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{PAEF}[(\pi_0,\pi_1),\nu]}(\mathcal{A}') \le 2^n/(2^n-1)^2$.

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 $t \le n$. An instance SAEF[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 SAEF[F] is n bits. The encryption and decryption algorithms are defined in Figure 8 and the encryption algorithm is illustrated in Figure 7.

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 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

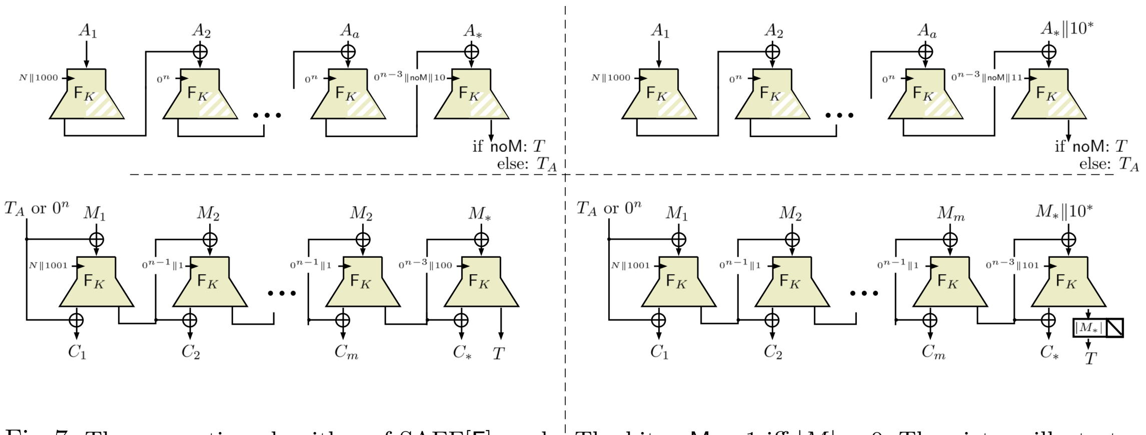

Fig. 7: The encryption algorithm of SAEF[F] mode. The bit $\mathsf{noM} = 1$ iff $|M| = 0$. The picture illustrates the processing of AD when length of AD is a multiple of n (**top left**) and when the length of AD is not a multiple of n (**top right**), and the processing of the message when length 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.

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*

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

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 Appendix C. As with PAEF, we first replace F with a pair of independent random tweakable permutations $\pi_0 = (\pi_{\mathsf{T},0} \leftarrow\$ \operatorname{Perm}(n))_{\mathsf{T}\in\{0,1\}^t}$ and $\pi_1 = (\pi_{\mathsf{T},1} \leftarrow\$ \operatorname{Perm}(n))_{\mathsf{T}\in\{0,1\}^t}$. We let SAEF$[(\pi_0, \pi_1), \nu]$ denote the SAEF mode that uses π_0, π_1 instead of F.

For confidentiality, we further replace tweakable permutations π_0, π_1 by random "tweakable" functions f_1, f_0, 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 in the tweakable permutations π_0 and π_0 by a 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 input collision occurs in any of the encryption queries (i.e., the same input is fed to the same tweakable permutation/function when processing two different blocks). 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

```
function $\mathcal{E}(K, N, A, M)$
    $A_1, \ldots, A_a, A_* \xleftarrow{n} A$
    $M_1, \ldots, M_m, M_* \xleftarrow{n} M$
    $\mathsf{noM} \leftarrow 0$
    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        $\triangleright$ 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 $C_1 \| \ldots \| C_m \| C_* \| \mathsf{left}_{|M_*|}(T)$
end function
```

```
function $\mathcal{D}(K, N, A, C)$
    $A_1, \ldots, A_a, A_* \xleftarrow{n} A$
    $C_1, \ldots, C_m, C_*, T \leftarrow \mathsf{csplit\text{-}b}_n C$
    $\mathsf{noM} \leftarrow 0$
    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        $\triangleright$ 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 $M_1 \| \ldots \| M_m \| \mathsf{left}_{|T|}(M_*)$
end function
```

Fig. 8: The SAEF[F] AEAD scheme.

blocks. This allows for *reducing* the computational cost if the unevaluated fork can be switched off (as in $\mathsf{ForkSkinny}$) 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 $\text{RPAEF}[\mathsf{F}, \nu] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ 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 $\text{PAEF}[\mathsf{F}, \nu]$ 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".

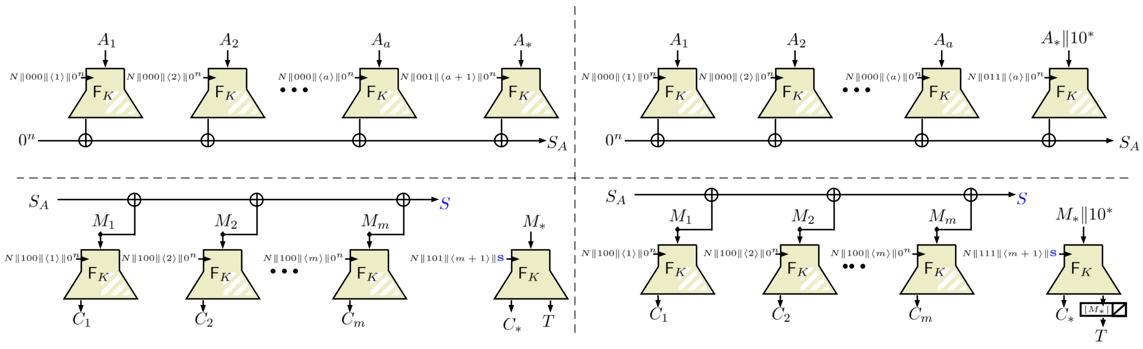
A_1
A_2
A_a
A_*
$N \| 000 \| \langle 1 \rangle \| 0^n$
$N \| 000 \| \langle 2 \rangle \| 0^n$
$N \| 000 \| \langle a \rangle \| 0^n$
$N \| 001 \| \langle a+1 \rangle \| 0^n$
F_K
0^n
S_A
$A_* \| 10^*$
$N \| 011 \| \langle a \rangle \| 0^n$
S
M_1
M_2
M_m
M_*
$N \| 100 \| \langle 1 \rangle \| 0^n$
$N \|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

Fig. 9: The encryption algorithm of $\text{RPAEF}[\mathsf{F}]$ 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*

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

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 Appendix D. For both confidentiality and authenticity, we first replace F with a pair of independent random tweakable permutations π_0, π_1, 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 processed using a unique tweak-permutation combination, and thus are uniformly distributed. We have $\mathbf{Adv}^{\mathbf{priv}}_{\text{PAEF}[(\pi_0,\pi_1),\nu]}(\mathcal{A}) = 0$.

For authenticity, we combine a case analysis and the same result of Bellare et al. [20] as used for PAEF to obtain the bound. The largest probability of forgery occurs in a case when the final tweak reuses the nonce, flags and the counter from a previous query, and the adversary may forge both by a collision on the checksum (with probability $1/(2^n - 1)$), and guessing the tag otherwise (with probability 2^{-n}). This is bounded by $2/(2^n - 1)$.

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 Appendix E). 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 [55].

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 [18] as the most fairly comparable TBC mode of operation based on SKINNY.

Hardware synthesis. To stimulate future comparison, the hardware synthesis results (ASIC) are obtained with the open source cell library NANGATE45NM in typical operating conditions. All designs face an identical synthesis flow and are synthesized with `Synopsys Design Compiler 2017.N3`, using `compile`. We allow the use of Scan flip-flops (FF). All designs are governed by the exact same assumptions: the key and plaintext are assumed to be available from a 128-bit bus at the interface and are stored directly in the computational registers, while the nonce is stored at the level of the mode. While this interface can be optimized for a target usecase, note that changes and optimizations to this interface affect all the compared designs identically.

Implementations. For SKINNY-AEAD, we use the publicly available SKINNY round-based encryption implementations[11]. 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. Figure 10 presents hardware synthesis results (ASIC) and cycle counts for encrypting a blocks of associated data and m blocks of message for the AEAD schemes under consideration. 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): $A_{total} = A_{prim} + A_{mode}$. As the area overhead associated with the modes of operation is largely dominated by the storage elements of the mode, it can be estimated in first order by counting 7.67 GE per bit of storage (size of a Scan FF in NANGATE45NM). This estimate

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

includes all state that is required to compute ciphertext and tag, as well as storage for the nonce (if applicable) and the block counter (if applicable). As an example, the number of storage bits for Paef-128-256 is $128+96+29 = 253$ bits and for Skinny-AEAD-M5 is $256+96+24 = 376$ bits.

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 [49], 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).

Implementation (round-based)	Area [GE] $A = A_{prim} + A_{mode}$	$f_{\max}$ [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	$6288 = 3895 + 2393$	1075	96	96	144	48	96	**96**	$48(\lceil\frac{a}{2}\rceil+\lceil\frac{m}{2}\rceil+1)$
Paef-64-192	**4205** $= 3246 + 959$	**1265**	**63**	126	189	**40**	103	166	$40(a+1.575m)$
Paef-64-192 (//)	**4811** $= 3852 + 959$	**1265**	**40**	**80**	**120**	**40**	**80**	120	$40(a+m)$

Implementation (round-based)	Area [GE] $A = A_{prim} + A_{mode}$	$f_{\max}$ [MHz]	Nb. cycles for encrypting $(a+m)$ **128-bit** blocks						
			$a=0$			$a=1$			General $(m\geq1)$
			$m=1$	$m=2$	$m=3$	$m=0$	$m=1$	$m=2$	
Sk-Aead m5	$6778 = 3895 + 2883$	1075	96	144	192	96	144	192	$48(a+m+1)$
Paef-128-256	$7189 = 5248 + 1941$	1053	**75**	150	225	**48**	**123**	198	$48(a+1.562m)$
Paef-128-256 (//)	$8023 = 6082 + 1941$	1042	**48**	**96**	**144**	**48**	**96**	**144**	$\boldsymbol{48(a+m)}$
Saef-128-256 (//)	$7064 = 6082 + 982$	1042	**48**	**96**	**144**	**48**	**96**	**144**	$\boldsymbol{48(a+m)}$
Rpaef (aggr.)	$8203 = 7244 + 959$	1052	**87**	**135**	**183**	**48**	**135**	**183**	$\boldsymbol{48(a+m)+39}$
Sk-Aead m1-2	$8210 = 5020 + 3190$	1000	112	168	224	112	168	224	$56(a+m+1)$
Paef-128-288	**7989** $= 5803 + 2186$	971	**87**	174	261	**56**	**143**	230	$56(a+1.553m)$
Paef-128-288 (//)	$9308 = 7122 + 2186$	962	**56**	**112**	**168**	**56**	**112**	**168**	$\boldsymbol{56(a+m)}$
Rpaef (cons.)	**8178** $= 7219 + 959$	**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 [18]) 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.

8 Discussion and open questions

We presented three modes of operation for our novel tweakable forkcipher primitive ForkSkinny. Each of the three modes reduces to a single call to the used forkcipher F in the case that the input only consists of a single block of data (either AD or message, but not both). This, together with an appropriate instantiation of F yields concrete AEAD schemes that excel in short-message encryption and decryption performance. Apart from its practical ramifications, the theory in this work sets the field of exploration of the novel forkcipher structure which by itself is an important contribution in symmetric cryptography.

Starting from a FIL PRI. As mentioned in Section 1, we also considered designing the VIL AEAD schemes as modes of operation of a FIL (tweakable) PRI, but we encountered a setbacks when heading in this direction. The technique we use to process the final message block in PAEF, SAEF and RPAEF requires the parallel-permutation structure of a forkcipher. When our finalization gets replaced with a n-to-$2n$ bit PRI, an encryption of a single-bit message would result in a ciphertext with $2n - 1$ bits of expansion; an attempt to truncate the ciphertext would render decryption impossible. Thus, we were unable to design an AE mode of a FIL PRI, that would simultaneously use a single primitive call for the shortest queries and have a *constant* stretch. Whether such a mode of operation exists is another interesting open question.

Constructing a FIL PRI. The second question is if we can find better instances of a FIL PRI. Our preferred choice was lightweight-based and thus the ForkSkinny. Our concrete idea to build a forkcipher was founded in the novel iterate-fork-iterate approach, yet we have not investigated other advantageous generic design paradigms. Also, as evidenced by the result in Section 6.9, there is an unavoidable birthday-type quantitative gap between the PRI security, and the kind of security that ForkSkinny inherently possesses. A direct instance of a *true* FIL tweakable PRI is another question we leave open.

Iterate-multifork-iterate. Another, interesting research direction would be to generalize the IFI paradigm to fork into μ branches (IFI[μ]). This direction would be interesting from the cryptanalytic point of view (to see how security degrades with an increasing number of branches), as well as from the application point of view (asymptotic reduction of PRP cost to $\approx 1/2$).

Novel Forkcipher Instantiations For very resource constrained IoT devices in which our ForkSkinny-based instances could not be considered a fitting lightweight option, our SAEF, PAEF and RPAEF can be further instantiated using a forkcipher based on any off-the-shelf lightweight blockcipher. The crux would be a careful realization of the tweakable forkcipher (possibly following the IFI framework).

Beyond AEAD. This work opens a new design space in symmetric cryptography and is naturally accompanied with plenty of open research questions. Possible direction is the research on the broader (than AEAD) application space for forkciphers and cnstructions in those domains. Forkcipher is undoubtedly an interesting primitive beyond AEAD applications and provides a number of examples for potential applications which can offer both efficiency and security optimizations compared to classical symmetric primitive structures: 1. GF multiplication-less BBB secure MACs; 2. design of streamcipher-like primitives (a generalized forkcipher with B branches produces a key stream in $(B + 1)/2$ BC calls); 3. BBB secure PRFs; 4. a generalized forkcipher with B branches and $B-1$ near-uniform blocks computing in $(B+1)/2$ BC calls following the CENC framework [33]).

Ackowledgement 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 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., 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)
11. 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)
12. 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)
13. 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
14. Avanzi, R.: Method and apparatus to encrypt plaintext data. US patent 9294266B2 (2013), `https://patents.google.com/patent/US9294266B2/`
15. 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)
16. 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`
17. 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)
18. 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)
19. 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)
20. Bellare, M., Goldreich, O., Mityagin, A.: The Power of Verification Queries in Message Authentication and Authenticated Encryption. IACR Cryptology ePrint Archive **2004**, 309 (2004)
21. 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)
22. 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)
23. 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)
24. 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)
25. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 409–426. Springer (2006)
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., Biryukov, A., Shamir, A.: Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. In: Stern, J. (ed.) EUROCRYPT '99. LNCS, vol. 1592, pp. 12–23. Springer (1999)
29. Biham, E., Shamir, A.: Differential cryptanalysis of des-like cryptosystems. J. Cryptology **4**(1), 3–72 (1991)

30. 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)
31. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: ASCON v1.2 (2014), `"https://competitions.cr.yp.to/round3/asconv12.pdf"`
32. 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)
33. Iwata, T.: New Blockcipher Modes of Operation with Beyond the Birthday Bound Security. In: Robshaw, M.J.B. (ed.) FSE 2006. LNCS, vol. 4047, pp. 310–327. Springer (2006)
34. Jean, J., Nikolić, I., Peyrin, T.: KIASU v1 (2014), `"https://competitions.cr.yp.to/round1/kiasuv1.pdf"`
35. Jean, J., Nikolić, I., Peyrin, T., Seurin, Y.: Deoxys v1.41 v1 (2016), `"https://competitions.cr.yp.to/round3/deoxysv141.pdf"`
36. Jean, J.: TikZ for Cryptographers. `https://www.iacr.org/authors/tikz/` (2016)
37. 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)
38. 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)
39. Knudsen, L.: Deal-a 128-bit block cipher. complexity **258**(2), 216 (1998)
40. Kranz, T., Leander, G., Wiemer, F.: Linear cryptanalysis: Key schedules and tweakable block ciphers. IACR Trans. Symmetric Cryptol. **2017**(1), 474–505 (2017)
41. Krovetz, T., Rogaway, P.: OCB v1.1 (2014), `"https://competitions.cr.yp.to/round3/ocbv11.pdf"`
42. 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)
43. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. In: CRYPTO 2002. pp. 31–46 (2002)
44. Liu, G., Ghosh, M., Song, L.: Security analysis of SKINNY under related-tweakey settings (long paper). IACR Trans. Symmetric Cryptol. **2017**(3), 37–72 (2017)
45. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT '93. LNCS, vol. 765, pp. 386–397. Springer (1993)
46. 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)
47. Moghaddam, A.E., Ahmadian, Z.: New automatic search method for truncated-differential characteristics: Application to midori and skinny. Cryptology ePrint Archive, Report 2019/126 (2019), `https://eprint.iacr.org/2019/126`
48. 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)
49. NIST: DRAFT Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process. `https://csrc.nist.gov/Projects/Lightweight-Cryptography` (2018)
50. 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`
51. 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)
52. 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)
53. Rogaway, P.: Authenticated-Encryption with Associated-Data. In: ACM CCS 2002. pp. 98–107 (2002)
54. Rogaway, P.: Practice-oriented provable security and the social construction of cryptography. IEEE Security & Privacy **14**(6), 10–17 (2016)
55. Rogaway, P., Shrimpton, T.: A Provable-Security Treatment of the Key-Wrap Problem. In: EUROCRYPT 2006. pp. 373–390 (2006)
56. Sadeghi, S., Mohammadi, T., Bagheri, N.: Cryptanalysis of reduced round SKINNY block cipher. IACR Trans. Symmetric Cryptol. **2018**(3), 124–162 (2018)
57. Shi, D., Sun, S., Derbez, P., Todo, Y., Sun, B., Hu, L.: Programming the demirci-selçuk meet-in-the-middle attack with constraints. In: Peyrin, T., Galbraith, S.D. (eds.) ASIACRYPT 2018. LNCS, vol. 11273, pp. 3–34. Springer (2018)
58. Song, L., Qin, X., Hu, L.: Boomerang connectivity table revisited. Cryptology ePrint Archive, Report 2019/146 (2019), `https://eprint.iacr.org/2019/146`
59. 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)

60. Todo, Y.: Structural evaluation by generalized integral property. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 287–314. Springer (2015)
61. 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)
62. Wagner, D.A.: The boomerang attack. In: Knudsen, L.R. (ed.) FSE '99. LNCS, vol. 1636, pp. 156–170. Springer (1999)
63. Whiting, D., Housley, R., Ferguson, N.: Counter with CBC-MAC (CCM). IETF RFC 3610 (Informational) (Sep 2003), `http://www.ietf.org/rfc/rfc3610.txt`
64. Wu, H.: ACORN v3 (2014), `"https://competitions.cr.yp.to/round3/acornv3.pdf"`
65. Wu, H., Huang, T.: MORUS v2 (2014), `"https://competitions.cr.yp.to/round3/morusv2.pdf"`
66. Wu, H., Preneel, B.: AEGIS v1.1 (2014), `"https://competitions.cr.yp.to/round3/aegisv11.pdf"`
67. Zhang, P., Zhang, W.: Differential cryptanalysis on block cipher skinny with MILP program. Security and Communication Networks **2018**, 3780407:1–3780407:11 (2018)
68. Zhang, W., Rijmen, V.: Division cryptanalysis of block ciphers with a binary diffusion layer. IET Information Security **13**(2), 87–95 (2019)

A Generic Validation of the Iterate-Fork-Iterate Construction

Theorem 4. *Fix a blocklength n and a tweaklength t. Then for any adversary $\mathcal{A}$ that makes at most q queries we have that*

$$\mathbf{Adv}^{\mathbf{prtfp}}_{\mathrm{IFI}[p,p_0,p_1]}(\mathcal{A}) = 0$$

where p, p_0, p_1 are random tweakable permutations with n-bit blocks and t-bit tweaks.

Proof. We use the game Γ from Figure 11 to show that the IFI construction used with a triple of random tweakable permutations yields a perfect forkcipher. For any partially defined permutation π, we let $\mathcal{D}(\pi) \subseteq \{0,1\}^n$ denote those domain points with a defined image, and we let $\mathcal{R}(\pi) \subseteq \{0,1\}^n$ denote those range points with a defined preimage. For simplicity, we will denote $\mathrm{IFI}[p, p_0, p_1]$ as F.

We first prove by an induction over adversary's queries that at any point during the execution of game Γ, and for any $\mathsf{T} \in \{0,1\}^t$ the following properties hold:

1. $\mathcal{D}(\pi_{\mathsf{T},0}) = \mathcal{D}(\pi_{\mathsf{T},1}) = \mathcal{D}(p_{\mathsf{T}})$,
2. $\mathcal{R}(p_{\mathsf{T}}) = \mathcal{D}(p_{\mathsf{T},0}) = \mathcal{D}(p_{\mathsf{T},1})$,
3. $\mathcal{R}(\pi_{\mathsf{T},0}) = \mathcal{R}(p_{\mathsf{T},0})$,
4. $\mathcal{R}(\pi_{\mathsf{T},1}) = \mathcal{R}(p_{\mathsf{T},1})$,
5. $p_{\mathsf{T},0}(p_{\mathsf{T}}(M)) = \pi_{\mathsf{T},0}(M)$ and $p_{\mathsf{T},1}(p_{\mathsf{T}}(M)) = \pi_{\mathsf{T},1}(M)$ for each $M \in \mathcal{D}(p_{\mathsf{T}})$

At the beginning of the game Γ, $p_{\mathsf{T}}, p_{\mathsf{T},0}, p_{\mathsf{T},1}, \pi_{\mathsf{T},0}$ and $\pi_{\mathsf{T},1}$ are undefined for every $\mathsf{T} \in \{0,1\}^t$, so all four properties are trivially true. Then, assuming that all five properties are true, we examine the effect of $\mathcal{A}$'s Enc and Dec queries. Note that the value of the selector s does not influence the computations that are updating the permutations $p, p_0, p_1, \pi_0, \pi_1$, and thus have no effect on the properties we wish to examine.

When $\mathcal{A}$ makes an Enc(T, M, s) query, and $\pi_{\mathsf{T},0}(M) \neq \perp$ (i.e., $M \in \mathcal{D}(\pi_{\mathsf{T},0})$) then none of the partial permutations is extended, and the properties are trivially preserved by the induction assumption.

If $\pi_{\mathsf{T},0}(M) = \perp$, then by property 1 the images of $\pi_{\mathsf{T},1}(M)$ and $p_{\mathsf{T}}(M)$ are undefined as well. We assign a new image to M in each of the three partial permutations, so property 1 is preserved. The value Y is included in $\mathcal{R}(p_{\mathsf{T}}), \mathcal{D}(p_{\mathsf{T},0})$ and $\mathcal{D}(p_{\mathsf{T},1})$, so property 2 is preserved as well. Similarly, $\mathcal{R}(\pi_{\mathsf{T},0})$ and $\mathcal{R}(p_{\mathsf{T},0})$ both get extended by the same value Z_0, and similarly $\mathcal{R}(\pi_{\mathsf{T},1})$ and $\mathcal{R}(p_{\mathsf{T},1})$ both get extended by Z_1. Thus properties 3 and 4 are preserved as well. Finally, if property 5 held before the current query then it also holds after it is made, as $p_{\mathsf{T}}(M) = Y$, $p_{\mathsf{T},0}(Y) = Z_0 = \pi_{\mathsf{T},0}(M)$ and $p_{\mathsf{T},1}(Y) = Z_1 = \pi_{\mathsf{T},1}(M)$.

When $\mathcal{A}$ makes a Dec(T, C, β) query and $\pi^{-1}_{\mathsf{T},\beta}(C) \neq \perp$, no changes are made to the partial permutations and all properties are trivially preserved. Otherwise, the value X extends the domains of p_{T}, $\pi_{\mathsf{T},0}$ and $\pi_{\mathsf{T},1}$, preserving property 1. The range of p_{T} is extended by the value Y, as are the domains of $p_{\mathsf{T},0}$ and $p_{\mathsf{T},1}$, preserving property 2. The adversarial input C is added to both $\mathcal{R}(\pi_{\mathsf{T},\beta})$ and $\mathcal{R}(p_{\mathsf{T},\beta})$, and the value $Z_{\beta\oplus 1}$ extends both $\mathcal{R}(\pi_{\mathsf{T},\beta\oplus 1})$ and $\mathcal{R}(p_{\mathsf{T},\beta\oplus 1})$, so the

```
proc initialize
    bad ← false
    for $\mathsf{T} \in \{0,1\}^t$ do
        $p_{\mathsf{T}} \leftarrow \bot$; $p_{\mathsf{T},0} \leftarrow \emptyset$; $p_{\mathsf{T},1} \leftarrow \emptyset$
        $\pi_{\mathsf{T},0} \leftarrow \bot$; $\pi_{\mathsf{T},1} \leftarrow \bot$
    end for

proc ENC($T, M, s$)
    if $\pi_{\mathsf{T},0}(M) = \bot$ then
        $Z_0 \leftarrow\$ \{0,1\}^n \backslash \mathcal{R}(\pi_{\mathsf{T},0})$
        $Z_1 \leftarrow\$ \{0,1\}^n \backslash \mathcal{R}(\pi_{\mathsf{T},1})$
        $\pi_{\mathsf{T},0}(M) \leftarrow Z_0$
        $\pi_{\mathsf{T},0}(M) \leftarrow Z_1$
        $\boxed{Y \leftarrow\$ \{0,1\}^n \backslash \mathcal{R}(p_{\mathsf{T}})}$
        $\boxed{p_{\mathsf{T}}(M) \leftarrow Y}$
        $\boxed{p_{\mathsf{T},0}(Y) \leftarrow Z_0}$
        $\boxed{p_{\mathsf{T},1}(Y) \leftarrow Z_1}$
    end if
    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 $\pi_{\mathsf{T},0}(M), \pi_{\mathsf{T},1}(M)$

proc DEC($T, C, \beta, s$)
    if $\pi^{-1}_{\mathsf{T},\beta}(C) = \bot$ then
        $X \leftarrow\$ \{0,1\}^n \backslash \mathcal{D}(\pi_{\mathsf{T},\beta})$
        $Z_{\beta\oplus 1} \leftarrow\$ \{0,1\}^n \backslash \mathcal{R}(\pi_{\mathsf{T},\beta\oplus 1})$
        $\pi^{-1}_{\mathsf{T},\beta}(C) \leftarrow X$
        $\pi_{\mathsf{T},\beta\oplus 1}(X) \leftarrow Z_{\beta\oplus 1}$
        $\boxed{Y \leftarrow\$ \{0,1\}^n \backslash \mathcal{D}(p_{\mathsf{T},\beta})}$
        $\boxed{p^{-1}_{\mathsf{T},\beta}(C) \leftarrow Y}$
        $\boxed{p^{-1}_{\mathsf{T}}(Y) \leftarrow X}$
        $\boxed{p_{\mathsf{T},\beta\oplus 1}(Y) \leftarrow Z_{\beta\oplus 1}}$
    end if
    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 $\pi^{-1}_{\mathsf{T},\beta}(C), \pi_{\mathsf{T},\beta\oplus 1}(\pi^{-1}_{\mathsf{T},\beta}(C))$
    end if
```

Fig. 11: Game Γ used in the proof of security of the IFI$[p, p_0, p_1]$ construction. The tweakable permutations p, p_0, p_1, π_0 and π_1 are initially undefined.

properties 3 and 4 are preserved. Finally, we have $p_{\mathsf{T}}(X) = Y$, $p_{\mathsf{T},\beta}(Y) = C = \pi_{\mathsf{T},\beta}(X)$ and $p_{\mathsf{T},\beta\oplus 1}(Y) = Z_{\beta\oplus 1} = \pi_{\mathsf{T},\beta\oplus 1}(X)$, so property 5 is preserved as well.

It is easy to see, that the games Γ and $\mathbf{prtfp\text{-}ideal}^{\mathcal{A}}_{\mathsf{F}}$ are equivalent. The framed lines in Figure 11 do not affect the outputs of oracle queries; Γ just lazily samples two tweakable random permutations π_0 and π_1, and uses them to reply the ENC and DEC queries the same way as in $\mathbf{prtfp\text{-}ideal}_{\mathsf{F}}$. Therefore $\Pr[\mathcal{A}^{\Gamma} \Rightarrow 1] = \Pr[\mathcal{A}^{\mathbf{prtfp\text{-}ideal}_{\mathsf{F}}} \Rightarrow 1]$.

At the same time, in a non-trivial ENC(T, M) query, we lazily sample an image Y of $p_{\mathsf{T}}(M)$, which was previously undefined due to property 1. The lines 3 and 4 do a correct lazy sampling of $p_{\mathsf{T},0}$ and $p_{\mathsf{T},1}$: the images $p_{\mathsf{T},0}(Y)$ and $p_{\mathsf{T},1}(Y)$ were previously undefined due to property 2, and the sampling of the images Z_0 and Z_1 is correct due to properties 3 and 4. Finally, due to property 5, we see that the ENC oracle actually implements the F construction.

Similarly, in a DEC(T, M, β) query, we sample a preimage Y of previously undefined $p^{-1}_{\mathsf{T},\beta}(C)$ (due to property 3 or 4). Then, the previously unassigned $p^{-1}_{\mathsf{T}}(Y)$ and $p_{\mathsf{T},\beta\oplus 1}(Y)$ (due to property 2) get a correctly sampled preimage X, resp. image $Z_{\beta\oplus 1}$ (and sampling is correct due to property 1 and property 3 or 4). Finally, the assignment is compatible with the F construction (due to property 5). Thus the games Γ and $\mathbf{prtfp\text{-}real}_{\mathsf{F}}$ are equivalent, and $\Pr[\mathcal{A}^{\Gamma} \Rightarrow 1] = \Pr[\mathcal{A}^{\mathbf{prtfp\text{-}real}_{\mathsf{F}}} \Rightarrow 1]$. This concludes the proof. □

B PAEF confidentiality and Integrity Proofs

Proof. Below we prove the confidentiality and authenticity of the PAEF mode. For both confidentiality and authenticity, we first replace F with a pair of independent random tweakable permutations π_0, π_1, i.e. $\pi_0 = (\pi_{\mathsf{T},0} \leftarrow\$ \mathrm{Perm}(n))_{\mathsf{T}\in\{0,1\}^t}$ is a collection of independent uniform elements of $\mathrm{Perm}(n)$ indexed by the elements of $\mathsf{T} \in \{0,1\}^t$ (and similarly $\pi_1 = (\pi_{\mathsf{T},1} \leftarrow\$ \mathrm{Perm}(n))_{\mathsf{T}\in\{0,1\}^t}$). We let PAEF$[(\pi_0, \pi_1), \nu]$ denote the PAEF mode that uses π_0, π_1 instead of F. We have that

$$\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{PAEF}[\mathsf{F},\nu]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{B}) + \mathbf{Adv}^{\mathbf{priv}}_{\mathrm{PAEF}[(\pi_0,\pi_1),\nu]}(\mathcal{A})$$

because a distinguisher $\mathcal{B}$ for F can perfectly simulate the games $\mathbf{priv\text{-}real}_{PAEF[\mathsf{F},\nu]}$ and $\mathbf{priv\text{-}real}_{PAEF[(\pi_0,\pi_1),\nu]}$ for $\mathcal{A}$ using its own oracles. In place of any F^ρ call, $\mathcal{B}$ has to make a decryption query followed by an encryption query. By copying $\mathcal{A}$'s output, $\mathcal{B}$ can achieve the same advantage as $\mathcal{A}$ does, with the same data complexity as $\mathcal{A}$ and a very similar running time. This implies that the gap between these games is bounded by $\mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{B})$. By a similar argument, we have that

$$\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{PAEF}[\mathsf{F},\nu]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{C}) + \mathbf{Adv}^{\mathbf{auth}}_{\mathrm{PAEF}[(\pi_0,\pi_1),\nu]}(\mathcal{A}).$$

For confidentiality, it is easy to see that in a nonce-respecting attack, every message block is processed with a unique tweak. Every ciphertext block and every tag is produced as the only image under and independent random permutation (or a substring of such), and thus uniformly distributed. The final block C_* of every ciphertext is produced as a xor-sum of outputs of π_0 and π_1, each produced with a unique tweak, and thus uniformly distributed. Since all ciphertexts are uniformly distributed we get perfect confidentiality and hence our result.

For authenticity, we analyse the probability of forgery for an adversary that makes a single decryption query against $\mathrm{PAEF}[(\pi_0,\pi_1),\nu]$ and then use a result of Bellare [20] to extend our result to multiple queries (still against $\mathrm{PAEF}[(\pi_0,\pi_1),\nu]$).

We will denote the encryption queries of $\mathcal{A}$ and the corresponding replies as (N^i, A^i, M^i) and C^i for $i = 1, \ldots, q$, where q is the number of encryption queries made by $\mathcal{A}$. For each i we let $C^i_1, \ldots, C^i_{m_i}, C^i_*, T = \mathsf{csplit\text{-}b}_n(C^i)$. We let (N, A, C) denote the only decryption query of $\mathcal{A}$ and we let $C_1, \ldots, C_m, C_*, T = \mathsf{csplit\text{-}b}_n(C)$. When the forgery (N, A, C) is made, we have two base cases. If the nonce N is fresh, then the forgery attempt is equivalent to guessing the value of a uniform string of n bits, and thus succeeds with probability 2^{-n}. This holds even if $|T| < n$, because the rightmost $(n - |T|)$ bits of the preimage of C_* under π_0 must have a specific value.

If N is reused, i.e. if $N = N^i$ for some $N^i \in \{N^1, \ldots, N^q\}$, then we perform a case analysis. Note that we can disregard all encryption queries except the ith, because their ciphertetxts are computed using independent random permutations. Every case assumes the negation of all previous case-conditions.

Case 1, $|C|_n \neq |C^i|_n$: We have several subcases.

- If $|C| = n$, then C is equal to a xor-sum of $\pi_{\mathsf{T},0}$ images from the associated data (denoted as T_A in Figure 6), such that we can possibly have $A^i = A$. However, due to the assumption in this case, we must have $|M^i| > 0$, so the xor-sum T_{A^i} computed in the ith encryption query is xor-masked with uniform bits produced by the processing of M^i_*. Therefore T_{A^i} is statistically independent of C^i, and the adversary has no information when trying to guess the value of the T_A sum. The probability of a successful forgery is 2^{-n}.
- When $|C| > n$, regardless if C has more or less blocks than C^i, the successful forgery is equivalent to guessing the value of an image under π_1 (respectively the value of n out of $2n$ bits produced by $\pi^{-1}_{\mathsf{T},0}(\mathsf{left}_n(T_*))$ and $\pi_{\mathsf{T},1}(\pi^{-1}_{\mathsf{T},0}(\mathsf{left}_n(T_*)))$) such that the tweak $\mathsf{T} = N\|110\|\langle m+1\rangle_{t-\nu-3}$ (respectively $\mathsf{T} = N\|111\|\langle m+1\rangle_{t-\nu-3}$) was not used before. The probability of this event is 2^{-n}.

The probability of a successful forgery in **Case 1** is at most 2^{-n}. In the following cases, $|C|_n = |C^i|_n$.

Case 2, $|A|_n \neq |A^i|_n$: Again, we have a few subcases to consider.

- If $|A|_n > |A^i|_n$, a successful forgery is equivalent to guessing an output value of $\pi_{\mathsf{T},0}$ with a previously unused tweak ($\mathsf{T} = N\|0b1\|\langle a+1\rangle_{t-\nu-3}$ for $b \in \{0,1\}$) thanks to $a > a_i$, succeeding with probability of 2^{-n}.
- If $0 < |A|_n < |A^i|_n$, then a successful forgery is still equivalent to guessing an output value of $\pi_{\mathsf{T},0}$ with a previously unused tweak ($\mathsf{T} = N\|0b1\|\langle a+1\rangle_{t-\nu-3}$ for $b \in \{0,1\}$), thanks to the three-bit domain-separation flag (which was set to 000 in the ith encryption query). This succeeds with probability 2^{-n}.
- Finally if $|A| = 0$, then $|A|_n \neq |A^i|_n$ implies that $|A^i|_n > 0$. Forging in this case is either equivalent to guessing the image $\pi_{(N\|011\|1),0}(10^{n-1})$ such that the random permutation $\pi_{(N\|011\|\langle 1\rangle_{t-\nu-3}),0}$ was evaluated on no more than a single other input $A^i_*\|10^* \neq 10^{n-1}$ in the whole game (if $|C| = n$), or to guessing the correct value for C_*. The former succeeds with probability at most $1/(2^n - 1)$, and the latter with probability at most 2^{-n} (because the corresponding output of $\pi_{\mathsf{T},0}$ was masked by T_{A^i}).

Thus the probability of a successful forgery in this case is at most $1/(2^n - 1)$. In the remaining cases, we have $|C|_n = |C^i|_n > 1$ and $|A|_n = |A^i|_n > 0$.

Case 3, $|C| \neq |C^i|$ and $|T| = n$ or $|T^i| = n$: In this case, the forgery verification will use $\pi_{\mathsf{T},1}$ with a fresh tweak T because the "incomplete-block" bit of the three-bit flag will have different values in the processing of the decryption query, and in the processing of the i^{th} encryption query. The forgery succeeds with probability 2^{-n}.

Case 4, $|A| \neq |A^i|$ and $|A_*| = n$ or $|A^i_*| = n$: This is analogous with the previous case; the probability of forgery is 2^{-n}. In the remaining cases, we have $|C|_n = |C^i|_n > 1$, $|A|_n = |A^i|_n > 0$ and $|T| > 0, |T^i| > 0, |A_*| > 0, |A^i_*| > 0$.

Case 5, $|C| \neq |C^i|$ and $|T| < n$ and $|T^i| < n$: In this case, both the encryption query and the decryption query use the same tweak T to process M^i_* and C_*, T, respectively. There are two conditions for the forgery to succeed. First, the preimage $X = \pi^{-1}_{\mathsf{T},0}(C_* \oplus S)$ (as per line 38 in Figure 5) must be equal to $W \| 10^{n-|T|-1} \neq M^i_* \| 10^{n-|T^i|-1}$ (noting that the case condition implies $|T| \neq |T^i|$) for some $W \in \{0,1\}^{|T|}$. This is no easier than finding a fresh value whose preimage falls into a set of size $2^{|T|}$. With a single image of $\pi^{-1}_{\mathsf{T},0}$ already used, this succeeds with probability bounded by $(2^{|T|})/(2^n - 1)$. *Secondly*, the image $Y = \pi_{\mathsf{T},1}(X)$ must be equal to $T \| Z$ for some $Z \in \{0,1\}^{n-|T|}$, *conditioned on X having the correct format.* This is equivalent to guessing a fresh image under $\pi_{\mathsf{T},1}$ with $(n - |T|)$ free bits. As a single image of $\pi_{\mathsf{T},1}$ has been used already, this happens with probability at most $(2^{n-|T|})/(2^n - 1)$. The probability of a successful forgery in this case is therefore bounded by $(2^{|T|})/(2^n - 1) \cdot (2^{n-|T|})/(2^n - 1) = 2^n/(2^n - 1)^2$.

Case 6, $|A| \neq |A^i|$ and $|A_a| < n$ and $|A^i_a| < n$: In this case, the final blocks A_* and A^i_* are processed by the same random permutation $\pi_{\mathsf{T},0}$, but as $A_* \| 10^{n-|A_*|} \neq A^i_* \| 10^{n-|A^i_*|}$, successfully forging in this case is equivalent to guessing the yet unsampled image $\pi_{\mathsf{T},0}(A_* \| 10^{n-|A_*|})$. With a single image of $\pi_{\mathsf{T},0}$ used before, this happens with probability at most $1/(2^n - 1)$.

Case 7, $|C| = |C^i|$ and $|A| = |A^i|$: In this case, there must be at least a single block of either AD or ciphertext where the two queries differ. We investigate the following subcases.

- If the forgery N, A, C differs from N, A^i, C^i only in $C_* \| T$, then , if we ran the decryption algorithm on N, A^i, C^i and N, A, C in parallel, the values S^i and S used on the line 38 of the decryption algorithm in Figure 5 would be the same, and thus necessarily $(C_* \oplus S) \| T \neq (C^i_* \oplus S^i) \| T^i$. The probability of a successful forgery is at $(2^n - 1)^{-1}$ if $|T| = n$ (inverse of $C_* \oplus S$ has not yet been sampled) and at most $2^n/(2^n - 1)^2$ otherwise (by a similar argument as in **Case 5**).
- If $A, C \| T$ and $A^i, C^i \| T^i$ differ in a single block, such that $C_* \| T = C^i_* \| T^i$, a forgery is impossible (because $\pi_{\mathsf{T},0}$ and $\pi_{\mathsf{T},1}$ are all permutations).
- If there are at least two blocks in $A_1, \ldots A_a, A_*, C_1, \ldots, C_m, C_*, T$ that differ from the corresponding blocks in $A^i_1, \ldots A^i_a, A^i_*, C^i_1, \ldots, C^i_m, C^i_*, T^i$, then the forgery can succeed in two ways. The first is if $(C_* \oplus S) \| T = (C^i_* \oplus S^i) \| T^i$. This happens with probability at most $1/(2^n - 1)$, as there will be at least one index j for which $A_j \neq A^i_j$ (or $C_j \neq C^i_j$), and for which $\pi_{\mathsf{T},0}(A_j) \oplus \pi_{\mathsf{T},0}(A^i_j)$ (respectively $\pi_{\mathsf{T},1}(\pi^{-1}_{\mathsf{T},0}(C_j)) \oplus \pi_{\mathsf{T},1}(\pi^{-1}_{\mathsf{T},1}(C^i_j)))$ would have to take a particular value. The probability follows from the fact that whatever T, the random permutations $\pi_{\mathsf{T},0}$ and $\pi_{\mathsf{T},1}$ were sampled only once. The second way is if $(C_* \oplus S) \| T \neq (C^i_* \oplus S^i) \| T^i$ but the verification still succeeds. This is analogous to **Case 5**.

The probability of a successful forgery in this case is bounded by $2^n/(2^n - 1)^2$.

Thus a single forgery succeeds with probability no greater than $2^n/(2^n - 1)^2$. By applying the result of Bellare [20], we can bound the probability of a successful forgery among q_v decryption queries as $(q_v \cdot 2^n)/(2^n - 1)^2$. □

C SAEF Confidentiality and Integrity Proofs

Proof (Proof of Theorem 2). The security analysis of SAEF is slightly more involved than in the case of PAEF. We first tackle confidentiality and then integrity.

Confidentiality of SAEF. We first replace the forkcipher F with a pair of tweakable permutations π_0 and π_1. I.e. $\pi_0 = (\pi_{\mathsf{T},0} \leftarrow_\$ \mathrm{Perm}(n))_{\mathsf{T} \in \{0,1\}^\tau}$ is a collection of independent uniform elements of

$\mathrm{Perm}(n)$ indexed by the elements of $\mathsf{T} \in \{0,1\}^\tau$ (and similarly for $\pi_1 = (\pi_{\mathsf{T},1} \leftarrow\$ \mathrm{Perm}(n))_{\mathsf{T}\in\{0,1\}^\tau}$). We let $\mathrm{SAEF}[\pi_0, \pi_1]$ denote the SAEF mode that uses π_0, π_1 instead of F. This replacement implies the following inequality:

$$\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{SAEF}[\mathsf{F}]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{B}) + \mathbf{Adv}^{\mathbf{priv}}_{\mathrm{SAEF}[\pi_0,\pi_1]}(\mathcal{A})$$

by a similar argument as in the proof of Theorem 1.

We now further replace the two families of random permutations π_0 and π_1 with families of *random functions* f_0 and f_1 with the same signature. I.e. $f_b = (f_{\mathsf{T},b} \leftarrow\$ \mathrm{Func}(n))_{\mathsf{T}\in\{0,1\}^\tau}$ for $b \in \{0,1\}$. Denoting the SAEF mode using these random functions by $\mathrm{SAEF}[f_0, f_1]$, we have that

$$\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{SAEF}[\pi_0,\pi_1]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{priv}}_{\mathrm{SAEF}[f_0,f_1]}(\mathcal{A}) + 2 \cdot \frac{(\sigma - q)^2}{2^{n+1}}$$

because all but the first block (be it AD or message) of each query are processed using a tweak of the form $0^{\tau-3} \| b_0 b_1 b_2$ with $b_0, b_1, b_2 \in \{0,1\}$. As there are no more than σ blocks of data in total, each of the permutations $\pi_{\mathsf{T},0}$ and $\pi_{\mathsf{T},1}$ processes σ_{T} blocks with $\sum_{\mathsf{T}\in\{0,1\}^\tau} \sigma_{\mathsf{T}} = \sigma$. Replacing each $\pi_{\mathsf{T},0}$ by $f_{\mathsf{T},0}$ augments the bound by at most $\sigma_{\mathsf{T}}(\sigma_{\mathsf{T}} - 1) \cdot 2^{-n-1}$ by the RP-RF switching lemma [25] and a standard hybrid argument. A sum of all these augmentations is upper bounded by $(\sigma - q)^2/2^{n+1}$, noting that there are at least q tweak values T for which $\pi_{\mathsf{T},0}$ is applied to at most a single block. Another term $(\sigma - q)^2/2^{n+1}$ needs to be added to account for the replacement of $\pi_{\mathsf{T},1}$ for all T.

We now bound $\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{SAEF}[f_0,f_1]}(\mathcal{A})$. For this, we use the games G_0 and G_1 defined in Figure 12. In both games, the set $\mathcal{D}_{\mathsf{T}}$ collects the domain points, on which the functions $f_{\mathsf{T},0}$ and $f_{\mathsf{T},1}$ were already evaluated. It is easy to verify that G_0 actually implements $\mathbf{priv\text{-}real}_{\mathrm{SAEF}[f_0,f_1]}$, as the flag bad and the sets $\mathcal{D}_{\mathsf{T}}$ have no influence on the outputs of Enc. It is also possible to verify that $\Pr[\mathcal{A}^{\mathbf{priv\text{-}ideal}_{\mathrm{SAEF}[f_0,f_1]}} \Rightarrow 1] = \Pr[\mathcal{A}^{G_1} \Rightarrow 1]$: unless bad is set, every ciphertext block C_i is an xor of images of a distinct input to two random functions, and T is simply produced by applying a random function to a fresh input. Thus, all the output bits of Enc are uniform. Once bad is set, all the ciphertext blocks and each value of Δ is replaced by a uniform string, so the simulation is perfect. Thus we have $\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{SAEF}[f_0,f_1]}(\mathcal{A}) \leq \Pr[\mathcal{A}^{G_0} \Rightarrow 1] - \Pr[\mathcal{A}^{G_1} \Rightarrow 1]$.

We also have that G_0 and G_1 are identical until bad, so by the Fundamental lemma of game-playing [25] we have that $\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{SAEF}[f_0,f_1]}(\mathcal{A}) \leq \Pr[\mathcal{A}^{G_0} \text{ sets } \mathsf{bad}]$, where $\mathcal{A}^{G_0}$ sets bad denotes the event that $\mathsf{bad} = \mathsf{true}$ when $\mathcal{A}$ issues its final output. We bound $\Pr[\mathcal{A}^{G_0} \text{ sets } \mathsf{bad}]$ by union bound, iterating over the probability that the i^{th} query sets bad, given that bad was not set before.

For an encryption query (N, A, M), the initial block of that query is processed with a tweak $N \| 1 b_0 b_1 b_2$, with the corresponding set $\mathcal{D}_{N\|1b_0b_1b_2}$ empty, making it impossible to set bad. Each remaining block (be it AD or message) is masked with the Δ value before it is fed to $f_{\mathsf{T},b}$ (for $b \in \{0,1\}$ and some T). If bad has not been set before, Δ is a uniform n-bit string. Thus each such block can set bad with probability $|\mathcal{D}_{\mathsf{T},b}|/2^n$ for $b \in \{0,1\}$ and some T will be uniformly distributed due to the Δ mask produced by $f_{N\|1b_0b_1b_2,1}$. There are almost $(\sigma - q)$ blocks that can set bad when fed to $f_{\mathsf{T},b}$, and for each we have $|\mathcal{D}_{\mathsf{T},b}| \leq (\sigma - q)$. The total probability of setting bad is thus no more than $(\sigma - q)/2^n$, completing the proof of the confidentiality bound.

Integrity of SAEF. We again replace the forkcipher F with a pair of tweakable permutations $\pi_0 = (\pi_{\mathsf{T},0} \leftarrow\$ \mathrm{Perm}(n))_{\mathsf{T}\in\{0,1\}^\tau}$ and $\pi_1 = (\pi_{\mathsf{T},1} \leftarrow\$ \mathrm{Perm}(n))_{\mathsf{T}\in\{0,1\}^\tau}$, such that we have

$$\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{SAEF}[\mathsf{F}]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{C}) + \mathbf{Adv}^{\mathbf{auth}}_{\mathrm{SAEF}[\pi_0,\pi_1]}(\mathcal{A})$$

by a similar argument as in the proof of Theorem 1.

We additionally replace π_0, π_1 by $\tilde{\pi}_0, \tilde{\pi}_1$, which implement a random permutation for some tweaks, and a random function for others. More precisely, we let $\mathsf{Flag}_A \subset \{0,1\}^t$ denote the set of all tweaks whose 3-bit flag is in the set $\{000, 010, 011, 110, 111\}$ (i.e., tweaks used in the AD processing), and we let $\mathsf{Flag}_M = \{0,1\}^t \backslash \mathsf{Flag}_A$ be the set of all tweaks used in the message processing. Then we define

$$\tilde{\pi}_0 = (\tilde{\pi}_{\mathsf{T},0})_{\mathsf{T}\in\{0,1\}^\tau} \text{ s.t. } \tilde{\pi}_{\mathsf{T},0} \leftarrow\$ \mathrm{Func}(n) \text{ if } \mathsf{T} \in \mathsf{Flag}_A \text{ and } \tilde{\pi}_{\mathsf{T},0} \leftarrow\$ \mathrm{Perm}(n) \text{ otherwise}$$

$$\tilde{\pi}_1 = (\tilde{\pi}_{\mathsf{T},1})_{\mathsf{T}\in\{0,1\}^\tau} \text{ s.t. } \tilde{\pi}_{\mathsf{T},1} \leftarrow\$ \mathrm{Func}(n) \text{ if } \mathsf{T} \in \mathsf{Flag}_M \text{ and } \tilde{\pi}_{\mathsf{T},1} \leftarrow\$ \mathrm{Perm}(n) \text{ otherwise} .$$

```
proc initialize
    for $\mathsf{T} \in \{0,1\}^\tau$ do
        $f_{\mathsf{T},0} \leftarrow\!\$\ \mathrm{Func}(n)$
        $f_{\mathsf{T},1} \leftarrow\!\$\ \mathrm{Func}(n)$
        $\mathcal{D}_\mathsf{T} \leftarrow \emptyset$
    end for
    $\mathsf{bad} \leftarrow \mathsf{false}$

proc Enc$(N, A, M)$
    $A_1, \ldots, A_a, A_* \xleftarrow{n} A$
    $M_1, \ldots, M_m, M_* \xleftarrow{n} M$
    $W \leftarrow (N, A)$
    $\mathsf{noM} \leftarrow 0$
    if $|M| = 0$ then $\mathsf{noM} \leftarrow 1$
    $\Delta \leftarrow 0^n$; $\mathsf{T} \leftarrow N \| 0^{\tau-\nu-4} \| 1$
    for $i \leftarrow 1$ to $a$ do
        $\mathsf{T} \leftarrow \mathsf{T} \| 000$
        if $A_i \oplus \Delta \in \mathcal{D}_\mathsf{T}$ then
            $\mathsf{bad} \leftarrow \mathsf{true}$
        end if
        $\mathcal{D}_\mathsf{T} \leftarrow \mathcal{D}_\mathsf{T} \cup (A_i \oplus \Delta)$
        $\Delta \leftarrow f_{\mathsf{T},0}(A_i \oplus \Delta)$
        if $\mathsf{bad} = \mathsf{true}$ then
            $\boxed{\Delta \leftarrow\!\$\ \{0,1\}^n}$
        end if
        $\mathsf{T} \leftarrow 0^{\tau-3}$
    end for
    if $|A_*| = n$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| \mathsf{noM} \| 10$
        if $A_* \oplus \Delta \in \mathcal{D}_\mathsf{T}$ then
            $\mathsf{bad} \leftarrow \mathsf{true}$
        end if
        $\mathcal{D}_\mathsf{T} \leftarrow \mathcal{D}_\mathsf{T} \cup (A_* \oplus \Delta)$
        $\Delta \leftarrow f_{\mathsf{T},0}(A_* \oplus \Delta)$
        if $\mathsf{bad} = \mathsf{true}$ then
            $\boxed{\Delta \leftarrow\!\$\ \{0,1\}^n}$
        end if
        $\mathsf{T} \leftarrow 0^{\tau-3}$
    else if $|A_*| > 0$ or $|M| = 0$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| \mathsf{noM} \| 11$
        if $A_* \oplus \Delta \in \mathcal{D}_\mathsf{T}$ then
            $\mathsf{bad} \leftarrow \mathsf{true}$
        end if
        $\mathcal{D}_\mathsf{T} \leftarrow \mathcal{D}_\mathsf{T} \cup ((A_* \| 10^*) \oplus \Delta)$
        $\Delta \leftarrow f_{\mathsf{T},0}((A_* \| 10^*) \oplus \Delta)$
        if $\mathsf{bad} = \mathsf{true}$ then
            $\boxed{\Delta \leftarrow\!\$\ \{0,1\}^n}$
        end if
        $\mathsf{T} \leftarrow 0^{\tau-3}$
    end if
    for $i \leftarrow 1$ to $m$ do
        $\mathsf{T} \leftarrow \mathsf{T} \| 001$
        if $M_i \oplus \Delta \in \mathcal{D}_\mathsf{T}$ then
            $\mathsf{bad} \leftarrow \mathsf{true}$
        end if
        $\mathcal{D}_\mathsf{T} \leftarrow \mathcal{D}_\mathsf{T} \cup (M_i \oplus \Delta)$
        $C_i \leftarrow f_{\mathsf{T},0}(M_i \oplus \Delta) \oplus \Delta$
        $\Delta \leftarrow f_{\mathsf{T},1}(M_i \oplus \Delta)$
        if $\mathsf{bad} = \mathsf{true}$ then
            $\boxed{C_i, \Delta \leftarrow\!\$\ \{0,1\}^n \times \{0,1\}^n}$
        end if
        $\mathsf{T} \leftarrow 0^{\tau-3}$
    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
    if $M_* \oplus \Delta \in \mathcal{D}_\mathsf{T}$ then
        $\mathsf{bad} \leftarrow \mathsf{true}$
    end if
    $C_* \leftarrow f_{\mathsf{T},0}(\mathsf{pad10}(M_*) \oplus \Delta) \oplus \Delta$
    $T \leftarrow f_{\mathsf{T},1}(\mathsf{pad10}(M_*) \oplus \Delta)$
    if $\mathsf{bad} = \mathsf{true}$ then
        $\boxed{C_*, T \leftarrow\!\$\ \{0,1\}^n \times \{0,1\}^n}$
    end if
    return $C_1 \| \ldots \| C_m \| C_* \| \mathsf{left}_{|M_*|}(T)$
```

Fig. 12: The games G_0 and G_1 for bounding $\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{SAEF}[f_0,f_1]}$. The game G_0 does *not* contain the boxed statement, while G_1 does.

In other words, we replace those random permutations that produce the Δ masks with random functions. We have

$$\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{SAEF}[\pi_0,\pi_1]}(\mathcal{A}) \le \mathbf{Adv}^{\mathbf{auth}}_{\mathrm{SAEF}[\tilde{\pi}_0,\tilde{\pi}_1]}(\mathcal{A}) + \frac{(\sigma - q + 1)^2}{2^{n+1}}$$

by a similar argument as in the proof of SAEF's confidentiality; the difference here is that $\mathcal{A}$ may force a permutation $\pi_{N\|b_0b_1b_2,1}$ to be used $\sigma - q + 1$ times by making all decryption queries with N.

To bound $\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{SAEF}[\tilde{\pi}_0,\tilde{\pi}_1]}(\mathcal{A})$, we consider the games G_2 and G_3 in Figures 13 and 14. It is easy to see that the game G_2 actually implements the game $\mathbf{auth}_{\mathrm{SAEF}[\tilde{\pi}_0,\tilde{\pi}_1]}$, because the sets $\mathcal{D}_\mathsf{T}$ for $\mathsf{T} \in \{0,1\}^\tau$ and the flag bad have no effect on the outputs of the game. Moreover, unless bad is set to true, the games G_2 and G_3 execute the same code. Thus, by the Fundamental lemma of game-playing [25], we have that $\Pr[\mathcal{A}^{G_2} \text{ forges}] - \Pr[\mathcal{A}^{G_3} \text{ forges}] \leq \Pr[\mathcal{A}^{G_2} \text{ sets } \mathsf{bad}]$ and consequently $\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{SAEF}[\tilde{\pi}_0,\tilde{\pi}_1]}(\mathcal{A}) \leq \Pr[\mathcal{A}^{G_2} \text{ sets } \mathsf{bad}] + \Pr[\mathcal{A}^{G_3} \text{ forges}]$.

Transition from G_2 to G_3. The flag bad being set means that for some $\mathsf{T} \in \{0,1\}^\tau$, the pair of permutations/functions $\tilde{\pi}_{\mathsf{T},0}$ and $\tilde{\pi}_{\mathsf{T},1}$ were used twice on the same input in an encryption query, beyond a trivial prefix of the two queries. Informally speaking, this event may allow the adversary to forge trivially by simply truncating the ciphertext, or the associated data used in an encryption query with such a collision. We disallow this kind of victory in the game G_3.

Some of the conditions that can set bad use predicates $P^i_{\mathrm{A}}(W,Q)$, $P^*_{\mathrm{A}}(W,Q)$ and $P^i_{\mathrm{M}}(W,Q)$. These predicates return true if the current query is, up to the currently processed block, **not** a blockwise prefix of some previous query. More precisely, the predicate $P^i_{\mathrm{A}}(W,Q)$ (with $W = (N,A,C)$ or (N,A)) returns *false* if and only if(1) $Q(N) \neq \emptyset$ and (2) there is a tuple (N,A') or (N,A',C') such that $A_j = A'_j$ for $j = 1,\ldots,i$. The predicate $P^*_{\mathrm{A}}(W,Q)$ is the same as the predicate $P^i_{\mathrm{A}}(W,Q)$ except condition (2) becomes that there is a tuple (N,A') or (N,A',C') such that $A_j = A'_j$ for $j = 1,\ldots,a$ and $A_* = A'_*$. Finally the predicate $P^i_{\mathrm{A}}(W,Q)$ returns *false* if and only if $P^*_{\mathrm{A}}(W,Q)$ is false, and if additionally $C_j = C'_j$ for $j = 1,\ldots,i$. Note that the three predicates generate a monotonic sequence when a query is processed; once one predicate returns true, all will return true in the same query. Note also that in the decrytpion queries, checking the collisions in the domain of any $\tilde{\pi}_{\mathsf{T},0}$ processing message blocks is equivalent with checking the collisions in the range, as each such $\tilde{\pi}_{\mathsf{T},0}$ is a permutation. Similarly as in the proof of confidentiality bound, we bound $\Pr[\mathcal{A}^{G_2} \text{ sets } \mathsf{bad}]$ by the union bound, iterating over the probability that the i^{th} query sets bad, given that bad was not set before.

In an encryption query (N,A,M), the flag bad can be set during AD processing only after $P^i_{\mathrm{A}}(W,Q)$ (or $P^*_{\mathrm{A}}(W,Q)$) are returning true. The first block A_i (or A_*), for which the predicate is true comes right after the longest blockwise prefix with previous queries, so the current mask $\Delta = \Delta'$ for the corresponding Δ' in the previous query (N,A') that yields the common prefix, but $A_i \neq A'_i$ (or $A_* \neq A'_*$). The value of Δ' is statistically independent of the ciphertexts returned to $\mathcal{A}$, and so $A_i \oplus \Delta \in \mathcal{D}_{\mathsf{T}}$ (or $\mathsf{pad10}(A_*) \oplus \Delta \in \mathcal{D}_{\mathsf{T}}$) falls into $\mathcal{D}_{\mathsf{T}}$ with probability $|\mathcal{D}_{\mathsf{T}}|/2^n \leq (\sigma - q)/2^n$ by a similar argument as in the confidentiality proof of SAEF. For all the consequent blocks B of AD or message, if bad is not set before B is being processed, the Δ value that is used to mask B is a uniformly distributed string, so $B \oplus \Delta \in \mathcal{D}_{\mathsf{T}}$ with probability $|\mathcal{D}_{\mathsf{T}}|/2^n \leq (\sigma - q)/2^n$ as well.

In a decryption query (N,A,C), bad can only be set after the first time $P^i_{\mathrm{A}}(W,Q)$, $P^*_{\mathrm{A}}(W,Q)$, or $P^i_{\mathrm{M}}(W,Q)$ return true. Similarly as in an encryption query, the first block B for which this occurs will be masked by a reused Δ, but this Δ will be independent of the observed ciphertexts (even if B is a message block, because $\neg\mathsf{bad}$ implies that each ciphertext block was computed with a fresh uniform mask). For the consequent blocks, $\neg\mathsf{bad}$ implies that Δ is fresh and uniformly distributed. Thus $B \oplus \Delta \in \mathcal{D}_{\mathsf{T}}$ with probability $|\mathcal{D}_{\mathsf{T}}|/2^n \leq (\sigma - q)/2^n$.

By summing over all σ blocks, we get $\Pr[\mathcal{A}^{G_2} \text{ sets } \mathsf{bad}] \leq \sigma(\sigma - q)/2^n$.

Forgery in G_3. We proceed to bounding $\Pr[\mathcal{A}^{G_3} \text{ forges}]$. We carry out the analysis for an adversary $\mathcal{A}'$ that makes a single verification query, and then obtain $\Pr[\mathcal{A}^{G_3} \text{ forges}] \leq q_v \cdot \Pr[\mathcal{A}'^{G_3} \text{ forges}]$, referring to a result by Bellare to support the claim [20]. We establish the bound by the means of a case analysis.

In what follows, we let $(N^i,A^i,M^i),C^i$ denote the i^{th} encryption query made by $\mathcal{A}'$, and (N,A,C) denote the only decryption query. For each i, we let $C^i_1,\ldots,C^i_m,C^i_*,T^i \leftarrow \mathsf{csplit\text{-}b}_n(C^i)$ and we let $C_1,\ldots,C_m,C_*,T \leftarrow \mathsf{csplit\text{-}b}_n(C)$. Additionally, we will refer to the values of the Δ variable. We will indicate by $\Delta_{A,j}$ the j^{th} value that the variable Δ takes when processing the j^{th} block of A from the decryption query (N,A,C), and by $\Delta_{M,j}$ the j^{th} value that the variable Δ takes when processing the j^{th} block of the ciphertext C. We note that we can have $j = *$ and that $\Delta_{A,1} = 0^n$. We define $\Delta^i_{A,j}$ and $\Delta^i_{M,j}$ in a similar way for (N^i,A^i,M^i).

Case 1, $A = \varepsilon$ and $|C|_n \leq 2$, or $|A|_n = 1$ and $|C|_n = 1$: We have two sub-cases.

Case 1.1, $\nexists N^i$ such that $N = N^i$: In this case, the forgery equals to guessing n random bits, as the verification uses $\tilde{\pi}_{N\|b_0b_1b_2,0}$ and $\tilde{\pi}_{N\|b_0b_1b_2,0}$, which have not been sampled before because of the freshness of the nonce.

Case 1.2, $\exists N^i$ such that $N = N^i$ but $|A|_n + |C|_n > 2$: Also in this case, the forgery equals to guessing n random bits, as the verification uses $\pi_{N\|b_0b_1b_2,0}$ and $\tilde{\pi}_{N\|b_0b_1b_2,0}$, which have not been sampled before because the N was not used with the binary flags $b_0b_1b_2$.

Case 1.3, $\exists N^i$ such that $N = N^i$ and $|A|_n + |C|_n \leq 2$: $\mathcal{A}'$ knows a at most a single image under each $\tilde{\pi}_{N\|b_0b_1b_2,0}$ (processes C_*) and $\tilde{\pi}_{N\|b_0b_1b_2,0}$ (processes T). If the forgery attempt is with an AD block, or a ciphertext corresponding to a complete message block, the adversary has to guess a fresh image under $\tilde{\pi}_{N\|b_0b_1b_2,0}$, succeeding with probability 2^{-n}. If $\mathcal{A}'$ tries to forge with a ciphertext corresponding to an incomplete message block, the freshly sampled preimage $M_* = \tilde{\pi}^{-1}_{N\|101,0}(C_* \oplus \Delta_{M,*})$ will need to be of the form $M_* = Z\|10^*$ for some $Z \in \{0,1\}^{|T|}$ *and* simultaneously, the first $|T|$ bits of the freshly sampled image $Y = \tilde{\pi}_{N\|101,1}(M_*)$ will need to be equal to T. This happens with probability no greater than $((2^{|T|} - 1) \cdot (2^{n-|T|}))/((2^n - 1) \cdot 2^n) \leq 1/(2^n - 1)$.

The probability of forgery in this case is no more than $1/(2^n - 1)$.

The following cases assume the negation of the condition in **Case 1** (i.e., the forgery attempt consist of more than a single block in total).

Case 2: The tag computation is not done right after the trivial prefix with (N, A^i, C^i). More formally, we have the following subcases:

Case 2.1, $|C|_n = 1$ and $P^a_{\mathsf{A}}(W, Q) =$ true: In this case, the tag is verified in AD processing using A_* and a mask $\Delta_{A,*}$. Due to the condition in this case (and the fact that a domain collision on $\tilde{\pi}_{\mathsf{T},0}$ sets bad and ends the game), Δ is computed as an image of $\tilde{\pi}_{\mathsf{T},0}$ evaluated on a fresh input, and thus uniform. The forgery can either succeed if $A_* \oplus \Delta_{A,*}$ equals to a value $A^j_* \oplus \Delta^j_{A,*}$ that has already been fed to $\tilde{\pi}_{\mathsf{T},0}$ in the jth encryption query (then $\mathcal{A}'$ can reuse C^j_*). As $j \in \{1, \ldots, q\}$ (T is used at most once per query), this happens with probability at most $q/2^n$. If this collision does not succeed, then the adversary must guess a fresh image under $\tilde{\pi}_{\mathsf{T},0}$, which succeeds with probability 2^{-n}. The total forgery probability in this case is bounded by $(q + 1)/2^n$.

Case 2.2, $|C|_n > 1$ and $P^m_{\mathsf{M}}(W, Q) =$ true: In this case, the tag is verified in message processing using C_*, tweak $\mathsf{T} \in \{0^{\tau-3}\|100, 0^{\tau-3}\|101\}$ and a mask $\Delta_{M,*}$. Similarly as in **Case 2.1**, the Δ mask is a uniform string, and the forgery can either succeed if $C_* \oplus \Delta_{M,*}$ is equal to an already-used range point $C^j_* \oplus \Delta^j_{M,*}$ of $\pi_{\mathsf{T},0}$ (allowing $\mathcal{A}'$ to reuse the corresponding tag), or by guessing a correct value and length of the tag. The former succeeds with probability at most $q/2^n$. For the latter, we explore two brief subcases.

Case 2.2.1, $|T| = n$. In this case, the fact that $C_* \oplus \Delta_{M,*}$ is fresh implies that $M_* \oplus \Delta_{M,*}$ has not been fed to $\tilde{\pi}_{\mathsf{T},1}$ before, and a successful forgery equals to guessing a value of a uniform n-bit string. This happens with probability at most 2^{-n}.

Case 2.2.2, $|T| < n$. In this case, the yet unknown preimage $M_* = \pi^{-1}_{\mathsf{T},0}(C_* \oplus \Delta_{M,*})$ must have the form $M_* = Z\|10^{n-|T|-1}$ for some $Z \in \{0,1\}^{|T|}$, and the yet unknown image $\tilde{\pi}_{\mathsf{T},1}(M_* \oplus \Delta_{M,*})$ has to be equal to $T\|Y$ for some $Y \in \{0,1\}^{n-|T|}$. This happens with probability at most $(2^{|T|}/(2^n - \sigma)) \cdot (2^{n-|T|}/2^n) \leq 2/2^n$.

The total probability of forgery in **Case 2.2** is bounded by $(q + 2)/2^n$.

The probability of forgery in **Case 2** is at most $(q + 2)/2^n$.

Case 3: In the final case, the tag verification is done right after the trivial prefix with (N, A^i, C^i). More formally, we have the following subcases:

Case 3.1, $|C|_n = 1$ and $P^a_{\mathsf{A}}(W, Q) =$ false: In this case, the tag is verified in AD processing using A_*, right after the trivial prefix with the ith encryption query, using a tweak $\mathsf{T} \in \{0^{\tau-3}\|110, 0^{\tau-3}\|111\}$ and a mask $\Delta_{A,*} = \Delta^i_{A,*}$ (for the corresponding mask in the ith encryption query). We must have that $A_* \neq A^i_*$ (otherwise the forgery attempt would be invalid), so C^i_* can't be reused (as necessarily $C_* \neq C^i_*$). $\mathcal{A}'$ may attempt to force $A_* \oplus \Delta_{A,*} = A^j_* \oplus \Delta_{A,*}$ and reuse C^j_* for $j \neq i$, but this happens with probability at most $q/2^n$, similarly as in **Case 2.1**. This is because $\Delta_{A,*} = \Delta^j_{A,*}$ is statistically independent of the ciphertexts observed by the adversary. Otherwise $\mathcal{A}'$ can forge by guessing the correct value for C_* succeeding with probability 2^{-n}. The total probability of forgery in this case is no more than $(q + 1)/2^n$.

```
proc initialize
    for $\mathsf{T} \in \{0,1\}^\tau$ do
        if $\mathsf{T} \in \mathsf{Flag}_A$ then
            $\tilde{\pi}_{\mathsf{T},0} \leftarrow\$ \mathrm{Func}(n)$
        else
            $\tilde{\pi}_{\mathsf{T},0} \leftarrow\$ \mathrm{Perm}(n)$
        end if
        if $\mathsf{T} \in \mathsf{Flag}_M$ then
            $\tilde{\pi}_{\mathsf{T},1} \leftarrow\$ \mathrm{Func}(n)$
        else
            $\tilde{\pi}_{\mathsf{T},1} \leftarrow\$ \mathrm{Perm}(n)$
        end if
        $\mathcal{D}_\mathsf{T} \leftarrow \emptyset$
    end for
    for $N \in \{0,1\}^\nu$ do $Q(N) \leftarrow \emptyset$
    $\mathsf{bad} \leftarrow \mathsf{false}$

proc $\mathrm{Enc}(N, A, M)$
    $A_1, \ldots, A_a, A_* \xleftarrow{n} A$
    $M_1, \ldots, M_m, M_* \xleftarrow{n} M$
    $\mathsf{noM} \leftarrow 0$
    if $|M| = 0$ then $\mathsf{noM} \leftarrow 1$
    $\Delta \leftarrow 0^n$; $\mathsf{T} \leftarrow N\|1$
    for $i \leftarrow 1$ to $a$ do
        $\mathsf{T} \leftarrow \mathsf{T}\|000$
        if $A_i \oplus \Delta \in \mathcal{D}_\mathsf{T}$ and $\mathsf{P}^i_a(W, Q)$ then
            $\mathsf{bad} \leftarrow \mathsf{true}$
        end if
        $\mathcal{D}_\mathsf{T} \leftarrow \mathcal{D}_\mathsf{T} \cup (A_i \oplus \Delta)$
        $\Delta \leftarrow \tilde{\pi}_{\mathsf{T},0}(A_i \oplus \Delta)$
        $\mathsf{T} \leftarrow 0^{\tau-3}$
    end for
    if $|A_*| = n$ then
        $\mathsf{T} \leftarrow \mathsf{T}\|\mathsf{noM}\|10$
        if $A_* \oplus \Delta \in \mathcal{D}_\mathsf{T}$ and $\mathsf{P}^*_a(W, Q)$ then
            $\mathsf{bad} \leftarrow \mathsf{true}$
        end if
        $\mathcal{D}_\mathsf{T} \leftarrow \mathcal{D}_\mathsf{T} \cup (A_* \oplus \Delta)$
        $\Delta \leftarrow \tilde{\pi}_{\mathsf{T},0}(A_* \oplus \Delta)$
        $\mathsf{T} \leftarrow 0^{\tau-3}$
    else if $|A_*| > 0$ or $|M| = 0$ then
        $\mathsf{T} \leftarrow \mathsf{T}\|\mathsf{noM}\|11$
        if $(A_*\|10^*) \oplus \Delta \in \mathcal{D}_\mathsf{T}$ and $\mathsf{P}^*_a(W, Q)$ then
            $\mathsf{bad} \leftarrow \mathsf{true}$
        end if
        $\mathcal{D}_\mathsf{T} \leftarrow \mathcal{D}_\mathsf{T} \cup ((A_*\|10^*) \oplus \Delta)$
        $\Delta \leftarrow \tilde{\pi}_{\mathsf{T},0}((A_*\|10^*) \oplus \Delta)$
        $\mathsf{T} \leftarrow 0^{\tau-3}$
    end if
    for $i \leftarrow 1$ to $m$ do
        $\mathsf{T} \leftarrow \mathsf{T}\|001$
        if $M_i \oplus \Delta \in \mathcal{D}_\mathsf{T}$ then
            $\mathsf{bad} \leftarrow \mathsf{true}$
        end if
        $\mathcal{D}_\mathsf{T} \leftarrow \mathcal{D}_\mathsf{T} \cup (M_i \oplus \Delta)$
        $C_i \leftarrow \tilde{\pi}_{\mathsf{T},0}(M_i \oplus \Delta) \oplus \Delta$
        $\Delta \leftarrow \tilde{\pi}_{\mathsf{T},1}(M_i \oplus \Delta)$
        $\mathsf{T} \leftarrow 0^{\tau-3}$
    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
    if $\mathsf{pad10}(M_i) \oplus \Delta \in \mathcal{D}_\mathsf{T}$ then
        $\mathsf{bad} \leftarrow \mathsf{true}$
    end if
    $\mathcal{D}_\mathsf{T} \leftarrow \mathcal{D}_\mathsf{T} \cup (\mathsf{pad10}(M_i) \oplus \Delta)$
    $C_* \leftarrow \tilde{\pi}_{\mathsf{T},0}(\mathsf{pad10}(M_*) \oplus \Delta) \oplus \Delta$
    $T \leftarrow \tilde{\pi}_{\mathsf{T},1}(\mathsf{pad10}(M_*) \oplus \Delta)$
    $Q(N) \leftarrow Q(N) \cup ((A_1, .., A_*), (C_1, .., C_m))$
    return $C_1\| \ldots \|C_m\|C_*\|\mathsf{left}_{|M_*|}(T)$
```

Fig. 13: The games G_2 and G_3 for bounding $\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{SAEF}[\tilde{\pi}_0,\tilde{\pi}_1]}$ (continued in Figure 14). The game G_2 does *not* contain the boxed statements, while G_3 does.

Case 3.2, $|C|_n > 1$ and $P^m_{\mathsf{M}}(W, Q) =$ false: In this case, the tag is verified in message processing right after the trivial prefix with the ith encryption query, using C_*, tweak $\mathsf{T} \in \{0^{\tau-3}\|100, 0^{\tau-3}\|101\}$ and a mask $\Delta_{M,*}$. This case is analogous to **Case 2.2**, except that $\Delta_{M,*} = \Delta^i_{M,*}$ has already been used before. Yet, $\Delta_{M,*} = \Delta^i_{M,*}$ is statistically independent from the observed ciphertexts (if bad is not set, every ciphertext block is equal to an image of $\pi_{\mathsf{T},0}$ masked with an independent uniform string). Thus the argumentation of **Case 2.2** carries over, and the probability of forgery in **Case 3.2** is no more than $(q + 2)/2^n$.

By taking the maximum over all cases, the probability that a single-decryption-query adversary $\mathcal{A}'$ forgers in the game G_3 is at most $(q + 2)/2^n$. The adversary $\mathcal{A}$ making q_v decryption queries thus forges with probability bounded by $q_v \cdot (q+2)/2^n$. By back-substituting all the previous equalities, we obtain the claimed result.

```
proc Dec$(N, A, C)$
    if $\mathsf{bad} = \mathsf{true}$ then
        $\boxed{\textbf{return } \bot}$
    end if
    $W \leftarrow (N, A, C)$
    $Q(N) \leftarrow Q(N) \cup ((A_1, .., A_*))$
    $A_1, \ldots, A_a, A_* \xleftarrow{n} A$
    $C_1, \ldots, C_m, C_*, T \leftarrow \mathsf{csplit\text{-}b}_n(C)$
    $\mathsf{noM} \leftarrow 0$
    if $|C| = n$ then $\mathsf{noM} \leftarrow 1$
    $\Delta \leftarrow 0^n$; $\mathsf{T} \leftarrow N \| 1$
    for $i \leftarrow 1$ to $a$ do
        $\mathsf{T} \leftarrow \mathsf{T} \| 000$
        if $A_i \oplus \Delta \in \mathcal{D}_\mathsf{T}$ and $\mathsf{P}^i_a(W, Q)$ then
            $\mathsf{bad} \leftarrow \mathsf{true}$
            $\boxed{\textbf{return } \bot}$
        end if
        $\mathcal{D}_\mathsf{T} \leftarrow \mathcal{D}_\mathsf{T} \cup (A_i \oplus \Delta)$
        $\Delta \leftarrow \tilde{\pi}_{\mathsf{T},0}(A_i \oplus \Delta)$
        $\mathsf{T} \leftarrow 0^{\tau-3}$
    end for
    if $|A_*| = n$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| \mathsf{noM} \| 10$
        if $A_* \oplus \Delta \in \mathcal{D}_\mathsf{T}$ and $\mathsf{P}^*_a(W, Q)$ then
            $\mathsf{bad} \leftarrow \mathsf{true}$
            $\boxed{\textbf{return } \bot}$
        end if
        $\mathcal{D}_\mathsf{T} \leftarrow \mathcal{D}_\mathsf{T} \cup (A_* \oplus \Delta)$
        $\Delta \leftarrow \tilde{\pi}_{\mathsf{T},0}(A_* \oplus \Delta)$
        $\mathsf{T} \leftarrow 0^{\tau-3}$
    end if
    if $|A_*| > 0$ or $|T| = 0$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| \mathsf{noM} \| 11$ and $\mathsf{P}^*_a(W, Q)$
        if $(A_* \| 10^*) \oplus \Delta \in \mathcal{D}_\mathsf{T}$ and $\mathsf{P}^*_a(W, Q)$ then
            $\mathsf{bad} \leftarrow \mathsf{true}$
            $\boxed{\textbf{return } \bot}$
        end if
        $\mathcal{D}_\mathsf{T} \leftarrow \mathcal{D}_\mathsf{T} \cup ((A_* \| 10^*) \oplus \Delta)$
        $\Delta \leftarrow \tilde{\pi}_{\mathsf{T},0}((A_* \| 10^*) \oplus \Delta)$
        $\mathsf{T} \leftarrow 0^{\tau-3}$
    end if
    for $i \leftarrow 1$ to $m$ do
        $\mathsf{T} \leftarrow \mathsf{T} \| 001$
        $M_i \leftarrow \pi^{-1}_{\mathsf{T},0}(C_i \oplus \Delta, 0) \oplus \Delta$
        if $M_i \oplus \Delta \in \mathcal{D}_\mathsf{T}$ and $\mathsf{P}^i_m(W, Q)$ then
            $\mathsf{bad} \leftarrow \mathsf{true}$
            $\boxed{\textbf{return } \bot}$
        end if
        $\mathcal{D}_\mathsf{T} \leftarrow \mathcal{D}_\mathsf{T} \cup (M_i \oplus \Delta)$
        $\Delta \leftarrow \tilde{\pi}_{\mathsf{T},1}(\pi^{-1}_{\mathsf{T},0}(C_i \oplus \Delta, 0))$
        $\mathsf{T} \leftarrow 0^{\tau-3}$
    end for
    if $|T| = n$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| 100$
    else if $|T| > 0$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| 100$
    else
        if $C_* \neq \Delta$ then return $\bot$
        return $\varepsilon$
    end if
    $M_* \leftarrow \tilde{\pi}^{-1}_{\mathsf{T},0}(C_* \oplus \Delta) \oplus \Delta$
    $T' \leftarrow \tilde{\pi}_{\mathsf{T},1}(M_* \oplus \Delta)$
    $T' \leftarrow \mathsf{left}_{|T|}(T')$; $P \leftarrow \mathsf{right}_{n-|T|}(M_*)$
    if $T' \neq T$ return $\bot$
    if $P \neq \mathsf{left}_{n-|T|}(10^{n-1})$ return $\bot$
    return $M_1 \| \ldots \| M_m \| \mathsf{left}_{|T|}(M_*)$
```

Fig. 14: The games G_2 and G_3 for bounding $\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{SAEF}[\tilde{\pi}_0, \tilde{\pi}_1]}$ (continued from Figure 13). The game G_2 does *not* contain the boxed statements, while G_3 does. The predicates P_A and P_M are defined in Section C.

D RPAEF confidentiality and Integrity Proofs

Proof. Below we prove the confidentiality and authenticity of the RPAEF mode. As for PAEF, we replace F with a pair of independent random tweakable permutations π_0 and π_1, obtaining

$$\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{RPAEF}[\mathsf{F},\nu]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{B}) + \mathbf{Adv}^{\mathbf{priv}}_{\mathrm{RPAEF}[(\pi_0,\pi_1),\nu]}(\mathcal{A})$$

and

$$\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{RPAEF}[\mathsf{F},\nu]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{C}) + \mathbf{Adv}^{\mathbf{auth}}_{\mathrm{RPAEF}[(\pi_0,\pi_1),\nu]}(\mathcal{A}).$$

We have $\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{RPAEF}[(\pi_0,\pi_1),\nu]}(\mathcal{A}) = 0$; similarly as in PAEF, every ciphertext block and every tag is produced with a fresh tweak, and thus uniformly distributed.

For authenticity, we analyse the probability of forgery for an adversary that makes a single decryption query against $\mathrm{RPAEF}[(\pi_0,\pi_1),\nu]$ and then use a result of Bellare [20] to obtain a bound for multiple decryption queries (against $\mathrm{RPAEF}[(\pi_0,\pi_1),\nu]$).

As before, we will denote the encryption queries of $\mathcal{A}$ and the corresponding replies as (N^i, A^i, M^i) and C^i for $i = 1,\ldots,q$, where q is the number of encryption queries made by $\mathcal{A}$. For each i we let $C^i_1,\ldots,C^i_{m_i},C^i_*,T = \mathsf{csplit\text{-}b}_n(C^i)$. We let (N, A, C) denote the only decryption query of $\mathcal{A}$ and we let $C_1,\ldots,C_m,C_*,T = \mathsf{csplit\text{-}b}_n(C)$. We further let $c = t - (\nu + n + 3)$.

Similarly to PAEF, there are two base cases to consider; forging with a fresh nonce N (which is equivalent to guessing the value of a uniform string of n bits) succeeding with probability 2^{-n}, and forging with a reused N.

If N is reused, i.e. if $N = N^i$ for some $N^i \in \{N^1,\ldots,N^q\}$, then we perform a case analysis, disregarding all encryption queries except the i^{th}, because their ciphertetxts are computed using independent random permutations. Every case assumes the negation of all previous case-conditions. We note that for the forgery to be valid, we must have $(A^i, C^i) \neq (A, C)$.

Case 1, $|C|_n \neq |C^i|_n$: We have several subcases.

- If $|C| = n$, then C is equal to a xor-sum of $\pi_{\mathsf{T},1}$ images from the associated data (denoted as S_A in Figure 9), such that we can possibly have $A^i = A$. Because we must have $|M^i| > 0$, the xor-sum S_{A^i} computed in the i^{th} encryption query is masked with uniform bits C^i_*. The probability of a successful forgery which is equivalent to guessing S_A, is 2^{-n}.
- When $|C| > n$, regardless if C has more or less blocks than C^i, the forgery attempt succeeds with probability 2^{-n} as C^* and T are processed with a tweak $\mathsf{T} = N\|110\|\langle m+1\rangle_c\|S$ (respectively $\mathsf{T} = N\|111\|\langle m+1\rangle_c\|S$) that was not used before.

The probability of a successful forgery in **Case 1** is at most 2^{-n}.

In the following cases, $|C|_n = |C^i|_n$.

Case 2, $|C| = |C^i| = n$: In this special case, $C^i_* = S^i_A$ and $C_* = S_A$ (referring to Figure 9). Valid forgery requiring $A \neq A^i$, one of the following conditions must be true:

- If $|A|_n \neq |A^i|_n$, then forging is equivalent to guessing the image $\pi_{(N\|b\|\langle a+1\rangle_c\|0^n),0}(A_*)$ with $b \in \{001, 011\}$, such that this tweak has not been used before, succeeding with probability 2^{-n}.
- Otherwise, if $|A_*| = n$ and $|A^i_*| < n$, or $|A_*| < n$ and $|A^i_*| = n$, then as in the previous subcase, forging is equivalent to guessing the image created with a fresh tweak (thanks to the domain separation flag), succeeding with probability 2^{-n}.
- Otherwise, if there is $1 \leq j \leq a$ such that $A_j \neq A^i_j$, then forging is equivalent to successfully guessing the image $\pi_{(N\|000\|\langle j\rangle_c\|0^n),0}(A_j)$, which has not been sample before, succeeding with probability at most $1/(2^n - 1)$.
- Finally we can have $A_* \neq A^i_*$, in which case forging is equivalent to guessing the image $\pi_{(N\|b\|\langle a+1\rangle_c\|0^n),0}(A_*)$ with $b \in \{001, 011\}$, which succeeds with probability at most $1/(2^n - 1)$

The probability of forging in this case is at most $1/(2^n - 1)$.

In the following cases, we have $|C|_n = |C^i|_n > 1$, so the tag T is non-empty, and the final ciphertext block C_* and the tag T are processed by an F call with a tweak containing the checksum S in its last n bits.

Case 3, $|T| = n$ and $|T^i| < n$ or $|T| < n$ and $|T^i| = n$: In this case, the final verification check of C_*, T will be done using a tweak $N\|b\|\langle m+1\rangle c\|S$ with $b \in \{101, 111\}$ that has not been used before (thanks to the flag b). The forgery succeeds with probability 2^{-n}.

In the following cases, the tweak used to process the final ciphertext block C_* and the tag T will contain the same nonce, the same 3-bit flag, and the same counter as the tweak that produced C_*^i, T^i. In all cases, the checksum in the forgery attempt S may or may not collide with the checksum S^i.

Case 4, $A \neq A^i$: $\mathcal{A}$ can succeed in forging either by forcing a collision $S = S^i$ (in which case it can set $(C_*, T) = (C_*^i, T^i)$), or else if $S \neq S^i$ the forgery succeeds with probability 2^{-n}. The probability of $S = S^i$ is at most $1/(2^n - 1)$ by a similar argument as in **Case 2**, and the probability of forgery in this case is bounded by $2^{-n} + 1/(2^n - 1) \leq 2/(2^n - 1)$ by a union bound.

In the remaining cases, we have $A = A^i$.

Case 5, $|C| \neq |C^i|$ and $|T| < n$ and $|T^i| < n$: We consider two subcases.
- If $S = S^i$, both the encryption query and the decryption query use the same tweak T to process M_*^i and C_*, T, respectively. There are two conditions for the forgery to succeed. First, the preimage $X = \pi_{\mathsf{T},0}^{-1}(C_*)$ must be equal to $W\|10^{n-|T|-1} \neq M_*^i\|10^{n-|T^i|-1}$ (noting that the case condition implies $|T| \neq |T^i|$) for some $W \in \{0,1\}^{|T|}$. This is no easier than finding a fresh value whose preimage falls into a set of size $2^{|T|}$. With a single image of $\pi_{\mathsf{T},0}^{-1}$ already used, this succeeds with probability bounded by $(2^{|T|})/(2^n - 1)$. *Secondly*, the image $Y = \pi_{\mathsf{T},1}(X)$ must be equal to $T\|Z$ for some $Z \in \{0,1\}^{n-|T|}$, *conditioned on X having the correct format.* This is equivalent to guessing a fresh image under $\pi_{\mathsf{T},1}$ with $(n - |T|)$ free bits. As a single image of $\pi_{\mathsf{T},1}$ has been used already, this happens with probability at most $(2^{n-|T|})/(2^n - 1)$. The probability of a successful forgery in this subcase is therefore bounded by $(2^{|T|})/(2^n - 1) \cdot (2^{n-|T|})/(2^n - 1) = 2^n/(2^n - 1)^2$.
- If $S \neq S^i$, the forgery succeeds with probability 2^{-n}.

The probability of a forgery in this case is at most $2^n/(2^n - 1)^2$ (because the checksum (non-)collision is treated as subcases).

In the following cases, we have $|C| = |C^i|$.

Case 6, $|C| = |C^i|$ and $A = A^i$: In this case, there must be at least a single block of ciphertext where the two queries differ. We investigate the following subcases.
- If the forgery N, A, C differs from N, A^i, C^i only in $C_*\|T$, then , if we ran the decryption algorithm on N, A^i, C^i and N, A, C in parallel, the values S^i and S used on the line 38 of the decryption algorithm in Figure 5 would be the same, resulting in identical tweaks in the final F calls of both queries. The probability of a successful forgery is at $(2^n - 1)^{-1}$ if $|T| = n$ (inverse of $C_* \oplus S$ has not yet been sampled) and at most $2^n/(2^n - 1)^2$ otherwise (by a similar argument as in **Case 5**).
- If there is some $1 \leq j \leq m$ such that $C_j \neq C_j^i$, the adversary may reuse C_*^i, T^i if $S = S^i$, or the forgery is equivalent to guessing a random string otherwise. The latter succeeds with probability 2^{-n}. The collision in the former case occurs with probability at most $1/(2^n - 1)$, because the preimage $\pi_{(N\|100\|\langle j\rangle_c\|0^n),0}(C_j)$ is unknown.

The probability of a successful forgery in this case is bounded by $1/(2^n - 1) + 2^{-n} \leq 2/(2^n - 1)$ otherwise.

Thus a single forgery succeeds with probability no greater than $2/(2^n - 1)$. By applying the result of Bellare [20], we can bound the probability of a successful forgery among q_v decryption queries as $(2 \cdot q_v)/(2^n - 1)$. □

E Deterministic MiniAE

In this section, we demonstrate that when used in a minimalistic "mode" of operation, a secure forkcipher yields a miniature AE scheme for fixed-size messages, which achieves PRI security [55].

PRI security of an AEAD scheme. Informally speaking, the best possible security that an AEAD scheme *with a fixed stretch* can achieve is to be (computationally) indistinguishable from a random injection from $\mathcal{N}\times\mathcal{A}\times\mathcal{M}$ to $\mathcal{C}$, because any AE scheme that is correct, must also be injective. This intuition is formalized as follows. The advantage of an adversary $\mathcal{A}$ in distinguishing an AEAD scheme Π with ciphertext expansion τ from a random injection with the same signature is defined as

$$\mathbf{Adv}^{\mathbf{pri}}_{\Pi}(\mathcal{A}) = \Pr[\mathcal{A}^{\mathbf{pri\text{-}real}_\Pi} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathbf{pri\text{-}ideal}_\Pi} \Rightarrow 1]$$

with the games $\mathbf{pri\text{-}real}_\Pi$ and $\mathbf{pri\text{-}ideal}_\Pi$ defined in Figure 15.

```
proc initialize                         pri-real_Π
$K \leftarrow\$ \mathcal{K}$

proc Enc($N, A, M$)
return $\mathcal{E}(K, N, A, M)$

proc Dec($N, A, C$)
return $\mathcal{D}(K, N, A, C)$
```

```
proc initialize                         pri-ideal_Π
for $N, A \in \mathcal{N}\times\mathcal{A}$ do
    $f_{N,A} \leftarrow\$ \mathrm{Inj}(\tau)$

proc Enc($N, A, M$)
return $f_{N,A}(M)$

proc Dec($N, A, C$)
if $\exists M \in \mathcal{M}$ s.t. $f_{N,A}(M) = C$ then
    return $M$
else
    return $\perp$
```

Fig. 15: Pseudo-random injection (PRI) security games for a scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with ciphertext expansion τ.

Given a tweakable forkcipher F with $\mathcal{T} = \{0,1\}^t$ and a $1 \le \nu < t$, we define the AEAD scheme $\mathrm{MAE}[\mathsf{F},\nu] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ (as in "mini AE") with $\mathcal{K} = \{0,1\}^k$. The message space $\mathcal{M} = \{0,1\}^n$ is given by the block-size of F, the nonce space $\mathcal{N} = \{0,1\}^\nu$ and the AD space $\mathcal{A} = \{0,1\}^\alpha$ with $\alpha = t - \nu$, so the parameter ν allows to make a trade-off between the nonce and AD sizes. The ciphertext expansion is n. The encryption and the decryption algorithm are defined in Figure 16.

The MAE mode captures the immediate intuition behind the "AE-potential" of a forkcipher: just use the redundancy contained in the right output block as a "tag".

```
1: function $\mathcal{E}(K, N, A, M)$
2:     return $\mathsf{F}_K^{N\|A,\mathsf{b}} M$
3: end function
```

```
1: function $\mathcal{D}(K, N, A, C\|T)$
2:     $M, T' = \mathsf{F}^{-1}{}_K^{N\|A,0,\mathsf{b}}(C)$
3:     if $T = T'$ then return $M$
4:     return $\perp$
5: end function
```

Fig. 16: The $\mathrm{MAE}[\mathsf{F},\nu]$ AEAD scheme.

Security of MAE. We have the following statement about the security of MAE.

Theorem 5. *Let F be a tweakable forkcipher with $\mathcal{T} = \{0,1\}^t$, let $1 \le \nu < t$ and let $1 \le \tau \le n$. Then for adversary $\mathcal{A}$ whose queries lie in the proper domains of the encryption and decryption algorithms and who makes q encryption queries and q_v decryption queries such that $q + q_v \le 2^{n-1}$, we have*

$$\mathbf{Adv}^{\mathbf{pri}}_{\mathrm{MAE}[\mathsf{F},\nu,\tau]}(\mathcal{A}) \le \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{B}) + \frac{(q+q_v)^2}{2^n}$$

for some adversary $\mathcal{B}$ who makes at most twice as many queries in total as $\mathcal{A}$, and who runs in time given by the running time of $\mathcal{A}$ plus an overhead that is linear in the total number $\mathcal{A}$'s queries.

Proof. We first replace F by a pair of tweakable random permutations $\pi_0 = (\pi_{\mathsf{T},0} \leftarrow\$ \operatorname{Perm}(n))_{\mathsf{T}\in\{0,1\}^t}$ and $\pi_1 = (\pi_{\mathsf{T},1} \leftarrow\$ \operatorname{Perm}(n))_{\mathsf{T}\in\{0,1\}^t})$. Letting $\mathrm{MAE}[\pi_0, \pi_1]$ denote the MAE mode that uses π_0, π_1 instead of F, we have

$$\mathbf{Adv}^{\mathbf{pri}}_{\mathrm{MAE}[\mathsf{F}]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{B}) + \mathbf{Adv}^{\mathbf{pri}}_{\mathrm{MAE}[\pi_0,\pi_1]}(\mathcal{A})$$

by a similar argument as in the proof of Theorem 1. In the rest of the analysis, we will refer to $\mathrm{MAE}[\pi_0, \pi_1$ simply by Π.

For the rest of the analysis, we use the game Γ_0 and Γ_1 defined in Figure 17. We claim that $\Pr[\mathcal{A}^{\mathbf{pri\text{-}real}_\Pi} \to 1] = \mathcal{A}^{\Gamma_1} \to 1]$ and that $\Pr[\mathcal{A}^{\mathbf{pri\text{-}ideal}_\Pi} \to 1] = \Pr[\mathcal{A}^{\Gamma_0} \to 1]$, which yields

$$\mathbf{Adv}^{\mathbf{pri}}_{\mathrm{MAE}[\pi_0,\pi_1]}(\mathcal{A}) \leq \Pr[\mathcal{A}^{\Gamma_1} \to 1] - \Pr[\mathcal{A}^{\Gamma_0} \to 1].$$

It is easy to verify the latter equality; with the boxed statements removed, the code in Figure 17 implements a family of random injections indexed by (N, A) by lazy sampling. In particular, note that the probability that non-trivial decryption query succeeds in Γ_0 is $\Pr[b' = 1 \wedge b'' = 1] = 1 \cdot (2^n - |f_{N,A}|)^2/(2^n - |f_{N,A}|) \cdot (2^{2n} - |f_{N,A}|)$ which is equal to the probability of finding a preimage of a random injection for which $|f_{N,A}|$ range points with known images (or known to have no preimages).

The former equality holds, because the framed lines in game Γ_1 make sure that f does in fact implement MAE based on a family of *pairs* of random permutations indexed by (N, A). First, there is an implicit bijection between $\{0,1\}^t$ and $\mathcal{N} \times \mathcal{A}$, so they are interchangeable. Then, the conditions of lines 7 and 7 make sure that the functions $\pi_{(N,A),0}$ and $\pi_{(N,A),1}$ defined by $\pi_{(N,A),0}(M) = \mathsf{left}_n(f(m))$ and $\pi_{(N,A),1}(M) = \mathsf{right}_n(f(m))$. The framed lines following the line 8 make sure that the distribution of ciphertext is the same as when produced by a pair of random permutations. The boxed statement after line 10 rejects ciphertexts that can never be produced by $\mathrm{MAE}[\pi_0, \pi_1]$. The boxed statements after line 20 make sure that the probability that a non-trivial decryption query succeeds is the same as for $\mathrm{MAE}[\pi_0, \pi_1]$.

In addition, the games Γ_0 and Γ_1 are identical until bad, so we have $\Pr[\mathcal{A}^{\Gamma_1} \to 1] - \Pr[\mathcal{A}^{\Gamma_0} \to 1] \leq \Pr[\mathcal{A}^{\Gamma_1}$ sets $\mathsf{bad}]$ by the Fundamental lemma of gameplaying [25]. We define bad^i for $i = 1, \ldots, q + q_v$ to be the event bad is set to true in the i^{th} query made by the adversary. We further let bad^i_1 denote the event that bad^i is true due to line 8, bad^i_2 denote the event that bad^i is true due to line 10 and bad^i_3 denote the event that bad^i is true due to line 20. Then we have that $\Pr[\mathcal{A}^{\Gamma_1}$ sets $\mathsf{bad}] \leq \sum_{i=1}^{3} \sum_{j=1}^{q+q_v} \Pr[\mathsf{bad}^i_j]$.

We have that

$$\Pr[\mathsf{bad}^i_1] \leq (i-1) \cdot \frac{2 \cdot 2^n - 1}{2^{2n} - i + 1} \leq (i-1) \cdot \frac{2^{n+1}}{2^n(2^n - 1)} \leq \frac{2 \cdot (i-1)}{2^n - 1}$$

because if bad was not set previously, there are at most $i - 1$ elements in both $\mathcal{R}_l(f_{N,A})$ and $\mathcal{R}_r(f_{N,A})$ for any (N, A), and for each X element of either $\mathcal{R}_l(f_{N,A})$ or $\mathcal{R}_r(f_{N,A})$, there are at most $2^n - 1$ elements of $\{0,1\}^{2n} \backslash \mathcal{R}_\perp(f_{N,A})$ that collide with X on their n leftmost, or respectively rightmost bits. The rest follows from the assumption $(q + q_v) \leq 2^n$ implied by $q + q_v \leq 2^{n-1}$. Summing over i, we get that $\sum_{i=1}^{q+q_v} \Pr[\mathsf{bad}^i_1] \leq 2 \cdot (q + q_v)^2/2 \cdot (2^n - 1)$.

Then, we have that

$$\Pr[\mathsf{bad}^i_2] \leq \cdot \frac{2^n}{2^{2n} - i + 1} \leq \frac{2^n}{2^n(2^n - 1)} \leq \frac{1}{2^n - 1}$$

because in the i^{th} query, we have $0 \leq |f_{N,A}| \leq i - 1$ for any (N, A), and this determines the parameter of the Bernouli variable which can set bad. The inequality follows using the assumption $(q + q_v) \leq 2^n$ implied by $(q + q_v) \leq 2^{n-1}$. Summing over i, we get that $\sum_{i=1}^{q+q_v} \Pr[\mathsf{bad}^i_2] \leq (q + q_v)/(2^n - 1)$.

Finally, we have that

$$\Pr[\mathsf{bad}^i_3] = \frac{1}{2^n - |f_{N,A}|} \cdot \left(1 - \frac{(2^n - |f_{N,A}|)^2}{2^{2n} - |f_{N,A}|}\right) \leq \frac{1}{2^n - |f_{N,A}|} \leq \frac{1}{2^n - (i-1)} \leq \frac{1}{2^{n-1}}$$

because bad^i_j occurs in the i^{th} query if and only if $b' = b'' = 1$. The final inequality then follows from the assumption $(q + q_v) \leq 2^{n-1}$. Summing over i, we get that $\sum_{i=1}^{q+q_v} \Pr[\mathsf{bad}^i_3] \leq (q + q_v)/(2^{n-1})$.

The claimed bound is obtained by adding up the sums $\sum_{j=1}^{q+q_v} \Pr[\mathsf{bad}^i_j]$ for $j = 1, 2, 3$.

```
proc initialize
    for $N, A \in \mathcal{N} \times \mathcal{A}$ do
        $f_{N,A} = \emptyset$
    end for
    $\mathsf{bad} \leftarrow \mathsf{false}$

proc $\mathrm{Enc}(N, A, M)$
    if $\exists\ C$ s.t. $(M, C) \in f_{N,A}$ then
        return $C$
    end if
    $C \leftarrow\$ \{0,1\}^{2n} \backslash \mathcal{R}_{\perp}(f_{N,A})$
    $C_l \leftarrow \mathsf{left}_n(C)$; $C_r \leftarrow \mathsf{right}_n(C)$
    if $C_l \in \mathcal{R}_l(f_{N,A})$ or $C_r \in \mathcal{R}_r(f_{N,A})$ then
        $\mathsf{bad} \leftarrow \mathsf{true}$
        if $C_l \in \mathcal{R}_l(f_{N,A})$ then
            $\boxed{X \leftarrow\$ \{0,1\}^n \backslash \mathcal{R}_l(f_{N,A})}$
            $\boxed{C \leftarrow X \| C_r}$
        end if
        if $C_r \in \mathcal{R}_r(f_{N,A})$ then
            $\boxed{X \leftarrow\$ \{0,1\}^n \backslash \mathcal{R}_r(f_{N,A})}$
            $\boxed{C \leftarrow C_l \| X}$
        end if
    end if
    $f_{N,A} \leftarrow f_{N,A} \cup \{(M, C)\}$
    return $C$

proc $\mathrm{Dec}(N, A, C \| T)$
    if $\exists\ M$ s.t. $(M, C) \in f_{N,A}$ then
        return $M$
    end if
    $M \leftarrow\$ \{0,1\}^n \backslash \mathcal{D}(f_{N,A})$
    $C_l \leftarrow \mathsf{left}_n(C)$; $C_r \leftarrow \mathsf{right}_n(C)$
    if $C_l \in \mathcal{R}_l(f_{N,A})$ or $C_r \in \mathcal{R}_r(f_{N,A})$ then
        $b \leftarrow\$ \mathrm{Be}\left(\frac{2^n - |f_{N,A}|}{2^{2n} - |f_{N,A}|}\right)$
        if $b = 1$ then
            $\mathsf{bad} \leftarrow \mathsf{true}$
            $\boxed{\textbf{return } \perp}$
            $f_{N,A} \leftarrow f_{N,A} \cup \{(M, C)\}$
            return $M$
        end if
    else
        $b' \leftarrow\$ \mathrm{Be}\left(\frac{1}{2^n - |f_{N,A}|}\right)$
        $b'' \leftarrow\$ \mathrm{Be}\left(\frac{(2^n - |f_{N,A}|)^2}{2^{2n} - |f_{N,A}|}\right)$
        if $b' = 1$ then
            if $b'' = 0$ then
                $\mathsf{bad} \leftarrow \mathsf{true}$
                $\boxed{f_{N,A} \leftarrow f_{N,A} \cup \{(M, C)\}}$
                $\boxed{\textbf{return } M}$
            else
                $f_{N,A} \leftarrow f_{N,A} \cup \{(M, C)\}$
                return $M$
            end if
        end if
    end if
    $f_{N,A} \leftarrow f_{N,A} \cup \{(\perp, C)\}$
    return $\perp$
```

Fig. 17: The games $\varGamma_0$ and $\varGamma_1$ for bounding $\mathbf{Adv}^{\mathbf{pri}}_{\mathrm{MAE}[\pi_0,\pi_1]}$. The game $\varGamma_1$ does *not* contain the boxed statements, while $\varGamma_0$ does. The games implement the (partially defined) injective funcitons $f_{N,A} : \{0,1\}^n \to \{0,1\}^{2n}$ as initially-empty sets of preimage-image pairs; a pair $(\perp, C)$ signifies that C has no premiage under the given function. We define the domain, range, and the "left" and "right" range of any $f_{N,A}$ as $\mathcal{D}(f_{N,A}) = \{M \in \{0,1\}^n | \exists (M, C) \in f_{N,A}\}$, $\mathcal{R}(f_{N,A}) = \{C \in \{0,1\}^{2n} | \exists (M, C) \in f_{N,A} \text{ s.t. } M \neq \perp\}$, $\mathcal{R}_l(f_{N,A}) = \{L \in \{0,1\}^n | \exists \text{ some } L \| X \in \mathcal{R}(f_{N,A})\}$ and $\mathcal{R}_r(f_{N,A}) = \{R \in \{0,1\}^n | \exists \text{ some } X \| R \in \mathcal{R}(f_{N,A})\}$. We additionally define the extended range $\mathcal{R}_{\perp}(f_{N,A}) = \{C \in \{0,1\}^{2n} | \exists (M, C) \in f_{N,A}\}$. $\mathrm{Be}\,(p)$ denotes a random variable with Bernouli distribution with $\Pr[\mathrm{Be}\,(p) = 1] = p$.

F Description and Security Analysis of ForkSkinny

F.1 Detailed Description of ForkSkinny

ForkSkinny is based on `SKINNY`, a family of lightweight tweakable block ciphers that was presented at Crypto 2016 by Beierle et al. [17]. The 6 variants described in [17] differ from the block size ($n = 64$ or $n = 128$ bits) and from the tweakey size ($z \times n$ bits, where z is either 1, 2 or 3). They are denoted as `SKINNY`-n-zn.

In a similar way, by ForkSkinny-n-zn we denote one variant of our cipher with a block size of n bits (either 64 or 128) and of $z \times n$ tweakey bits. We further consider versions where the tweakey size is not a multiple of the block size n. In general, ForkSkinny-n-t here will denote the ForkSkinny with n-bit block and t-bit tweakey. As detailed in Section 4 The two branches of ForkSkinny produce two ciphertexts each of length n bits.

The ciphers have a Substitution-Permutation-Network (SPN) structure, and the internal state is organised as a 4×4 matrix, where each cell is either a byte (when $n = 128$) or a nibble (when $n = 64$). The n-bit messages are loaded row-wisely in the internal state IS, as depicted below.

$$IS = \begin{bmatrix} m_0 & m_1 & m_2 & m_3 \\ m_4 & m_5 & m_6 & m_7 \\ m_8 & m_9 & m_{10} & m_{11} \\ m_{12} & m_{13} & m_{14} & m_{15} \end{bmatrix}$$

In the following, we review the most important aspects of the design of `SKINNY`, and refer to the original `SKINNY` specification [17] for more details.

Round Function ForkSkinny round function (see Figure 18) only differs slightly from the `SKINNY` one: it reuses the 5 operations described in `SKINNY`, but considers different round constants in the `AddConstants` step to take into account the fact that more rounds are iterated.

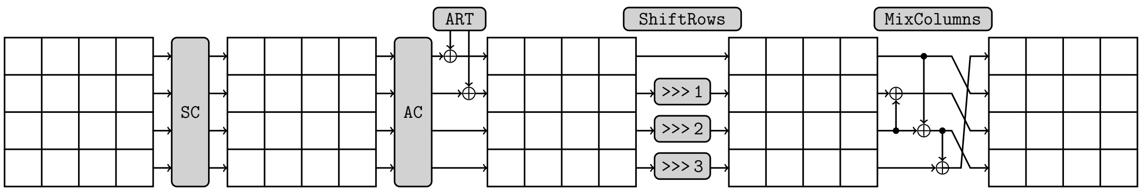

Fig. 18: Structure of every round in ForkSkinny, made of the five operations `SubCells` (`SC`), `AddConstants` (`AC`), `AddRoundTweakey` (`ART`), `ShiftRows` (`SR`) and `MixColumns` (`MC`), as it is done in `SKINNY`. (Figure credits: [36]).

The round function operations are the following (see Figure 18):

- `SubCells` (`SC`): each of the 16 words of the internal state is modified by a 4×4 (if $n = 64$) or 8×8 Sbox (if $n = 128$). The definition of the Sboxes is recalled below. ForkSkinny reuses the Sboxes of `SKINNY` without any change.
- `AddConstants` (`AC`): A LFSR is used to produce constants that are added in the first 3 cells of the first column. Since in total ForkSkinny iterates more rounds than `SKINNY`, we changed the definition of the LFSR to avoid repetitions.
- `AddRoundTweakey` (`ART`): Exactly as in `SKINNY`, the addition of the tweakey material is done in the first two lines of the state.
- `ShiftRows` (`SR`): The second line of the internal state is right rotated by 1 cell, the third line is right rotated by 2 cells, and the last line is right rotated by 3 cells.
- `MixColumns` (`MC`): This operation modifies each column by multiplying it with a binary matrix M, given by:

$$M = \begin{pmatrix} 1\,0\,1\,1 \\ 1\,0\,0\,0 \\ 0\,1\,1\,0 \\ 1\,0\,1\,0 \end{pmatrix}$$

Note that all the rounds are identical, and in particular that no whitening keys are used. The Sboxes are defined as follows:

```
/* SKINNY-64 Sbox */
const unsigned char S4[16] = {12,6,9,0,1,10,2,11,3,8,5,13,4,14,7,15};

/* SKINNY-128 Sbox */
uint8_t S8 [256] = {
0x65, 0x4c, 0x6a, 0x42, 0x4b, 0x63, 0x43, 0x6b, 0x55, 0x75, 0x5a, 0x7a, 0x53, 0x73, 0x5b, 0x7b,
0x35, 0x8c, 0x3a, 0x81, 0x89, 0x33, 0x80, 0x3b, 0x95, 0x25, 0x98, 0x2a, 0x90, 0x23, 0x99, 0x2b,
0xe5, 0xcc, 0xe8, 0xc1, 0xc9, 0xe0, 0xc0, 0xe9, 0xd5, 0xf5, 0xd8, 0xf8, 0xd0, 0xf0, 0xd9, 0xf9,
0xa5, 0x1c, 0xa8, 0x12, 0x1b, 0xa0, 0x13, 0xa9, 0x05, 0xb5, 0x0a, 0xb8, 0x03, 0xb0, 0x0b, 0xb9,
0x32, 0x88, 0x3c, 0x85, 0x8d, 0x34, 0x84, 0x3d, 0x91, 0x22, 0x9c, 0x2c, 0x94, 0x24, 0x9d, 0x2d,
0x62, 0x4a, 0x6c, 0x45, 0x4d, 0x64, 0x44, 0x6d, 0x52, 0x72, 0x5c, 0x7c, 0x54, 0x74, 0x5d, 0x7d,
0xa1, 0x1a, 0xac, 0x15, 0x1d, 0xa4, 0x14, 0xad, 0x02, 0xb1, 0x0c, 0xbc, 0x04, 0xb4, 0x0d, 0xbd,
0xe1, 0xc8, 0xec, 0xc5, 0xcd, 0xe4, 0xc4, 0xed, 0xd1, 0xf1, 0xdc, 0xfc, 0xd4, 0xf4, 0xdd, 0xfd,
0x36, 0x8e, 0x38, 0x82, 0x8b, 0x30, 0x83, 0x39, 0x96, 0x26, 0x9a, 0x28, 0x93, 0x20, 0x9b, 0x29,
0x66, 0x4e, 0x68, 0x41, 0x49, 0x60, 0x40, 0x69, 0x56, 0x76, 0x58, 0x78, 0x50, 0x70, 0x59, 0x79,
0xa6, 0x1e, 0xaa, 0x11, 0x19, 0xa3, 0x10, 0xab, 0x06, 0xb6, 0x08, 0xba, 0x00, 0xb3, 0x09, 0xbb,
0xe6, 0xce, 0xea, 0xc2, 0xcb, 0xe3, 0xc3, 0xeb, 0xd6, 0xf6, 0xda, 0xfa, 0xd3, 0xf3, 0xdb, 0xfb,
0x31, 0x8a, 0x3e, 0x86, 0x8f, 0x37, 0x87, 0x3f, 0x92, 0x21, 0x9e, 0x2e, 0x97, 0x27, 0x9f, 0x2f,
0x61, 0x48, 0x6e, 0x46, 0x4f, 0x67, 0x47, 0x6f, 0x51, 0x71, 0x5e, 0x7e, 0x57, 0x77, 0x5f, 0x7f,
0xa2, 0x18, 0xae, 0x16, 0x1f, 0xa7, 0x17, 0xaf, 0x01, 0xb2, 0x0e, 0xbe, 0x07, 0xb7, 0x0f, 0xbf,
0xe2, 0xca, 0xee, 0xc6, 0xcf, 0xe7, 0xc7, 0xef, 0xd2, 0xf2, 0xde, 0xfe, 0xd7, 0xf7, 0xdf, 0xff
};
```

Round Constants. As explained in Section 4, we use 7-bit round constants. For completeness Table 2 give the value used in each round.

Table 2: Constants used in ForkSkinny.

Rounds	Constants
1 - 16	01,03,07,0F,1F,3F,7E,7D,7B,77,6F,5F,3E,7C,79,73
17 - 32	67,4F,1E,3D,7A,75,6B,57,2E,5C,38,70,61,43,06,0D
33 - 48	1B,37,6E,5D,3A,74,69,53,26,4C,18,31,62,45,0A,15
49 - 64	2B,56,2C,58,30,60,41,02,05,0B,17,2F,5E,3C,78,71
65 - 80	63,47,0E,1D,3B,76,6D,5B,36,6C,59,32,64,49,12,25
81 - 87	4A,14,29,52,24,48,10

Tweakey. Again, the tweakey schedule works similarly to what is done in SKINNY, that is based on the TWEAKEY framework [37]. The first operation consists in filling the tweakey state, which is view as a collection of 4×4 matrices of the same cell-size as the considered internal state. If the cipher uses material that is not the key (that is, strictly a tweak), this one is positioned first in $TK1$, row wisely, and then is set the key (if that leaves an incomplete matrix we fill it with zeros). We denote these matrices by $TK1$, $TK2$ and $TK3$ (if any). As suggested in the SKINNY specification, when there is some tweak material, we add an extra 1 in the constant matrix from AddConstants, every round at line 0, column 2, to the second bit).

If the tweakey size is not a multiple of the state size but leaves 2 empty rows in the last tweakey matrix (as it is the case for ForkSkinny-128-192), instead of filling the remaining cells with zeros we simply don't use these cells, which allows to save some memory, some LFSR applications and also some XORs.

As can be seen on Figure 19, during the AddRoundTweakey step the first two rows of each tweakey are exclusive-ored together and then to the internal state. To update the tweakey arrays for the next round, each tweakey word is first modified by a cell-permutation P_T, given by:

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

and which effect on the cell positioning is as follows:

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix} \xrightarrow{P_T} \begin{bmatrix} 9 & 15 & 8 & 13 \\ 10 & 14 & 12 & 11 \\ 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \end{bmatrix}$$

Each cell (except the ones in TK1) is then linearly modified by a LFSR, following the definitions given in Table 3.

Table 3: LFSR used to update $TK2$ and $TK3$.

TK	cell size	LFSR
TK2	4	$(x_3\|\|x_2\|\|x_1\|\|x_0) \rightarrow (x_2\|\|x_1\|\|x_0\|\|x_3 \oplus x_2)$
	8	$(x_7\|\|x_6\|\|x_5\|\|x_4\|\|x_3\|\|x_2\|\|x_1\|\|x_0) \rightarrow (x_6\|\|x_5\|\|x_4\|\|x_3\|\|x_2\|\|x_1\|\|x_0\|\|x_7 \oplus x_5)$
TK3	4	$(x_3\|\|x_2\|\|x_1\|\|x_0) \rightarrow (x_0 \oplus x_3\|\|x_3\|\|x_2\|\|x_1)$
	8	$(x_7\|\|x_6\|\|x_5\|\|x_4\|\|x_3\|\|x_2\|\|x_1\|\|x_0) \rightarrow (x_0 \oplus x_6\|\|x_7\|\|x_6\|\|x_5\|\|x_4\|\|x_3\|\|x_2\|\|x_1)$

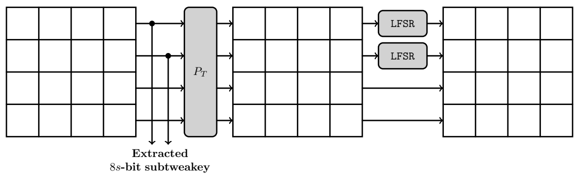

Fig. 19: Tweakey schedule of ForkSkinny, replicating the one of SKINNY. (Figure credits: [36])

We further detail here the other proposed variants given in Table 1:

ForkSkinny-*64-192:* This member of ForkSkinny has block size $n = 64$ and tweakey size $t = 3n$ bits. The 192-bit tweakey contains 64-bit tweak and the rest are key bits.

ForkSkinny-*128-192:* This has block size $n = 128$ and tweakey size $t = 3n/2$ bits. The 192-bit tweakey contains 64-bit tweak and the rest are key bits. Note that the design of SKINNY allows to use tweakey such that $n < t < 2n$. In such cases, the $2n - t$ bits of the tweakey are set to 0.

ForkSkinny-*128-256:* For this version of ForkSkinny we use $n = 128$ with tweakey size $t = 2n$. The 256-bit tweakey contains 128 bits of tweak and the rest are key bits.

ForkSkinny-*128-288:* For this version of ForkSkinny we use $n = 128$, with tweakey size $t = 9n/4$. The 288-bit tweakey contains 160 bits of tweak and the rest are key bits. Note that the design of SKINNY allows to use tweakey such that $2n < t < 3n$. In such cases, SKINNY proposal recommends to set the $3n - t$ bits of the tweakey to 0.

ForkSkinny-*128-384:* This member also has a block and a key of 128 bits. It uses 3 blocks of tweakey ($t = 3n$).

G Security Analysis of **ForkSkinny**

G.1 Arguments deduced from the Security of SKINNY

As noted previously, the security analyses of SKINNY directly transfer to ForkSkinny in the scenario where an attacker try to attack the cipher from the knowledge of both M and C_0. Consequently, to justify the security of this part of our construction we give an overview of the main attacks published so far: Table 4 details how many rounds can be reached together with the complexities of the attacks (note that we focus our review on the versions of SKINNY with the same parameters as in our ForkSkinny candidates).

Table 4: Complexities of the main previous cryptanalyses of SKINNY-64-192, SKINNY-128-256 and SKINNY-128-384. The letters indicate if it is in the Related (R) or Single (S) tweakey scenario.

Version	Technique	Rounds	Time	Data	Memory	ref.
SKINNY-64-192	Rect.(R)	27/40	$2^{165.5}$	$2^{63.5}$	2^{80}	[44]
SKINNY-64-192	Impossib.(S)	22/40	$2^{183.97}$	$2^{47.84}$	$2^{74.84}$	[61]
SKINNY-128-256	Impossib.(R)	23/48	$2^{251.47}$	$2^{124.47}$	2^{248}	[44]
SKINNY-128-256	Impossib.(S)	20/48	$2^{245.72}$	$2^{92.1}$	$2^{147.1}$	[61]
SKINNY-128-384	Rect.(R)	27/56	2^{331}	2^{123}	2^{155}	[44]
SKINNY-128-384	Impossib.(S)	22/56	$2^{373.48}$	$2^{92.22}$	$2^{147.22}$	[61]
SKINNY-128-384	DS-MITM.(S)	22/56	$2^{382.46}$	2^{96}	$2^{330.99}$	[57]

Other previous works discussed distinguishers only, without converting them into attacks. We summarize them in Table 5.

Table 5: Probabilities of the main previous distinguishers of SKINNY-64-192, SKINNY-128-256 and SKINNY-128-384. The letters indicate if it is in the Related (R) or Single (S) tweakey scenario.

Version	Type of distinguisher	Rounds	Probability	ref.
SKINNY-64-192	Boomerang (R)	22/40	$2^{-42.98}$	[58]
SKINNY-64-192	Differential (S)	20/40	$2^{-176.74}$	[12]
SKINNY-64	Truncated (S)	10/40	2^{-40}	[47]
SKINNY-64	Integral (S)	10/40	n/a	[68]
SKINNY-64	zero-correlation (S)	10/40	n/a	[56]
SKINNY-128-256	Boomerang (R)	18/48	$2^{-77.83}$	[58]
SKINNY-128	zero-correlation (S)	10/48	n/a	[56]
SKINNY-128-384	Boomerang (R)	22/56	$2^{-48.30}$	[58]
SKINNY-128	zero-correlation (S)	10/56	n/a	[56]

We also recall in Table 6 the bounds on the number of active Sboxes that were provided in the SKINNY specification.

G.2 Truncated Attacks

Truncated attacks are a variant of differential attacks where an attacker focuses on the activity pattern of differences instead of on their exact value. In most cases, these patterns correspond to stating which Sbox-size words are inactive and which are potentially active. A truncated differential can be easily used in an attack if its probability is higher than the probability to observe such an input and output patterns for a random permutation.

The resistance of SKINNY against truncated differential attacks has been studied in a recent ePrint report that uses Milp techniques [47]. The authors proved that the best truncated differential trails existing on 10-round SKINNY-64 have a probability of 2^{-40}. This implies that on 20 rounds

Table 6: Lower bounds on the number of active Sboxes in SKINNY, in the single key (SK) and Related-tweakey (TK1, TK2 and TK3) models, as given in [17].

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
SK	1	2	5	8	12	16	26	36	41	46	51	55	58	61	66
TK1	0	0	1	2	3	6	10	13	16	23	32	38	41	45	49
TK2	0	0	0	0	1	2	3	6	9	12	16	21	25	31	35
TK3	0	0	0	0	0	0	1	2	3	6	10	13	16	19	24

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
SK	75	82	88	92	96	102	108	(114)	(116)	(124)	(132)	(138)	(136)	(148)	(158)
TK1	54	59	62	66	70	75	79	83	85	88	95	102	(108)	(112)	(120)
TK2	40	43	47	52	57	59	64	67	72	75	82	85	88	92	96
TK3	27	31	35	43	45	48	51	55	58	60	65	72	77	81	85

there are no truncated differential trails of probability higher than 2^{-80}, so no straightforward distinguisher of this type can be deduced for 20 rounds. In a similar way to what we did for simple differentials, these results can be used to prove the resistance of ForkSkinny. The result in [47] combined with the large number of rounds of the instances we consider (our proposal derived from SKINNY-64 has parameters $r_{\mathsf{init}} = 17$ and $r_0 = r_1 = 23$), make us confident that our proposals are immune to this type of attacks.

G.3 Impossible Differential

Impossible differential attacks [28, 39] make use of a couple of differences (α, β) that verifies that for all possible keys two messages with a Xor difference equal to α cannot produce two messages that differ by β after a given number of rounds r of encryption.

To turn this distinguisher into a key recovery, an attacker appends some rounds before and after the impossible differential. She then makes a guess on the value of some key bits to check if the differences α and β are observed together. If this is the case, the guess is wrong for sure (since it leads to a situation that is impossible), so the corresponding keys are discarded. Once the search space has been sufficiently reduced, the attack is usually finalised with an exhaustive search.

In case the impossible differential is of the truncated type, we can easily give an upper bound on its number of rounds. This study was provided in the SKINNY specification, where it was shown that a miss-in-the-middle (in the special case where the contradiction is that one cell is active for sure from one direction but inactive from the other direction) can at most reach 11 rounds in the single-tweakey model.

In following works, the study was extended to the related-tweakey scenario, and for this the number of rounds covered by the distinguisher was extended to 12 rounds for TK1, 14 rounds for TK2 and 16 rounds for TK3 [44].

What remains to be done is the study of the case where the impossible differential is positioned around the forking point. A good first estimate consists in looking at the single key truncated impossible differential case, where the contradiction comes from one active cell obtained from one direction and one inactive cell coming from the other direction. We start by looking for the maximum number of rounds for which one word at least remains inactive or active, both for the cases:

1. decryption rounds only (corresponding to going from C_0 or C_1 up to before the forking point)
2. decryption rounds followed by encryption rounds (corresponding to going from C_0 or C_1 and decrypting and then continuing over the forking point with encryption.)

To evaluate the second case, we look at all the possibilities for the number of rounds before the forking point.

The results are provided in Table 7. If we leave out the necessary requirement that the position of the active cell of one path has to correspond to the position of the inactive path of the other cell, we obtain that no truncated impossible differential can cover more than $7 + 5 = 6 + 6 = 12$ rounds.

Table 7: Maximum number of rounds covered with a truncated differential path until we lose all information.

information	case 1	case 2
inactive	5	6
active	6	7

Since this approximation (that is optimistic for the attacker) is close to what was obtained for SKINNY (and that SKINNY has comfortable security margins), we are confident about the resistance of ForkSkinny against this type of attacks.

In the related tweakey scenario, an attacker can easily increase the number of rounds of the distinguisher by creating blank rounds (that is with no differences at all), simply by choosing carefully the value of the tweakey difference. However, this trick is limited by the properties of SKINNY Tweakey Schedule, namely the $p-1$ cancellation property of [37]: only a single difference cancellation can happen every 15 rounds for TK2, and only two difference cancellations can happen for TK3. Since only half of the tweakey material is used every round this implies that at most 3 consecutive rounds with no tweakey differences can be constructed every 30 rounds for TK2, and 5 for TK3. Even in the case where these free rounds can be exploited both at the beginning and at the end, the securiy margins chosen in SKINNY are sufficient.

G.4 Boomerang Attacks

In the classical boomerang attack [62] the adversary produces a quartet of plaintexts/ciphertexts $\{(P_i)\}_{i=0}^{4}$ such that $\bigoplus P_i = 0$, satisfying $\bigoplus E(P_i) = 0$, where E is typically a block cipher. Boomerang attacks can also be adapted in the related-key model, which are known as the related-key boomerang attacks. The success of classic boomerang attacks depends on the probability of differential propagation in a block cipher. Usually a boomerang attack combines two high probability differentials which exist on reduced number of rounds. Suppose that in a block cipher two differentials exist with probabilities p and q on r_1 and r_2 round respectively. Then as a first approximation we can evaluate the probability of the boomerang distinguisher for $r_1 + r_2$ rounds of $E_{r_2} \circ E_{r_1}$ to p^2q^2, where E_r denotes r round of the encryption function E. In ForkSkinny such attacks can not be applied to the full version due to the large number of active Sboxes. The related-key boomerang attack is more relevant, since it may lead to a forgery attack against the AE scheme. In ForkSkinny, we can always find a difference between the round-tweakeys (immediately after the forking step) which are used in the two different branches of the forkcipher. Using such related round-tweakeys if an adversary can find RTK boomerang attack then it will lead to the forgery of the AE scheme. The idea of such attack is depicted in the Fig 20. Such an attack [16] was also found on an earlier forkcipher instantation. However, it is not possible to find a similar boomerang attack on ForkSkinny which may lead to forgery attack.

G.5 Meet-in-the-Middle Attack

In a (basic) Meet-in-the-Middle attack, the attacker looks for a decomposition of the cipher in two parts so that the computation of each part only requires a fraction of the master key. She then computes a part of the internal state from the plaintext up to the end of the first part of the cipher, and computes the same part from the ciphertext up to the beginning of the second part. The correct value for the guessed key bits is among the hypotheses that lead to a match. A good starting point to obtain a first approximation of the resistance of a cipher to Meet-in-the-Middle attacks consists in looking at its diffusion.

The diffusion of a cipher corresponds to the number of rounds d that are required for any input bit to influence all the bits of the internal state. In case the key size corresponds to the block size and that all the key material is used in every rounds, having a cipher with diffusion equal to d means that any output bit after d rounds is an expression depending on all the key bits, which prevents the previous MitM attacks when more than $(d-1)+(d-1)$ rounds are used.

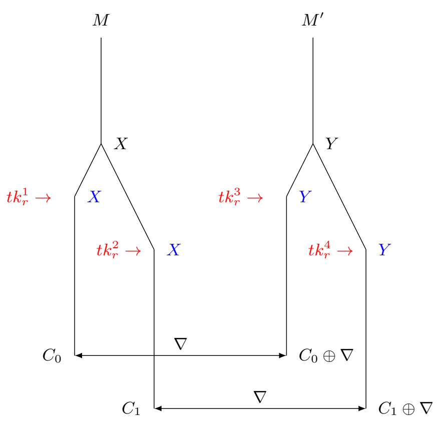

Fig. 20: RTK boomerang attack against forkcipher producing forgery for single block. Here tk_r^1 and tk_r^2 are two round keys after forking which introduce the tweakey difference. X and Y are states of the ForkSkinny after forking.

For SKINNY the diffusion delay is equal to 6, which would lead to a first estimate of 10 rounds for a partial matching. However, we must take into account the fact that only half of the tweakey material is used in each round, that the key addition is made after the non-linear operation and that the forking point in a reconstruction operation has a lower diffusion[12]), which adds some rounds to the first estimate.

In any case the obtained numbers are far from the chosen number of rounds. Moreover, recent results by Shi et al. [57] showed that with the improvements resulting from the Demirci-Selçuk techniques a total of 22 rounds out of the 56 of SKINNY-128-384 can be attacked. This supports that the number of rounds we chose are sufficient to thwart these types of attacks.

G.6 Integral Attack

ForkSkinny has two components ForkSkinny_0 and ForkSkinny_1 which produce C_0 and C_1, respectively from M. The security of these components follow directly from the analysis of SKINNY. The integral cryptanalysis against SKINNY can be directly applied to ForkSkinny_0 and ForkSkinny_1. SKINNY specification describes an integral distinguisher for 10 rounds. This can be applied to both reduced round ForkSkinny_0 and ForkSkinny_1. When applied to these components, the integral distinguisher can only cover less than r_{init} rounds prior to forking step. For the key recovery attack, it is possible to add 4 rounds to this integral distinguisher which allows an adversary to mount an attack against 14 rounds of SKINNY. Again, this key recovery attack can only cover less than r_{init} rounds, prior to forking in different ForkSkinny-n-t. In the reconstruction, an adversary has to cover at least 27 rounds in the encryption direction (following the forking point). Hence, it is not possible to use the integral attack against the full reconstruction in ForkSkinny. Complexities of the integral attacks against round reduced ForkSkinny remain the same as described in the specification of SKINNY [17].

Division Property. The division property was introduced as a generalization of the integral property by Todo [60]. SKINNY specification analyses show that the division property has significant margin against an attack that uses it. The generic analysis of SPN ciphers described in [60] leads to only 6 rounds of division property. Taking the resistance of SKINNY against division property into account, we are confident that ForkSkinny has a sufficient security margin against the same type of attacks.

G.7 Algebraic Attack

By following the analysis of SKINNY we can show that algebraic attacks pose no threats to full ForkSkinny. ForkSkinny uses the same Sboxes of sizes 4 bits and 8 bits with algebraic degree $a = 3$

[12] The diffusion delay could also increase if the forking point chains two tweakeys that depend on the same half of the tweakey material. To avoid this we opted for values of r_1 that are odd.

and $a = 6$, respectively, as in SKINNY. In a single key setting, 7 consecutive rounds of SKINNY have 26 active Sboxes, so for all variants of ForkSkinny, we obtain that every output bit after r rounds has an expected degree of at least $a \cdot 26 \cdot \lfloor \frac{r}{7} \rfloor \ggg n$. As it has been shown in the specification [17], writing the set of quadratic equations corresponding to the encryption under the smaller SKINNY variant leads to more equations in more variables than what is obtained for a fixed-key AES permutation, an observation that transfers to ForkSkinny rounds. One could fear that the fork structure might help simplifying the system (by taking into account the plaintext and the two ciphertexts all together), but we believe that the resulting gain does not compensate the very large number of rounds that each version count.

G.8 Invariant Subspace Cryptanalysis

As its name indicates, this type of attacks relies on subspaces that remain invariant while going through the round functions. As stated in SKINNY specification, this cryptanalysis was shown efficient in cases where the cipher has no key schedule, that is when the same key is added every round. Given the fact that the tweakey schedule in ForkSkinny is not of this form and that round constants are used, we believe that our proposal is safe against this kind of attacks.

G.9 On the applicability of the Techniques Devised on ForkAES to ForkSkinny

The recent article [16] provided a series of techniques to attack reduced versions of ForkAES. Their best result breaks ForkAES with $r_0 = r_1 = 4$ (independently of the number of rounds in r_{init}), while the initial proposal for the parameters of ForkAES were $r_{\mathsf{init}} = r_0 = r_1 = 5$. The techniques used were of various types: rectangle, impossible differential, reflection differential and (impossible-differential) yoyo. Note that ForkAES makes use of a tweak in the way KIASU [34] does, and that these attacks reach one more round than the best results published so far on KIASU-BC.

To achieve this, their authors took advantage of the sequence of operations that are done at the forking point in the reconstruction scenario, and in particular of the fact that the diffusion at this point is weaker than in encryption rounds. Second, they combined this with the freedom provided by the simple tweak used in the cipher to obtain inactive rounds.

The main thing that differs between ForkAES and ForkSkinny is the security margin: the sequence of operations linking M to C_0 (and C_0 to C_1) in ForkAES has 10 rounds, while 8 rounds of KIASU-BC have previously been attacked. On the other hand, the security margin for SKINNY is much more important, which should render an attack impossible on the full version of the cipher. Moreover, we fixed the parameters so that the number of rounds connecting C_0 to C_1 is higher than the number of rounds of the corresponding version of SKINNY, compensating a bit the weaker diffusion at the forking point. Additionally we also introduce a branch constant so that the state of the two branches after forking has a difference. The security analysis we conducted confirms that the set of parameters we chose are reasonable.

H Games for Defining Nonce-based AEAD Security

The games **priv-real**$_\Pi$, **priv-ideal**$_\Pi$ and **auth**$_\Pi$ can be found in Figure 21

```
proc initialize                          $\boxed{\textbf{priv-real}_{\Pi}}$
$K \leftarrow_{\$} \mathcal{K}$
$\mathcal{X} \leftarrow \varnothing$

proc Enc$(N, A, M)$
if $N \in \mathcal{X}$ then
  return $\perp$
$\mathcal{X} \leftarrow \mathcal{X} \cup \{N\}$
$C \leftarrow \mathcal{E}(K, N, A, M)$
return $C$
```

```
proc initialize                          $\boxed{\textbf{priv-ideal}_{\Pi}}$
$\mathcal{X} \leftarrow \varnothing$

proc Enc$(N, A, M)$
if $N \in \mathcal{X}$ then
  return $\perp$
$\mathcal{X} \leftarrow \mathcal{X} \cup \{N\}$
$C \leftarrow_{\$} \{0,1\}^{|M|+\tau}$
return $C$
```

```
proc initialize                          $\boxed{\textbf{auth}_{\Pi}}$
$K \leftarrow_{\$} \mathcal{K}$
$\mathcal{X} \leftarrow \varnothing$, $\mathcal{Y} \leftarrow \varnothing$

proc Enc$(N, A, M)$
if $N \in \mathcal{X}$ then
    return $\perp$
$\mathcal{X} \leftarrow \mathcal{X} \cup \{N\}$
$C \leftarrow \mathcal{E}(K, N, A, M)$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(N, A, C)\}$
return $C$

proc Dec$(N, A, C)$
if $(N, A, C) \in \mathcal{Y}$ then
    return $\perp$
return $\mathcal{D}(K, N, A, C)$
```

Fig. 21: Security games for a nonce-based AE $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with ciphertext expansion τ.