Forking a Blockcipher for Authenticated Encryption of Very Short Messages

Elena Andreeva[1], Reza Reyhanitabar[2], Kerem Varici[1] and Damian Vizár[3]

[1] imec-COSIC, KU Leuven, Belgium
elena.andreeva@esat.kuleuven.be, kerem.varici@esat.kuleuven.be
[2] Elektrobit Automotive GmbH, Germany
reza.reyhanitabar@elektrobit.com
[3] CSEM, Switzerland
damian.vizar@csem.ch

Abstract. Highly efficient encryption and authentication of *short* messages has been identified as an essential requirement for enabling security in constrained computation and communication scenarios such as the CAN FD in automotive systems (with maximum message length of 64 bytes), massive IoT and critical communication domains of 5G, and Narrowband IoT (NB-IoT), to mention some. Accordingly, NIST has specified, as a design requirement in the lightweight cryptography project, that AEAD submissions shall be "optimized to be efficient for short messages (e.g., as short as 8 bytes)". We propose AEAD schemes that exceed in efficiency over all previous general-purpose modular AEAD designs at processing (very) short inputs. The main ingredient in our solution is a new low-level primitive, called a tweakable *forkcipher*, which we introduce and formalize in this paper. We give an instance of the tweakable forkcipher and dub it ForkAES. It is based on the tweakable blockcipher KIASU, which relies on the round function of AES and uses the TWEAKEY framework to derive round keys from a 128-bit secret key and a 64-bit tweak. Finally, we demonstrate the applicability of a tweakable forkcipher by designing several provably-secure nonce-based AEAD modes of operation, optimized to be efficient for short messages. Considering the AES block size (16 bytes) as a reference, our new AE schemes can beat *all* known schemes for single-block messages while still performing better than majority of the existing schemes for combined message and associated data lengths up to 4 blocks. While ForkAES as a concrete instantiation for a forkcipher is based on KIASU, we note that our solution provides a general recipe for lightweight AEAD for short messages, even for very resource-constrained scenarios in which AES may not be considered a lightweight option. In those environments, our schemes can be instantiated using a forkcipher that is realized based on the best off-the-shelf lightweight blockcipher, following the TWEAKEY framework.

Keywords: Authenticated encryption, short messages, lightweight cryptography, forkcipher, ForkAES.

1 Introduction

Authenticated encryption (AE) aims at achieving two fundamental security goals of symmetric-key cryptography: confidentiality (privacy) and integrity (together with authentication) for data at rest and data in transit.

Historically, these two goals were achieved by generic composition of an encryption scheme (for confidentiality) and a message authentication code (MAC) [**?**, **?**]. For instance, *old* versions of major security protocols such as TLS, SSH and IPsec included variants of

generic composition method, namely MAC-then-Encrypt, Encrypt-and-MAC and Encrypt-then-MAC schemes, respectively. But it turned out that this approach is neither the most efficient (as it needs processing the whole message twice) nor the most robust to security and implementation issues [**?**, **?**, **?**, **?**]; rather it is easy for practitioners to get it wrong even when using the best known method among three, i.e. Encrypt-then-MAC, following standards [**?**].

The notion of AE as a primitive in its own right—integrating encryption and authentication functionalities and exposing a single abstract interface—put forth by Bellare and Rogaway [**?**] and independently by Katz and Yung [**?**] in 2000. The notion was further enhanced by Rogaway [**?**] to authenticated encryption with associated data (AEAD). Being able to process associated data (AD) is now a default requirement for any authenticated encryption scheme; therefore we use AE and AEAD interchangeably. After attracting near two decades of research and standardization activities, recently fostered by the CAESAR competition (2014–2018) [**?**], we have now 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 seven finalists of CAESAR).

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

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

Therefore, designing efficient AEAD for short messages is a timely and compelling objective as also evidenced by NIST's first call for submissions (May 14, 2018) for lightweight cryptography [**?**], 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)".

Motivating Use Cases. The need for high-performance and low-latency processing of short messages can be identified as an essential requirement in several security and safety critical use cases in different domains. Examples are Secure Onboard Communication (SecOC) in automotive systems [**?**], handling of short data bursts in critical communication and massive IoT domains of 5G [**?**], and Narrowband IoT (NB-IoT) [**?**, **?**] systems. For instance, the new CAN FD standard (ISO 11898-1) for vehicle bus technology [**?**, **?**], which is expected to be implemented in most cars by 2020, allows for a payload up to 64 bytes. In NB-IoT standard [**?**, **?**] 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). 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. In use cases such as smart parking lots the actual data to be sent is just a bit (for "free" or "occupied" status), so a minimum allowed TBS size of 2 bytes (16 bits) would suit the application.

Our Goal. Our main objective is to construct secure, modular AEAD schemes that exceed 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 focused) 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 AE. 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 their underlying FIL primitives can provide. Modular AEAD designs typically confine the AE security to the mode of operation only; the used lower-level primitives, such as (tweakable) block ciphers, cryptographic permutations and compression functions, are never meant to posses any AE-like features, and in particular they are never expanding as required for providing ciphertext integrity in AEAD. Therefore, a VIL AEAD scheme Π designed as a mode of operation for an FIL primitive F plays two roles: not only it extends the domain of its underlying FIL primitive but also it transforms and boosts the security property of the primitive to match to the AEAD security notion. A natural question then arises, whether by explicitly decoupling these two roles of the AEAD mode we can have more efficient designs and more transparent security proofs.

Forkcipher: a New Primitive. The first, most obvious approach to resolving this 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 [**?**, **?**, **?**]. 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) [**?**, **?**]. 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.*[1] 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 of efficient AEAD for short messages.

We introduce a different kind of low-level primitive—calling it a tweakable **forkcipher**—that does yield the most efficient AEAD design 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.[2] When keyed and tweaked, a forkcipher maps an n bit input block to $2n$ output bits. Intuitively, evaluating a secure tweakable forkcipher on an input X is equivalent to evaluating *two independent* tweakabkle permutations on X but with an *amortized computational cost.*

We give an instance of the tweakable forkcipher and dub it ForkAES. It is based on the tweakable blockcipher KIASU [**?**, **?**], which relies on the round function of AES and uses the TWEAKEY framework to derive round keys from a 128-bit secret key and a 64-bit tweak. To obtain ForkAES, we apply our newly proposed *iterate-fork-iterate* paradigm: when encrypting a block X, we derive 17 $(3 \cdot 5 + 2)$ round keys, and first transform X into X' using 5 AES rounds with 5 round keys. Then, we fork the encryption process by applying two parallel paths each comprising 5 AES rounds with 5 round keys, followed by

[1] See Section 5.10 for a brief discussion.

[2] We can 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.

the final whitenings using the remaining two sub-keys. The iterate-fork-iterate paradigm is conceptually easy and we give arguments about its security.

The New AEAD Schemes. We demonstrate the applicability of tweakable forkciphers by designing three provably secure nonce-based AEAD modes of operation, all suitable for short messages, but having some different features.

Our first scheme, called **PAEF** (Parallel AE from a forkcipher), is fully parallelizable, but its security completely falls apart with the first nonce repetition. This mode will be useful in applications where longer queries occur more often, and where one of the parties is able to perform parallel computations.

Our second scheme, called **SAEF** (Sequential AE from a forkcipher), has an online computable encryption, but is not parallelizable. While its integrity and confidentiality bounds in the nonce-respecting setting are inferior to those of the parallel mode, we conjecture that a nonce-misusing universal forgery will require data complexity at the birthday bound. In addition, SAEF also lends itself well to low-overhead implementations, as it does not need to store any block counters. In a typical setting where very short queries occur with an overwhelming probability, this mode is our primary recommendation.

Our third scheme, called **fGCM** (forkcipher-based GCM), is inspired by the blockcipher-based standard scheme GCM, but differs from it in some critical details. It is almost fully parallelizable, and it beats the other two modes in efficiency for plaintexts of more than n bits, but this requires an n-bit increase of the memory footprint to store a derived key in order to achieve full efficiency.

Performance Comparison: No One-size-fits-all Mode. Regarding the state of the art in AE designs, it appears that aiming for a *provably secure* AE scheme that achieves the best performance for both very long and very short message scenarios is a very ambitious goal. Instead we aimed at addressing the specific problem of high-performance AE for very short inputs. All three proposed modes, PAEF, SAEF and fGCM, can be efficiently implemented when instantiated with ForkAES. The ForkAES-based instances of our schemes beat all of the existing blockcipher-based AEAD modes when instantiated with AES, for the shortest queries. The performance comparison (in terms of the number of AES128 calls) of encrypting very short queries (up to 4 blocks) with AES-GCM [**?**, **?**], AES-CCM [**?**], AES-OCB [**?**], AES-CLOC [**?**], Deoxys-I [**?**] and KIASU$^{\neq}$ [**?**] is summarized in Figure 1. Our new AE schemes can beat all known schemes for single-block messages while still performing better than most of the existing schemes up to 4-block messages, and then leaving the competition to some of the general-purpose schemes such as AES-OCB.

To be Modular and Provably Secure, or not to be. In this work we followed the well-established modular AE design approach for data of arbitrary lengths in the provable security framework. However, we note that there seems to be no consensus in the cryptography community whether AE schemes can claim higher merits for being modular and provably secure or not. For instance, we note that 3 out of 7 CAESAR [**?**] finalists, namely ACORN, AEGIS and MORUS are novel monolithic designs from scratch and do not follow the provable security paradigms. Nevertheless, we choose as our design paradigm the modular and provable security methodology for its several well-advertised benefits [**?**, **?**]. Interestingly, the class of provably secure AE designs includes all currently standardized AE schemes (e.g. GCM and CCM) as well as the majority of CAESAR finalists.

Recipe for Lightweight AEAD for Short Messages. For very resource constrained IoT devices in which AES could not be considered a lightweight option—which seems to be a driving motivation behind the current NIST lightweight cryptography standardization—our proposed SAEF, PAEF and fGCM modes can be instantiated using a forkcipher which is based on any off-the-shelf lightweight blockcipher. The crux would be a careful realization of the forkcipher following the TWEAKEY framework.

scheme	Cost of encryption in (# of AES rounds)/10									
	$a=0$				$a=1$				$a=2$	
	m=1	m=2	m=3	m=4	m=0	m=1	m=2	m=3	m=1	m=2
GCM	2+2	3+3	4+4	5+5	1+2	2+3	3+4	4+5	2+4	3+5
CCM	4	6	8	10	3	5	7	9	6	8
OCB3	3	4	5	6	3	4	5	6	5	6
CLOC	3	5	7	9	2	4	6	8	5	7
Deoxys-I	2.8	4.2	5.6	7	2.8	4.2	5.6	7	5.6	7
KIASU$^{\neq}$	2	3	4	5	2	3	4	5	4	5
PAEF	1.5	3	4.5	6	1.5	2.5	4	5.5	3.5	5
SAEF	1.5	3	4.5	6	1.5	2.5	4	5.5	3.5	5
fGCM	1.5+1	2.5+2	3+3	5+4	1.5+2	1.5+2	2.5+3	3+4	1.5+3	2.5+4

Figure 1: The comparison of performance of several existing single-pass AE schemes and the forkcipher modes presented in Section 5. Here $a = |A|_n$ and $m = |M|_n$, and the cells of the form "$b+g$" mean "$b \cdot 10$ AES rounds and g multiplications in $\mathsf{GF}(2^{128})$." GCM, CCM, OCB3 and CLOC are assumed to be instantiated with AES. For schemes that compute a derived key that is the same for all queries (OCB and GCM), the complexity of this step is ignored.

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 $\{0,1\}^{\leq n}$ denote the set of all strings fo length at most 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 $0 \leq \chi \leq |X|$. We let $(L, R) = \mathsf{lsplit}_{X,n}$ denote splitting a string $X \in \{0,1\}^*$ into two parts such that $L = \mathsf{left}_{\min(|X|,n)}(X)$ and $R = \mathsf{right}_{|X|-|L|}(X)$. In particular, for $n \geq |X|$ we have $(X, \varepsilon) = \mathsf{lsplit}_{X,n}$. We further let $(M', M_*) = \mathsf{msplit}_n(M)$ denote a splitting of a string $M \in \mathit{bits}*$ 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 $(C', C_*, T) = \mathsf{csplit}_n(C)$ 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 $C'_1, \ldots, C'_m, C_*, T \leftarrow \mathsf{csplit\text{-}b}_n(C)$ denote a version of $\mathsf{csplit}_n(C)$, where the string C' further gets partitioned into $|C'|_n$ blocks of n bits, such that $C' = C'_1 \| \ldots \| C'_m$.

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 $|X|_n = \lceil X/n \rceil$. Given a (possibly implicit) positive integer n and an $X \in \{0,1\}^*$, we let $X \| 10^*$ denote $X \| 10^{n-(|X| \bmod n)-1}$ for simplicity.

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. We let $(n)_q$ denote the falling factorial $n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-q+1)$.

3 ForkAES

We design an expanding primitive with fixed input length which we dub ForkAES. It is best described by its name: it is an iterated design with a 128-bit input block, and its internal state is *forked* after half of the rounds to produce *two* redundant 128-bit output blocks. We also add a tweak to facilitate the design of simple modes of operation. ForkAES is obtained by combining two ingredients: the KIASU [**?**, **?**] tweakable blockcipher (which is, in turn, a derivative of AES, hence the name), and our newly proposed *iterate-fork-iterate* paradigm.

3.1 Specification

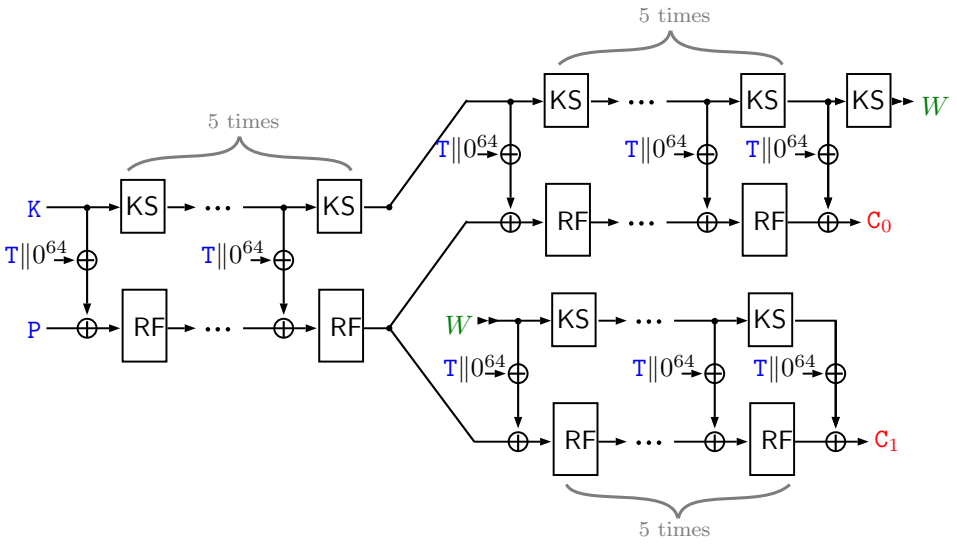

Figure 2: Illustration of an encryption by ForkAES. A 128 bit plaintext $\mathtt{P}$, a 128 bit key $\mathtt{K}$ and 64 bit tweak $\mathtt{T}$ (all in blue) are used to compute a 256 bit ciphertext $\mathtt{C} = \mathtt{C}_0 \| \mathtt{C}_1$ (in red). RF denotes a single iteration of the AES round function and KS denotes a single iteration of the AES keyschedule.

ForkAES is a deterministic cryptographic algorithm which takes a 128-bit plaintext $\mathtt{P}$, a 64-bit tweak $\mathtt{T}$ and a 128-bit secret key $\mathtt{K}$ as input, and outputs a 256-bit ciphertext $\mathtt{C}$ (i.e., $\mathsf{ForkAES}(\mathtt{K}, \mathtt{T}, \mathtt{P}) = \mathtt{C}$).

It is based on the tweakable blockcipher KIASU. In KIASU, a round function based on the SubBytes, Shiftrows and Mixcolumn operations of AES is iteratively applied to the plaintext block. Following the TWEAKEY framework [**?**], the secret key and tweak are used to generate subkeys which are xored to the intermediate internal state before every application of the round function.

Iterate-fork-iterate. How ForkAES differs from both AES and KIASU is that after half of the rounds, the encryption is forked and two copies of the internal states are further processed with different sets of *independent* subkeys. The additional required subkeys are generated by doing the necessary number of extra iterations of the key schedule (beyond what would have been done in the original (tweakable) blockcipher). We call this simple mechanism of turning an iterated (tweakable) blockcipher into a forkcipher the *iterate-fork-iterate* construction paradigm.

Inverse algorithms. Associated to ForkAES are the decryption algorithm $\mathsf{ForkAES}^{-1}$ and the *reconstruction* algorithm $\mathsf{ForkAES}^{\rho}$. Because the two output blocks produced by ForkAES are redundant, either one of them is sufficient for decryption. The decryption algorithm thus takes a secret key $\mathtt{K}$, a tweak $\mathtt{T}$, a half-ciphertext $\mathtt{C}$ of 128 bits, and a bit b that indicates whether this is the left half or the right half, and inverts the "fork" indicated by b and then the initial common processing. For every $\mathtt{K}, \mathtt{P} \in \{0,1\}^{128}$ and every $\mathtt{T} \in \{0,1\}^{64}$ we have

$$\mathtt{P} = \mathsf{ForkAES}^{-1}(\mathtt{K}, \mathtt{T}, \mathsf{left}_n(\mathsf{ForkAES}(\mathtt{K}, \mathtt{T}, \mathtt{P})), 0) = \mathsf{ForkAES}^{-1}(\mathtt{K}, \mathtt{T}, \mathsf{right}_n(\mathsf{ForkAES}(\mathtt{K}, \mathtt{T}, \mathtt{P})), 1)\,.$$

Similarly, the redundancy can be used to recompute one output block from the other which is what the reconstruction algorithm does. It takes a secret key $\mathtt{K}$, a tweak $\mathtt{T}$, a half-ciphertext $\mathtt{C}$ of 128 bits, and a bit b that indicates whether this is the left half or the right half, inverts the indicated "fork", and then recomputes the other one. For every $\mathtt{K}, \mathtt{P} \in \{0,1\}^{128}$ and every $\mathtt{T} \in \{0,1\}^{64}$ we have

$$\mathsf{ForkAES}^{\rho}(\mathtt{K}, \mathtt{T}, \mathsf{left}_n(\mathsf{ForkAES}(\mathtt{K}, \mathtt{T}, \mathtt{P})), 0) = \mathsf{right}_n(\mathsf{ForkAES}(\mathtt{K}, \mathtt{T}, \mathtt{P}))$$

and

$$\mathsf{ForkAES}^{\rho}(\mathtt{K}, \mathtt{T}, \mathsf{right}_n(\mathsf{ForkAES}(\mathtt{K}, \mathtt{T}, \mathtt{P})), 1) = \mathsf{left}_n(\mathsf{ForkAES}(\mathtt{K}, \mathtt{T}, \mathtt{P}))$$

The formal algorithmic description of all three algorithms is given in Figure 3, and the encryption operation is illustrated in Figure 2.

To generate of round keys, we set the secret key as the first round key, iterate the key schedule of AES 16 times, and xor the tweak to the 8 leftmost bytes of each round key. This is exactly what is done in KIASU, except we iterate the key schedule 6 more times. The round key generation algorithm is described in Figure 3.

3.2 Security Evaluation

In this section, we briefly discuss the security of ForkAES against the most important cryptanalytic attacks. We only consider classical black-box attacks, i.e., we do not consider side-channel attacks.

Differential Cryptanalysis. Differential crpyptanalysis is one of the most powerful security analysis methods and showing the security of a cipher against it is essential part of the security evaluation. For a cipher based on the Substitution Permutation Network (SPN) the analysis is relatively easy and well-understood and it is based on counting the number of active s-boxes over the cipher rounds. When the active s-boxes reach a certain threshold then the cipher is assumed to be secure against differential cryptanalysis. For example, in the case of AES in the single-key model, one can guarantee at least 25 active s-boxes for a differential path of four rounds due to the careful choice of a permutation layer (which is a diffusion matrix with branching number five). If each active s-box reaches the maximal differential probability of the AES S-box $p_{max} = 2^{-6}$, then the probability of the differential path becomes $2^{-150} < 2^{-128}$. Hence, four AES rounds already provide enough protection. Since our ForkAES design uses the AES round function, we can easily deduce that our design will provide enough security in this setting after four rounds against differential attacks in the single-key model.

Related-TWEAKEY Attacks. The extra freedom provided from key K (in our case tweak T as well) makes the security evaluation of ciphers against related-key (in our case related-tweakey) attacks more challenging. Over the years, many search algorithms [?, ?, ?, ?, ?, ?] were given to compute an upper bound for the related-key differential characteristics. The KIASU designers gave a comprehensive related-key analysis for KIASU by extending the search algorithms to cover the related-tweak option and we summarize their results in Table 1. Our design is based on the KIASU algorithm and its tweakey schedule and thus a closer inspection reveals that the latter results also apply to ForkAES.

Algorithm $\mathsf{ForkAES}(\mathtt{K}, \mathtt{T}, \mathtt{P})$
  $K_0, \ldots, K_{16} \leftarrow \mathsf{KeySched}(\mathtt{K}, \mathtt{T})$
  $S \leftarrow \mathtt{P}$
  for $i \leftarrow 0$ **to** 4 **do**
    $S \leftarrow S \oplus K_i$
    $S \leftarrow \mathsf{AESrnd}(S)$
  end for
  $S_0 \leftarrow S$; $S_1 \leftarrow S$
  for $i \leftarrow 5$ **to** 9 **do**
    $S_0 \leftarrow S_0 \oplus K_i$
    $S_0 \leftarrow \mathsf{AESrnd}(S_0)$
  end for
  $\mathtt{C}_0 \leftarrow S_0 \oplus K_{10}$
  for $i \leftarrow 11$ **to** 15 **do**
    $S_1 \leftarrow S_1 \oplus K_i$
    $S_1 \leftarrow \mathsf{AESrnd}(S_1)$
  end for
  $\mathtt{C}_1 \leftarrow S_1 \oplus K_{16}$
  return $\mathtt{C}_0 \| \mathtt{C}_1$
end Algorithm

Algorithm $\mathsf{ForkAES}^{-1}(\mathtt{K}, \mathtt{T}, \mathtt{C}, b)$
  $K_0, \ldots, K_{16} \leftarrow \mathsf{KeySched}(\mathtt{K}, \mathtt{T})$
  $S \leftarrow \mathtt{C} \oplus K_{10+b\cdot 6}$
  for $i \leftarrow 9 + b \cdot 6$ **to** $5 + b \cdot 6$ **do**
    $S \leftarrow \mathsf{AESrnd}^{-1}(S)$
    $S \leftarrow S \oplus K_i$
  end for
  for $i \leftarrow 4$ **to** 0 **do**
    $S \leftarrow \mathsf{AESrnd}^{-1}(S)$
    $S \leftarrow S \oplus K_i$
  end for
  return S
end Algorithm

Algorithm $\mathsf{ForkAES}^{\rho}(\mathtt{K}, \mathtt{T}, \mathtt{C}, b)$
  $K_0, \ldots, K_{16} \leftarrow \mathsf{KeySched}(\mathtt{K}, \mathtt{T})$
  $b' \leftarrow b \oplus 1$
  $S \leftarrow \mathtt{C} \oplus K_{10+b\cdot 6}$
  for $i \leftarrow 9 + b \cdot 6$ **to** $5 + b \cdot 6$ **do**
    $S \leftarrow \mathsf{AESrnd}^{-1}(S)$
    $S \leftarrow S \oplus K_i$
  end for
  for $i \leftarrow 5 + b' \cdot 6$ **to** $9 + b' \cdot 6$ **do**
    $S \leftarrow S \oplus K_i$
    $S \leftarrow \mathsf{AESrnd}(S)$
  end for
  $\mathtt{C}' \leftarrow S \oplus K_{10+b'\cdot 6}$
  return $\mathtt{C}'$
end Algorithm

Algorithm $\mathsf{KeySched}(\mathtt{K}, \mathtt{T})$
  $K_0 \leftarrow \mathtt{K} \oplus \mathsf{Rwfy}(\mathtt{T} \| 0^{64})$
  $W_0, \ldots, W_3 \xleftarrow{32} \mathtt{K}$
  for $i \leftarrow 1$ **to** 16 **do**
    $\mathsf{tmp} \leftarrow \mathsf{RotWord}(\mathsf{W}[3])$
    $W'_0 \leftarrow W_0 \oplus \mathsf{SubWord}(\mathsf{tmp}) \oplus \mathsf{Rcon}[i]$
    for $j \leftarrow 1$ **to** 3 **do**
      $W'_j \leftarrow W_j \oplus W'_{j-1}$
    end for
    for $j \leftarrow 0$ **to** 3 **do**
      $W_j \leftarrow W'_j$
    end for
    $K_i \leftarrow W_0 \| W_1 \| W_2 \| W_3 \oplus \mathsf{Rwfy}(\mathtt{T} \| 0^{64})$
  end for
  return $K_0, \ldots, K_{16}$
end Algorithm

Algorithm $\mathsf{AESrnd}(S)$
  $S \leftarrow \mathsf{SubBytes}(S)$
  $S \leftarrow \mathsf{ShifRows}(S)$
  $S \leftarrow \mathsf{MixColumns}(S)$
  return S
end Algorithm

Algorithm $\mathsf{AESrnd}^{-1}(S)$
  $S \leftarrow \mathsf{iMixColumns}(S)$
  $S \leftarrow \mathsf{iShifRows}(S)$
  $S \leftarrow \mathsf{iSubBytes}(S)$
  return S
end Algorithm

Figure 3: The algorithms $\mathsf{ForkAES}$, $\mathsf{ForkAES}^{-1}$ and $\mathsf{ForkAES}^{\rho}$. The function $Y = \mathsf{Rwfy}(X)$ (from "rowify") is a byte-transposition of a 128-bit string X that maps $Y_i = X_{4*(i \bmod 4)+\lfloor i/4 \rfloor}$.

Table 1: Upper bounds on probabilities of related-TWEAKEY differential characteristics [**?**, Table 4.1].

Rounds	Active S-boxes	Probability (upper bound)	Method
1	0	2^0	trivial
2	0	2^0	trivial
3	1	2^{-6}	Matsui's
4	8	2^{-48}	Matsui's
5	≥ 14	2^{-84}	Matsui's
7	≥ 22	2^{-132}	extended split $(3R + 4R)$

Meet-in-the-Middle Attack. There are numerous meet-in-the-middle attacks performed against AES [?, ?, ?] and for all those attacks the key schedule plays an important role. In these attacks partial encryption/decryption is done by guessing keys to prepare pre-computed tables. To reduce the amount of guessed key bytes (and respective attack complexities), the existing linear relations of the AES key schedule are exploited. In our design, we use KIASU as our core encryption operation which in turn replies on the AES cipher with the tweak addition to key schedule. The tweak is a fixed and known constant value T and therefore the existing meet-in-the-middle attacks for AES-128 will apply to both KIASU and our design.

Security Against Other Attacks. Our forkcipher ForkAES is based purposely on the AES block cipher regarding round function and key schedule designs. Moreover, we borrow the KIASU tweak (tweak and key) treatment to support the use of the additional tweak input in our design. Since we do not introduce any novel design complexities, the security of our forkcipher design can be reduced to the security of the AES and KIASU ciphers for further type of attacks.

4 Forkcipher

In this section, we formalize the syntax and security goals of a *forkcipher*; the *kind* of FIL and expanding tweakable primitive that ForkAES is. We recall that in ForkAES, the forward computation starts with a single input X, but forks into two independent branches in the middle, resulting in a double output block $C_0 \| C_1$. This makes it possible to compute the original preimage X from either of the two output block-halves C_0 or C_1, or to reconstruct one half from another. These are the basic properties captured in the concept of a forkcipher.

When keyed and tweaked and given a block X of n bits, a forkcipher computes its image under two permutations $C_0 = \pi_0(X)$ and $C_1 = \pi_1(X)$ *simultaneously*, returning $2n$ bit block $C_0 \| C_1$. The "inverse" forkcipher, consists of two algorithms. They each take a binary flag b and an n bit block C as input, and return n bits: one computes the inverse of the input under the permutation selected by b, i.e. $X' = \pi_b^{-1}(C)$ and the other reconstructs the *image* $C' = \pi_{b\oplus 1} \circ \pi_b^{-1}(C)$ of the putative input under the other permutation. This is illustrated in Figure 4.

When used with a random key, each of these permutations would be perfectly random for an ideal forkcipher, i.e., it would implement a *pair* of independent random permutations for every tweak. We thus 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 models two instances of a secure tweakable blockcipher that are used in parallel. In the same spirit, one could instantiate a forkcipher by a tweakable blockcipher used with two independent keys (or a tweak-space separation mechanism).

The main novelty in a forkcipher is that it aims to provide the same security as a pair of tweakable blockciphers at a reduced computational cost. Yet this amortization of computation has nothing to do with the security goals and syntax; these only model what kind of object a forkcipher inevitably is, and which security properties it aspires to achieve.

4.1 Syntax

A forkcipher is a triple of deterministic algorithms, the encryption[3] algorithm $\mathsf{F} : \{0,1\}^k \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^{2n}$, the inversion algorithm $\mathsf{F}^{-1}\{0,1\}^k \times \mathcal{T} \times \{0,1\}^n \times \{0,1\} \to \{0,1\}^n$

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

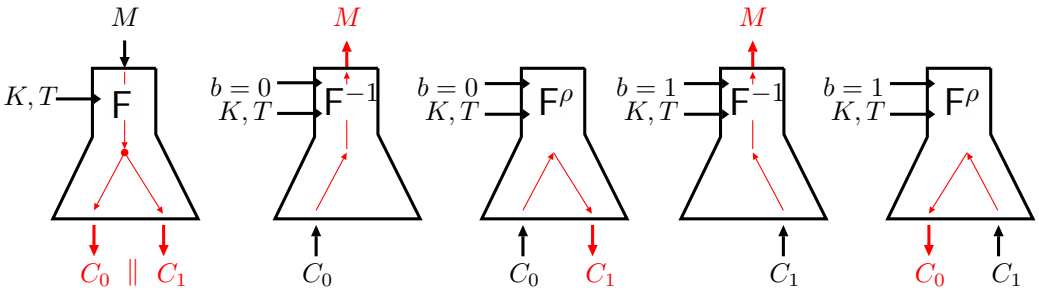

Figure 4: Forward and backward execution of a forkcipher.

and the tag reconstruction algorithm $\mathsf{F}^{\rho}\{0,1\}^k \times \mathcal{T} \times \{0,1\}^n \times \{0,1\} \to \{0,1\}^n$. 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 $K, T, M, \beta \in \{0,1\}^k \times \mathcal{T} \times \{0,1\}^n \times \{0,1\}$ we have

$$\mathsf{F}^{-1}(K, T, \mathsf{F}(K, T, M)[(\beta \cdot n) \ldots (\beta \cdot n + n - 1)], \beta) = M$$

and

$$\mathsf{F}(K, T, M)[((1-\beta)\cdot n) \ldots ((1-\beta)\cdot n+n-1)] = \mathsf{F}^{\rho}(K, T, \mathsf{F}(K, T, M)[(\beta \cdot n) \ldots (\beta \cdot n+n-1)], \beta).$$

We again focus on two specific forms of $\mathcal{T}$ only: when $\mathcal{T} = \{0,1\}^t$ for some positive t, and when $\mathcal{T}$ is an empty set. If $\mathcal{T} = \emptyset$, then we call F simply forkcipher, and we omit $\mathcal{T}$ from the functional notation of both F and F^{-1}, as well as the tweaks from the input arguments of the two algorithms.

4.2 Security Definition

We define the security of forkciphers by indistiguishability from the closest, most natural idealized version of the primitive.

We formalize the security of forkciphers through the notion of a pseudorandom tweakable forked permutation, with help of security games defined in Figure 5. A forked permutation is a pair of oracles, that internally make use of two permutations, s.t. the "left" permutation is used in the usual way (as a permutation), and the "right" permutation is always *forked* from the preimage of the left permutation, no matter if the former is used 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** and define the advantage of $\mathcal{A}$ at distinguishing F from a random tweakable injection in a *chosen ciphertext attack* as

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

4.3 Iterate-Fork-Iterate: A Generic Validation

In order to rule out that the iterate-fork-iterate construction succumbs to a *generic* attack (i.e., attacks that do not use any property of the iterated primitive, but only treat it as a blackbox), we carry out a generic analysis in the spirit of provable security. This result does not, of course, imply the security of ForkAES, or any other instance of forkcipher, because the iterated primitive is not a secure pseudorandom permutation. It does show, however, that the construction itself does not introduce any exploitable weakness.

Game **prtfp-real**$_{\mathsf{F}}$

$K \leftarrow\$ \{0,1\}^k$
$b \leftarrow \mathcal{A}^{\textsc{Enc},\textsc{Dec}}$
return b

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

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

Game **prtfp-ideal**$_{\mathsf{F}}$

for $T \in \mathcal{T}$ **do** $\pi_{T,0}, \pi_{T,1} \leftarrow\$ \mathrm{Perm}(n)$
$b \leftarrow \mathcal{A}^{\textsc{Enc},\textsc{Dec}}$
return b

Oracle $\textsc{Enc}(T, M)$
return $\pi_{T,0}(M) \| \pi_{T,1}(M)$

Oracle $\textsc{Dec}(T, C, \beta)$
return $\pi^{-1}_{T,\beta}(C)$

Figure 5: Games **prtfp-real**, and **prtfp-ideal** used to define security of a (strong) forkcipher.

IFI: a generalized ForkAES. We define the IFI construction which combines three tweakable permutations (i.e., three tweak-indexed collections of random permutations) in the same same way ForkAES applies the three groups of Tweakey-AESround iterations. Fix the block length n and the tweak length t. Formally, for three tweakable random permutations p, p_0, p_1 (i.e. $p = (p^{\mathsf{T}} \leftarrow\$ \mathrm{Perm}(n))_{\mathsf{T}\in\{0,1\}^t}$ is a collection of independent uniform elements of $\mathrm{Perm}(n)$ indexed by the elements of $\mathsf{T} \in \{0,1\}^t$, and similar applies for p_0 and p_1), the forkcipher $\mathsf{F} = \mathrm{IFI}[p, p_0, p_1]$ is defined by $\mathsf{F}^{\mathsf{T}}(X) = p_{\mathsf{T},0}(p_{\mathsf{T}}(X)) \| p_{\mathsf{T},1}(p_{\mathsf{T}}(X))$, $\mathsf{F}^{-1^{\mathsf{T}}}(C, b) = p^{-1}_{\mathsf{T}}(p^{-1}_{\mathsf{T},b}(C))$ and $\mathsf{F}^{\rho^{\mathsf{T}}}(C, b) = p_{\mathsf{T},b\oplus 1}(p^{-1}_{\mathsf{T},b}(C))$. We note that the three tweakable random permutations act as a key for $\mathrm{IFI}[p, p_0, p_1]$ and we omit them for the sake of simplicity.

The analysis. We are going to analyze the indistinguishability of the IFI construction from a forked random permutation. Since both objects an adversary is attempting to distinguish are information-theoretic, we can assume w.l.o.g. that the adversaries are computationally unbounded and deterministic.

Theorem 1. *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.

Proof. We use the game Γ from Figure 6 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.

When $\mathcal{A}$ makes an $\textsc{Enc}(\mathsf{T}, M)$ 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 $\textsc{Dec}(\mathsf{T}, C, \beta)$ 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 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 6 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 $\textsc{Enc}(\mathsf{T}, M)$ 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 $\textsc{Dec}(\mathsf{T}, M, \beta)$ 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.

□

5 Tweakable Forkcipher Modes

We demonstrate the applicability of forkciphers by designing provably secure AE modes of operation for a tweakable forkcipher. The designs are motivated by the following objectives: (1) the resulting AE scheme must be able to process strings of *arbitrary length* but (2) it must be most efficient for encryption queries whose total number of blocks (in AD and message) is very small, e.g. below four.

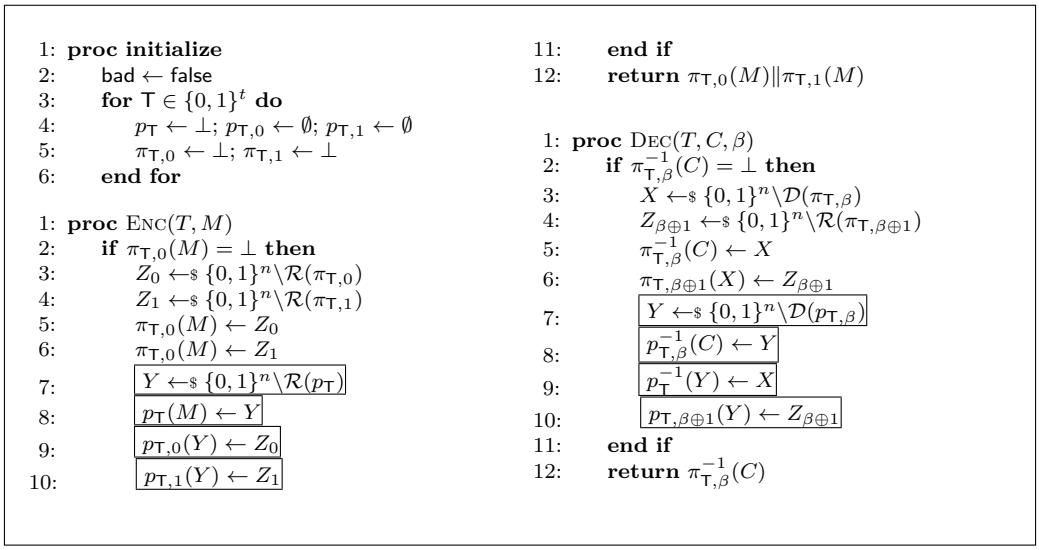

Figure 6: Game Γ used in the proof of security of the IFI$[p, p_0, p_1]$ construction. The tweakable permutations p, p_0, p_1, π_0 and π_1 are initially undefined.

We define three nonce-based AE modes of operation for a tweakable forkcipher. One of the modes is fully parallelizable, but its security falls apart with the first nonce repetition. This mode will be useful in applications where longer queries occur more often, and where one of the parties has more powerful computational capabilities.

The second mode has an online computable encryption, but is not parallelizable. While its integrity and confidentiality bounds in the nonce-respecting setting are inferior to those of the parallel mode, we conjecture that a nonce-misusing universal forgery will require data complexity at the birthday bound. In addition to the resilience to accidental nonce reuse, the sequential mode also lends itself well to low-overhead implementations, as it does not require to store any message-block counter.

The third mode is inspired by the blockcipher-based scheme GCM [**?**], but differs from it in some critical details. It is almost fully parallelizable, and it beats the other two modes in efficiency for plaintexts of more than n bits, but this increased efficiency comes at the price of an n-bit larger memory footprint necessary to store a derived authentication key.

While the first two of our AE schemes appear to be computationally less efficient, we stress that all three can be efficiently implemented when instantiated with ForkAES. The ForkAES-based instances will even beat all of the existing AE modes for the shortest queries (see Section 5.8).

A small AE primitive. While a secure forkcipher does not directly capture any integrity, we show in Section 5.9 that a secure forkcipher *can* be used as an AEAD scheme with fixed length messages and AD *in the natural way*, provably delivering robust AE security guarantees.

5.1 Syntax and Security of AE

Our modes target the security of nonce-based AE with associated data, following the syntax proposed by Rogaway [**?**].

A scheme for authenticated encryption with associated data is a triplet $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. The key space $\mathcal{K}$ is a finite set endowed with the uniform distribution. 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

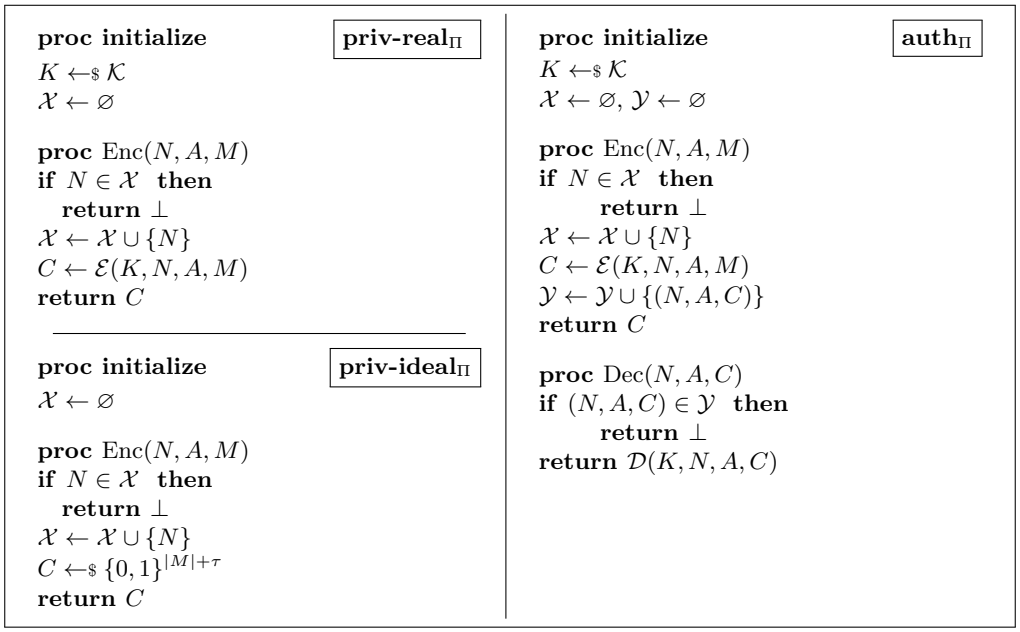

Figure 7: Games for defining the security of a nonce based AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with ciphertext expansion τ.

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 signalize 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 use the two-requirement definition of AE security. We model a chosen plaintext attack of an adversary $\mathcal{A}$ against the confidentiality of a nonce-based AE scheme Π with the help of the security games **priv-real** and **priv-real** in Figure 7. We define the advantage of $\mathcal{A}$ in breaking the confidentiality of Π as

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

We model a chosen ciphertext attack against the integrity of Π with help of the game **auth** in Figure 7. We define the advantage of $\mathcal{A}$ in breaking the integrity of Π as

$$\mathbf{Adv}^{\mathbf{priv}}_{\Pi}(\mathcal{A}) = \Pr[\mathcal{A}^{\mathbf{auth}_\Pi} \text{forges}]$$

where "$\mathcal{A}$ forges" denotes the event that there is a decryption query that returns a value other than $\perp$.

5.2 Parallel AE from a Forkcipher

We define the nonce based AEAD scheme PAEF (as in "Parallel AE from a Forkcipher"). PAEF is parameterized by a forkcipher F (as defined in Section 4) 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

PAEF[F, ν] $= (\mathcal{K}, \mathcal{E}, \mathcal{D})$ has $\mathcal{K} = \{0,1\}^k$ and the encryption and decryption algorithms as defined in Figure 8. Its nonce space is $\mathcal{N} = \{0,1\}^\nu$, and its message and AD space are respectively $\mathcal{M} = \{0,1\}^{\leq n \cdot 2^{(t-\nu-3)}}$ and $\mathcal{A} = \{0,1\}^{\leq n \cdot 2^{(t-\nu-3)}}$. The ciphertext expansion of PAEF[F, ν] is n. The encryption algorithm is illustrated in Figure 9.

In an encryption query, AD and message are partitioned into blocks of n bits. Each block is processed with exactly one call to F using a tweak that is composed of

1. a three-bit flag $f_0 \| f_1 \| f_2$,
2. the nonce,
3. a $(t - \nu - 3)$-bit encoding of the block index (indexing is individual for both AD and message),

so the nonce-length is a parameter that allows to make a trade-off between 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, the bit $f_1 = 1$ 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 being processed and is incomplete. The ciphertext blocks are the "left" output blocks of F from message blocks, and an xor of all "right" output blocks of F is xored to the rightmost n bits of the call to F that processes the final message 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.

5.3 Security of PAEF

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

Theorem 2. *Let F be a tweakable forkcipher with $\mathcal{T} = \{0,1\}^t$, and let $0 < \nu \leq t - 4$. Then for any nonce-respecting adversary $\mathcal{A}$ whose queries lie in the proper domains of the encryption and decryption algorithms and who makes at most q_v decryption queries, we have*

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

and

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

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

Proof. 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

Algorithm $\mathcal{E}(K, N, A, M)$
$A_1, \ldots, A_a, A_* \xleftarrow{n} A$
$M_1, \ldots, M_m, M_* \xleftarrow{n} M$
$T \leftarrow 0^n$
for $i \leftarrow 1$ **to** a **do**
  $\mathsf{T} \leftarrow 000 \| N \| \langle i \rangle_{t-\nu-3}$
  $T \leftarrow T \oplus \mathsf{right}_n(\mathsf{F}_K^{\mathsf{T}}(A_i))$
end for
if $|A_*| = n$ **then**
  $\mathsf{T} \leftarrow 001 \| N \| \langle a+1 \rangle_{t-\nu-3}$
  $T \leftarrow T \oplus \mathsf{right}_n(\mathsf{F}_K^{\mathsf{T}}(A_*))$
else if $|A_*| > 0$ **or** $|M| = 0$ **then**
  $\mathsf{T} \leftarrow 011 \| N \| \langle a+1 \rangle_{t-\nu-3}$
  $T \leftarrow T \oplus \mathsf{right}_n(\mathsf{F}_K^{\mathsf{T}}(A_* \| 10^*))$
end if ▷ Do nothing if $A = \varepsilon, M \neq \varepsilon$
for $i \leftarrow 1$ **to** m **do**
  $\mathsf{T} \leftarrow 100 \| N \| \langle i \rangle_{t-\nu-3}$
  $C_i, T' \leftarrow \mathsf{lsplit}_n(\mathsf{F}_K^{\mathsf{T}}(M_i))$
  $T \leftarrow T \oplus T'$
end for
if $|M_*| = n$ **then**
  $\mathsf{T} \leftarrow 101 \| N \| \langle m+1 \rangle_{t-\nu-3}$
  $R \leftarrow \mathsf{F}_K^{\mathsf{T}}(M_*)$
else if $|M_*| > 0$ **then**
  $\mathsf{T} \leftarrow 111 \| N \| \langle m+1 \rangle_{t-\nu-3}$
  $R \leftarrow \mathsf{left}_{n+|M_*|}(\mathsf{F}_K^{\mathsf{T}}(M_* \| 10^*))$
else
  $R \leftarrow 0^n$
end if
$R \leftarrow (0^{|M_*|} \| T) \oplus R$
return $C_1 \| \ldots \| C_m \| R$
end Algorithm

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

Figure 8: The PAEF$[\mathsf{F}, \nu]$ AEAD scheme. Here $\langle i \rangle_\ell$ is the cannonical encoding of an integer i as an ℓ-bit string.

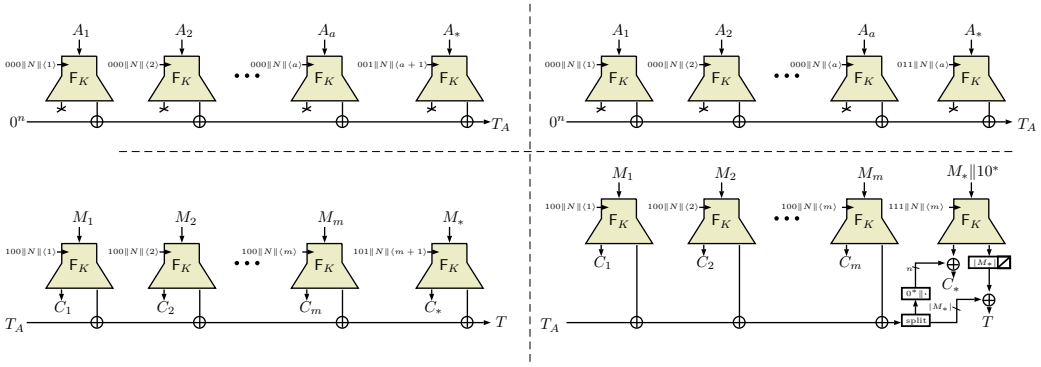

Figure 9: The encryption algorithm of PAEF[F] mode. The picture illustrates the processing of AD when length of AD is a multiple of n (**top left**) and when the length of AD is not a multiple of n (**top right**), and the processing of the message when length of the message is a multiple of n (**bottom left**) and when the length of message is not a multiple of n (**bottom right**)

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

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

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 is produced as the only image under and independent random permutation, and thus uniformly distributed. The final $|M_*| + n$ bits of every ciphertext are produced as xor-sum that always contains (a part of) the concatenated output of π_0 and π_1 applied to he last message block with a unique tweak, such that this part is not used for any ciphertext block. Since all ciphertexts are uniformly distributed we get perfect privacy and hence our privacy result.

For authenticity, we analyse the probability of forgery for an adversary that makes a single decryption query against PAEF$[(\pi_0, \pi_1), \nu]$ and then use a result of Bellare [**?**] to extend our result to multiple queries (still against 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 $C^i_1, \ldots, C^i_m, C^i_*, T = \mathsf{csplit\text{-}b}_n(C^i)$. 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, thus succeeds with probability 2^{-n}. This holds even if $|T| < 0$, because the rightmost $(n - |T|)$ bits of the final image under the inverse of 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 i^{th}, 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 a xor-sum of $\pi_{\mathsf{T},1}$ images from the associated data (denoted as T_A in Figure 9), 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 i^{th} encryption query is xor-masked with uniform bits produced by 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 $\pi_{\mathsf{T},0}^{-1}(\mathsf{left}_n(T_*))$ and $\pi_{\mathsf{T},1}(\pi_{\mathsf{T},0}^{-1}(\mathsf{left}_n(T_*)))$ such that the tweak $\mathsf{T} = 110\|N\|\langle m+1\rangle_{t-\nu-3}$ (respectively $\mathsf{T} = 111\|N\|\langle m+1\rangle_{t-\nu-3}$) 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 $\pi_{\mathsf{T},1}$ with a previously unused tweak ($\mathsf{T} = 0b1\|N\|\langle a+1\rangle_{t-\nu-3}$ for $b \in \{0,1\}$) thanks to $a > a_i$, succeeding with probability of 2^{-n}.
- If $0 < |A|_n < |A^i|_n$ but more than 0, then a successful forgery is still equivalent to guessing an output value of $\pi_{\mathsf{T},1}$ with a previously unused tweak ($\mathsf{T}0b1\|N\|\langle a+1\rangle_{t-\nu-3}$ for $b \in \{0,1\}$), thanks to the three-bit domain-separation flag (which was set to 000 in the i^{th} 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 equivalent to guessing the image $\pi_{(011\|N\|1),1}(10^{n-1})$, such that the random permutation $\pi_{(011\|N\|1),1}$ was evaluated on no more than a single other input $A_*^i\|10^* \neq 10^{n-1}$ in the whole game. This succeeds with probability at most $1/(2^n-1)$.

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 $X = \pi_{\mathsf{T},0}^{-1}(C_* \oplus (0^{n-|T|}\mathsf{left}_{|T|}(\bar{T}))$ (as per line 21 in Figure 8) 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 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 $\pi_{\mathsf{T},1}$, 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 $\pi_{\mathsf{T},1}(A_*\|10^{n-|A_*|})$. With a single image of $\pi_{\mathsf{T},1}$ 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 $\bar{T}^i$ and $\bar{T}$ used on the line 21 of the decryption algorithm in Figure 8 would be the same, and thus necessarily $(C_*\|T) \oplus (0^{n-|T|}\|\bar{T}) \neq (C^i_*\|T^i) \oplus (0^{n-|T|}\|\bar{T}^i)$. The probability of a successful forgery is at most $2^n/(2^n-1)^2$ 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 $(C_*\|T) \oplus (0^{n-|T|}\|\bar{T}) = (C^i_*\|T^i) \oplus (0^{n-|T|}\|\bar{T}^i)$. 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 $\pi_{\mathsf{T},1}(A_j) \oplus \pi_{\mathsf{T},1}(A^i_j)$ (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 $\pi_{\mathsf{T},1}$ and $\pi^{-1}_{\mathsf{T},1}$ we sampled only once. The second way is if $(C_*\|T) \oplus (0^{n-|T|}\|\bar{T}) \neq (C^i_*\|T^i) \oplus (0^{n-|T|}\|\bar{T}^i)$ but the verification still succeeds. This is analogous to **Case 5**.

The probability of a successful forgery in this case is therefore 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 [**?**], we can bound the probability of a successful forgery among q_v decryption queries as $(q_v \cdot 2^n)/(2^n-1)^2$. □

5.4 Sequential AE from a Forkcipher

We now define SAEF (as in "Sequential AE from a Forkcipher," pronounce as "safe"), a nonce-based AEAD scheme parameterized by a tweakable forkcipher F (as defined in Section 4) with $\mathcal{T} = \{0,1\}^t$ for a positive $t \leq n$. An instance SAEF[F] $= (\mathcal{K}, \mathcal{E}, \mathcal{D})$ has a key space $\mathcal{K} = \{0,1\}^k$, nonce space $\mathcal{N} = \{0,1\}^{t-3}$, and the AD and message spaces are both $\{0,1\}^*$ (although the maximal message length influences the security). The ciphertext expansion of SAEF[F] is n. The encryption and decryption algorithms are defined in Figure 11 and the encryption algorithm is illustrated in Figure 12.

In an encryption query, first AD and then message 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, and
2. a three-bit flag f.

The flag f takes different values for processing of different types of data blocks in the encryption algorithm. These are given in Figure 10.

The "right" output 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

value of f	type of block processed by the encryption algorithm
000	processing non-final AD block
010	processing final complete AD block
011	processing final incomplete AD block
110	processing final complete AD block to produce tag
111	processing final incomplete AD block to produce tag
001	processing non-final message block
100	processing final complete message block
101	processing final incomplete message block

Figure 10: The binary flag values used in the SAEF mode for forkcipher.

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 as the encryption, except that the "right" outputs of F in the message processing part are reconstructed from ciphertext blocks (using the F reconstruction algorithm). Only if the verification succeeds will the actual decryption take place.

5.5 Security of SAEF

For the security analysis of SAEF, we will work with adversarial resources that are more fine-grained than it is usual. The reason for this is that the security in a typical setting, most of the messages will be very short. We will show that this can greatly improve security of SAEF, if we account for adversarial resources per query length.

For an adversary $\mathcal{A}$, we will watch the total number of encryption and decryption queries, q and q_v respectively. For data complexity, we will work with a vector $\boldsymbol{q} \in \mathbb{N}^L$ with $L = \{\ell \mid \exists A \in \mathcal{A}, M \in \mathcal{M} \text{ s.t. } |A|_n = \ell \text{ or } |M|_n = \ell\}$. In other words, $\boldsymbol{q}$ is a vector of non-negative integers whose elements are indexed by a subset of those integers that can be the number of either message or AD blocks in a query to SAEF[F]. The vector $\boldsymbol{q}$ associated to $\mathcal{A}$ will then contain for each ℓ the maximal number $\boldsymbol{q}_\ell$ of queries (N, A, M) made by $\mathcal{A}$ with no more than $\ell = \max(|A|_n, |M|_n)$ blocks of either the message or AD. We further let $\ell_{\max} = \max\{|A|_n + |X|_n|\ \mathcal{A} \text{ queries } (N, A, X) \text{ or } (N, A, X\|T)\}$ denote the maximal number of blocks of AD and message *together* in a single query. We finally let σ denote the maximal total data complexity of $\mathcal{A}$, measured in the total number of blocks of AD, messages, and possibly ciphertexts in all $\mathcal{A}$'s queries. It is then easy to see that $\sum_{\ell \in L \cap \{0,\ldots,\ell_{\max}\}} \ell \cdot \boldsymbol{q}_\ell \le 2\sigma$ and that $q + q_v \le \sum_{\ell \in L} \boldsymbol{q}_\ell$.

We state the formal claim about the nonce-based AE security of SAEF in Theorem 3.

Theorem 3. *Let F be a tweakable forkcipher with $\mathcal{T} = \{0,1\}^t$. Then for any nonce-respecting adversary $\mathcal{A}$ whose resources are bounded by $q, q_v, \sigma, \ell_{\max}$ and $\boldsymbol{q}$ (in the sense we just defined) such that $\ell_{\max} \le 2^n/4$, we have*

$$\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{SAEF}[\mathsf{F}]}(\mathcal{A}) \le \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{B}) + \sum_{\ell \in L \cap \{0,\ldots,\ell_{\max}\}} 3 \cdot \frac{\boldsymbol{q}_\ell \cdot \ell \cdot (\ell-1)}{2^n},$$

$$\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{SAEF}[\mathsf{F}]}(\mathcal{A}) \le \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{C}) + \sum_{\ell \in L \cap \{0,\ldots,\ell_{\max}\}} \frac{\boldsymbol{q}_\ell \cdot 3\ell \cdot (\ell-1)}{2^n} + \frac{q_v \cdot (2\ell_{\max}+1)}{2^n} + \frac{q_v}{(2^n-1)}$$

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.

Algorithm $\mathcal{E}(K, N, A, M)$
$A_1, \ldots, A_a, A_* \xleftarrow{n} A$
$M_1, \ldots, M_m, M_* \xleftarrow{n} M$
$\mathsf{noM} \leftarrow 0$
if $|M| = 0$ **then** $\mathsf{noM} \leftarrow 1$
$\Delta \leftarrow 0^n$; $\mathsf{T} \leftarrow 000 \| N$
for $i \leftarrow 1$ **to** a **do**
  $\Delta \leftarrow \mathsf{right}_n(\mathsf{F}^{\mathsf{T}}_K(A_i \oplus \Delta))$
end for
if $|A_*| = n$ **then**
  $\mathsf{T} \leftarrow \mathsf{noM} \| 10 \| N$
  $\Delta \leftarrow \mathsf{right}_n(\mathsf{F}^{\mathsf{T}}_K(A_* \oplus \Delta))$
else if $|A_*| > 0$ **or** $|M| = 0$ **then**
  $\mathsf{T} \leftarrow \mathsf{noM} \| 11 \| N$
  $\Delta \leftarrow \mathsf{right}_n(\mathsf{F}^{\mathsf{T}}_K((A_* \| 10^*) \oplus \Delta))$
end if ▷ Do nothing if $A = \varepsilon, M \neq \varepsilon$
$\mathsf{T} \leftarrow 001 \| N$
for $i \leftarrow 1$ **to** m **do**
  $S \leftarrow \mathsf{F}^{\mathsf{T}}_K(M_i \oplus \Delta) \oplus (\Delta \| 0^n)$
  $C_i, \Delta \leftarrow \mathsf{lsplit}_n(S)$
end for
if $|M_*| = n$ **then**
  $\mathsf{T} \leftarrow 100 \| N$
  $R \leftarrow \mathsf{F}^{\mathsf{T}}_K(M_* \oplus \Delta) \oplus (\Delta \| 0^n)$
else if $|M_*| > 0$ **then**
  $\mathsf{T} \leftarrow 101 \| N$
  $R \leftarrow \mathsf{F}^{\mathsf{T}}_K((M_* \| 10^*) \oplus \Delta) \oplus (\Delta \| 0^n)$
  $R \leftarrow \mathsf{left}_{n+|M_*|}(R)$
else
  $R \leftarrow \Delta$
end if
return $C_1 \| \ldots \| C_m \| R$
end Algorithm

Algorithm $\mathcal{D}(K, N, A, C)$
$A_1, \ldots, A_a, A_* \xleftarrow{n} A$
$C_1, \ldots, C_m, C_*, T \mathsf{csplit\text{-}b}_n C$
$\mathsf{noM} \leftarrow 0$
if $|C| = n$ **then** $\mathsf{noM} \leftarrow 1$
$\Delta \leftarrow 0^n$; $\mathsf{T} \leftarrow 000 \| N$
for $i \leftarrow 1$ **to** a **do**
  $\Delta \leftarrow \mathsf{right}_n(\mathsf{F}^{\mathsf{T}}_K(A_i \oplus \Delta))$
end for
if $|A_*| = n$ **then**
  $\mathsf{T} \leftarrow \mathsf{noM} \| 10 \| N$
  $\Delta \leftarrow \mathsf{right}_n(\mathsf{F}^{\mathsf{T}}_K(A_* \oplus \Delta))$
else if $|A_*| > 0$ **or** $|T| = 0$ **then**
  $\mathsf{T} \leftarrow \mathsf{noM} \| 11 \| N$
  $\Delta \leftarrow \mathsf{right}_n(\mathsf{F}^{\mathsf{T}}_K((A_* \| 10^*) \oplus \Delta))$
end if ▷ Do nothing if $A = \varepsilon, M \neq \varepsilon$
$\mathsf{T} \leftarrow 001 \| N$
for $i \leftarrow 1$ **to** m **do**
  $M_i \leftarrow \mathsf{F}^{-1\,\mathsf{T}}_K(C_i \oplus \Delta, 0) \oplus \Delta$
  $\Delta \leftarrow \mathsf{F}^{\rho\,\mathsf{T}}_K(C_i \oplus \Delta, 0)$
end for
if $|T| = n$ **then**
  $\mathsf{T} \leftarrow 100 \| N$
  $\bar{T} \leftarrow T$
else if $|T| > 0$ **then**
  $\mathsf{T} \leftarrow 101 \| N$
  $\bar{T} \leftarrow 10^{n-|T|-1} \| T$
else
  if $\Delta \neq C_*$ **then return** $\perp$
  return ε
end if
$\tilde{T} \leftarrow \mathsf{left}_{|T|}(\mathsf{F}^{\rho\,\mathsf{T}}_K(C_* \oplus \Delta, 0))$
$M_* \leftarrow \mathsf{F}^{-1\,\mathsf{T}}_K(C_* \oplus \Delta, 0) \oplus \Delta$
if $\bar{T} \neq \mathsf{right}_{n-|T|}(M_*) \| \tilde{T}$ **then**
  return $\perp$
end if
$M_* \leftarrow \mathsf{left}_{|T|}(M_*)$
return $M_1 \| \ldots \| M_m \| M_*$
end Algorithm

Figure 11: The SAEF[F] AEAD scheme.

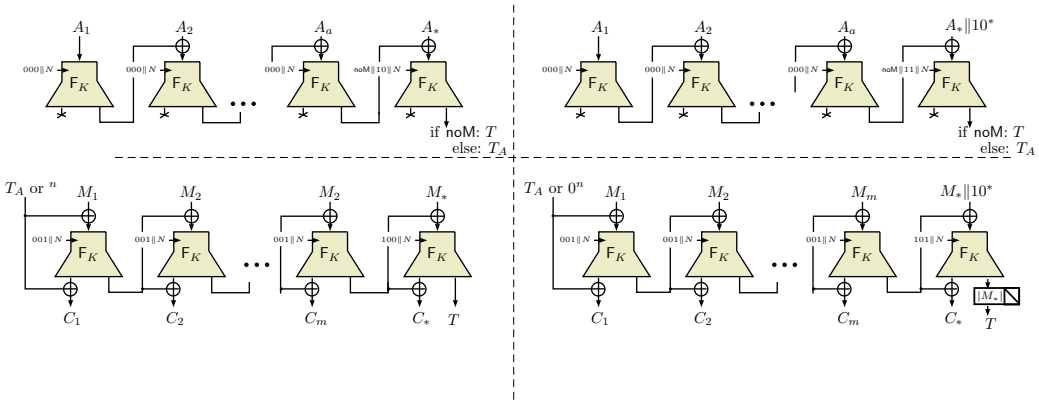

Figure 12: The encryption algorithm of SAEF[F] mode. The bit $\mathsf{noM} = 1$ iff $|M| = 0$. The picture illustrates the processing of AD when length of AD is a multiple of n (**top left**) and when the length of AD is not a multiple of n (**top right**), and the processing of the message when length of the message is a multiple of n (**bottom left**) and when the length of message is not a multiple of n (**bottom right**)

Corollary 1. *From* $\sum_{\ell \in L \cap \{0,\ldots,\ell_{\max}\}} \ell \cdot \boldsymbol{q}_\ell \leq 2\sigma$ *it immediately follows that*

$$\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{SAEF}[\mathsf{F}]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{B}) + 12 \cdot \frac{\sigma^2}{2^n},$$
$$\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{SAEF}[\mathsf{F}]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{C}) + \ 12 \cdot \frac{\sigma^2}{2^n} + 4 \cdot \frac{q_v \cdot (2\ell^2_{\max} + 1)}{2^n} + \frac{q_v}{(2^n - 1)}.$$

Remark 1 (Interpretation of the bounds in Theorem 3 and Corollary 1). Corollary 1 clearly shows that in general, the security of SAEF is birthday bounded, which is unusual for a mode of operation for a tweakable primitive. However, SAEF was designed especially for applications where the encryption queries are very *short*. The security guarantees given by the fine-grained bound in Theorem 3 become very competitive in this case.

For example, in an (IoT-like) application where q_v is reasonably small, and where we know for certain that the vast majority of queries consists of a single block of AD and a single message block, and a tiny portion $\gamma \ll q$ of messages is expected to contain a small number of message blocks β, the contribution of the short queries vanishes and the security bounds of SAEF become near optimal:

$$\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{SAEF}[\mathsf{F}]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{B}) + 3 \cdot \frac{\gamma \cdot \beta \cdot (\beta - 1)}{2^n}$$
$$\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{SAEF}[\mathsf{F}]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{C}) + 3 \cdot \frac{\gamma \cdot \beta \cdot (\beta - 1)}{2^n} + \cdot \frac{q_v \cdot (\beta^2 + 1)}{2^n} + \frac{q_v}{(2^n - 1)}$$

Remark 2 (Sequential encryption vs forgeries.). The sequential structure of SAEF causes a birthday-type deterioration of the security bounds in in the nonce-respecting setting. However, we conjecture that it also lends a degree for resistance to forgeries up to the birthday bound in the nonce-misuse setting to SAEF. Right now this is an unproven claim, and we intend to prove it for the full version of the paper.

Here is an intuition behind this claim. In the case of nonce-reuse, issuing queries with no AD allows the adversary to learn direct input-"left" output pairs for F. However, the adversary never learns the "right" outputs of F for the non-final blocks of message. Therefore, finding a collision on the input to F that would allow a forgery should still take a query complexity at the birthday bound.

Proof of Theorem 3. 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. $\pi_0 = (\pi_{\mathsf{T},0} \leftarrow\!\$\ \mathrm{Perm}(n))_{\mathsf{T}\in\{0,1\}^t}$ is a collection of independent uniform elements of $\mathrm{Perm}(n)$ indexed by the elements of $\mathsf{T} \in \{0,1\}^t$ (and similarly for $\pi_1 = (\pi_{\mathsf{T},1} \leftarrow\!\$\ \mathrm{Perm}(n))_{\mathsf{T}\in\{0,1\}^t}$). 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:

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

by a similar argument as in the proof of Theorem 2.

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. $f_b = (f_{\mathsf{T},b} \leftarrow\!\$\ \mathrm{Func}(n))_{\mathsf{T}\in\{0,1\}^t}$ for $b \in \{0,1\}$. Denoting the SAEF mode using these random functions by $\mathrm{SAEF}[f_0, f_1]$, we have that

$$\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{SAEF}[\pi_0,\pi_1]}(\mathcal{A}) \le \mathbf{Adv}^{\mathbf{priv}}_{\mathrm{SAEF}[f_0,f_1]}(\mathcal{A}) + \sum_{\ell\in L\cap\{0,\ldots,\ell_{\max}\}} 2\cdot\frac{\boldsymbol{q}_\ell\cdot\ell\cdot(\ell-1)}{2^n}$$

because each query uses a distinct nonce, and thus for each N, the functions $f_{\mathsf{T},0}$ and $f_{\mathsf{T},1}$ with $\mathsf{T} = b_0b_1b_2\|N$ and $b_0, b_1, b_2 \in \{0,1\}$ are always used only in a single query. Moreover, for $b_0b_1b_2 \in \{0,1\}^3\backslash\{000, 001\}$, the associated random functions are used at most once, so their distribution is indistinguishable from that of the corresponding random permutations. For $b_0b_1b_2 \in \{000, 001\}$, there will be exactly $\boldsymbol{q}_\ell$ values of N for which both $f_{\mathsf{T},0}$ and $f_{\mathsf{T},1}$ will be used at most ℓ times. Replacing such a $\pi_{b_0b_1b_2\|N,b}$ by $f_{b_0b_1b_2\|N,b}$ augments the bound by at most $\ell\cdot(\ell-1)\cdot 2^{-n-1}$ by the RP-RF switching lemma [**?**] and a standard hybrid argument. By summing over ℓ, $b_0b_1b_2 \in \{000, 001\}$ and $b \in \{0,1\}$, we obtain the bound.

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 13. 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 image of a distinct input to $f_{\mathsf{T},0}$. The final blocks C_* and T are an xor of the sum of previous "right" blocks, and of bits produced by two random functions $f_{10b\|N,0}$ and $f_{10b\|N,1}$ that are each used exactly once (due to the non-repetition of the nonces and the domain separation flags $10b$ for $b \in \{0,1\}$). 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 $\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{SAEF}[f_0,f_1]}(\mathcal{A}) \le \Pr[\mathcal{A}^{G_0} \Rightarrow 1] - \Pr[\mathcal{A}^{G_1} \Rightarrow 1]$.

We also have that G_0 and G_1 are identical until bad, so by the Fundamental lemma of gameplaying [**?**] we have that $\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{SAEF}[f_0,f_1]}(\mathcal{A}) \le \Pr[\mathcal{A}^{G_0}$ 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 $\Pr[\mathcal{A}^{G_0}$ 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) with $\ell = |M|_n$, all the sets $\mathcal{D}_{b_0b_1b_2\|N}$ used in the query are initially empty, due to the uniqueness of N. The flag bad can only be set due to a collision on $\mathcal{D}_{001\|N}$. We use the tweak $001\|N$ when processing the message. With the first message block, the adversary can possibly choose the input to $f_{001\|N,1}$, but this cannot cause a collision because $\mathcal{D}_{001\|N}$ is empty. Then for every next block M_i, the input to $f_{001\|N,1}$ is masked by the current value Δ. If bad has not been set

before, Δ is a fresh, uniform value, so the probability that $(M_i \oplus \Delta) \in \mathcal{D}_{001\|N}$ is at most $|\mathcal{D}_{001\|N}| \cdot 2^{-n} = (i-1) \cdot 2^{-n}$. The probability that bad gets set due to AD processing is then bounded by $\ell \cdot (\ell-1) \cdot 2^{-n-1}$ by summing over $i = 2, \ldots, \ell$.

We know that for each ℓ, there are exactly $\boldsymbol{q}_\ell$ queries (N, A, M) with $\ell = \max(|A|_n, |M|_n)$ and for $\ell > \ell_{\max}$, $\boldsymbol{q}_\ell = 0$. So, by applying the union bound over all queries, we get $\Pr[\mathcal{A}^{G_0} \text{ sets } \mathsf{bad}] \leq \sum_{\ell \in L \cap \{0,\ldots,\ell_{\max}\}} \boldsymbol{q}_\ell \cdot \ell \cdot (\ell-1) \cdot 2^{-n}$. Combined with the previously derived inequalities, this completes the proof of the confidentiality bound.

```
proc initialize
    for T ∈ {0,1}^t do
        f_{T,0} ←$ Func(n)
        f_{T,1} ←$ Func(n)
        D_T ← ∅
    end for
    bad ← false

proc Enc(N, A, M)
    A_1, ..., A_a, A_* ←^n A
    M_1, ..., M_m, M_* ←^n M
    noM ← 0
    if |M| = 0 then noM ← 1
    Δ ← 0^n; T ← 000‖N
    for i ← 1 to a do
        Δ ← f_{T,1}(A_i ⊕ Δ)
    end for
    if |A_*| = n then
        T ← noM‖10‖N
        Δ ← f_{T,1}(A_* ⊕ Δ)
    else if |A_*| > 0 or |M| = 0 then
        T ← noM‖11‖N
        Δ ← f_{T,1}((A_*‖10*) ⊕ Δ)
    end if
    T ← 001‖N
    for i ← 1 to m do
        if M_i ⊕ Δ ∈ D_T then
            bad ← true
        end if
        D_T ← D_T ∪ (M_i ⊕ Δ)
        C_i ← f_{T,0}(M_i ⊕ Δ) ⊕ Δ
        Δ ← f_{T,1}(M_i ⊕ Δ)
        if bad = true then
            [C_i‖Δ ←$ {0,1}^{2n}]
        end if
    end for
    if |M_*| = n then
        T ← 100‖N
        C_* ← f_{T,0}(M_* ⊕ Δ) ⊕ Δ
        T ← f_{T,1}(M_* ⊕ Δ)
    else if |M_*| > 0 then
        T ← 101‖N
        C_* ← f_{T,0}((M_*‖10*) ⊕ Δ) ⊕ Δ
        T ← f_{T,1}((M_*‖10*) ⊕ Δ)
        T ← left_{|M_*|}(T)
    else
        T ← Δ
    end if
    return C_1‖...‖C_m‖C_*‖T
```

Figure 13: 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.

Integrity of SAEF. We again replace the forkcipher F with a pair of tweakable permutations $\pi_0 = (\pi_{\mathsf{T},0} \leftarrow\$ \mathrm{Perm}(n))_{\mathsf{T} \in \{0,1\}^t}$ and $\pi_1 = (\pi_{\mathsf{T},1} \leftarrow\$ \mathrm{Perm}(n))_{\mathsf{T} \in \{0,1\}^t}$, 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 2.

We additionally replace the tweakable permutation π_1 by a tweakable function f_1 with the same signature, yielding

$$\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{SAEF}[\pi_0,\pi_1]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{auth}}_{\mathrm{SAEF}[\pi_0,f_1]}(\mathcal{A}) + \sum_{\ell \in L \cap \{0,\ldots,\ell_{\max}\}} \frac{\boldsymbol{q}_\ell \cdot \ell \cdot (\ell-1)}{2^n}$$

by a similar argument as in the proof of SAEF's confidentiality.

To bound $\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{SAEF}[\pi_0,f_1]}(\mathcal{A})$, we consider the games G_2 and G_3 in Figure 14. It is easy to see that the game G_2 actually implements the game $\mathbf{auth}_{\mathrm{SAEF}[\pi_0,f_1]}$, because

the sets $\mathcal{D}_{\mathsf{T}}$ for $\mathsf{T} \in \{0,1\}^t$ 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 gameplaying [?], we have that $\Pr[\mathcal{A}^{G_2} \text{ forges}] - \Pr[\mathcal{A}^{G_3} \text{ forges}] \leq \Pr[\mathcal{A}^{G_2} \text{ sets bad}]$ and consequently $\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{SAEF}[\pi_0,f_1]}(\mathcal{A}) \leq \Pr[\mathcal{A}^{G_2} \text{ sets bad}] + \Pr[\mathcal{A}^{G_3} \text{ forges}]$.

Transition from G_2 to G_3. The flag bad being set means that for some $\mathsf{T} \in \{0,1\}^t$, the permutation $\pi_{\mathsf{T},0}$ and the function $f_{\mathsf{T},1}$ were used twice on the same input in an encryption 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.

Similarly as in the proof of confidentiality bound, we bound $\Pr[\mathcal{A}^{G_2} \text{ sets bad}]$ 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) with $\ell = \max(|A|_n, |M|_n)$, the flag bad can only be set by a collision on set $\mathcal{D}_{b\|N}$ with $b \in \{000, 001\}$. For the first block of AD (and the first block of message) there can be no collision because $\mathcal{D}_{000\|N}$ is initially empty (as is $\mathcal{D}_{001\|N}$). For each subsequent block A_i with $2 \leq i < \ell$, A_i is xored with a mask Δ. Given that bad has not been set before, Δ is an image of a fresh input under $f_{(000\|N),1}$. Therefore, the probability that A_i sets bad is bounded by $|\mathcal{D}_{000\|N}| \cdot 2^{-n} \leq (i-1) \cdot 2^{n-1}$ due to the fact $i < \ell \leq \ell_{\max}$ and due to the assumption $\ell_{\max} \leq 2^{n-1}$. Summing over i, we get that AD processing cannot set bad with a probability bigger than $\ell \cdot (\ell - 1) \cdot 2^{-n}$. The probability that bad is set while processing message blocks is bounded by $\ell \cdot (\ell - 1) \cdot 2^{-n}$ by a similar argument. Summing over all queries and knowing that for each ℓ, there are exactly $\boldsymbol{q}_\ell$ queries with $\ell = \max(|A|_n, |M|_n)$, we get that $\Pr[\mathcal{A}^{G_2} \text{ sets bad}] \leq \sum_{\ell \in \mathbb{L} \cap \{0,\ldots,\ell_{\max}\}} \boldsymbol{q}_\ell \cdot 2\ell \cdot (\ell - 1) \cdot 2^{-n}$.

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 [?].

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).

We bound $\Pr[\mathcal{A}^{G_3} \text{ forges}]$ by the following case analysis.

Case 1, $N \neq N^i$ for all $1 \leq i \leq q$: In this case, T is either compared to an output of $f_{(b\|N),1}$ with $b \in \{100, 110, 111\}$, or the rightmost $n - |T|$ bits of the preimage $\pi^{-1}_{(101\|N),0}(C_*)$ must have a specific value, and T is compared to $|T|$ bits of an image under $f_{(101\|N),1}$. Because all these random permutations (respectively functions) have not yet been sampled, the probability that the forgery succeeds is at most 2^{-n}.

In all the remaining cases, we assume that $\exists 1 \leq i \leq q : N = N^i$, i.e. the nonce N is reused from the i^{th} encryption query.

Case 2, $|T| = n$ and $|T^i| < n$, or $|T| < n$ and $|T^i| = n$: If $|T| = n$, then it must be equal to an image under a so far unused random function $f_{(100\|N),1}$ (because of the three-bit flag 100). Thus forging is equivalent to guessing the value of n random bits and succeeds with probability 2^{-n}.

If on the other hand $|T| < n$, then either the rightmost $n - |T|$ bits of the preimage $\pi^{-1}_{(101\|N),0}(C_*)$ must have a specific value and T is compared to $|T|$ bits of an image under

$f_{(101\|N),1}$, or C_* is compared to the output of $f_{(11b\|N),1}$ for $b \in \{0,1\}$ (if $|T| = 0$). With each of these three permutations unused in the experiment until now, the forgery succeeds with probability 2^{-n}.

Case 3, $|T| = n \wedge |T^i| = n$**:** The tag T is compared to an image under the random function $f_{(100\|N),1}$, which was also used to produce T^i. The adversary can forge in two ways, either by making sure that $C_* \oplus \Delta_{M,*} = C^i_* \oplus \Delta^i_{M,*}$ and reusing $T = T^i$, or by trying to guess the tag $T \neq T^i$ without this collision. Abusing the notation, we therefore have $\Pr[\mathcal{A}^{G_3} \text{ forges}|\textbf{Case 3}] \leq \Pr[\mathsf{coll}|\textbf{Case 3}] + \Pr[\mathcal{A}^{G_3} \text{ forges}|\neg\mathsf{coll} \wedge \textbf{Case 3}]$ where coll denotes the event $C_* \oplus \Delta_{M,*} = C^i_* \oplus \Delta^i_{M,*}$ conditioned on the event that $\mathsf{bad} \neq \mathsf{true}$ in game G_3.

We have that $\Pr[\mathcal{A}^{G_3} \text{ forges}|\neg\mathsf{coll} \wedge \textbf{Case 3}] \leq 2^{-n}$, because forging is equivalent to guessing the value of a fresh image under a random function. To bound $\Pr[\mathsf{coll}|\textbf{Case 3}]$, we investigate four subcases:

Case 3.1, $|A^i_*| = n$ **and** $|A_*| < n$ **or** $|A^i_*| < n$ **and** $|A_*| = n$**:** For the event coll to occur, there either are two indices $1 \leq j \leq m^i$ and $1 \leq j' \leq m$ for which $M^i_j \oplus \Delta^i_{M,j} = M'_j \oplus \Delta_{M,j'}$, or there are no such two indices and yet $C_* \oplus \Delta_{M,*} = C^i_* \oplus \Delta^i_{M,*}$ (in the first case, $\mathcal{A}'$ may simply reuse a part of C^i). Omitting the condition "**Case 3.1**", the probability of coll is bounded by

$$\Pr[\mathsf{coll}|\textbf{Case 3.1}] \leq \sum_{j,j'} \Pr[\mathsf{coll}_{j,j'}] + \Pr[\mathsf{coll}_*]$$

where $\mathsf{coll}_{j,j'}$ denotes the event $M^i_j \oplus \Delta^i_{M,j} = M'_j \oplus \Delta_{M,j'}$ conditioned on $\nexists \bar{j} : M^i_{\bar{j}} \oplus \Delta^i_{M,\bar{j}} = M_{j'-1} \oplus \Delta_{M,j'-1}$ (if $j' > 1$), and where coll_* denotes the event $C_* \oplus \Delta_{M,*} = C^i_* \oplus \Delta^i_{M,*}$ conditioned on $\nexists \bar{j}', \bar{j} : M^i_{\bar{j}} \oplus \Delta^i_{M,\bar{j}} = M_{\bar{j}'} \oplus \Delta_{M,\bar{j}'}$.

For $j' = 1$, $\mathcal{A}$ may force $\Delta_{M,1} = 0^n$ by setting $A = \varepsilon$, making $\mathsf{coll}_{j,1}$ equivalent to $\Delta^i_{M,j} = C_1 \oplus C^i_1$. However, all the masks $\Delta^i_{M,j}$ were generated uniformly, either due to the $\mathsf{bad} \neq \mathsf{true}$ condition for $j > 1$ or because $A^i \neq \varepsilon$ for $j = 1$, so the probability that any of them is equal to $C_1 \oplus C^i_1$ is 2^{-n}. If $A \neq \varepsilon$, then $\Delta_{M,1}$ is produced by a fresh random function, and therefore uniformly distributed, and the probability of $\mathsf{coll}_{j,1}$ is thus 2^{-n}.

For $j' > 1$, the condition $\nexists \bar{j} : M^i_{\bar{j}} \oplus \Delta^i_{M,\bar{j}} = M_{j'-1} \oplus \Delta_{M,j'-1}$ implies that $\Delta_{M,j'}$ is uniformly distributed, and so the probability of $\mathsf{coll}_{j,j'}$ is 2^{-n} for any j.

Finally, the event coll_* is equivalent to guessing the value of random n-bit string, and occurs with probability 2^{-n}.

As we have $j \leq m^i \leq \ell_{\max}$ and $j' \leq m \leq \ell_{\max}$, the probability of coll in this subcase is at most $(\ell^2_{\max} + 1) \cdot 2^{-n}$, and a forgery succeeds with probability at most $(\ell^2_{\max} + 2) \cdot 2^{-n}$.

Case 3.2, $|A^i_*| = n$ **and** $|A| = n$**:** The analysis is very similar as as in **Case 3.1**, except that now both $\Delta_{M,1}$ and $\Delta^i_{M,1}$ are produced by the same random function $f010\|N, 1$. We therefore need to consider the possibility, that $\mathcal{A}'$ forces $\Delta_{M,1} = \Delta^i_{M,1}$ by triggering a non-trivial collision $\Delta_{A,*} \oplus A_* = \Delta^i_{A,1} \oplus A^i_*$. If no such collision happens, the analysis proceeds exactly as in **Case 3.1**.

The analysis of the collision in AD processing is almost identical to the analysis of the collision in the message processing, except that we trivially have $\Delta_{A,1} = \Delta^i_{A,1} = 0^n$. This does not help the adversary in achieving a *non-trivial* collision, because such a collision can only occur past the trivial blockwise common prefix of A and A^i.

Denoting by coll_A the event $\Delta_{A,*} \oplus A_* = \Delta^i_{A,1} \oplus A^i_*$, we have

$$\Pr[\mathsf{coll}_A | \textbf{Case 3.2}] \leq \sum_{j,j'} \Pr[\mathsf{coll}_{j,j'}] + \Pr[\mathsf{coll}_*]$$

where $\mathsf{coll}_{j,j'}$ denotes the event $A^i_{\bar{j}} \oplus \Delta^i_{A,j} = A'_{\bar{j}} \oplus \Delta_{A,j'}$ conditioned on $\nexists \bar{j} : A^i_{\bar{j}} \oplus \Delta^i_{A,\bar{j}} = A_{j'-1} \oplus \Delta_{A,j'-1}$ (if $j' > 1$), and where coll_* denotes the event $A_* \oplus \Delta_{A,*} = A^i_* \oplus \Delta^i_{A,*}$ conditioned on $\nexists \bar{j}', \bar{j} : A^i_{\bar{j}} \oplus \Delta^i_{A,\bar{j}} = A_{\bar{j}'} \oplus \Delta_{A,\bar{j}'}$, such that we iterate over $\min\{\bar{j} | A^i_{\bar{j}} \neq A_{\bar{j}}\} \leq j' \leq m$ and $1 \leq j \leq m^i$.

The probability of each $\mathsf{coll}_{j,j'}$ is at most 2^{-n} by a similar argument as in the analysis of **Case 3.1** for all pairs $j'j$ except for $j = j' = \min\{\bar{j} | A^i_{\bar{j}} \neq A_{\bar{j}}\}$; for $j' = \min\{\bar{j} | A^i_{\bar{j}} \neq A_{\bar{j}}\}$ the event $\Delta_{A,j'} \oplus A_{j'} = \Delta^i_{A,j'} \oplus A^i_{j'}$ is impossible. There are never more than $\ell_{\max}$ possible values for j'.

We deduce that the probability of forger in this case is bounded by $(2\ell^2_{\max} + 2) \cdot 2^{-n}$.

Case 3.3, $0 < |A^i_*| < n$ **and** $0 < |A| < n$**:** Because of the use of injective padding in AD processing, the analysis of this case is almost identical with the analysis of **Case 3.2**, and a forgery succeeds with probability bounded by $(2\ell^2_{\max} + 1) \cdot 2^{-n}$.

Case 3.4, $A = A^i = \varepsilon$**:** In this special case, we have $\Delta_{M,1} = \Delta^i_{M,1} = 0^n$. The analysis in this case is the same as in case **Case 3.1**, except that we iterate over $\min\{\bar{j} | C^i_{\bar{j}} \neq C_{\bar{j}}\} \leq j' \leq m$, and deduce that a forgery succeeds with a probability bounded by $(\ell^2_{\max} + 2) \cdot 2^{-n}$.

Summing up, the probability of forgery in **Case 3** is bounded by $(2\ell^2_{\max} + 2) \cdot 2^{-n}$.

Case 4, $|T| = 0$ **and** $|T^i| = 0$**:** In this case, both messages are empty. We have the following subcases:

Case 4.1, $|A_*| = n$ **and** $|A^i_*| < n$**, or** $|A_*| < n$ **and** $|A^i_*| = n$**:** In this case, the value C_* is compared to an image of a random function that has not been sampled in the experiment. A forgery succeeds with probability 2^{-n} in this case.

Case 4.2, $|A_*| = n$ **and** $|A^i_*| = n$**:** In this case, $\mathcal{A}'$ can manage to forge either by triggering the collision $\Delta_{A,*} \oplus A_* = \Delta^i_{A,1} \oplus A^i_*$, or by guessing the tag otherwise. By a similar argument as **Case 3.2**, the probability of forgery is no more than $(\ell^2_{\max} + 2) \cdot 2^{-n}$ in this case.

Case 4.3, $|A_*| < n$ **and** $|A^i_*| < n$**:** Due to the use of the injective 10^* padding, this case is symmetric to **Case 4.2** and the probability of forgery is no more than $(\ell^2_{\max} + 2) \cdot 2^{-n}$.

The probability of forgery in this case is no more than $(\ell^2_{\max} + 2) \cdot 2^{-n}$.

Case 5, $0 < |T| < n$ **and** $0 < |T^i| < 0$**:** In this case, we have the following subcases:

Case 5.1, $|T| = |T^i|$**:** In this case, the adversary may forge either by forcing $C_* \oplus \Delta_{M,*} = C^i_* \oplus \Delta^i_{M,*}$ and reusing the tag, or by guessing a fresh valid pair (C_*, T). The analysis is very similar as in **Case 3**, except that the probability of forgery conditioned on $C_* \oplus \Delta_{M,*} \neq C^i_* \oplus \Delta^i_{M,*}$ changes.

In particular, the freshly sampled preimage $X = \pi^{-1}_{101\|N,0}(C_* \oplus \Delta_{M,*})$ will need to be of the form $X = Z\|10^*$ for some $Z \in \{0,1\}^{|T|}$ *and* simultaneously, the first $|T|$ bits of the freshly sampled image $Y = f_{101\|N,1}(X)$ 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 total probability of forgery is no greater than $(2\ell_{\max} + 1) \cdot 2^{-n} + (2^n - 1)^{-1}$

Case 5.2, $|T| \neq |T^i|$**:** As in the previous case, the adversary may attempt to trigger $C_* \oplus \Delta_{M,*} = C^i_* \oplus \Delta^i_{M,*}$. If this does not succeed, the probability of forgery is bounded by $(2^n - 1)/2^{2n}$ as in **Case 5.1**.

If this does succeed, the forgery is not as easily granted as before, as $\mathcal{A}'$ also needs $C_* \oplus \Delta_{M,*} = Z\|10^*$ for some $Z \in \{0,1\}^{|T|}$. Because clearly

$$\Pr[C_* \oplus \Delta_{M,*} = C^i_* \oplus \Delta^i_{M,*} \textbf{ and } C_* \oplus \Delta_{M,*} = Z\|10^*] \leq \Pr[C_* \oplus \Delta_{M,*} = C^i_* \oplus \Delta^i_{M,*}],$$

We conclude that the probability of forgery in this case is also bounded by $(2\ell_{\max} + 1) \cdot 2^{-n} + (2^n - 1)^{-1}$.

The probability of forgery in this case is no more than $(2\ell_{\max} + 1) \cdot 2^{-n} + (2^n - 1)^{-1}$.

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 $(2\ell_{\max}+1)\cdot 2^{-n}+(2^n-1)^{-1}$. The adversary $\mathcal{A}$ making q_v decryption queries thus forges with probability bounded by $q_v \cdot (2\ell_{\max} + 1) \cdot 2^{-n} + q_v \cdot (2^n - 1)^{-1}$. By back-substituting all the previous equalities, we obtain the claimed result. □

5.6 Modifying GCM for a Forkcipher

We finally define fGCM (as "forkcipher-based GCM,"), a nonce-based AEAD scheme parameterized by a tweakable forkcipher F (as defined in Section 4) with $\mathcal{T} = \{0,1\}^t$ for a positive t. An instance fGCM[F] $= (\mathcal{K}, \mathcal{E}, \mathcal{D})$ has a key space $\mathcal{K} = \{0,1\}^k$, nonce space $\mathcal{N} = \{0,1\}^{t-2}$, and the AD and message spaces are both $\{0,1\}^{\leq 2^{n/2}}$ (although the maximal message length influences the security). The ciphertext expansion of fGCM[F] is n. The encryption and decryption algorithms are defined in Figure 15 and the encryption algorithm is illustrated in Figure 16.

The scheme fGCM is inspired by the original Galois-Counter Mode [**?**] in that it processes all-but-last message block using counter mode, and computes a polynomial hash of thus-computed ciphertext blocks and the AD. Unlike the original GCM, the output of the GHASH is used as a masking offset for a primitive call, that encrypts the final ciphertext block and simultaneously produces the authentication tag.

In an encryption query, the message processed in blocks of n bits. The first $\max(0, \lceil |M|/n \rceil - 1)$ message blocks are encrypted with counter mode, using $\lceil \max(0, \lceil |M|/n \rceil - 1)/2 \rceil$ calls to F (because each F call produces $2n$ pseudorandom bits). Then, AD and the string formed by concatenating the just-computed ciphertext blocks is processed by GHASH to obtain a masking value Δ. The key for GHASH is derived as $\mathsf{left}_n(\mathsf{F}(K, 0^t, 0^n))$ The final message block is padded to n bits if needed, and then processed by a single F call, whose input and "left" output are xor-masked by Δ. The first $|M| \bmod n + n$ bits of the output of this call form the final ciphertext block and the tag.

Each call to F made during the encryption is using a tweak that is composed of

1. the nonce, and
2. a two-bit flag f.

The flag f takes different values for processing of different types of data blocks in the encryption algorithm:

$f = 01$ when in counter mode,

$f = 10$ in the final call if $|M| \equiv 0 \pmod n$,

$f = 11$ in the final call if $|M| \not\equiv 0 \pmod n$.

The decryption recomputes the value Δ, with it decrypts the final ciphertext block and and verifies the integrity of the inputs. If the verification succeeds, the rest of the ciphertext gets decrypted, otherwise the error symbol is returned.

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

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** $\mathsf{noM} \leftarrow 1$
  $\Delta \leftarrow 0^n$; $\mathsf{T} \leftarrow 000 \| N$
  for $i \leftarrow 1$ **to** a **do**
    if $A_i \oplus \Delta \in \mathcal{D}_{\mathsf{T}}$ **then**
      $\mathsf{bad} \leftarrow \mathsf{true}$
    end if
    $\mathcal{D}_{\mathsf{T}} \leftarrow \mathcal{D}_{\mathsf{T}} \cup (A_i \oplus \Delta)$
    $\Delta \leftarrow f_{\mathsf{T},1}(A_i \oplus \Delta)$
  end for
  if $|A_*| = n$ **then**
    $\mathsf{T} \leftarrow \mathsf{noM} \| 10 \| N$
    $\Delta \leftarrow f_{\mathsf{T},1}(A_* \oplus \Delta)$
  else if $|A_*| > 0$ **or** $|M| = 0$ **then**
    $\mathsf{T} \leftarrow \mathsf{noM} \| 11 \| N$
    $\Delta \leftarrow f_{\mathsf{T},1}((A_* \| 10^*) \oplus \Delta)$
  end if
  $\mathsf{T} \leftarrow 001 \| N$
  for $i \leftarrow 1$ **to** m **do**
    if $M_i \oplus \Delta \in \mathcal{D}_{\mathsf{T}}$ **then**
      $\mathsf{bad} \leftarrow \mathsf{true}$
    end if
    $\mathcal{D}_{\mathsf{T}} \leftarrow \mathcal{D}_{\mathsf{T}} \cup (M_i \oplus \Delta)$
    $C_i \leftarrow \pi_{\mathsf{T},0}(M_i \oplus \Delta) \oplus \Delta$
    $\Delta \leftarrow f_{\mathsf{T},1}(M_i \oplus \Delta)$
  end for
  if $|M_*| = n$ **then**
    $\mathsf{T} \leftarrow 100 \| N$
    $C_* \leftarrow \pi_{\mathsf{T},0}(M_* \oplus \Delta) \oplus \Delta$
    $T \leftarrow f_{\mathsf{T},1}(M_* \oplus \Delta)$
  else if $|M_*| > 0$ **then**
    $\mathsf{T} \leftarrow 101 \| N$
    $C_* \leftarrow \pi_{\mathsf{T},0}((M_* \| 10^*) \oplus \Delta) \oplus (\Delta)$
    $T \leftarrow f_{\mathsf{T},1}((M_* \| 10^*) \oplus \Delta)$
    $T \leftarrow \mathsf{left}_{|M_*|}(T)$
  else
    $T \leftarrow \Delta$
  end if
  $T \leftarrow \Delta$
  return $C_1 \| \ldots \| C_m \| C_* \| T$

proc $\mathrm{Dec}(N, A, C)$
  if $\mathsf{bad} = \mathsf{true}$ **then**
    $\boxed{\textbf{return } \perp}$
  end if
  $A_1, \ldots, A_a, A_* \xleftarrow{n} A$
  $C_1, \ldots, C_m, C_*, T \leftarrow \mathsf{csplit\text{-}b}_n(C)$
  $\mathsf{noM} \leftarrow 0$
  if $|C| = n$ **then** $\mathsf{noM} \leftarrow 1$
  $\Delta \leftarrow 0^n$; $\mathsf{T} \leftarrow 000 \| N$
  for $i \leftarrow 1$ **to** a **do**
    $\Delta \leftarrow f_{\mathsf{T},1}(A_i \oplus \Delta)$
  end for
  if $|A_*| = n$ **then**
    $\mathsf{T} \leftarrow \mathsf{noM} \| 10 \| N$
    $\Delta \leftarrow f_{\mathsf{T},1}(A_* \oplus \Delta)$
  end if
  if $|A_*| > 0$ **or** $|T| = 0$ **then**
    $\mathsf{T} \leftarrow \mathsf{noM} \| 11 \| N$
    $\Delta \leftarrow f_{\mathsf{T},1}((A_* \| 10^*) \oplus \Delta)$
  end if
  $\mathsf{T} \leftarrow 001 \| N$
  for $i \leftarrow 1$ **to** m **do**
    $M_i \leftarrow \pi^{-1}_{\mathsf{T},0}(C_i \oplus \Delta, 0) \oplus \Delta$
    $\Delta \leftarrow f_{\mathsf{T},1}(\pi^{-1}_{\mathsf{T},0}(C_i \oplus \Delta, 0))$
  end for
  if $|T| = n$ **then**
    $\mathsf{T} \leftarrow 100 \| N$
    $\bar{T} \leftarrow T$
  else if $|T| > 0$ **then**
    $\mathsf{T} \leftarrow 100 \| N$
    $\bar{T} \leftarrow 10^{n-|T|-1} \| T$
  else
    if $\Delta \neq C_*$ **then return** $\perp$
    return $\perp$
  end if
  $M_* \leftarrow \pi^{-1}_{\mathsf{T},0}(C_* \oplus \Delta) \oplus \Delta$
  $\tilde{T} \leftarrow \mathsf{left}_{|T|}(f_{\mathsf{T},1}(M_* \oplus \Delta))$
  if $\bar{T} \neq \mathsf{right}_{|T|}(M_*) \| \tilde{T}$ **then**
    return $\perp$
  end if
  $M_* \leftarrow \mathsf{left}_{|T|}(M_*)$
  return $M_1 \| \ldots \| M_m \| M_*$

Figure 14: The games G_2 and G_3 for bounding $\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{SAEF}[\pi_0, f_1]}$. The game G_2 does *not* contain the boxed statements, while G_3 does.

Algorithm $\mathcal{E}(K, N, A, M)$
$L \leftarrow \mathsf{left}_n(\mathsf{F}_K^{0^t}(0^n))$
$M', M_* \leftarrow \mathsf{msplit}_n(M)$
$S \leftarrow \varepsilon$
for $i \leftarrow 1$ **to** $\lceil |M'|/2n \rceil$ **do**
  $S \leftarrow S \| \mathsf{F}_K^{01\|N}(\langle i \rangle_n)$
end for
$C' \leftarrow M' \oplus \mathsf{left}_{|M'|}(S)$
$\Delta \leftarrow \mathrm{GHASH}(L, A, C')$
if $|M_*| = n$ **then**
  $\mathsf{T} \leftarrow 10 \| N$
else
  $M_* \leftarrow M_* \| 10^*$
  $\mathsf{T} \leftarrow 11 \| N$
end if
$R \leftarrow \mathsf{F}_K^{\mathsf{T}}(M_* \oplus \Delta) \oplus (\Delta \| 0^n)$
$C \leftarrow C \| \mathsf{left}_{|M_*|+n}(R)$
return C
end Algorithm

Algorithm $\mathrm{GHASH}(L, A, C)$
$\alpha \leftarrow n - (|A| \bmod n)$
$\gamma \leftarrow n - (|C| \bmod n)$
$X \leftarrow A \| 0^\alpha \| C \| 0^\gamma \| \langle |A| \rangle_{n/2} \| \langle |C| \rangle_{n/2}$
$X_1, \ldots, X_x \xleftarrow{n} X$
$Y \leftarrow 0^n$
for $j \leftarrow 1$ **to** x **do**
  $L \cdot (Y \oplus X_j)$ ▷ in $\mathsf{GF}(2^n)$
end for
return Y
end Algorithm

Algorithm $\mathcal{D}(K, N, A, C)$
$L \leftarrow \mathsf{left}_n(\mathsf{F}_K^{0^t}(0^n))$
$C', C_*, T \leftarrow \mathsf{csplit}_n(C)$
$\Delta \leftarrow \mathrm{GHASH}(L, A, C')$
if $|T| = n$ **then**
  $\mathsf{T} \leftarrow 10 \| N$
  $\bar{T} \leftarrow T$
else
  $\mathsf{T} \leftarrow 11 \| N$
  $\bar{T} \leftarrow 10^{n-|T|-1} \| T$
end if
$\tilde{T} \leftarrow \mathsf{left}_{|T|}(\mathsf{F}^{\rho\mathsf{T}}_K(C_* \oplus \Delta, 0))$
$M_* \leftarrow \mathsf{F}^{-1\,\mathsf{T}}_K(C_* \oplus \Delta, 0) \oplus \Delta$
if $\bar{T} \neq \mathsf{right}_{n-|T|}(M_*) \| \tilde{T}$ **then**
  return $\perp$
end if
$M_* \leftarrow \mathsf{left}_{|T|}(M_*)$
$S \leftarrow \varepsilon$
for $i \leftarrow 1$ **to** $\lceil |C'|/2n \rceil$ **do**
  $S \leftarrow S \| \mathsf{F}_K^{01\|N}(\langle i \rangle_n)$
end for
$M' \leftarrow C' \oplus \mathsf{left}_{|C'|}(S)$
return $M' \| M_*$
end Algorithm

Figure 15: The fGCM[F] AEAD scheme. Refer to Section 2 for the definitions of string manipulation notation.

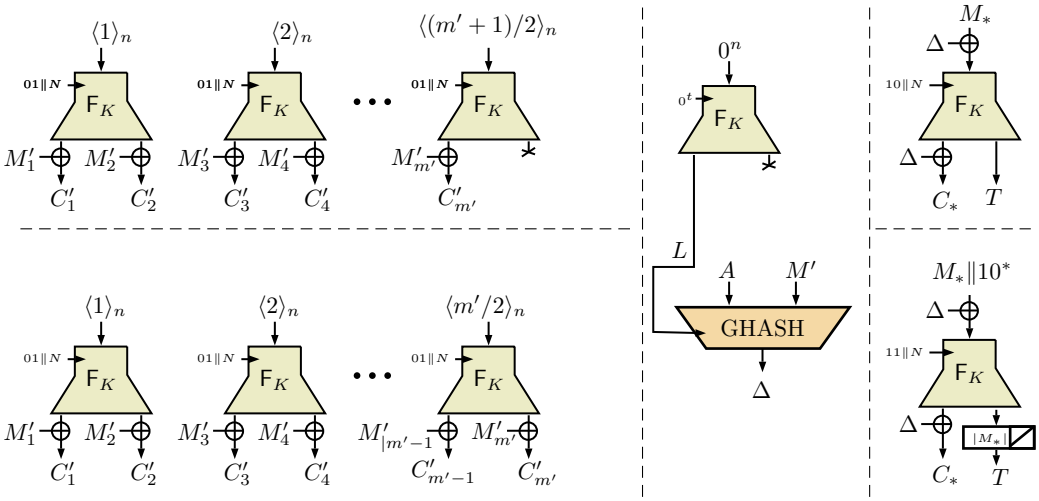

Figure 16: The encryption algorithm of fGCM[F] mode (see Figure 15 for the definition of M'). We have $m' = |M'|_n$. The picture illustrates the processing of M' when length of M' is an odd multiple of n (**top left**) and when the length of M' is an even multiple of n (**bottom left**), the processing of the final message block M_* when length of the message M is a multiple of n (**top right**) and when the length of message M is either zero or not a multiple of n (**bottom right**), and the computation of the value Δ (**middle**).

5.7 Security of fGCM

For the security analysis of fGCM, we will again work with the previously defined fine-grained adversarial resources. We state the formal claim about the nonce-based AE security of fGCM in Theorem 4.

Theorem 4. *Let* F *be a tweakable forkcipher with* $\mathcal{T} = \{0,1\}^t$*. Then for any nonce-respecting adversary* $\mathcal{A}$ *whose resources are bounded by* $q, q_v, \sigma, \ell_{\max}$ *and* $\boldsymbol{q}$ *(in the sense we just defined) such that* $\ell_{\max} \leq 2^n/4$*, we have*

$$\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{fGCM}[\mathsf{F}]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{B}) + \sum_{\ell \in L \cap \{0,\ldots,\ell_{\max}\}} \frac{\boldsymbol{q}_\ell \cdot \ell \cdot (\ell-1)}{2^n},$$

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

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*.*

Corollary 2. *From* $\sum_{\ell \in L \cap \{0,\ldots,\ell_{\max}\}} \ell \cdot \boldsymbol{q}_\ell \leq 2\sigma$ *it immediately follows that*

$$\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{fGCM}[\mathsf{F}]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{B}) + 4 \cdot \frac{\sigma^2}{2^n}$$

Remark 3 (Interpretation of the bounds in Theorem 4 and Corollary 2). According to Corollary 2, the security of fGCM is birthday bounded in the general case. However, when most of the encryption queries are very *short*, the security guarantees given by the fine-grained bound in Theorem 4 become very competitive.

Revisiting the example where q_v is reasonably small, and where the vast majority of queries consists of a single block of AD and a single message block, and a tiny portion $\gamma \ll q$ of messages is expected to contain a small number of message blocks β, the security

bounds of fGCM become near optimal:

$$\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{fGCM}[\mathsf{F}]}(\mathcal{A}) \le \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{B}) + \frac{\gamma \cdot \beta \cdot (\beta - 1)}{2^n}$$
$$\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{fGCM}[\mathsf{F}]}(\mathcal{A}) \le \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{C}) + \frac{q_v \cdot \beta}{2^n} + \frac{q_v \cdot 2^n}{(2^n - 1)^2}.$$

The proof of Theorem 4 relies on the AXU property of GHASH stated in Lemma 1.

Lemma 1 (GHASH [**?**])**.** *For any positive integers $n, \ell_{\max}$ such that $\ell_{\max} \cdot n < 2^{n/2}$, for any pairs of strings $(A, C'), (\tilde{A}, \tilde{C}') \in (\{0,1\}^{\le \ell_{\max} \cdot n})^2$ such that $(A, C') \neq (\tilde{A}, \tilde{C}')$, and for any $c \in \{0,1\}^n$ we have that*

$$\Pr[L \leftarrow\!\$\, \{0,1\}^n : \mathrm{GHASH}(L, A, C') \oplus \mathrm{GHASH}(L, \tilde{A}, \tilde{C}') = c] \le \frac{\ell_{\max} + 1}{2^n}$$

with GHASH *defined in Figure 15.*

Proof of Theorem 4. We first tackle confidentiality and then integrity of fGCM.

Confidentiality of fGCM. As before, we first replace the forkcipher F with a pair of tweakable permutations π_0 and π_1 (i.e., $\pi_0 = (\pi_{\mathsf{T},0} \leftarrow\!\$\, \mathrm{Perm}(n))_{\mathsf{T} \in \{0,1\}^t}$ is a collection of independent uniform elements of $\mathrm{Perm}(n)$ indexed by the elements of $\mathsf{T} \in \{0,1\}^t$, and similarly for $\pi_1 = (\pi_{\mathsf{T},1} \leftarrow\!\$\, \mathrm{Perm}(n))_{\mathsf{T} \in \{0,1\}^t}$). We let $\mathrm{fGCM}[\pi_0, \pi_1]$ denote the fGCM mode that uses π_0, π_1 instead of F, and deduce the following inequality:

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

by a similar argument as in the proof of Theorem 2.

We 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., $f_b = (f_{\mathsf{T},b} \leftarrow\!\$\, \mathrm{Func}(n))_{\mathsf{T} \in \{0,1\}^t}$ for $b \in \{0,1\}$). Denoting the fGCM mode using these random functions by $\mathrm{fGCM}[f_0, f_1]$, we have that

$$\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{fGCM}[\pi_0,\pi_1]}(\mathcal{A}) \le \mathbf{Adv}^{\mathbf{priv}}_{\mathrm{fGCM}[f_0,f_1]}(\mathcal{A}) + \sum_{\ell \in L \cap \{0,\ldots,\ell_{\max}\}} 2 \cdot \frac{\boldsymbol{q}_\ell \cdot \ell \cdot (\ell - 1)}{2^n}$$

because each query uses a distinct nonce, and thus for each N, the functions $f_{\mathsf{T},0}$ and $f_{\mathsf{T},1}$ with $\mathsf{T} = b_0 b_1 \| N$ and $b_0 b_1 \in \{0,1\}^2$ are always used only in a single query. For $b_0 b_1 \in \{00, 10, 11\}$, the associated random functions are used at most once, so their distribution is indistinguishable from that of the corresponding random permutations. For $b_0 b_1 = 01$, there will be exactly $\boldsymbol{q}_\ell$ values of N for which both $f_{\mathsf{T},0}$ and $f_{\mathsf{T},1}$ will be used at most ℓ times. Replacing such a $\pi_{01\|N,b}$ by $f_{01\|N,b}$ augments the bound by at most $\ell \cdot (\ell - 1) \cdot 2^{-n-1}$ by the RP-RF switching lemma [**?**] and a standard hybrid argument. By summing over ℓ, and $b \in \{0,1\}$, we obtain the bound.

Bounding $\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{fGCM}[f_0,f_1]}(\mathcal{A})$ is relatively straightforward, because in each ciphertext C that $\mathcal{A}$ sees, one part of C is computed as an xor of message bits and images of the random functions $f_{01\|N,0}$ and $f_{01\|N,1}$. Since none of the functions is ever evaluated twice on any input, all these bits are uniformly distributed. The remainder of C consists of a concatenation of the images of two random functions, that are only evaluated once during the whole experiment. Every ciphertext is thus distributed uniformly, and we have $\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{fGCM}[f_0,f_1]}(\mathcal{A}) = 0$.

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

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

by a similar argument as in the proof of Theorem 2.

To bound $\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{fGCM}[\pi_0,\pi_1]}(\mathcal{A})$, we first use the result of Bellare [**?**] that states that

$$\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{fGCM}[\pi_0,\pi_1]}(\mathcal{A}) \leq q_v \cdot \mathbf{Adv}^{\mathbf{auth}}_{\mathrm{fGCM}[\pi_0,\pi_1]}(\mathcal{A}')$$

where $\mathcal{A}'$ is an adversary with identical resources to those of $\mathcal{A}$, except that $\mathcal{A}'$ only makes a single decryption query.

What remains is to bound $\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{fGCM}[\pi_0,\pi_1]}(\mathcal{A}')$. The adversary $\mathcal{A}'$ makes q encryption queries N^i, A^i, M^i and receives ciphertexts C^i in response for $i = 1, \ldots, q$. For each i we define ${M'}^i, M^i_* = \mathsf{msplit}_n(M^i)$ and ${C'}^i, C^i_*, T^i = \mathsf{csplit}_n(C^i)$, and we let Δ^i denote the output of GHASH in the i^{th} query. $\mathcal{A}'$ also makes a single decryption query (N, A, C), for which we define $C', C_*, T = \mathsf{csplit}_n(C)$, and let Δ denote the output of GHASH evaluation in the decryption query. We proceed with the following case analysis.

Case 1, $N \neq N^i$ for all $1 \leq i \leq q$**:** T is either compared to an output of $\pi_{(b_0b_1\|N),1}$ with $b_0b_1 \in \{10, 11\}$, or the rightmost $n - |T|$ bits of the preimage $\pi^{-1}_{(11\|N),0}(C_*)$ must have a specific value, and T is compared to $|T|$ bits of an image under $\pi_{(11\|N),1}$. Because all these random permutations have not yet been sampled, the probability that the forgery succeeds is at most 2^{-n}.

In all the remaining cases, we assume that $\exists 1 \leq i \leq q : N = N^i$, i.e., the nonce N is reused from the i^{th} encryption query.

Case 2, $|T| = n \wedge |T^i| < n$ **or** $|T| < n \wedge |T^i| = n$**:** If $|T| = n$, the tag T is compared to an image under the random permutation $\pi_{(10\|N),1}$. However, no image under this permutation was used in the game (because we only used $\pi_{(11\|N),1}$). Thus forging is equivalent to guessing the value of n random bits and succeeds with probability 2^{-n}.

If on the other hand $|T| < n$, the rightmost $n - |T|$ bits of the preimage $\pi^{-1}_{(11\|N),0}(C_*)$ must have a specific value, and T is compared to $|T|$ bits of an image under $\pi_{(11\|N),1}$. These two permutation being unused until now, the forgery succeeds with probability 2^{-n}.

Case 3, $|T| = n \wedge |T^i| = n$**:** We will investigate two subcases:

Case 3.1, $(A, C') \neq (A^i, {C'}^i)$**:** The forgery can succeed in two ways in this case. If the adversary reuses the tag, i.e. $T = T^i$, then the forgery succeeds only if $\pi_{(10\|N),1}(\pi^{-1}_{(10\|N),0}(C_* \oplus \Delta)) = \pi_{(10\|N),1}(\pi^{-1}_{(10\|N),0}(C^i_* \oplus \Delta^i))$, which is equivalent with the event $\Delta \oplus \Delta_i = C_* \oplus C^i_*$. As $(A, C') \neq (A^i, {C'}^i)$, the probability of this event is no more than $\ell_{\max}/2^n$ due to the AXU property of GHASH (note that $\ell_{\max}$ is the maximal number of blocks in M, but C' always has 1 block less than M).

Otherwise, if $T \neq T^i$, the forgery succeeds if and only if $C_* \oplus \Delta \neq C^i_* \oplus \Delta_i \wedge \pi_{(10\|N),1}(\pi^{-1}_{(10\|N),0}(C_* \oplus \Delta)) = T$. Because $\pi_{(10\|N),1} \circ \pi^{-1}_{(10\|N),0}$ is a random permutation, and it was only evaluated on a single image, the probability of this event is at most $1/(2^n - 1)$.

Because the two ways of forging in this case are mutually exclusive, the probability of forgery is bounded by $\max(\ell_{\max}/2^n, 1/(2^n - 1))$.

Case 3.2, $(A, C') = (A^i, {C'}^i)$**:** Here, we must have $(C_*, T) \neq (C^i_*, T^i)$ for the forgery to be valid, but we also have $\Delta = \Delta^i$. This implies that $C_* \neq C^i_*$ and $T \neq T_i$ (because the tags are produced as images of permutations). A successful forgery implies that $\pi_{(10\|N),1}(\pi^{-1}_{(10\|N),0}(C_* \oplus \Delta)) = T$, such that the random permutation $\pi_{(10\|N),1} \circ$

$\pi^{-1}_{(10\|N),0}$ was only evaluated on $C^i_* \oplus \Delta^i$ before, which occurs with probability at most $1/(2^n - 1)$.

The probability of forgery in this case is no more than $\max(\ell_{\max}/2^n, 1/(2^n - 1))$.

Case 4, $|T| < n \wedge |T^i| < n$**:** Again, we have two subcases:

Case 4.1, $(A, C') \neq (A^i, C'^i)$**:** Similarly to **Case 3**, $\mathcal{A}'$ can succeed in two ways in this case. The adversary may succeed in creating the collision $C_* \oplus \Delta = C^i_* \oplus \Delta_i$ and forge by setting $T = T^i$, which happens with probability bounded by $\ell_{\max}/2^n$ due to the AXU property of GHASH.

Otherwise, if $C_* \oplus \Delta \neq C^i_* \oplus \Delta_i$, the forgery succeeds if and only if $\pi^{-1}_{(10\|N),0}(C_* \oplus \Delta) \oplus \Delta = Y\|10^{n-|T|-1} \wedge \pi_{(10\|N),1}(\pi^{-1}_{(10\|N),0}(C_* \oplus \Delta)) = T\|Z$ for arbitrary $Y \in \{0,1\}^{|T|}$ and $Z \in \{0,1\}^{n-|T|}$. The probability of the event $\pi^{-1}_{(10\|N),0}(C_* \oplus \Delta) \oplus \Delta = Y\|10^{n-|T|-1}$ is at most $\{0,1\}^{|T|}/(2^n - 1)$ because $\pi^{-1}_{(10\|N),0}$ has only been evaluated on a single point $C^i_* \oplus \Delta_i \neq C_* \oplus \Delta$ before. The probability of the event $\pi_{(10\|N),1}(Y\|10^{n-|T|-1}) = T\|Z$ conditioned on $\pi^{-1}_{(10\|N),0}(C_* \oplus \Delta) = Y\|10^{n-|T|-1}$ is then at most $|\{0,1\}^{n-|T|}|/(2^n - 1)$, because $\pi_{(10\|N),1}$ was also only evaluated on a single input different from $Y\|10^{n-|T|-1}$. Multiplying the two terms yields $2^n/(2^n - 1)^2$.

Because the two ways of forging in this case are not clearly mutually exclusive (both allow $T = T^i$), the probability of forgery is bounded by $\ell_{\max}/2^n + 2^n/(2^n - 1)^2$.

Case 4.2, $(A, C') = (A^i, C'^i)$**:** We must have $(C_*, T) \neq (C^i_*, T^i)$ for the forgery to be valid, ad we also have $\Delta = \Delta^i$. This implies that $C_* \neq C^i_*$ (because forging with $C_* = C^i_*$ and $T \neq T^i_*$ is impossible), which in turn implies $C_* \oplus \Delta \neq C^i_* \oplus \Delta_i$. By a similar argument as in the second part of **Case 4.1**, the probability of forgery in this case is bounded by $2^n/(2^n - 1)^2$.

The probability of forgery in this case is thus no more than $\ell_{\max}/2^n + 2^n/(2^n - 1)^2$.

By taking the maximum over all cases, the probability that a single-decryption-query adversary $\mathcal{A}'$ manages to forge $\ell_{\max} \cdot 2^{-n} + 2^n \cdot (2^n - 1)^{-2}$. The adversary $\mathcal{A}$ making q_v decryption queries thus forges with probability bounded by $q_v \cdot \ell_{\max} \cdot 2^{-n} + q_v \cdot 2^n \cdot (2^n - 1)^{-2}$. By back-substituting all the previous inequalities, we obtain the claimed result. □

5.8 Efficient Instantiation

Judging by the generic definitions of PAEF and SAEF in Figures 8 and 11, these two modes look anything but efficient. First, half of the output of every forkcipher call is being wasted when processing the associated data, and the decryption algorithms have to call both the inversion and the reconstruction algorithm of the forkcipher for every ciphertext block.

However, both PAEF and SAEF were designed with our proposed instance for tweakable forkcipher, the ForkAES, in mind. Addressing the two issues just raised, and referring to Section 3, with ForkAES

- The computational cost of "wasting" half of the output corresponds to 10 AES rounds; we evaluate the first five rounds to compute state from which the computation is forked, and then we only need to compute the five more rounds to compute the "right" output. Thus in the AD processing, PAEF and SAEF perform as any other rate-1 mode.

scheme	Cost of encryption in (# of AES rounds)/10									
	$a = 0$				$a = 1$				$a = 2$	
	m=1	m=2	m=3	m=4	m=0	m=1	m=2	m=3	m=1	m=2
GCM	2+2	3+3	4+4	5+5	1+2	2+3	3+4	4+5	2+4	3+5
CCM	4	6	8	10	3	5	7	9	6	8
OCB3	3	4	5	6	3	4	5	6	5	6
CLOC	3	5	7	9	2	4	6	8	5	7
Deoxys-I	2.8	4.2	5.6	7	2.8	4.2	5.6	7	5.6	7
KIASU$^{\neq}$	2	3	4	5	2	3	4	5	4	5
PAEF	1.5	3	4.5	6	1.5	2.5	4	5.5	3.5	5
SAEF	1.5	3	4.5	6	1.5	2.5	4	5.5	3.5	5
fGCM	1.5+1	2.5+2	3+3	5+4	1.5+2	1.5+2	2.5+3	3+4	1.5+3	2.5+4

Figure 17: The comparison of performance of GCM [**?**], CCM [**?**], OCB3 [**?**], CLOC [**?**], Deoxys-I [**?**], KIASU [**?**] and the AE schemes presented in Section 5. Here $a = |A|_n$ and $m = |M|_n$, and the cells of the form "$b + g$" mean "$b \cdot 10$ AES rounds and g multiplications in $\mathsf{GF}(2^{128})$." GCM, CCM, OCB3 and CLOC are assumed to be instantiated with AES. For schemes that compute a derived key that is the same for all queries (OCB and GCM), the complexity of this step is ignored.

- The computational cost of the inversion and the reconstruction algorithm can be factored when evaluated on *the same input*. We first compute the 5 inverse rounds that are common for both the reconstruction and the inversion algorithms to reach the forking state S, and then compute five more backwards round to compute the plaintext, and five other *forward* rounds to compute the other output half. The computational cost of this is exactly the same as the complete forward evaluation of ForkAES: 15 AE rounds.

In an optimized implementation, the actual interface of ForkAEs would be slightly different as well. The general rule for the implementations could be characterized as "implement only what you need." For example for the PAEF and SAEF modes, one would have both the usual forward interface ForkAES$(\cdot,\cdot,\cdot)$ with $2n$ output bits, but also a dedicated interface ForkAES-r$(\cdot,\cdot,\cdot)$ that evaluates only the right branch of the forward ForkAES, for the AD processing. On the other hand, one would have a dedicated "composite" inverse interface ForkAES$(\cdot,\cdot,\cdot)$ that would take the key, the tweak, and *the left output block*, and directly output $2n$ bits—the plaintext block, and the reconstructed right output block. Thus we would factor the inversion and reconstruction, and dispense with the indicator bit input.

In this light, the performance of SAEF[ForkAES] and PAEF[ForkAES, ν] looks much more promising, with the cost of both encryption and decryption algorithm being equivalent to about 1 AES evaluation per block of AD, and 1.5 AES evaluations per block of message.

PAEF and SAEF versus other modes. Of course, the factor of 1.5 is restrictive for long plaintexts, but SAEF and PAEF were designed to be efficient for *short messages*. To illustrate their efficiency, we compare the computational cost (in terms of the number of AES128 calls) of encrypting very short queries by PAEF and SAEF with that of GCM, CCM, OCB, CLOC and Deoxys-I in Figure 17.

Note also that in the plausible application scenarios (as those mentioned in Section 1), it is unlikely that a pipelined hardware accelerator for AES would be available. Therefore, the metric used in Table 17 yields an accurate comparison of actual performance of the mentioned AE modes.

5.9 Deterministic MiniAE

In the introduction, we stated that a forkcipher is nearly, but not exactly, an AE primitive: we clarify this statement in this section. It is easy to see that the syntax and security goals of a forkcipher, as proposed in Section 4, capture neither AE functionality nor AE security goals. Yet, *constructing* a secure PRI (with the same signature) from the forkcipher is trivial. 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 [**?**].

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 18.

$\boxed{\mathbf{pri\text{-}real}_{\Pi}}$	$\boxed{\mathbf{pri\text{-}ideal}_{\Pi}}$
proc initialize $K \leftarrow\!\!\$\ \mathcal{K}$	**proc initialize** **for** $N, A \in \mathcal{N} \times \mathcal{A}$ **do** $\quad f_{N,A} \leftarrow\!\!\$\ \mathrm{Inj}(\tau)$
proc $\mathrm{Enc}(N, A, M)$ **return** $\mathcal{E}(K, N, A, M)$	**proc** $\mathrm{Enc}(N, A, M)$ **return** $f_{N,A}(M)$

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

Figure 18: 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 $1 \leq \nu < t$, 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 19.

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".

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 $1 \leq \nu < t$ and let $1 \leq \tau \leq n$. 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 $q + q_v \leq 2^{n-1}$, we have*

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

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

1: **Algorithm** $\mathcal{D}(K, N, A, C\|T)$
2: **if** $T = \mathsf{F}^{\rho}{}_K^{N\|A}(C, 0)$ **then**
3: **return** $\mathsf{F}^{-1}{}_K^{N\|A}(C, 0)$
4: **end if**
5: **return** $\perp$
6: **end Algorithm**

Figure 19: The MAE$[\mathsf{F}, \nu]$ AEAD scheme.

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

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

by a similar argument as in the proof of Theorem 2. In the rest of the analysis, we will refer to 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 20. We claim that $\Pr[\mathcal{A}^{\mathbf{pri\text{-}real}_\Pi} \to 1] = \mathcal{A}^{\Gamma_1} \to 1]$ and that $\Pr[\mathcal{A}^{\mathbf{pri\text{-}ideal}_\Pi} \to 1] = \mathcal{A}^{\Gamma_0} \to 1]$, which yields

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

It is easy to verify the latter equality; with the boxed statements removed, the code in Figure 20 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 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 MAE$[\pi_0, \pi_1]$.

In addition, the games Γ_0 and Γ_1 are identical until bad, so we have $\Pr[\mathcal{A}^{\Gamma_1} \to 1] - \Pr[\mathcal{A}^{\Gamma_0} \to 1] \le \Pr[\mathcal{A}^{\Gamma_1}$ sets $\mathsf{bad}]$ by the Fundamental lemma of gameplaying [**?**]. 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 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 $\mathsf{bad}] \le \sum_{i=1}^{3} \sum_{j=1}^{q+q_v} \Pr[\mathsf{bad}^i_j]$.

We have that

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

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 $\sum_{i=1}^{q+q_v} \Pr[\mathsf{bad}_1^i] \leq 2 \cdot (q + q_v)^2/2 \cdot (2^n - 1)$.

Then, we have that

$$\Pr[\mathsf{bad}_2^i] \leq \cdot \frac{2^n}{2^{2n} - i + 1} \leq \frac{2^n}{2^n(2^n - 1)} \leq \frac{1}{2^n - 1}$$

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 $\sum_{i=1}^{q+q_v} \Pr[\mathsf{bad}_2^i] \leq (q + q_v)/(2^n - 1)$.

Finally, we have that

$$\Pr[\mathsf{bad}_3^i] = \frac{1}{2^n - |f_{N,A}|} \cdot \left(1 - \frac{(2^n - |f_{N,A}|)^2}{2^{2n} - |f_{N,A}|}\right) \leq \frac{1}{2^n - |f_{N,A}|} \leq \frac{1}{2^n - (i-1)} \leq \frac{1}{2^{n-1}}$$

because bad_j^i occurs in the i^{th} query if and only if $b' = b'' = 1$. The fina;l inequality then follows from the assumption $(q+q_v) \leq 2^{n-1}$. Summing over i, we get that $\sum_{i=1}^{q+q_v} \Pr[\mathsf{bad}_3^i] \leq (q + q_v)/(2^{n-1})$.

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

5.10 Discussion

We presented three modes of operation for a tweakable forkcipher. Each of the three nodes 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 AE schemes that excel in short-message encryption and decryption performance.

However, all three modes also share a certain deficiency: the technique we use to process arbitrary-length messages requires the ciphertext expansion to be exactly n bits. This can be limiting in many applications, in which bandwidth is limited. We leave solving this issue as an interesting open question.

Starting from a FIL PRI. As mentioned in Section 1, we also considered designing the VIL AEAD schemes as mods of operation of a FIL (tweakable) PRI, but we encountered several setbacks when heading in this direction.

Firstly, the FIL PRI does not seem suitable for designing modes that process *arbitrary* length messages. Out of the three modes we proposed, PAEF and SAEF cannot be recast as modes of a PRI, because they effectively discard one half of every call to the used primitive F. If F were a PRI, this would make decryption impossible. The parallel-permutation structure is crucial for these two modes. The fGCM, on the other hand, could be recast as a mode of a FIL PRI, as it does not require the computation of the inverse of F, except for the final message block. The problem here is that when used with a FIL PRI, fGCM could not have a constant ciphertext expansion. When instantiated 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 find a mode of operation for a FIL-PRI that would process arbitrary length messages with *constant* stretch. Whether such a mode of operation for forkciphers exists is another interesting open question.

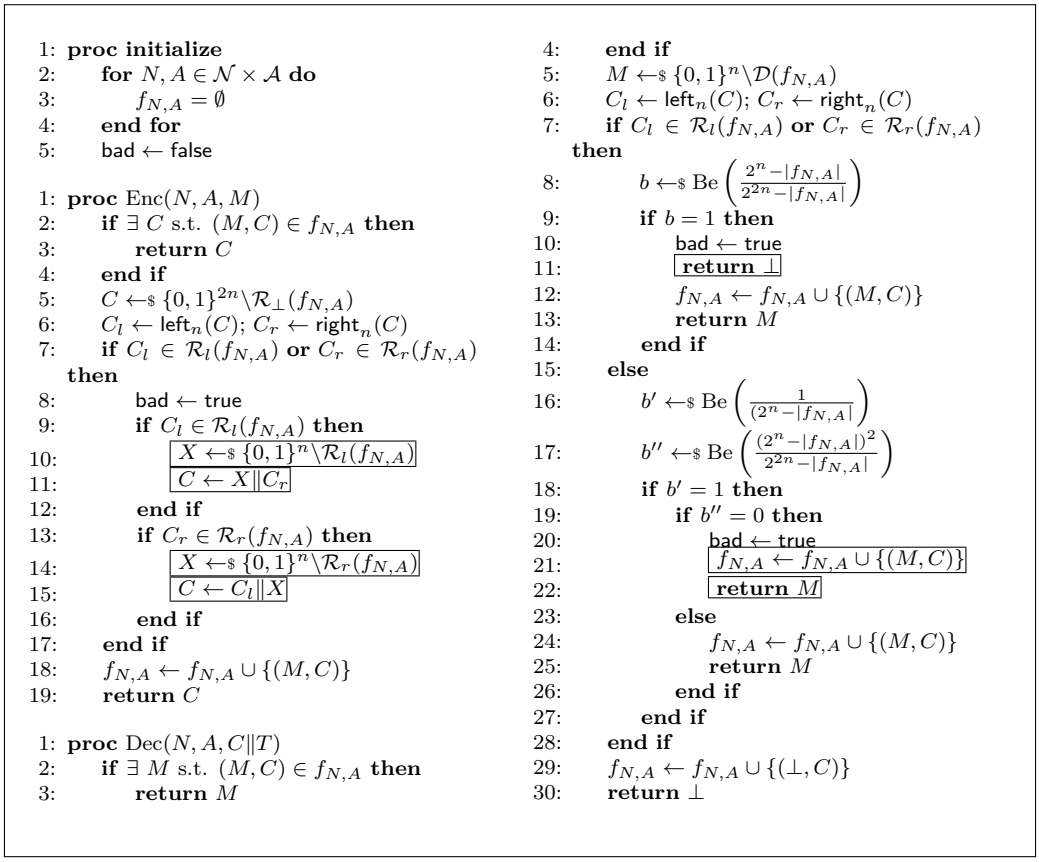

Figure 20: The games Γ_0 and Γ_1 for bounding $\mathbf{Adv}^{\mathbf{pri}}_{\mathrm{MAE}[\pi_0,\pi_1]}$. The game Γ_1 does *not* contain the boxed statements, while Γ_0 does. The games implement the (partially defined) injective funcitons $f_{N,A} : \{0,1\}^n \to \{0,1\}^{2n}$ as initially-empty sets 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 $\mathcal{D}(f_{N,A}) = \{M \in \{0,1\}^n | \exists (M, C) \in f_{N,A}\}$, $\mathcal{R}(f_{N,A}) = \{C \in \{0,1\}^{2n} | \exists (M, C) \in f_{N,A}$ s.t. $M \neq \perp\}$, $\mathcal{R}_l(f_{N,A}) = \{L \in \{0,1\}^n | \exists$ some $L \| X \in \mathcal{R}(f_{N,A})\}$ and $\mathcal{R}_r(f_{N,A}) = \{R \in \{0,1\}^n | \exists$ 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 $\Pr[\mathrm{Be}(p) = 1] = p$.

Constructing a FIL PRI. The second question is if we can find better instances of a FIL PRI. Our preferred choice was AES-based and thus the ForkAES. Yet, as evidenced by the result in Section 5.9, there is an unavoidable birthday-type quantitative gap between the PRI security, and the kind of security that ForkAES inherently possesses. A direct instance of a *true* FIL tweakable PRI is a question we leave open.

Acknowledgments.

This work was partly supported by the EU H2020 POMEGRANATE project, funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 708815, while Reza Reyhanitabar was a Marie Skłodowska-Curie Individual Fellow in KU Leuven.