Generic Attack on Duplex-Based AEAD Modes Using Random Function Statistics

Henri Gilbert, Louiza Khati, Rachelle Heim Boissier, Yann Rotella

Generic Attack on Duplex-Based AEAD Modes using Random Function Statistics

Henri Gilbert[1,2], Rachelle Heim Boissier[2], Louiza Khati[1], Yann Rotella[2]

[1]ANSSI, France
[2]Université Paris-Saclay, UVSQ, CNRS, Laboratoire de mathématiques de Versailles, 78000, Versailles, France

Abstract. Duplex-based authenticated encryption modes with a sufficiently large key length are proven to be secure up to the birthday bound $2^{\frac{c}{2}}$, where c is the capacity. However this bound is not known to be tight and the complexity of the best known generic attack, which is based on multicollisions, is much larger: it reaches $\frac{2^c}{\alpha}$ where α represents a small security loss factor. There is thus an uncertainty on the true extent of security beyond the bound $2^{\frac{c}{2}}$ provided by such constructions. In this paper, we describe a new generic attack against several duplex-based AEAD modes. Our attack leverages random functions statistics and produces a forgery in time complexity $\mathcal{O}(2^{\frac{3c}{4}})$ using negligible memory and no encryption queries. Furthermore, for some duplex-based modes, our attack recovers the secret key with a negligible amount of additional computations. Most notably, our attack breaks a security claim made by the designers of the NIST lightweight competition candidate Xoodyak. This attack is a step further towards determining the exact security provided by duplex-based constructions.

Keywords: Cryptanalysis · Symmetric cryptography · AEAD · Duplex-based constructions · NIST lightweight competition · Xoodyak · Random functions

1 Introduction

Authenticated Encryption (AE), which allows to encrypt and authenticate a plaintext message in a combined way, is one of the main workhorses of symmetric cryptography. AE often offers the option to authenticate, in addition to the plaintext message, some extra data which, unlike the plaintext, are transmitted unencrypted. AE is then renamed Authenticated Encryption with Associated Data (AEAD). A considerable research effort was devoted during the last years to the design and analysis of efficient and secure AEAD algorithms. Examples of AEAD mechanisms largely deployed over the Internet are AES-GCM and Chacha20-Poly1305. In 2014-2019, the three-round process of the CAESAR competition for authenticated encryption resulted in the selection of a portfolio of six AEAD mechanisms chosen to address the needs of the three following use cases: lightweight applications, high-performance applications and provision

of defense-in-depth features, e.g. nonce misuse resistance. Another symmetric algorithms selection initiative, the ongoing NIST lightweight cryptography standardization process, aims at selecting (families of) algorithms suitable for use in constrained environment that comprise at least an AEAD mechanism and optionally a cryptographic hashing mechanism. The process was launched in 2018 and has now reached its third round, where ten finalists are still being evaluated.

Most existing AEAD algorithms are either *block cipher-based* or *permutation-based*. In the first case, they result from the application of a suitable mode of operation to a block cipher or a tweakable block cipher. In the second case, they consist in the instantiation of a keyed mode of operation with a public permutation. In this paper, we focus on so-called *duplex-based* keyed modes. Seminal examples of such modes are SpongeWrap and MonkeyDuplex, both introduced by Bertoni, Daemen, Peeters and Van Assche [7,10]. Duplex-based modes can be viewed as an adaptation to the AEAD context of the *Sponge construction* for hash functions introduced by the same authors in [8,6]. After an initialization phase allowing to derive a b-bit state from a key and an IV value, they essentially iterate calls to a key-less public permutation P of the state space $\{0,1\}^b$ in alternance with an injection and/or extraction of data blocks on/from a dedicated r-bit part ($r < b$) of the current state value in order to absorb additional data, encrypt and absorb the plaintext, and produce an authentication tag.[1] The sizes r and $c = b - r$ bits of the state parts affected (resp. unaffected) by data injections or extractions are named the *rate* and the *capacity*. The corresponding state parts are referred to as the *outer* and the *inner* state.

The security of duplex-based constructions has been extensively studied during the last decade. Let us denote the total time complexity of an attack by $\mathcal{T} = \sigma_e + \sigma_d + q_P + t_{extra-op}$, where σ_e and σ_d respectively represent the number of online calls to P caused by the adversary's encryption requests and forgery attempts and q_P represents the number of offline queries to P or its inverse. The last term $t_{extra-op}$ represents the extra computations not taken into account in $\sigma_e + \sigma_d + q_P$, e.g. computations of primitives involved in the initialization and finalization, basic read/write operations in memory, random samplings. It is measured as an equivalent number of P computations.[2] Let us further denote by q_d the number of forgery attempts of the adversary. The initial security arguments for duplex constructions, which leveraged the indifferentiability of the sponge construction for hash functions, only allowed to guarantee the security of the duplex constructions as long as $\mathcal{T} \ll min\{2^{\frac{c}{2}}, 2^{\kappa}\}$ and $q_d \ll 2^{\tau}$, where κ and τ represent the key length and the tag length. In [27,28], Jovanovic *et al.*

[1] The name "duplex", that conveys the idea of a bidirectional process, reflects the fact that in such constructions both a data block injection and a data block extraction to/from the current state can potentially take place between two consecutive invocations of P.

[2] Note that in the attack considered in the sequel, $t_{extra-op}$ will in practice be negligible compared to $\sigma_e + \sigma_d + q_P$.

showed that in the nonce-respecting setting, the security of a series of duplex constructions can be ensured beyond the birthday bound $2^{\frac{c}{2}}$, namely as long as

$$\mathcal{T} \ll min\{2^{\frac{b}{2}}, \frac{2^c}{\alpha}, 2^\kappa\} \text{ and } q_d \ll 2^\tau, \tag{1}$$

where α represents a small constant upper bounded by r in [27] and a tighter, substantially smaller constant in [28]. The detail of the security proofs sections of [27,28] indicates however that the bounds (1) are only valid under the assumption that σ_d is strongly limited: if one wants to avoid implicit assumptions about σ_d, the bounding conditions (1) must be replaced by the more complete (though still simplified) conditions:

$$\mathcal{T} \ll min\{2^{\frac{b}{2}}, \frac{2^c}{\alpha}, \frac{2^c}{\sigma_d}, 2^\kappa\} \text{ and } q_d \ll 2^\tau. \tag{2}$$

Unlike the conditions (1), the complete bounding conditions (2) can hardly be considered as 'beyond-birthday bounds'. For $\sigma_d \geq 2^{\frac{c}{2}}$, the condition $\mathcal{T} \ll \frac{2^c}{\sigma_d}$ of (2) indeed implies $\mathcal{T} \ll 2^{\frac{c}{2}}$. Thus, the security of the considered duplex-based constructions without assumptions on σ_d can only be guaranteed as long as the birthday condition $\mathcal{T} \ll 2^{\frac{c}{2}}$ is met.

The bounding conditions (1) were used for dimensioning the family of duplex AEADs `NORX`, and led since 2014 to several other AEAD design proposals based on duplex-like constructions with a claimed security level strictly larger than $\frac{c}{2}$. Similar bounds were shown to still hold when instead of limiting the size of injected and extracted data blocks to r bits, one only limits the size of extracted data blocks to r bits but full-state data injections are permitted [30]. This leads to AEAD proposals where the efficiency of the associated data absorption phase is increased. A generalisation of the full-state keyed duplex of [30] with multi-user support, accompanied by a refined security analysis, was published in [20].

Generic attacks on duplex-based constructions. The best currently known generic attacks on duplex-based constructions are based on multicollisions. These attacks match the security bound $\frac{2^c}{\alpha}$ where α represents a small security loss factor as stated before. They are presented in detail in [28] and can be roughly outlined as follows. First, the adversary submits sufficiently many calls to an encryption oracle in order to find a multicollision among the outer state values of a number ρ of states. Once this has been achieved, an exhaustive search for a c-bit value matching the inner state value of one of these ρ states is likely to succeed after about $\frac{2^c}{\rho}$ trials.

Our contribution. The end of the former discussion on the security of duplex-based constructions showed that in situations where the possibility of forgery attempts of non-negligible data complexity σ_d is not precluded, the complete bounding conditions (2) only prevent the existence of generic attacks of total complexity $\mathcal{T} \ll 2^{\frac{c}{2}}$. It is therefore an open question whether there exists a

generic attack of total complexity $\mathcal{T}$ strictly comprised between $2^{\frac{c}{2}}$ and the complexity $\frac{2^c}{\alpha}$ of multicollision-based attacks.[3] In this paper, we provide a positive answer to the former open question by exhibiting a generic attack against a large family of duplex-based constructions of total time complexity $\mathcal{T}$ in $\mathcal{O}(2^{\frac{3c}{4}})$. We rely on the analysis of random function statistics to design a two-phase forgery attack with a precomputation phase of time complexity $\mathcal{O}(2^{\frac{3c}{4}})$ essentially equal to q_P and an actual forgery phase of time complexity $\mathcal{O}(2^{\frac{3c}{4}})$ essentially equal to σ_d. The value σ_d can also be viewed as the data complexity of the attack measured in ciphertext blocks. The attack requires no encryption queries, *i.e.* $\sigma_e = 0$, and $t_{extra-op}$ is negligible compared to q_P and σ_d. For some of these constructions, our attack recovers the secret key.

Our attack takes advantage of the following property shared by several authenticated encryption modes based on the duplex construction: for a ciphertext built by concatenating a fixed block multiple times, the decryption of a message consists in the iteration of a public function with domain and co-domain $\mathbb{F}_2^c$. This public function is fully determined by the value of the ciphertext block. We are therefore able to precompute some of its parameters offline. Forgery strategies that just assume a "near to average" behaviour of the iterated function, e.g a cycle length of the path generated by a random point of $\{0,1\}^c$ close to the expected value $\sqrt{\pi 2^c/8}$, do not seem to lead to forgery attack complexities better than $\mathcal{O}(2^c)$. We show that however, the following two-phase attack strategy allows an adversary to produce an existential forgery with a success probability close to 1 and a total (online and offline) computation time $\mathcal{T} = \mathcal{O}(2^{\frac{3c}{4}})$.

1) *In an offline phase*, a significant amount $\mathcal{O}(2^{\frac{3c}{4}})$ of precomputation is dedicated to the detection of a c-bit to c-bit function that possesses exceptional characteristics that exponentially deviate from an average behaviour, thus rendering its use in a forgery attack more efficient.

2) *In an online phase*, the iteration of the function identified in the offline phase is used to produce forgery attempts whose success probability is exceptionally high.

In slightly more detail, the offline phase aims at selecting a ciphertext block that determines a function whose graph possesses a large component in which all paths are terminated by the same exceptionally small cycle, of length at most a small predefined multiple of $2^{\frac{c}{4}}$. This is shown to imply that for long ciphertexts obtained by repeating the selected ciphertext block, the tag is the image of one of the values of the former small cycle by a known function with a probability close to 1. This in turn allows to mount a forgery attack of offline, online and total time complexity $\mathcal{O}(2^{\frac{3c}{4}})$.

The precomputation phase needs to be run only once to break the same construction with as many different keys as desired. Further, we can adjust the trade-off between the precomputation phase and the online phase in order to

[3] In other words, the questions whether the "birthday term" $\frac{2^c}{\sigma_d}$ is an artifact of the proofs and whether the capacity value c can be safely dimensioned well below $2s$, where s denotes the targeted security level remain open.

bring the complexity of the latter closer to $\mathcal{O}(2^{\frac{c}{2}})$ at the expense of significantly increasing the complexity of the former.

Previously published generic attacks against hash-based MACs or hash functions also rely on the statistics of random functions [32,29,33,4]. Such attacks also model, as the AEAD attack introduced here, one function that the considered construction allows to iterate as a random function. Yet, while previous attacks generally assume and exploit a "near to average" behaviour of a function selected at random, an essential feature of the attack presented here is that it selects and leverages instances of a function whose behaviour exceptionally deviates from average.

Our attack is applicable to several duplex-based modes such as monkeyWrap, monkeyDuplex or Motorist. Most notably, our attack is applicable to Cyclist, the mode of the Lightweight Cryptography NIST competition finalist Xoodyak [18]. With a key recovery attack of complexity 2^{148} applications of the state update function Xoodoo, we break the claim of achieving 184-bit security against plaintext recovery and forgery attacks made by the designers in [18, Corollary 2, p. 72]. Note that this does not threaten the 112-bit security level required by the NIST. Our results are detailed in Section 5 and displayed in Table 5.

Organization. The rest of this paper is organized as follows. Section 2 introduces definitions and results related to the statistics of random functions that are relevant for our attack and defines a 'vanilla' duplex-based AEAD mode that only captures those features of duplex-based constructions that are essential to understand our attack. Section 3 presents our attack and analyses its performance using the vanilla mode of Section 2 as the target in order to simplify its presentation. Section 4 presents experimental validations of essential features of the attack based on small scale implementations. Section 5 shows that the attack of Section 3 is applicable to several real-life duplex modes with minor adaptations. Section 5 also discusses variations encountered in AEADs such as Beetle [16] or Ascon [22] that, on the other hand, prevent the attack.

2 Preliminaries

In this section, we start by introducing key definitions and results related to the statistics of random functions. Next, we describe a simplified duplex-based authenticated encryption mode on which we will rely to describe our attack. This is followed by a short subsection on the security model on which we rely.

2.1 Preliminaries on random functions

Let $\mathfrak{F}_n$ be the set of all functions which map a finite set of size $n \in \mathbb{N}^*$ to itself. Without loss of generality, we consider the set $\{1, \ldots, n\}$. Each function f in $\mathfrak{F}_n$ determines a directed graph $G(f)$ in which a vertex goes from node i to node j, $i, j \in \{1, \ldots, n\}$, if and only if $f(i) = j$ [31,24]. In the following, for simplicity reasons, we say that a node belongs to the graph of a function when it belongs

to the set of nodes of this graph. We use the term "random function in $\mathfrak{F}_n$" to refer to a function selected uniformly at random in the set $\mathfrak{F}_n$.

For any f in $\mathfrak{F}_\mathfrak{n}$ and any node x in $G(f)$, we can iterate f and consider the set of *successors* of x

$$\mathcal{S}(x) = \{f^i(x) \mid 0 \leq i \leq n-1\}.$$

We denote by $s(x)$ the size of this set. Since the graph has a finite number of nodes, the sequence $\{f^i(x)\}_{i\geq 0}$ is eventually periodic. Graphically, it thus corresponds to a path linked to a *cycle* defined as

$$\mathcal{C}(x) = \{f^i(x) \mid \exists j > 0, f^i(x) = f^{i+j}(x)\}.$$

We denote by $\mu(x) = \#\mathcal{C}(x)$ the length of this cycle or *cycle length* and by $\lambda(x) = s(x) - \#\mathcal{C}(x)$ the length of this path or *tail length*. The tail length $\lambda(x)$ is the smallest integer i such that $f^i(x) \in \mathcal{C}(x)$. The set of all nodes $y \in G(f)$ such that $\mathcal{C}(y) = \mathcal{C}(x)$ forms a *connected component*. Since all nodes in the same connected component have the same cycle, the *cycle of a component* is well-defined.

Our cryptanalysis relies on the attacker's ability to find functions which have a large component with a small cycle. We formally characterize what should be understood by "large component with a small cycle" later on. To do so, we will need the two following definitions.

Definition 1 (ν-component). *Let $0 < \nu < \frac{1}{2}$. A ν-component is a component that has a cycle of size at most $n^{\frac{1}{2}-\nu}$.*

Definition 2 ((s,ν)-component). *Let $0 < \nu < \frac{1}{2}$, $0 < s < 1$. A (s,ν)-component is a ν-component whose size is greater or equal to ns.*

In order to estimate the complexity and success probability of our attack, we rely on the statistical analysis of random functions. Such an analysis has been extensively conducted in combinatorics [25,31,21,23,24]. In this paper, we will need the three following results.

Expectancy of cycle length and tail length for a random point [23]. For a random node x in the graph of a random function $f \in \mathfrak{F}_n$, Flajolet and Odlyzko have computed the asymptotic form of the expectancy of the cycle length $\mu(x)$ and tail length $\lambda(x)$ using generating functions. They obtained an expectancy of $\sqrt{\frac{\pi n}{8}}$ for both.

Probability for a random point to belong to a ν-component [25]. For a random node x belonging to the graph of a random function $f \in \mathfrak{F}_n$, Harris shows that the probability that $\mu(x)$ is smaller than $n^{\frac{1}{2}-\nu}$ is asymptotically

$$\begin{aligned} p_\nu &= 1 - e^{-\frac{1}{2n^{2\nu}}} + \frac{\sqrt{2\pi}}{n^\nu}\left[1 - \phi(n^{-\nu})\right] \\ &= \frac{\sqrt{2\pi}}{2n^\nu} + \mathcal{O}\left(\frac{1}{n^{2\nu}}\right) \end{aligned}$$

where $\phi(y) = \int_{-\infty}^{y} (2\pi)^{-\frac{1}{2}} e^{-\frac{1}{2}x^2} dx$. This corresponds to the probability for a random node x of the graph of a random function to belong to a ν-component. For example, for $\nu = \frac{1}{4}$ and c a positive integer such that $n = 2^c$, $p_\nu = \frac{\sqrt{2\pi}}{2} \times 2^{-\frac{c}{4}}$.

Probability for a random function to have a (s, ν)-component [21]. For a random $f \in \mathfrak{F}_n$, the probability $p_{s,\nu}$ that $G(f)$ has a (s, ν)-component has been estimated by DeLaurentis in a paper published at Crypto 1987 [21]. It is shown to be

$$p_{s,\nu} = \sqrt{\frac{2(1-s)}{\pi s}} n^{-\nu} \left[1 + \mathcal{O}\left(r_n(s)\right)\right]$$

where $r_n(s) = s^{-2} n^{-\frac{1}{2}-3\nu} + s^{\frac{-1}{2}} n^{-\nu} + n^{-\frac{1}{3}}$. Thus, $p_{s,\nu} \approx \sqrt{\frac{2(1-s)}{\pi s}} n^{-\nu}$. For example, if we take $\nu = \frac{1}{4}$, $s = 0.65$ as in Section 3.6, and c a positive integer such that $n = 2^c$, $p_{s,\nu} \simeq 0.6 \times 2^{-\frac{c}{4}}$.

Probability for a random point to belong to its component's cycle after $l-1$ applications of f. Let x be a random point of a random function. Harris [25] gives the asymptotic density function of the number of successors $s(x)$. More precisely, he provides the asymptotic density function f_1 of $\frac{s}{\sqrt{n}}$ for $x > 0$

$$f_1(x) = x e^{-\frac{x^2}{2}} .$$

For l a positive integer, let p_l be the probability that $f^{l-1}(x)$ is in the cycle, that is, the probability that $\lambda(x) \leq l - 1$. Since the number of successors is greater than the tail length, p_l is greater or equal to the probability for x to have strictly less than l successors. Thus, we have

$$p_l \geq 1 - e^{-\frac{l^2}{2n}} .$$

Notational conventions. For simplicity and readability reasons, when it comes to estimates resulting from statistics on random functions, we will use the sign "$\leq$" where it would be more rigorous to use the smaller or equivalent sign "$\lessapprox$" in the rest of the paper.

2.2 Description of a vanilla duplex-based AEAD mode

The duplex construction was designed by the KECCAK team as a tool to build authenticated encryption modes [7,9]. The first proposal of such a mode is SPONGEWRAP [7], published in 2011. Today, many modes are based on this construction. In this paper, we define the simplified authenticated encryption mode DUPLEXAEAD. DUPLEXAEAD shares its structure with the modes of several duplex-based AEAD schemes. This mode is not meant to be used in

practice. It is defined for readability reasons: its sole purpose is to make the description of our attack simpler. We show in Section 5 how our attack can be adapted to several real-life duplex-based modes.

DuplexAEAD is instantiated with a permutation P which operates on a b-bit state S divided into two parts. The first r bits of the state form the *outer state* $\overline{S}$, whilst the next $c = b - r$ bits form the *inner state* $\widehat{S}$. The state can thus be written as $S = \overline{S}||\widehat{S}$. As stated in Section 1, the parameter r is called the *rate* and the parameter c is called the *capacity*. DuplexAEAD also involves two other public functions, namely, an initialisation function P_{init} and a finalisation function P_{final}. The encryption algorithm $\mathcal{E}$ takes as input a κ-bit key K, a η-bit nonce N, a plaintext M and associated data A of variable length and returns a ciphertext C and a τ-bit tag T. The decryption algorithm $\mathcal{D}$ takes as input (K, N, A, C, T) and returns the plaintext M if the tag is valid. Otherwise, it returns $\perp$.

We assume for simplicity reasons that the length in bits of the plaintexts processed by DuplexAEAD is always divisible by r.[4] Thus, any plaintext M can be split into r-bit blocks, $M = M_0||\cdots||M_{l-1}$, where l is the plaintext length in number of r-bit blocks. The ciphertext's length is equal to the plaintext's length, and can thus also be written $C = C_0||\cdots||C_{l-1}$.

The mode works as follows (see Figure 1):

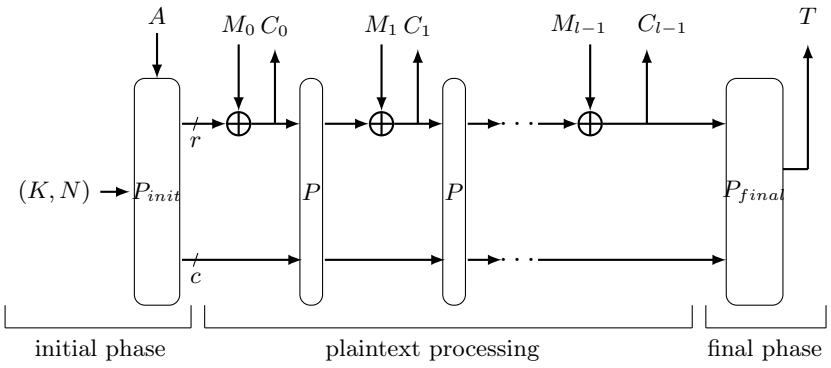

Fig. 1. DuplexAEAD in encryption mode.

[4] In practice, our cryptanalysis can be easily adapted to attack modes which can process plaintexts of arbitrary length, but this requires a short case-by-case analysis which we provide in Section 5. Note that the resulting adjustments have a negligible impact on the complexity of our attack and do not impact its success probability.

Initial phase. The encryption and decryption algorithms start by an initial phase. The key, the nonce and the associated data are processed by the initialisation function P_{init}. The state is set to the output of P_{init}.

Plaintext processing. In encryption mode, DuplexAEAD then processes the plaintext and generates the ciphertext block by block as follows:

1. The ciphertext block C_i is generated by XORing the outer state to the plaintext block M_i. That is, $C_i \leftarrow \overline{S} \oplus M_i$.
2. The outer state is set to the just computed ciphertext block. That is, $\overline{S} \leftarrow C_i$.
3. The permutation P is applied to the state. That is, $S \leftarrow P(S)$.

Ciphertext processing. In decryption mode, DuplexAEAD then processes the ciphertext. The plaintext M is constructed as the ciphertext is processed but will only be outputted at the end of the final phase if the tag is valid. This phase works as follows:

1. A plaintext block M_i is generated by XORing the outer state to the ciphertext block C_i. That is, $M_i \leftarrow \overline{S} \oplus C_i$.
2. The outer state is set to the value of the ciphertext block. That is, $\overline{S} \leftarrow C_i$.
3. The permutation P is applied to the state. That is, $S \leftarrow P(S)$.

Final phase. During encryption (resp. decryption), plaintext (resp. ciphertext) processing is followed by a final phase in which the finalisation function P_{final} takes as input the state and computes a τ-bit tag. The encryption algorithm returns a ciphertext and the corresponding tag. The decryption algorithm checks whether the tag is valid. If so, it returns the plaintext M. Otherwise, it returns $\perp$.

Domain separation. Actual duplex-based AE modes generally rely on domain separation for their security. In the case of DuplexAEAD, we assume that the way P_{init} and P_{final} are constructed from P ensures domain separation between the processing of (K, N), the processing of A, the processing of M and the computation of T. This can be done for example by XORing distinct constants to the inner state before P invocations in each phase.

2.3 Security model

For authenticated encryption, two security notions are involved, namely privacy and integrity. Since our attack aims at breaking integrity, we leave privacy aside. We do not fully formalise the security model but give some *simplified* reminders to the reader.[5] In the integrity setting, we have an adversary $\mathcal{A}$ who has access to the following oracles:

[5] For more details see for example [27].

- A primitive oracle $\mathcal{O}_P$ that allows to call the public permutation P or its inverse P^{-1}. It takes as input a value $v \in \mathbb{F}_2^b$ and outputs $P(v) \in \mathbb{F}_2^b$ (resp. $P^{-1}(v) \in \mathbb{F}_2^b$) for a call made to P (resp. P^{-1}).
- An encryption oracle $\mathcal{O}_{enc}$ that takes as inputs a nonce N, associated data A and a plaintext M and returns a ciphertext C and a tag T computed with a secret key K which is randomly sampled once for all (the same key is used by the decryption oracle).[6] It implements the encryption algorithm of the analysed AEAD scheme based on P.
- A decryption oracle $\mathcal{O}_{dec}$ that takes as input a nonce N, associated data A, a ciphertext C and a tag T and, using the key K, returns the corresponding plaintext if the verification is correct, $\perp$ otherwise. Similarly to the previous oracle, it implements the decryption algorithm of the analysed AEAD scheme.

We assume that the adversary $\mathcal{A}$ is **nonce-respecting**. Her goal is to provide a forgery, that is, an input (N, A, C, T) such that $\mathcal{O}_{dec}(N, A, C, T) \neq \perp$ where (C, T) was not outputted by the encryption oracle $\mathcal{O}_{enc}$ on an input $(N, A, \cdot)$ (for any plaintext). The probability to provide a forgery has to be negligible.

As already mentioned in Section 1, we denote by q_e, q_d and q_P the number of queries done to respectively the encryption oracle, the decryption oracle and the primitive oracle. We denote by σ_e the total number of plaintext blocks processed by the encryption oracle and by σ_d the total number of ciphertext blocks processed by the decryption oracle.

In this paper, we construct a generic forgery attack against several duplex-based authenticated encryption modes. Our attack is generic in the sense that we do not exploit the properties of the permutation P but only properties of the mode itself [9]. It does not rely on nonce misuse or the release of unverified plaintext. For some modes, our attack also recovers the secret key with a negligible amount of extra computation.

3 Description of the attack

In this section, we present a generic forgery attack against duplex-based authenticated encryption modes. For the sake of clarity, we first describe how the attack works on the simplified mode DuplexAEAD defined in Section 2.2. We show how to apply our attack to other authenticated encryption modes in Section 5.

3.1 Observation on duplex-based AEAD modes

We describe a simple property of DuplexAEAD that is shared with many other duplex-based AE modes: for a ciphertext built by concatenating a fixed block multiple times, the decryption of a plaintext consists in the iteration of a known function with domain and co-domain $\mathbb{F}_2^c$. This property is at the core of our attack. It is depicted in Figure 2.

[6] Each parameter space is well defined according to the analysed AEAD scheme.

Let $\ell \in \mathbb{N}^*$ and $\beta \in \mathbb{F}_2^r$. Let β_ℓ be the ciphertext equal to the concatenation of ℓ r-bit blocks of constant value β, that is

$$\beta_\ell = \underbrace{\beta||\cdots||\beta}_{\ell} .$$

During the ciphertext decryption, the value of the outer state at the input of the state update function P is equal to the current ciphertext block. Thus, the decryption of β_ℓ corresponds to the iteration of the function P_β defined as

$$\begin{aligned} P_\beta : \mathbb{F}_2^c &\longrightarrow \quad \mathbb{F}_2^c \\ x &\longmapsto \widehat{P(\beta||x)} . \end{aligned}$$

Indeed, let $x_0 \in \mathbb{F}_2^c$ be the value of the inner state obtained at the end of the initial phase from the key, the nonce and the associated data. After processing the first plaintext block, the value of the outer state is equal to the first ciphertext block β. Thus, the input of the state update function is exactly $\beta||x_0$. As a consequence, the value of the inner state going into the second application of P is $\beta||x_1$ where $x_1 = P_\beta(x_0)$. In turn, the third one is $\beta||x_2$ where $x_2 = P_\beta(x_1) = P_\beta^2(x_0)$. When the last plaintext block is constructed, right before the final phase, the state is thus of the form

$$\beta||P_\beta^{\ell-1}(x_0) = \beta||x_{\ell-1} .$$

Since the outer part of the state is equal to β, it is known to the attacker. In particular, to recover the value of the state before the final phase, an attacker only needs to determine the value of $x_{\ell-1}$. Since $T = P_{final}\,(\beta||x_{\ell-1})$, it is sufficient for the attacker to recover the value of $x_{\ell-1}$ to find a forgery (N, A, C, T). As we will show more rigorously later on, our cryptanalysis relies on the fact that without knowing x_0, an attacker is able to select β and ℓ such that she is able to both restrict and predict the space of all possible $x_{\ell-1} = P_\beta^{\ell-1}(x_0)$ with good probability.

3.2 High level description of the attack

Our attack aims at recovering the value of $x_{\ell-1}$. It was devised relying on the random functions statistics introduced in Section 2.1. Indeed, let $n = 2^c$. P is a random permutation on $\mathbb{F}_2^b$ and $c = b - r$ is significantly smaller than b. Thus, for any β randomly drawn from $\mathbb{F}_2^r$, we expect P_β to behave as a function randomly drawn from $\mathfrak{F}_n$.

The attack consists of two phases.

Precomputation phase. First, in a precomputation phase, an offline algorithm finds a value β such that $G(P_\beta)$ has a large component with a small cycle, that is, a (s, ν)-component with great s and ν.

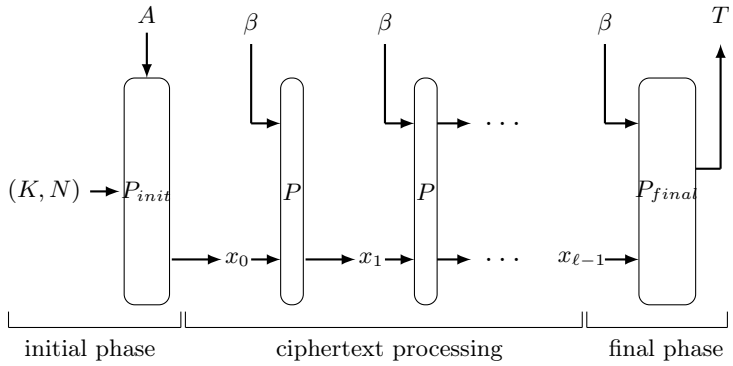

Fig. 2. Decrypting β_ℓ.

Online phase. Second, in an online phase, (N, A, C, T) queries are submitted to a decryption oracle, where the ciphertext is $C = \beta_\ell$ with ℓ sufficiently large and β is the output of the precomputation algorithm. Recall that β is chosen such that $G(P_\beta)$ has a large component with a small cycle. Since this component is large, it contains the unknown node x_0 with good probability. In that case, for a great enough value of ℓ, $x_{\ell-1}$ belongs to the cycle of this component, which is small. Thus, the number of all possible values for $x_{\ell-1}$ is reduced and can be efficiently exhausted by submitting (N, A, β_ℓ, T) queries to the decryption oracle where

1. tags are produced by applying P_{final} to a state such that $\overline{S} = \beta$ and such that the inner state $\widehat{S}$ belongs to the small cycle of P_β's large component;
2. nonces are either randomly sampled or arbitrary distinct values;[7]
3. the associated data is set to the empty string ε.[8]

Our final attack, which balances the computational cost of the offline and online phases, provides a forgery in time $\mathcal{O}(2^{\frac{3c}{4}})$ with a negligible amount of memory. In the rest of this section, we provide a detailed description together with an analysis of the complexity and success probability of our full attack.

In our complexity and success probability analysis, significant efforts were put towards limiting the use of heuristic assumptions. This is illustrated for example by the use of the probability $p_{s,\nu}$ [21] that a random function has a (s, ν)-component rather than a heuristic estimation of this value from the probability

[7] In decryption queries, nonces can be repeated even in a nonce-respecting setting. For the purpose of our attack, x_0 needs to behave as a point randomly sampled in the graph of P_β. Thus, we can either require nonces to be randomly sampled or to be arbitrary distinct values.

[8] For modes other than the full-state duplex, we could set the associated data to any value. In the case of the full-state duplex, the associated data must be chosen carefully (typically, the attack applies when the AD is set to the empty string).

p_ν that it has a ν-component. Although we did not manage to get entirely rid of all heuristic assumptions, we intended to back each assumption with heuristic reasoning and/or small scale experiments.

3.3 Precomputation phase

In this section, we show how to construct an offline algorithm which outputs β such that $G(P_\beta)$ has a large component with a small cycle. By 'having a large component with a small cycle', we mean having a (s, ν)-component as defined in Section 2.1. We will set the parameters s and ν to their final values later in the paper. The offline algorithm also outputs the length μ of the (s, ν)-component's cycle, as well as an element e of this cycle.

Our algorithm samples random values β from $\mathbb{F}_2^r$ and random values x in $\mathbb{F}_2^c$ and investigates whether or not the component on which x is located is a (s, ν)-component of $G(P_\beta)$.

When a random β and a random x are generated, we start by investigating the cycle length of the component on which x is located, that is, whether or not it is a ν-component. Cycle-finding algorithms such as Floyd's or Brent's algorithm provide a straightforward way to determine the component's cycle length. These algorithms are typically used to find collisions on functions with a negligible amount of memory. In fact, the collisions found by these algorithms are located within the functions' cycles. Thus, given a random $\beta \in \mathbb{F}_2^r$, which selects a random $P_\beta \in \mathfrak{F}_n$, and a random $x \in \mathbb{F}_2^c$, cycle-finding algorithms can be used to construct an algorithm which outputs both $\mu(x)$, the cycle length of the connected component on which x is located, and the value of a node in the cycle [26]. We call such an algorithm `cycle`. The algorithm `cycle` allows the attacker to determine whether or not the component on which x is located is a ν-component, *i.e.* a component with a cycle of the desired length $\mu(x) \leq n^{\frac{1}{2}-\nu}$.

To construct our final precomputation algorithm, two main issues remain to be solved.

Issue 1. If the random β and the random x investigated are such that the component on which x is located is not a (s, ν)-component, it does not necessarily mean that $G(P_\beta)$ does not have a (s, ν)-component. There could be an $x' \neq x$ such that x' belongs to a different component that has the desired cycle length and size. In particular, we must determine whether or not it is worth trying a different $x' \neq x$ when `cycle` returns a cycle length value that is greater than $n^{\frac{1}{2}-\nu}$ (x's component is not a ν-component). There are two imaginable strategies. The first strategy consists in trying a single random x for each β, and, whenever x has a component with a cycle size smaller than $n^{\frac{1}{2}-\nu}$, investigate whether or not the component has size greater or equal than sn. The second strategy would be to try several x's for each β. We stick to the first strategy. The following argument indeed suggests that the second strategy would be more costly. When conditioned by the failure of the first drawn x, the probability that a second x' succeeds with the same β is at most $\frac{p_{s,\nu}(1-s)}{1-p_{s,\nu}}$ whilst for a new random (β', x'), the

probability of success is at least $p_{s,\nu}s$. Thus, the first strategy is better whenever $s > \frac{1}{2-p_{s,\nu}}$ which is always true for our values of s (which are always greater than 0.5) and ν.

Issue 2. Although `cycle` allows to detect ν-components, it does not provide a way to detect (s,ν)-components. Being fully certain that a ν-component has the desired size would be prohibitingly costly. A strategy allowing to estimate that a ν-component is a (s,ν)-component with a sufficiently large probability has to be devised.[9] The algorithm `is_big` implements this strategy. Note that in this algorithm, the notation '$(.,z)$' in Step 4 means that the first output (corresponding to the cycle length) is ignored. Each time the algorithm detects a small cycle, it checks whether or not ω other random nodes belong to the same component. It then computes the proportion s_{obs} of these new randomly chosen points which belong to the desired component. If this number is above a threshold value $s+\delta$ strictly larger than s, it decides that x is likely to belong to a (s,ν)-component and returns β and the values of the cycle. We show that for $\delta = \frac{2.33}{2\sqrt{\omega}}$, a selected component has size greater than s with great probability. Suppose that we have drawn a value β such that β has a ν-component of unknown size $s_\beta n$. In that case, the random variable s_{obs} is the mean of ω Bernoulli variables which are equal to 1 with probability s_β and 0 with probability $1-s_\beta$. By the Central Limit Theorem (CLT), we have that for all $\epsilon \in \mathbb{R}$,

$$p\left[s_{obs} - s_\beta \leq \epsilon\sqrt{\frac{s_\beta(1-s_\beta)}{\omega}}\right] = p\left(Y \leq \epsilon\right)$$

where $Y \rightsquigarrow \mathcal{N}(0,1)$. Thus, since $\delta = 2.33\sqrt{\frac{1}{4\omega}}$ and since for $s_\beta < 1$, $s_\beta(1-s_\beta) \leq \frac{1}{4}$, we have that

$$s_{obs} - s_\beta \leq \delta$$

with probability $p_\delta \geq p\big(Y \leq 2.33\big) > 0.99$. Suppose that for a random β with a ν-component, we obtain a proportion s_{obs} of random x's in this component such that $s_{obs} \geq s+\delta$ as required in Step 10 of the algorithm `is_big`. We know that with probability $p_\delta \geq 0.99$:

$$s_\beta \geq s_{obs} - \delta \geq s + \delta - \delta = s\,.$$

As a conclusion, when our final offline algorithm `offline_search` returns a β, then it is such that $G(P_\beta)$ has a ν-component with probability 1, and the component considered has at least the desired size with probability $p_\delta \geq 0.99$.

[9] Given a point of the graph x, the introduction of an algorithm which estimates x's component's size is going to significantly complicate our attack's complexity analysis. We believe that we could have designed an attack without this extra algorithm by relying on the fact that a random component is large with great probability. However, unless increasing the complexity of the online phase, we would have then obtained lower success probabilities (roughly around 0.5 rather than close to 1).

Algo. `offline_search`(ω, s, ν)

1 : **while true do**
2 : $\quad \beta \xleftarrow{\$} \mathbb{F}_2^r;\ x \xleftarrow{\$} \mathbb{F}_2^c$
3 : $\quad (\mu, e) \leftarrow \texttt{cycle}(\beta, x)$
4 : $\quad$ **if** $\mu \leq n^{\frac{1}{2}-\nu}$ **then**
5 : $\quad\quad$ **if** `is_big`$(\omega, s, \beta, \mu, e)$
6 : $\quad\quad\quad$ **return** (β, μ, e)

Algo. `is_big`$(\omega, s, \beta, \mu, e)$

1 : $\delta \leftarrow \dfrac{2.33}{2\sqrt{\omega}}; j \leftarrow 0$
2 : **for** $i = 1..\omega$ **do**
3 : $\quad k \leftarrow 0;\ inside\_big \leftarrow$ **false**; $\ y \xleftarrow{\$} \mathbb{F}_2^c$
4 : $\quad (., z) \leftarrow \texttt{cycle}(\beta, y)$
5 : $\quad$ **while** $inside\_big =$ **false and** $\ k < \mu$ **do**
6 : $\quad\quad$ **if** $z = e$ **then** $inside\_big \leftarrow$ **true**
7 : $\quad\quad z \leftarrow P_\beta(z)$
8 : $\quad\quad k \leftarrow k + 1$
9 : $\quad$ **if** $inside\_big$ **then** $j \leftarrow j + 1$
10 : **if** $\dfrac{j}{\omega} \geq s + \delta$ **return true**
11 : **return false**

3.4 Analysis of the offline algorithm

In this section, we analyse the success probability and complexity of the offline algorithm. As stated in Section 2.3, the complexity is expressed as a number of calls to P. This algorithm is at the core of our attack. It only needs to be executed once. Indeed, once it has succeeded in finding β such that $G(P_\beta)$ has a (s, ν)-component, a duplex-based mode with any key using the permutation P can be attacked.

Complexity of the cycle-finding algorithm. We start by investigating the average complexity of the algorithm `cycle`. To construct this algorithm, we use Brent's cycle-detection algorithm as a tool [14]. For a random $\beta \in \mathbb{F}_2^r$ and a random $x \in \mathbb{F}_2^c$, Brent's algorithm recovers the cycle length $\mu(x)$ and a node of the cycle $e \in \mathcal{C}(x)$ after at most $2\max(\mu(x), \lambda(x)) + \mu(x)$ applications of P_β. For a random x, the cycle length $\mu(x)$ and the tail length $\lambda(x)$ have the same expectation of $\sqrt{\frac{\pi n}{8}}$ [24] (See Section 2.1). Thus, we estimate the average complexity of `cycle` in number of calls to P to be upper bounded by

$$\mathcal{T}_{\texttt{cycle}} = 3\sqrt{\frac{\pi n}{8}}\,.$$

Complexity of the offline algorithm. We wish to compute the complexity of the offline algorithm `offline_search`. First, the memory complexity is negligible. As for the time complexity, we compute the average number of applications of P after which the algorithm returns a block $\beta \in \mathbb{F}_2^r$.

First, we upper bound the average complexity of `is_big`. This algorithm is executed when a pair (β, x) such that x is located on a ν-component of $G(P_\beta)$

has been selected. It takes as input β, $\mu = \mu(x) \leq n^{\frac{1}{2}-\nu}$, an element e of the cycle of x's component, s and an integer ω. It performs ω times a computation that essentially consists in the generation of a random point of $G(P_\beta)$, one computation of the cycle algorithm, and at most μ point comparisons and $\mu - 1$ P_β invocations. The average complexity of `is_big` is thus upper bounded by

$$\left(\mathcal{T}_{\texttt{cycle}} + n^{\frac{1}{2}-\nu}\right)\omega\,.$$

The following estimation is heuristic, and implicitly relies on the assumption that the average complexity of `cycle` for a random point of a random function does not significantly differ from the average complexity of `cycle` for a random point of a fixed random function that has a ν-component. We provide a reasoning to justify why it seems very unlikely that this assumption would significantly distort the complexity at the end of the section.

We now compute the complexity of `offline_search`. First, we compute the complexity of one single iteration of a Step 1 loop of `offline_search`. Recall that p_ν is the probability for a random node $x \in \mathbb{F}_2^c$ of a random function to belong to a ν-component. Each time a pair (β, x) is generated, the algorithm `cycle` is executed. Then, the algorithm executes `is_big` when a pair is such that x belongs to a ν-component of $G(P_\beta)$, which happens with probability p_ν. Finally, we need to compute after how many Step 1 iterations the algorithm returns a block β on average. At each iteration, the probability to generate a random $\beta \in \mathbb{F}_2^r$ such that $G(P_\beta)$ has a (s,ν)-component is $p_{s,\nu}$. Given such β, the probability that the randomly drawn $x \in \mathbb{F}_2^c$ also belongs to this component is greater than s. Thus, on average, we select a node x in a (s,ν)-component after less than $\frac{1}{p_{s,\nu}s}$ iterations. Heuristically, one can thus expect the overall complexity of our algorithm to be of the following form:

$$\frac{1}{p_{s,\nu}s}\left[\mathcal{T}_{\texttt{cycle}} + p_\nu\left(\mathcal{T}_{\texttt{cycle}} + n^{\frac{1}{2}-\nu}\right)\omega\right]\,.$$

Yet, we need to adjust this expression. Recall that a (s,ν)-component is a ν-component of size *greater or equal* to sn. However, drawing a node in a ν-component of size exactly sn[10] is not enough to make the algorithm stop. Indeed, if a randomly drawn $\beta \in \mathbb{F}_2^r$ is such that $G(P_\beta)$ has a ν-component of size exactly sn, the probability that s_{obs} is greater than $s + \delta$ is smaller than 1% (here, we use the central limit theorem again). We thus need to adjust our computation. To do so, we lower bound the probability that a random pair (β, x) is selected by the probability that it satisfies the two following conditions:

(a) $G(P_\beta)$ has a (s^+,ν)-component with $s^+ \geq s + 2\delta$ (recall that $\delta = \frac{2.33}{2\sqrt{\omega}}$).
(b) x belongs to the (s^+,ν)-component, so that the algorithm `offline_search` randomly draws ω other values in $\mathbb{F}_2^c$ and computes s_{obs}.

Indeed, we show that satisfying these two conditions implies that $s_{obs} \geq s + \delta$ with great probability, which is exactly the condition that needs to be satisfied for the algorithm to return β.

[10] or greater than sn, but too close to sn to make the algorithm stop.

Assume that the algorithm randomly draws $\beta \in \mathbb{F}_2^r$ and $x \in \mathbb{F}_2^c$ such that conditions (a) and (b) are satisfied. Then by the central limit theorem, we have that for any $\epsilon \in \mathbb{R}$,

$$p\left[s_{obs} - s^+ \geq \epsilon\sqrt{\frac{s^+(1-s^+)}{\omega}}\right] = p\left(Y \geq \epsilon\right)$$

where $Y \rightsquigarrow \mathcal{N}(0,1)$. For $\epsilon = -2.33$, we thus have that with probability $p_\delta \geq 0.99$,

$$s_{obs} \geq s^+ - \delta \geq s + \delta\,.$$

Thus, the probability that a randomly chosen β is selected is lower bounded by $p_{s^+,\nu}s^+p_\delta$. It comes that the average complexity of the offline phase satisfies

$$\begin{aligned}
\mathcal{T}_{\texttt{offline}} &\leq \frac{1}{p_{s^+,\nu}s^+p_\delta}\left[\mathcal{T}_{\texttt{cycle}} + p_\nu\left(\mathcal{T}_{\texttt{cycle}} + n^{\frac{1}{2}-\nu}\right)\omega\right] \\
&= \frac{\mathcal{T}_{\texttt{cycle}}}{0.99}\left[\sqrt{\frac{\pi}{2(1-s^+)s^+}}n^\nu\right]\left[1 + p_\nu\omega\left(1 + \frac{n^{\frac{1}{2}-\nu}}{\mathcal{T}_{\texttt{cycle}}}\right)\right] \\
&\leq n^{\frac{1}{2}+\nu}\sqrt{\frac{2\pi}{(1-s^+)s^+}}\left[1 + p_\nu\omega\left(1 + \mathcal{O}(n^{-\nu})\right)\right] \\
&= n^{\frac{1}{2}+\nu}\sqrt{\frac{2\pi}{(1-s^+)s^+}}\left[1 + \frac{\sqrt{2\pi}\omega}{2n^\nu} + \mathcal{O}\left(\frac{1}{n^{2\nu}}\right)\right] \\
&= \left[\frac{1}{\sqrt{(1-s^+)s^+}}\right]\left[n^{\frac{1}{2}+\nu}\sqrt{2\pi} + n^{\frac{1}{2}}\omega\pi + \mathcal{O}\left(n^{\frac{1}{2}-\nu}\right)\right]\,.
\end{aligned}$$

For $s^+ = 0.73, \nu = \frac{1}{4}$, the last expression is equivalent to $\sqrt{\frac{2\pi}{(1-s^+)s^+}}n^{\frac{3}{4}} < 6n^{\frac{3}{4}}$.

The previously mentioned heuristic assumption on the similarity of the average tail length of a random point of a random function that has a ν-component and the average tail length of a random point of a random function only underlies the second appearance of $\mathcal{T}_{\texttt{cycle}}$ in the first bound on $\mathcal{T}_{\texttt{offline}}$ above. Even if these two statistics differed non-negligibly, this would not significantly distort the complexity. Since `is_big` is executed only when a ν-component is detected, $\mathcal{T}_{\texttt{cycle}}$ in its second occurence is multiplied by $p_\nu\omega \ll 1$ whilst it appears otherwise on its own.

Success probability of the offline algorithm. We consider that the algorithm is successful if the value β outputted by the offline algorithm is such that P_β has a (s,ν)-component. We have seen in Section 3.3 that when a β is selected (that is, $s_{obs} \geq s + \delta$), the central limit theorem guarantees that the real size of the ν-component investigated is greater than sn with probability p_δ. It comes that the success probability of the offline algorithm $p_{\texttt{offline}}$ is equal to $p_\delta \geq 0.99$.

3.5 Online phase

In this section, we describe and analyse the online algorithm `online_algo`. The online phase consists in submitting (N, A, C, T) queries to the decryption oracle $\mathcal{O}_{dec}$ with $C = \beta_\ell$ where β has been outputted by the offline algorithm. In the following, we describe how the tags are constructed and how we choose the value of ℓ.

Algo. `online_algo`(β, e, μ, ℓ, m)

1 : **for** $i = 1..m$ **do**
2 : $N \xleftarrow{\$} \mathbb{F}_2^\eta; y \leftarrow e$
3 : **for** $j = 1..\mu$ **do**
4 : $T \leftarrow P_{final}(\beta||y)$
5 : **if** $\mathcal{O}_{dec}(N, \varepsilon, \beta_\ell, T) \neq \perp$
6 : **return** $(N, \varepsilon, \beta_\ell, T)$
7 : **else** $y \leftarrow P_\beta(y)$
8 : **return false**

Recall that we denote by x_0 the inner state at the end of the decryption's initial phase, and that when the ciphertext C is equal to β_ℓ, the state before the final phase is equal to $\beta||P_\beta^{\ell-1}(x_0)$. Recall that β, which has been outputted by the offline algorithm, is such that $G(P_\beta)$ has a large component with a small cycle of this large component's cycle. Since this component is large, x_0 belongs to it with great probability. In that case, one can choose ℓ large enough for $P_\beta^\ell(x_0)$ to be likely to reach the small cycle of $G(P_\beta)$'s large component. The number of candidates for the state before the final phase, and thus for the possible tags, is thereby reduced.

More formally, assume that the offline algorithm has been successful, that is, it has outputted β, μ and e such that $G(P_\beta)$ has a (s, ν)-component with a cycle of length μ and such that e is an element of this cycle. If $G(P_\beta)$ has a (s, ν)-component, x_0 belongs to this component with probability greater than or equal to s. If x_0 belongs to the (s, ν)-component, $P_\beta^{\ell-1}(x_0)$ reaches the cycle with the probability p_ℓ introduced in Section 2.1. It comes that with probability at least $p_\ell s$, $P_\beta^{\ell-1}(x_0)$ belongs to the cycle. Here, we implicitly rely on the assumption that the random variable tail length of a random point of a random function such that this random point belongs to a (s, ν)-component has the same distribution as the tail length of a random point of a random function. Although we do not believe this assumption to be true in the sense that the distributions are not strictly equal, we believe that they are close enough for it not to impact our attack significantly. Small scale experiments described in Section 4 corroborate this assumption.

We now analyse Algorithm `online_algo`. For a nonce $N \in \mathbb{F}_2^\eta$, this online phase consists in submitting the following (N, A, C, T) queries to the decryption oracle:

$$\left(N, \varepsilon, \beta_\ell, P_{final}(\beta || P_\beta^i(e))\right) \quad \text{for } i = 0, \ldots, \mu - 1 \,.$$

Time complexity. For each nonce, the attacker submits μ decryption queries with a ciphertext of ℓ blocks, and for each decryption query, she must also apply P_{final} to the current state $\beta||y$ where y is the current cycle element and apply P_β to y in order to try exhaustively all the cycle elements. Since $\ell = \mathcal{O}(n^{\frac{1}{2}})$, the time complexity incurred by the above invocations of P_β and P_{final} at each decryption query is negligible compared to the time complexity of a decryption query with $C = \beta_\ell$, that can be approximated by ℓ applications of P. The time complexity for each nonce is thus well approximated by $\mu\ell$ and thus upper bounded by

$$n^{\frac{1}{2}-\nu}\ell \,.$$

Thus, for m nonces, the average complexity of the online phase in number of calls to P verifies

$$\mathcal{T}_{\texttt{online}} \leq n^{\frac{1}{2}-\nu} m\ell \,.$$

Probability of success. For each nonce, the probability of success is exactly the probability that $P_\beta^{\ell-1}(x_0)$ belongs to the cycle which we showed to be at least $p_\ell s$. If we repeat this experiment with m nonces, the probability of success is thus at least

$$p_{\texttt{online}} = 1 - (1 - p_\ell s)^m \,.$$

3.6 Complexity and success probability of the attack

The complexity of the attack is the sum of the complexities of the online algorithm and the offline algorithm. Note that in practice, the attacker only needs to run the precomputation offline algorithm once in order to be able to execute the online phase to attack the same duplex-based mode with any secret key. The average complexity $\mathcal{T}_{\texttt{online}} + \mathcal{T}_{\texttt{offline}}$ of the attack can be upper bounded by

$$n^{\frac{1}{2}-\nu} m\ell + \left\lceil \frac{1}{\sqrt{(1-s^+)s^+}} \right\rceil \left[n^{\frac{1}{2}+\nu}\sqrt{2\pi} + n^{\frac{1}{2}}\omega\pi \right] + \mathcal{O}\left(n^{\frac{1}{2}-\nu}\right) \,.$$

Similarly, the overall success probability of the attack is of the form

$$\begin{aligned} p_{\texttt{success}} = p_{\texttt{offline}} p_{\texttt{online}} &\geq p_\delta \left[1 - (1 - p_\ell s)^m\right] \\ &\geq 0.99(1 - e^{-mp_\ell s}) \quad (\text{since } \forall x \in \mathbb{R}, e^{-x} > 1 - x) \,. \end{aligned}$$

In order to have an overall probability of success $p_{\texttt{success}}$ greater than 0.95, we want $1 - e^{-mp_\ell s}$ to be greater than 0.96. Recall from Section 2.1 that $p_\ell \geq 1 - e^{\frac{-l^2}{2n}}$. We set $\ell = 3\sqrt{n}$ so that $p_\ell \geq 0.988$. We also set the following values:

- $\omega = 2^{10}$; thus $2\delta = \frac{2.33}{\sqrt{\omega}} < 0.08$;
- $s^+ = 0.73$; thus $s = s^+ - 2\delta = 0.65$.

Since $p_\ell \geq 0.988$ and $s = 0.65$, $p_{\texttt{online}}$ is greater than 0.96 for $m \geq 5$. We thus set $m = 5$. Thus, the complexity has the form

$$15n^{1-\nu} + 6n^{\frac{1}{2}+\nu} + \frac{4\pi 2^{10}}{\sqrt{3}} n^{\frac{1}{2}} + \mathcal{O}\left(n^{\frac{1}{2}-\nu}\right).$$

To balance the above expression, we set $\nu = \frac{1}{4}$, we get an attack of complexity at most $21n^{\frac{3}{4}} + \mathcal{O}(\sqrt{n}) = 21 \times 2^{\frac{3c}{4}} + \mathcal{O}(\sqrt{n})$ in number of calls to P.[11]

Note that the complexity of the online phase is at most $15n^{1-\nu}$. In particular, given β such that $G(P_\beta)$ has a component of size greater than ns with a cycle length close to 1, the online complexity can be brought close to $2^{\frac{c}{2}+4}$.

3.7 Key-recovery

For modes such that P_{init} is reversible for known nonce and associated data, our forgery attack also recovers the secret key with $\ell = \mathcal{O}(\sqrt{n})$ extra applications of P, which is negligible compared to the complexity of the forgery. Indeed, if the decryption oracle receives a forgery (N, A, C, T), it returns the corresponding plaintext. This allows the attacker to recover the state at the end of the initial phase with $\ell = \mathcal{O}(\sqrt{n})$ extra applications of P^{-1}.[12] As a consequence, if the function P_{init} is reversible for known nonce and associated data, the attacker recovers the secret key.

4 Small scale experiments

Our attack relies on the assumption that the P_β's derived from a public permutation P defined on $\mathbb{F}_2^b$ behave as random functions on $\mathbb{F}_2^c$. To statistically verify this assumption, we implemented some experiments using the permutation Xoodoo[12] as P. The main reason behind this choice is its use in the finalist of the NIST lightweight cryptography competition Xoodyak, but another permutation used in practice with a reasonably large value of b would have done just as well. We took toy values compared to Xoodyak for the capacity ($c \leq 40$) in order for computer experiments to remain easy to achieve. Since our attack relies mainly on the random function statistics results introduced in Section 2.1, we designed a test for each of these results. We also implemented the algorithm `offline_search` for small values of c.

[11] Note that $\nu = \frac{1}{4}$ is not the fully optimal choice in general. Rather, the optimal choice for ν is $\nu = \frac{1}{2}\left(\frac{1}{2} - \log_n\left(\sqrt{2\pi}/(\sqrt{s^+(1-s^+)}15)\right)\right)$. For example, for $n = 2^{128}$, the optimal choice is approximately 0.256.

[12] The plaintext also allows to verify that the recovered state before the final phase is correct, making the key recovery possible with only a negligible amount of extra computations regardless of the tag length.

Algo. `cycle_expectancy`(Ω)

1 : $tot \leftarrow 0$
2 : **for** $i = 1..\Omega$ **do**
3 : $\quad \beta \xleftarrow{\$} \mathbb{F}_2^r;\ x \xleftarrow{\$} \mathbb{F}_2^c$
4 : $\quad (\mu, e) \leftarrow \texttt{cycle}(\beta, x)$
5 : $\quad tot \leftarrow tot + \mu$
6 : **return** tot/Ω

Algo. `nu_components`(Ω, ν)

1 : $ctr \leftarrow 0$
2 : **for** $i = 1..\Omega$ **do**
3 : $\quad \beta \xleftarrow{\$} \mathbb{F}_2^r;\ x \xleftarrow{\$} \mathbb{F}_2^c$
4 : $\quad (\mu, e) \leftarrow \texttt{cycle}(\beta, x)$
5 : $\quad$ **if** $\mu \leq n^{\frac{1}{2}-\nu}$ **then** $ctr \leftarrow ctr + 1$
6 : **return** ctr/Ω

Expectancy of cycle/tail length. We wish to verify that for a random $\beta \in \mathbb{F}_2^r$ and for a random node $x \in G(P_\beta)$, the expectancy of the cycle length $\mu(x)$ and tail length $\lambda(x)$ are both equal to $\sqrt{\frac{\pi n}{8}}$, with $n = 2^c$. Note that the variance of the cycle length and tail length for a random node of a random function is equal to $\sigma_\mu^2 = n\left[\frac{2}{3} - \frac{2\pi}{16}\right]$ [25]. Regarding the cycle length, we use the algorithm `cycle_expectancy`. After Ω tries, by the Central Limit Theorem, the observed average cycle length `mean` outputted by the algorithm `cycle_expectancy` is such that with probability about 0.99:

$$\texttt{mean} \in \left[\sqrt{\frac{\pi n}{8}} - \frac{2.58\sigma_\mu}{\sqrt{\Omega}}; \sqrt{\frac{\pi n}{8}} + \frac{2.58\sigma_\mu}{\sqrt{\Omega}}\right].$$

Setting $\Omega = 14000$, we verify whether `mean` is in this interval in our tests (see Table 1). We use a similar algorithm and reasoning for the tail length.[13] As shown in Table 1, all our experimental results match the theory.

c	28	32	36	40
Expectancy	10267	41068	164274	657098
Confidence interval	[10080, 10454]	[40321, 41817]	[161283, 167266]	[645130, 669065]
tail `mean`	10323	40971	163732	654775
cycle `mean`	10255	41620	164445	650156

Table 1. Expectancy of cycle length and tail length

Probability for a random point to belong to a ν-component. We wish to verify that for a random $\beta \in \mathbb{F}_2^r$ and for a random node $x \in G(P_\beta)$, the

[13] Here, we make the assumption that the standard deviation for the tail length is the same as the cycle length's. This assumption does not affect our attack, it only affects how to interpret our test results.

probability that x belongs to a ν-component is p_ν. To do so, we use the algorithm `nu_components` and focus in practice on experiments where $\nu = \frac{1}{4}$ as it is the value used in our attack. Drawing Ω random (x, β) pairs and computing the proportion of such pairs such that x belongs to a ν-component of P_β amounts to computing the mean of Ω Bernoulli variables equal to 1 with probability p_ν and 0 with probability $1-p_\nu$. By the Central Limit Theorem, since the standard variation of the above variables is $\sigma_\nu = \sqrt{p_\nu(1-p_\nu)}$, we have that

$$\texttt{proportion} \in \left[p_\nu - \frac{2.58\sigma_\nu}{\sqrt{\Omega}}; p_\nu + \frac{2.58\sigma_\nu}{\sqrt{\Omega}} \right]$$

with probability ≥ 0.99. We verify in our experiments that `proportion` is indeed in this confidence interval for various values of c and Ω. Our results, which match the theory, are displayed in Table 2.

	c	28	32	36	40
	Ω	71344	142688	285376	570752
ν-component	Expectancy	0.009792	0.004896	0.002448	0.0012
	Conf. interval	$[0.0088, 0.0110]$	$[0.0044, 0.0054]$	$[0.0022, 0.0027]$	$[0.0011, 0.0013]$
	`proportion`	0.010162	0.004787	0.002400	0.001242
(s,ν)-component	$p_{s^+,\nu} s^+ p_\delta$	0.002740	0.001370	0.000685	0.000342
	$p_{s,\nu} s$	0.002973	0.001487	0.000743	0.000372
	`frequency`	0.003714	0.001647	0.000880	0.000517

Table 2. Probability for a random node in the graph of a random function to belong to a ν-component and experimental verification of (the detection of) the occurence of (s, ν)-components.

Probability for a random function to have a (s, ν)-component. Verifying that for a random $\beta \in \mathbb{F}_2^r$, the probability that $G(P_\beta)$ has a (s, ν)-component is $p_{s,\nu}$ is hard in practice as it is too costly to determine with probability 1 whether or not $G(P_\beta)$ has a (s, ν)-component. Instead, we make an indirect verification. We draw Ω random (x, β) pairs and compute the proportion `frequency` of such pairs such that:

- P_β has a ν-component (the component to which x belongs);
- the estimated size s_{obs} of this component (given by the proportion, among $\omega = 1000$, of newly chosen random points that belong to this component) is

at least $s+\delta$ with $\delta = \frac{2.33}{2\sqrt{\omega}}$, so that the actual size of the component is very likely to be at least sn.[14]

We compare the found `frequency` values with two values:

- the lower bound $p_{s,\nu}s$ on the probability that P_β has a (s,ν)-component and x belongs to it;
- the conservative lower bound $p_{s^+,\nu}s^+p_\delta$ on the average value of `frequency` used in the complexity estimate of the offline algorithm of Section 3.4, where $s^+ = s + 2\delta$.

This algorithm is easy to derive from the algorithm `offline_search` introduced in Section 3.3. We make this experiment for various values of Ω and c and with $s = 0.65$, $s^+ = 0.73$, as they are the values used in our attack. Our results match the theory and are displayed in Table 2. The difference (in favour of the adversary) between theoretical bounds and experimental values for `frequency` in Table 2 can be at least partially explained as follows. The value $p_{s,\nu}$ is the probability that a component is a ν-component of size *greater or equal* to s. The probability that a random point belongs to a (s,ν)-component is thus strictly greater than $p_{s,\nu}s$ as the proportion of the points belonging to a component of size greater than s is greater than s.

c	β ($b-c$ bits)	μ
28	`1473b86a 2607d6e5 5234df22 4c111c51 122e188f 37586e28` `5b74f306 40ac1d69 2bb9c59f 6e8479b7 3d6ec314 7`	10
32	`67b9632a 032ec1a3 1f3b4f8c 7c641f59 39e3cab6 3aaa4444` `73bf377d 7f1f6b35 6412ffb2 523d5180 54465a4f`	4
36	`59189691 3e3769f2 293b1b6f 0cc0af85 7d96b0a4 0e1c201b` `137523e8 11f61a60 6c06c85f 762716b7 276c730`	122
40	`7b2fb641 7874c3d6 171abbc2 231ebf22 4e6e1ad3 2d6df079` `7e6457aa 7816dd2a 011fe0f3 1de6ee24 56f1ed`	18

Table 3. We provide a few examples of obtained β values such that P_β has a (s,ν)-component with $\mu \ll 2^{\frac{c}{4}}$. Although we only provide one for each c, several values of β are available for each c.

[14] To reduce the execution time of the implementation, to detect whether or not two points x, x' are in the same component, we checked whether $\mu(x) = \mu(x')$ instead of doing the exhaustive verification made in `is_big`.

Some values of β. Lastly, we implemented the algorithm `offline_search` for small values of c and for $s = 0.65$. A few examples of obtained β values are displayed in Table 3.

Probability for a random point to belong to its component's cycle after $\ell - 1$ applications of the random function. In order to verify the applicability of Harris' result on the number of successors provided in Section 2.1 to points randomly drawn in a (s, ν)-component, *a fortiori* when the function is randomly sampled from the set of all P_β's rather than randomly sampled in $\mathfrak{F}_n$, we also conducted the following experiment. For all β values displayed in Table 3, we checked whether the points that were found to be in the (s, ν)-component had a tail length smaller than $\ell = 3\sqrt{n}$. For all these β, 100% of the points found to be in the (s, ν)-component have a tail length smaller than ℓ. This gives us reasonable confidence in the fact that this assumption is realistic and does not lead to a significant overestimation of the success probability of the attack.

5 Application to concrete duplex-based modes

In this section, we apply the attack previously described on the simplified DuplexAEAD mode to concrete AEAD duplex-based modes. Whilst our attack can be easily adapted to many of them as summarized in Table 5, others frustrate our attack. For all the modes presented in Table 5, our attack enables key recovery.

5.1 Highlights

For the attack to succeed, we identify two requirements:

1. In decryption mode, the ciphertext blocks must overwrite the outer state at least in part;
2. The tag must be determined by the state before the final phase, in such a way that a correct guess on this state gives us the correct tag.

As a consequence, we will show that in all modes for which the attack is applicable, the padding rule applied to the message does not matter, even though when the padding rule is made block by block the complexity of our attack can be slightly greater.

As a main concrete implication, we provide an attack on Xoodyak that breaks the claim of achieving 184-bit security against plaintext recovery and forgery attacks made by the designers in [18, Corollary 2, p. 72] but does not threaten the 112-bit security level required by NIST for the lightweight competition. The attack also breaks the penultimate `NORX` version `NORX` v2 [1] but not the more recent version `NORX` v3 [2].

Secondly we highlight two reasons that prevent our attack from applying to modes such as Beetle [16], Ascon [22] or `NORX` v3 [2] in Section 5.3. We mainly identify two reasons:

Mode	monkeyWrap/monkeyDuplex										Cyclist	Motorist	
Scheme	NORX v2[3]		Ketje				KNOT				Xoodyak	Keyak	
Instance	N-32	N-64	Jr	Sr	Mi	Ma	KNOT-AEAD				Xoodyak	River	Lake
b	512	1024	200	400	800	1600	256	384	384	512	384	800	1600
r	384	768	16	32	128	256	64	192	96	128	192	544	1344
r' [1]	382	766	14	30	126	254	64	192	96	128	192	544	1344
c	128	256	184	368	672	1344	192	192	288	384	192	256	256
Sec. level[2]	128	256	96	128	128	128	125	128	189	253	184	128	128
$\mathcal{T}$	2^{102}	2^{198}	2^{144}	2^{282}	2^{506}	2^{1014}	2^{148}	2^{148}	2^{220}	2^{292}	2^{148}	2^{196}	2^{196}

Table 4. Summary of our results. Our attack is the best known generic attack against the modes displayed in blue;[1] r' is the length of the outer state part that is overwritten by ciphertext blocks taking into a account a potential block by block padding such as in monkeyDuplex;[2] Claimed plaintext integrity security level for a key and tag of sufficient, potentially maximal, length;[3] Note that a more efficient (although not generic) attack has been devised in [15].

– the use of a linear application that prevents outer state overwriting such as the *feedback function* proposed in the Beetle mode [16] (requirement 1 is not fulfilled);[15]
– the use of the secret key in the final phase to produce the tag such as in Ascon and NORX v3 [22,2]. Indeed, in that case, a correct guess on the state before the final phase does not determine the tag (requirement 2 is not fulfilled).

5.2 Schemes to which the attack can be applied

The attack is applicable to the following duplex-based constructions: the Cyclist, monkeyDuplex and Motorist modes. Therefore, the attack is applicable to Xoodyak, Ketje, KNOT, NORX v2 and Keyak. This section is made to help the reader check attack details for all AEAD algorithms on which the attack is applicable.

[15] Although the first mode that uses a feedback function is COFB [17], this mode is out of scope as it is not duplex-based (in fact, it is not even permutation-based but block-cipher based).

Cyclist mode. A well known representative of the family of duplex-based ciphers is Xoodyak [18], a finalist of the NIST lightweight cryptography competition.[16] Xoodyak uses the permutation Xoodoo as its state update function.

As explained in Section 3.5, the specification of the initial phase does not influence the applicability of the attack. Hence, we only focus on the ciphertext processing and the final phase. After the initial phase, a byte is set to '80' and is XORed into the state, but this is only the case for the first block. Thus, we consider that the processing of the first ciphertext block belongs to the initial phase. Moreover, each time a ciphertext block overwrites the outer state, a byte set to '01' is XORed into the state at the bit positions $r+1, \ldots, r+8$[17], where $r = 24$ bytes $= 192$ bits [18, p. 68]. Thus, the permutation P on which we need to apply `offline_search` is

$$\begin{aligned} P : \mathbb{F}_2^{384} &\longrightarrow \mathbb{F}_2^{384} \\ s &\longmapsto \textsc{Xoodoo}[12](s \oplus 0^{192}||0^7||1||0^{184}) \,. \end{aligned}$$

If one considers the padding rule 10^* together with the interface provided by the Xoodyak authors, one can notice that all those transformations are deterministic in the value of the inner state just before processing the last ciphertext block. Hence, we can guess the last state value, and apply the final phase (considering the padding for the last block inside this transformation) to the last block and apply the attack. This means that for any tag length, our attack strategy provides a forgery with a complexity of $2^{148.4}$.

When a valid decryption query is provided to the decryption oracle, the corresponding plaintext is returned. The plaintext allows the attacker to check that her guess on the state before the final phase is correct as she can invert the whole process. Doing so, she can also recover the state just after processing the key. As long as the key is smaller than 44 bytes, it is copied entirely in the state and then A, N and M are processed. Thus, P_{init} is reversible for known N and A. As described in Section 3.7 the attack can thus be turned into a key recovery for Xoodyak.

For a t-byte tag and a κ-bit secret key with $\kappa \leq 192$, the authors claim that Xoodyak has a security strength level in bits of $\min(184, \kappa, 8t)$ in computation where the data is limited to $96+\kappa/2$. Our attack breaks this claim for $\kappa \geq 152$ and $t \geq 19$. Our attack not only produces a forgery but also recovers the secret key. Typically, for $\kappa = 192$ and $t = 24$, there should be no attack with a complexity under 2^{184} in time and 2^{192} in data. Yet, our attack has a complexity of $2^{148.4}$.

MonkeyDuplex: Ketje, KNOT and `NORX` v2. Ketje [12], KNOT-AEAD [34] and `NORX` v2 [1] use the mode monkeyDuplex defined originally in 2014 for

[16] For more details on Xoodyak's specification, we refer to [18] at page 62 for the keyed mode together with pages 67 and 68 for the full description of what is relevant for our analysis. `https://csrc.nist.gov/Projects/lightweight-cryptography/`

[17] Note that the rate called r in this paper is denoted by n in Xoodyak's specification.

KETJE [11]. However, those algorithms differ in the padding rule applied to the message.

- In KNOT-AEAD, the padding is made by 'appending a single 1 and the smallest number of 0's to the bit string such that the length of the padded bit string is a multiple of r bits'. Thus, the technique used for XOODYAK can be applied.
- In KETJE and in `NORX`, the padding is made block by block [12, 1.3, p. 4] and [1, p. 9-10] with the rule $\text{pad}10^*1[r](|M|)$ which is the multi-rate padding as defined in [7]. Both ciphers allow to process bit strings of length smaller or equal[18] to $r' = r - 2$. To deal with this padding rule, we consider that every plaintext block is of length exactly r', meaning that the padding rule is just the concatenation of 11 to each block. By doing the same technique together with the domain separation as for XOODYAK, that is considering that the position corresponding to those two bits are in fact inside the inner state and that the XOR of 11 is part of the KETJE inner permutation P (or part of the `NORX` inner permutation), we can apply our attack, by considering the effective rate to be $r' = r - 2$ instead of r.

For these three constructions, the final phase is deterministic in the inner state after processing the plaintext or the ciphertext so our attack can be applied. Both ciphers come with 4 different instances and our attack leads to the complexities listed in Table 5. Moreover, when the decryption oracle sends back the plaintext, a key recovery is possible as the state is directly initialized with key and nonce without applying any transformation.

Motorist: KEYAK. KEYAK [13] is a family of authenticated encryption schemes which is a third-round candidate of the CAESAR competition. The features of the Motorist mode that are of interest for our attack are:

- Decryption overwrites the outer state with ciphertext blocks values and in between there are applications of P where P is a KECCAK-p permutation.
- An encoding (in byte) of the size of the processed message blocks is XORed into the state at the $R_a + 1$ byte position, where R_a is the absorbing rate in bytes ($R_a = \frac{r}{8}$).
- For the tag generation, given a (public) tag length, the tag is determined by the state before the final phase.

Thus, the attack can also be applied on KEYAK instances by just changing the function that is iterated and considering the XOR of the bytes that encode the length of ciphertext and associated data to be part of the decryption function.

However, KEYAK comes with instances that can work independently (Pistons), in order to highly parallelize encryption and decryption of large amount of data. To do so, the authors of KEYAK propose to do several initialization in parallel, process independently 4, 8 or more strings and then mix everything

[18] This value is called ρ_{max} in [7, p. 335]

together at the end. However, if one would like to guess the tag, one would have to guess independently every inner state in all parallel instances as they are all initialized differently. So, our attack on KEYAK is also applicable on all instances but make sense only for the non-parallelized instances, that is RIVER KEYAK and LAKE KEYAK. For these instances, the key can also be recovered.

5.3 Modes that frustrate our attack

In this section, we will look into authenticated encryption modes to which the attack cannot be applied. Two main features frustrate our cryptanalysis. The first one consists of a final key addition just at the beginning of the final phase and the second one is the use of a *feedback function* as proposed in the mode Beetle [16].

Key-dependent final phase. Our attack allows to reduce the space of all possible values of the final state before the final phase. For the modes to which the attack applies, a correct guess on this state can be transformed into a forgery, and, under certain conditions, into a key recovery. However, some proposals such as Ascon [22] or the third version of `NORX` [2] for the CAESAR competition slightly change the final phase by making it not only dependent on the state after processing the plaintext or ciphertext, but also key dependent. As explained at the beginning of this section, the final tag must be fully determined by the state just before the final phase. When the key is involved in the final phase, even if the correct state value is guessed, an attacker does not know *a priori* the secret key and so the state recovery does not lead to a forgery attack.

Absence of outer state overwriting. In another line of research, Chakraborti et al. proposed in 2018 a family of AEAD constructions named Beetle [16]. Beetle is not strictly speaking a duplex-based construction as the use of a *feedback function* between two consecutive invocations of the permutation P avoids, when both a data injection and a data extraction take place, that the r-bit outer state value after the data injection be equal to the extracted data block as in traditional duplexing, and allows to render both values almost independent. Thus, the use of a feedback function prevents the attack by making the outer state overwriting impossible. Unlike duplex-based constructions which do not use a feedback function twist, the Beetle construction is claimed to provide a security level $\min\{\frac{b}{2}, c - \log(r), r\}$ without any restriction on decryption queries [16,3]. The Beetle construction is used in the Beetle and SPARKLE [5] finalists of the NIST lightweight cryptography standardization process.

The same effect also occurs in Subterranean 2.0 [19] which is a second round candidate of the NIST-lightweight competition and a duplex-like proposal. In order to generate the ciphertext blocks, two bits of the state are XORed to serve as a keystream. It prevents the attack in the same way that the feedback function of the Beetle mode does, as the attacker can not predict the value of the outer state before the next application of the update function.

6 Conclusion

In this paper, we provided a generic attack against several duplex-based AEAD modes. Constructed as a forgery attack, our cryptanalysis can be transformed into a key recovery for most modes to which it applies. It has a complexity equivalent to $\mathcal{O}(2^{\frac{3c}{4}})$ applications of the state update function and thus represents an improvement from previous generic attacks against duplex-based modes. It is also memory-less. Further, the complexity of the online phase can be brought close to $2^{\frac{c}{2}}$ at the cost of increasing the time complexity of the pre-computation phase (which needs to be run only once) above $\mathcal{O}(2^{\frac{3c}{4}})$.

Acknowledgements. We wish to thank Jérôme Plût and Hugues Randriam for our insightful conversations on the statistics of random functions.

References

1. Aumasson, J.P., Jovanovic, P., Neves, S.: NORX v2. Submission to the Caesar competition (2015), `https://competitions.cr.yp.to/round2/norxv20.pdf`
2. Aumasson, J.P., Jovanovic, P., Neves, S.: NORX v3. Submission to the Caesar competition (2016), `https://competitions.cr.yp.to/round3/norxv30.pdf`
3. Banik, S., Chakraborti, A., Iwata, T., Minematsu, K., Nandi, M., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: GIFT-COFB. Cryptology ePrint Archive, Report 2020/738 (2020), `https://eprint.iacr.org/2020/738`
4. Bao, Z., Guo, J., Wang, L.: Functional graphs and their applications in generic attacks on iterated hash constructions. IACR Transactions on Symmetric Cryptology **2018**(1), 201–253 (2018). doi:10.13154/tosc.v2018.i1.201-253
5. Beierle, C., Biryukov, A., Cardoso dos Santos, L., Großschädl, J., Perrin, L., Udovenko, A., Velichkov, V., Wang, Q.: Lightweight AEAD and hashing using the Sparkle permutation family. IACR Transactions on Symmetric Cryptology **2020**(S1), 208–261 (2020). doi:10.13154/tosc.v2020.iS1.208-261
6. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: On the indifferentiability of the sponge construction. In: Smart, N.P. (ed.) Advances in Cryptology – EUROCRYPT 2008. Lecture Notes in Computer Science, vol. 4965, pp. 181–197. Springer, Heidelberg, Germany, Istanbul, Turkey (Apr 13–17, 2008). doi:10.1007/978-3-540-78967-3_11
7. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Duplexing the sponge: Single-pass authenticated encryption and other applications. In: Miri, A., Vaudenay, S. (eds.) SAC 2011: 18th Annual International Workshop on Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 7118, pp. 320–337. Springer, Heidelberg, Germany, Toronto, Ontario, Canada (Aug 11–12, 2012). doi:10.1007/978-3-642-28496-0_19
8. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Sponge functions (2007), `https://keccak.team/files/SpongeFunctions.pdf`
9. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Cryptographic sponge functions (2011), `https://keccak.team/files/CSF-0.1.pdf`
10. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Permutation-based encryption, authentication and authenticated encryption (July 2012), `http://www.hyperelliptic.org/djb/diac/record.pdf`

11. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V., Keer, R.V.: Ketje v1. Submission to Caesar competition (2014), http://competitions.cr.yp.to/round1/ketjev1.pdf
12. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V., Keer, R.V.: Ketje v2. Round 3 candidate for the Caesar competition (2016), http://competitions.cr.yp.to/round3/ketjev2.pdf
13. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V., Keer, R.V.: Keyak v2. Submission to the Caesar competition (2016), https://competitions.cr.yp.to/round3/keyakv22.pdf
14. Brent, R.P.: An improved monte carlo factorization algorithm. In: BIT Numerical Mathematics. p. 176–184. Berlin, Heidelberg (1980). doi:https://doi.org/10.1007/BF01933190
15. Chaigneau, C., Fuhr, T., Gilbert, H., Jean, J., Reinhard, J.R.: Cryptanalysis of NORX v2.0. Journal of Cryptology **32**(4), 1423–1447 (Oct 2019). doi:10.1007/s00145-018-9297-9
16. Chakraborti, A., Datta, N., Nandi, M., Yasuda, K.: Beetle family of lightweight and secure authenticated encryption ciphers. IACR Transactions on Cryptographic Hardware and Embedded Systems **2018**(2), 218–241 (2018). doi:10.13154/tches.v2018.i2.218-241, https://tches.iacr.org/index.php/TCHES/article/view/881
17. Chakraborti, A., Iwata, T., Minematsu, K., Nandi, M.: Blockcipher-based authenticated encryption: How small can we go? In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2017. Lecture Notes in Computer Science, vol. 10529, pp. 277–298. Springer, Heidelberg, Germany, Taipei, Taiwan (Sep 25–28, 2017). doi:10.1007/978-3-319-66787-4_14
18. Daemen, J., Hoffert, S., Peeters, M., Van Assche, G., Van Keer, R.: Xoodyak, a lightweight cryptographic scheme. IACR Transactions on Symmetric Cryptology **2020**(S1), 60–87 (2020). doi:10.13154/tosc.v2020.iS1.60-87
19. Daemen, J., Massolino, P.M.C., Mehrdad, A., Rotella, Y.: The subterranean 2.0 cipher suite. IACR Transactions on Symmetric Cryptology **2020**(S1), 262–294 (2020). doi:10.13154/tosc.v2020.iS1.262-294
20. Daemen, J., Mennink, B., Assche, G.V.: Full-state keyed duplex with built-in multi-user support. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017, Part II. Lecture Notes in Computer Science, vol. 10625, pp. 606–637. Springer, Heidelberg, Germany, Hong Kong, China (Dec 3–7, 2017). doi:10.1007/978-3-319-70697-9_21
21. DeLaurentis, J.M.: Components and cycles of a random function. In: Pomerance, C. (ed.) Advances in Cryptology – CRYPTO'87. Lecture Notes in Computer Science, vol. 293, pp. 231–242. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 1988). doi:10.1007/3-540-48184-2_21
22. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.2: Lightweight authenticated encryption and hashing. Journal of Cryptology **34**(3), 33 (Jul 2021). doi:10.1007/s00145-021-09398-9
23. Flajolet, P., Odlyzko, A.M.: Random mapping statistics. In: Quisquater, J.J., Vandewalle, J. (eds.) Advances in Cryptology – EUROCRYPT'89. Lecture Notes in Computer Science, vol. 434, pp. 329–354. Springer, Heidelberg, Germany, Houthalen, Belgium (Apr 10–13, 1990). doi:10.1007/3-540-46885-4_34
24. Flajolet, P., Sedgewick, R.: Analytic Combinatorics. Cambridge University Press (2009), http://www.cambridge.org/uk/catalogue/catalogue.asp?isbn=9780521898065

25. Harris, B.: Probability Distributions Related to Random Mappings. The Annals of Mathematical Statistics **31**(4), 1045 – 1062 (1960). doi:10.1214/aoms/1177705677, https://doi.org/10.1214/aoms/1177705677
26. Joux, A.: Algorithmic Cryptanalysis. Chapman and Hall/CRC (2009). doi:https://doi.org/10.1201/9781420070033
27. Jovanovic, P., Luykx, A., Mennink, B.: Beyond $2^{c/2}$ security in sponge-based authenticated encryption modes. In: Sarkar, P., Iwata, T. (eds.) Advances in Cryptology – ASIACRYPT 2014, Part I. Lecture Notes in Computer Science, vol. 8873, pp. 85–104. Springer, Heidelberg, Germany, Kaoshiung, Taiwan, R.O.C. (Dec 7–11, 2014). doi:10.1007/978-3-662-45611-8_5
28. Jovanovic, P., Luykx, A., Mennink, B., Sasaki, Y., Yasuda, K.: Beyond conventional security in sponge-based authenticated encryption modes. Journal of Cryptology **32**(3), 895–940 (Jul 2019). doi:10.1007/s00145-018-9299-7
29. Leurent, G., Peyrin, T., Wang, L.: New generic attacks against hash-based MACs. In: Sako, K., Sarkar, P. (eds.) Advances in Cryptology – ASIACRYPT 2013, Part II. Lecture Notes in Computer Science, vol. 8270, pp. 1–20. Springer, Heidelberg, Germany, Bengalore, India (Dec 1–5, 2013). doi:10.1007/978-3-642-42045-0_1
30. Mennink, B., Reyhanitabar, R., Vizár, D.: Security of full-state keyed sponge and duplex: Applications to authenticated encryption. In: Iwata, T., Cheon, J.H. (eds.) Advances in Cryptology – ASIACRYPT 2015, Part II. Lecture Notes in Computer Science, vol. 9453, pp. 465–489. Springer, Heidelberg, Germany, Auckland, New Zealand (Nov 30 – Dec 3, 2015). doi:10.1007/978-3-662-48800-3_19
31. Moon, J.W.: Counting Labelled Trees. Canadian Mathematical Congress 1970, William Clowes and Sons (1970)
32. Peyrin, T., Sasaki, Y., Wang, L.: Generic related-key attacks for HMAC. In: Wang, X., Sako, K. (eds.) Advances in Cryptology – ASIACRYPT 2012. Lecture Notes in Computer Science, vol. 7658, pp. 580–597. Springer, Heidelberg, Germany, Beijing, China (Dec 2–6, 2012). doi:10.1007/978-3-642-34961-4_35
33. Peyrin, T., Wang, L.: Generic universal forgery attack on iterative hash-based MACs. In: Nguyen, P.Q., Oswald, E. (eds.) Advances in Cryptology – EUROCRYPT 2014. Lecture Notes in Computer Science, vol. 8441, pp. 147–164. Springer, Heidelberg, Germany, Copenhagen, Denmark (May 11–15, 2014). doi:10.1007/978-3-642-55220-5_9
34. Zhang, W., Ding, T., Yang, B., Bao, Z., Xiang, Z., Ji, F., Zhao, X.: KNOT. Round 2 candidate for the NIST Lightweight Cryptography project (2019), https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/knot-spec-round.pdf