Can Caesar Beat Galois?

Robustness of CAESAR Candidates Against Nonce Reusing and High Data Complexity Attacks

Serge Vaudenay and Damian Vizár(✉)

EPFL, Lausanne, Switzerland
damian.vizar@epfl.ch

Abstract. The Competition for Authenticated Encryption: Security, Applicability and Robustness (CAESAR) has as its official goal to "identify a portfolio of authenticated ciphers that offer advantages over [the Galois-Counter Mode with AES]" and are suitable for widespread adoption." Each of the 15 candidate schemes competing in the currently ongoing 3rd round of CAESAR must clearly declare its security claims, i.e. whether it can tolerate nonce misuse, and what is the maximal data complexity for which security is guaranteed. These claims appear to be valid for all 15 candidates. Interpreting "Robustness" in CAESAR as the ability to mitigate damage when security guarantees are void, we describe attacks with 64-bit complexity or above, and/or with nonce reuse for each of the 15 candidates. We then classify the candidates depending on how powerful does an attacker need to be to mount (semi-)universal forgeries, decryption attacks, or key recoveries. Rather than invalidating the security claims of any of the candidates, our results provide an additional criterion for evaluating the security that candidates deliver, which can be useful for e.g. breaking ties in the final CAESAR discussions.

Keywords: Authenticated encryption · CAESAR competition
Forgery · Decryption attack · Key recovery · Birthday bound
Nonce misuse

1 Introduction

Authenticated encryption (AE) is a symmetric key primitive that simultaneously ensures confidentiality, integrity and authenticity of encrypted messages [4,29] and typically also allows to authenticate a public string, the associated data, along with the message [37]. During the two decades of its existence, AE has been not just a frequent research object but also a frequently used tool (e.g. in IEEE 802.11i, IPsec ESP and IKEv2, NIST SP 800-38D, ANSI C12.22, and ISO/IEC 19772:2009), especially because most practical applications of symmetric key cryptography require both confidentiality and integrity at the same time.

In 2013, the Competition for Authenticated Encryption: Security, Applicability and Robustness (CAESAR) was announced. The reason for its launch was,

B. Preneel and F. Vercauteren (Eds.): ACNS 2018, LNCS 10892, pp. 476–494, 2018.
https://doi.org/10.1007/978-3-319-93387-0_25

in part, a startling amount of recently discovered issues with the applications of symmetric cryptography, and with the most popular AE schemes CCM [27,43] and GCM (Galois Counter Mode) [32]. The security misses in the applications constituted practically exploitable vulnerabilities [7] and for CCM and GCM, concerns were expressed about their applicability [39], the security proofs [25] or their susceptibility to serious attacks when not used correctly [16,28].

Thus CAESAR's main goal was set to "identify a portfolio of authenticated ciphers that offer advantages over AES-GCM and are suitable for widespread adoption" [6]. GCM instantiated with the AES blockcipher has been used as a reference that ought to be surpassed by the CAESAR candidates, while the name of the competition spells out the properties the candidates are expected to guarantee: security, applicability and robustness. Out of 57 submissions to the first round of CAESAR, 15 candidates still compete in the 3rd round [5]. The security claims of each of them are supported by solid cryptanalysis and/or security proofs, and are generally believed to be sound.

Table 1. An overview of 3rd round CAESAR candidates based on their *claimed* security guarantees w.r.t the nonce misuse and quantitative security; 64-bit-bound refers to about 2^{64} processed bits. For security in presence of nonce misuse, we consider MRAE [38], OAE [17] or RAE [21]. For each candidate, we consider an instance with 128-bit secret key. Deoxys II is listed twice due to its graceful degradation of security.

	Up to 64-bit-bound	Beyond 64-bit-bound
Unique nonces	OCB, NORX, Jambu, CLOC& SILC	Tiaoxin, Morus, Keyak, Ketje, Deoxys I& II, Ascon, AEGIS, ACORN
Nonce misuse	Deoxys II, COLM, AEZ	-

64-bit Bound and Nonce-Misuse. All of CAESAR candidates must accept a nonce, a secret key, AD and a message as an input. The nonce is akin to an initialization vector, and it can be assumed to have a unique value for every encryption query. The candidates are allowed to request that the nonce must not repeat in order for their security guarantees to apply. This is the case for 12 3rd round CAESAR candidates. AEZ and Deoxys guarantee no degradation of authenticity, and the minimal (and unavoidable [38]) degradation of confidentiality[1] even if the nonces are misused, i.e. repeated. COLM guarantees a weaker version of confidentiality protection in presence of nonce misuse, so called online -misuse resistance [17]. Each candidate must also specify how much data can be securely processed with a single secret key. Most CAESAR candidates guarantee security up to the so called birthday-bound; for AES-based AE schemes, this means processing no more than about 2^{64} blocks of data per key and making no

[1] As the encryption is required to be a deterministic algorithm, repeating all inputs unavoidably means repeating the ciphertexts as well.

more than 2^{64} encryption queries. In this paper, we use the 64-bit data/query complexity as a reference threshold for comparison of candidates, denoted by 64-bit-bound.

In Table 1, we categorize the 3rd round candidates, as well as CCM and GCM, based on their security claims w.r.t. the nonce misuse and quantitative security. We consider a scheme to claim security against nonce reuse if it targets MRAE [38], OAE [17] or RAE [21] security. For each candidate, we consider an instance with a 128-bit secret key.

Robustness: (In)security Beyond Guarantees. All CAESAR candidates clearly state what security properties do they guarantee as long as the conditions on the nonces or data limits are respected. However, they give little or no information on the actual impact of attacks that violate these usage conditions.

This is what we aim to determine in this work. We take the liberty to interpret robustness of AE schemes as the ability to resist powerful attacks, possibly beyond the limitations guaranteed by the designers, and analyze the security of all 15 third round CAESAR candidates against attacks with very high data complexity, and against nonce-misuse attacks. In order to make the result comparable, we consider instances using secret keys of 128 bits, and use the 64-bit-bound (i.e. the "birthday bound" of AES-GCM) as a point of reference.

An Overview. For each candidate we describe one or more attacks, unless relevant attacks already exist. We sort the CAESAR candidates into six categories based on the adversarial powers necessary to break them: **(A)** Those for which we have a nonce-respecting universal forgery *and* a decryption attack at the 64-bit-bound. **(B)** Others for which we have a nonce-respecting universal forgery *and* a decryption attack above the 64-bit-bound, but below exhaustive search. **(C)** Those for which we have a reusable forgery *and* a reusable decryption attack with small complexity, possibly with nonce-misuse. **(D)** Others for which we have a forgery *or* a decryption attack with small complexity, possibly with nonce-misuse. **(E)** Others for which we have a forgery *or* a decryption attack at the 64-bit-bound, possibly with nonce-misuse. **(F)** Remaining ones. Our results are summarized in Table 2. For each candidate, we indicate the type of attack, the query complexity[2], whether the attack needs nonce misuse, and whether it is reusable. All attacks presented in Table 2 succeed with high probability.

The categories can be ordered by a decreasing level of resilience as follows: $(\mathbf{F}) \geq (\mathbf{E}) \geq (\mathbf{D}) \geq (\mathbf{C})$ and $(\mathbf{F}) \geq (\mathbf{E}) \geq (\mathbf{B}) \geq (\mathbf{A})$. The categories $(\mathbf{A})$ and $(\mathbf{C})$ are incomparable (same for $(\mathbf{B})$ and $(\mathbf{D})$), as the impacted schemes succumb to different kinds of misuse. However, the attacks in category $(\mathbf{C})$ may be seen as a more serious threat than those in $(\mathbf{A})$, as they are much more likely in practice.

Our Contribution. Table 2 sheds more light on the actual impact of nonce-reuse/high-data attacks, and arguably provides much more information than the guarantees provided by the authors (summarized in Table 1). This can be very useful to break ties at the end of 3rd round of CAESAR competition. Some of

[2] The time and memory complexities of the attacks mentioned in the Table 2 are small multiples/small powers of the query complexity.

Table 2. A summary of attacks on 3rd round CAESAR candidates and their clustering based on the type of attack. The categories (**A**), (**B**), (**C**), (**D**), (**E**) and (**F**) are listed from top to bottom. The column "source" lists the sections and/or bibliography references that describe the relevant attacks. The comments "(N, A)", "(N)" and "(A)" in the reusability column (see Sect. 2) mean that the reusability is limited to fixed values of the listed parameters. The values in the column "nonce-reuse" indicate maximal number of times any nonce is used (so 1 means nonce respecting), q denotes the number of independent forgeries made in a single attack, and m is used as a parameter. #The attack applies only if $|N| > 128$.

	Algorithm	Source(s)	Type of attack	Nonce-reuse	# Queries	Reusable
A	**AES-GCM** [32]#	4	Univ. forgery	1	$3 \cdot 2^{64}$	Yes
	AEZ [22]	5, [12]	Key recovery	1	$3 \cdot 2^{64}$	
	OCB [30]	6, [15]	Univ. forgery & CCA decryp.	1	2 (one w/ 2^{64} blocks)	Yes
	AES-OTR [34]	3, 7	Univ. forgery & CPA decryp.	1	2 (one w/ 2^{64} blocks)	Yes
B	CLOC [24]	8	Univ. forgery & CPA decryp.	1	2^{80}	Yes
C	**AES-GCM** [32]	3, 4, [28]	Univ. forgery & CPA decryp.	2	2	Yes
	Deoxys-I [26]	3	Univ. forgery & CCA decryp.	3	3	Yes (A)
	OCB [30]	3	Univ. forgery & CCA decryp.	2	2	Yes (A)
	Tiaoxin [35]	10	Key recovery	30	30	
	AEGIS-128 [47]	11	Univ. forgery & CPA decryp.	15	15	Yes (N, A)
	ACORN-128 [44]	12	Univ. forgery & CPA decryp.	586	586	Yes (N, A)
	Ketje Sr [9]	13	Key recovery	50	50	
	MORUS 640 [45]	14	Univ. forgery & CPA decryp.	8	8	Yes (N)
D	**AES-CCM** [43]	3	CPA decryp.	2	1	
	CLOC & SILC [24]	3	CPA decryp	2	1	No
	JAMBU [46]	3	CPA decryp.	$1 + \lvert C\rvert/64$	$\lvert C\rvert/64$	No
	NORX32-4-1 [2]	3	CPA decryp.	$1 + \lvert C\rvert/384$	$\lvert C\rvert/384$	No
	Ascon-128 [14]	3	CPA decryp.	$1 + \lvert C\rvert/64$	$\lvert C\rvert/64$	No
	Lake Keyak [10]	3	CPA decryp.	$1 + \lvert C\rvert/1344$	$\lvert C\rvert/1344$	No
E	COLM [1]	3	Semi-univ. forgery	$1 + q$	2^{64}	Yes (N, A)
F	Deoxys-II [26]	9	Semi-univ. forgery & CCA decryp.	2^m	2^{128-m}	Yes (A)

these attacks can also be viewed as disturbingly powerful (e.g. low-complexity key recoveries). Taking into consideration the circumstances that led to the start of CAESAR competition, we do not think that schemes that succumb to such attacks should be recommended as CAESAR finalists (in this sense, not every candidate for CAESAR can beat Galois).

The attacks we present also shed more light on the weaknesses and strengths of different constructions. For example, many designs in cat. (**C**) use aggressively optimized state update functions which give up the key (or secret state) with the slightest nonce reuse, which we find worrisome. The collection of generic attacks in Sect. 3 is especially helpful to identify common security phenomena related to certain construction principles, such as the decryption attacks for streamciphers, or easy nonce-reusing forgeries on ciphertext-translation based schemes.

We found it interesting that the state recovery on AEGIS and Tiaoxin works thanks to the differential properties of the AES Sbox. The "E_K oracle" attack on CLOC is nonce respecting because CLOC processes the nonce in a place that is usual for the last associated data block. COLM, in turn, resists to nonce-respecting collision attacks thanks to having the nonce restricted to 64 bits. Finally, we have not seen the trade-off between the degree of nonce-reuse and the attack complexity used for Deoxys-II in the literature before.

Disclaimer and Open Problems. We understand that **none** of the attacks we present violates the security claims of any of the CAESAR candidates. That is not the goal of our work. Our goal is to determine to what degree will the security of respective candidates deteriorate *after* the guarantees become void.

We leave the investigation of security of CAESAR candidates within other adversarial models (such as related-key security, release of unverified plaintext or multi-user security) as open problems.

Related Work. The (in)security of GCM mode was treated by a number of works [20,25,36,40], in particular Joux authored the "forbidden" nonce misusing attack [28]. Collision attack similar to ours, or inspiring ours, were described for previous versions of AEZ by Fuhr et al. [19], and Chaigneau and Gilbert [12]. Collision attack on OCB were given by Ferguson [15] and Sun et al. [41]. Reusable forgery attacks on OCB, OTR and COLM were described by Forler et al. [18]. Collision-based attacks on COPA and ELmD (the predecessors of COPA) were described by Bay et al. [3] and Lu [31]. Bost and Sanders found a flaw in the masking scheme of an earlier version of OTR [11], Huang and Wu described a collision based forgery [23]. Mileva et al. describe a nonce misusing distinguisher attack for MORUS [33]. The collision-based forgeries on NORX, Ascon and Keyak are matching Lemma 2 of the work on provable generic security of full-state keyed duplex by Daemen et al. [13].

Organization of the Paper. In Sect. 2 we introduce notations, AE syntax and the attack model. In Sect. 3 we give generic attacks that apply to several schemes that share a particular structure. Then in Sects. 4 to 14, we address attacks specific to GCM and several CAESAR candidates, each in a separate section. For descriptions of CCM, GCM, and the CAESAR candidates in, we refer the reader either to the full version of this paper [42], or to the respective submission documents [5].

2 Preliminaries

When presenting the CAESAR candidates, we try to respect the original notations but deviate a bit to unify the notation of the common input/output values. Hence, the secret key is denoted by K, the nonce (or IV) is denoted by N, the associated data (AD) is denoted by A, the plaintext is denoted by M, the ciphertext is denoted by C, and the tag (if any) is denoted by T. We further use τ to denote the ciphertext expansion/stretch, which is in most cases the same as the tag length.

Notations. All strings are binary strings. We let ε denote the empty string and $|X|$ the length of a string X in bits. For two strings X, Y with $|X| = |Y|$, we let $X\&Y$ denote the bitwise AND of X and Y and $X \oplus Y$ the bitwise xor. We let $\{0,1\}^n$ denote the set of all strings of n bits, and let $\{0,1\}^* = \bigcup_{n\in\{0,1,2,\ldots\}}\{0,1\}^n$. Each of the candidates internally partitions the inputs into blocks of constant size. We use several symbols to denote the length of the blocks, e.g. n, r or ν, in order to respect the notation of each candidate as much as possible. We use subscript to index blocks in a query and superscript to index queries, e.g. M_i^j is the i^{th} message block in j^{th} query. We let $M_1, \ldots, M_\ell \xleftarrow{n} M$ denote the partitioning of a string M into blocks of n bits, except for $1 \leq |M_\ell| \leq n$, such that $\ell = \lceil |M|/n \rceil$. We let $|M|_n = \lceil |M|/n \rceil$. With a slight abuse of notation, we let $X0^*1$ denote extending a string X with the smallest number of zero bits followed by a "1" that will yield a string whose length is a multiple of a block size, when a block size is implicit from the context. We let $\mathsf{msb}_a(X)$ denote the a most significant bits of a string X, and similar applies to lsb_a. We let $\mathrm{enc}_n(a)$ denote the n-bit canonical encoding of an integer $0 \leq a \leq 255$. For blockcipher-based schemes, we let E denote the underlying blockcipher.

Syntax. A scheme for authenticated encryption (AE) Π consists of a key space $\mathcal{K} \subset \{0,1\}^*$ (for most candidates $\mathcal{K} = \{0,1\}^k$ for a positive k), and two deterministic algorithms $\mathcal{E}$ and $\mathcal{D}$. The encryption algorithm maps a key, a nonce, associated data (AD) and a message (K, N, A, M) to a ciphertext $C = \mathcal{E}(K, N, A, M)$, such that $|C| = |M| + \tau$ where the stretch is either a constant parameter, or user-selectable (only for candidate AEZ). For most candidates, the ciphertext consists of a core ciphertext and a tag, i.e. $\mathcal{E}(K, N, A, M) = C\|T$ with $|T| = \tau$. The decryption algorithm $\mathcal{D}$ that maps (K, N, A, C) (or $(K, N, A, C\|T)$) to a message M or to an error symbol $\perp$, if the authentication fails. It is required that for every valid input tuple (K, N, A, M), we have $M = \mathcal{D}(K, N, A, \mathcal{E}(K, N, A, M))$. We denote the sets of nonces, AD and messages valid for Π by $\mathcal{N}$, $\mathcal{A}$ and $\mathcal{M}$ respectively.

Attack Model. We focus on three types of attacks: decryption attacks, (semi) universal forgeries and key recovery attacks. To make the results comparable, for each candidate we attack an instance that uses 128-bit keys (i.e. $\mathcal{K} = \{0,1\}^{128}$), and we define our attacks models to correspond to the 128-bit security level.

In each type of attack on a scheme Π, an attacker $\mathscr{A}$ has blackbox oracle access to an instance of the encryption and the decryption algorithms $\mathcal{E}_K, \mathcal{D}_K$

of Π that use a secret key K unknown to $\mathscr{A}$. We call $\mathscr{A}$ *nonce respecting* if each encryption query it makes uses a distinct nonce. We say that $\mathscr{A}$ mounts a *chosen plaintext attack* (CPA) if it never makes a decryption query, otherwise we say $\mathscr{A}$ mounts a *chosen ciphertext attack* (CCA).[3] $\mathscr{A}$ is free to make any queries beyond the explicit restrictions.

For each attack, we keep track of the data complexity (in blocks of some constant size) and/or the query complexity, the maximal number (over the values of the nonce) of encryption queries made with the same nonce. We call a forgery (resp. decryption) attack *reusable* if, after having forged (resp. decrypted) for the first time, the query and computational complexity of the consequent forgeries (resp. decryptions) are significantly lower than the complexity of the initial forgery (resp. decryption).

(Semi)-universal Forgery. $\mathscr{A}^{\mathcal{E}_K,\mathcal{D}_K}(N,A,M)$ receives an a nonce, AD and a message and tries to produce a decryption query (N,A,C) that will correctly decrypt to M, such that C was not an output of a previous encryption query made with N,A. We call the forgery *semi-universal* if $\mathscr{A}$ only gets target AD and message (i.e. $\mathscr{A}^{\mathcal{E}_K,\mathcal{D}_K}(A,M)$) or target message only (i.e. $\mathscr{A}^{\mathcal{E}_K,\mathcal{D}_K}(M)$) and is allowed to use arbitrary values for the remaining inputs.

Decryption Attack. $\mathscr{A}^{\mathcal{E}_K,\mathcal{D}_K}(N,A,C)$ receives a nonce, AD and ciphertext-tuple that is an encryption of a secret random message M of fixed length $\mu \geq 128$, and tries to produce M.

Key Recovery. $\mathscr{A}^{\mathcal{E}_K,\mathcal{D}_K}()$ tries to compute K.

3 Generic Attacks

In this section, we list attacks that trivially apply to certain construction principles, rather than being construction-specific. Nevertheless, these attacks are relevant for the comparison of "robustness" of CAESAR candidates.

CPA Decryption: Streamciphers (Nonce Reuse, Constant Complexity). AE schemes that produce a core ciphertext C and a tag T such that $C = M \oplus f(K,N,|M|)$ (or $C = M \oplus f(K,N,A,|M|)$), i.e. the message is xored with a sequence of masking bits derived as a function of the nonce and the secret key (or the nonce, secret key and AD) will necessarily succumb to this attack. To decrypt $(N,A,C\|T)$, we make a single encryption query $f(K,N,A,|M|)\|T' = \mathcal{E}_K(N,A,0^{|C|})$ that reveals the key stream and compute $M = C \oplus f(K,N,A,|M|)$. This attack applies to **CCM**, **GCM**.

CPA Decryption: Self-synchronizing Streamciphers (Nonce Reuse, Tiny Complexity). The previous attack can be adapted to AE schemes that produce the core ciphertext C block by block, by xoring the current message block with masking bits dependent on the key, the nonce, AD and the previous message blocks. I.e. $M_1,\ldots,M_\ell \xleftarrow{n}$ and then $C_i =$

[3] Note that a forgery is always a CCA, due to the final decryption query.

$M_i \oplus f(K, N, A, M_1 \| \dots M_{i-1}, |M_i|)$, where the value of n depends on the scheme. To decrypt $(N, A, C\|T)$, we make $|C|_n = \lceil |C|/n \rceil$ encryption queries as follows:

1: Compute $C_1, \dots, C_\ell \xleftarrow{n} C$.
2: **for** $i \leftarrow 1$ **to** ℓ **do**
3: Query $C'\|T' \leftarrow \mathcal{E}_K(N, A, M_1\| \dots \|M_{i-1}\|0^{|C_i|})$.
4: Compute $C'_1, \dots, C'_i \xleftarrow{n} C'$ and then $M_i \leftarrow C'_i \oplus C_i$.
5: **end for**

This attack applies to **CLOC**, **SILC**, **AEGIS**, **ACORN**, **MORUS**, **Ketje**, **NORX**, **Ascon**, **Keyak** and **JAMBU**.

Semi-universal Forgery: AD Preprocessing (Nonce-Reuse, Varying Complexity). Several candidates internally process an encryption query (K, N, A, M) by first computing a value $V = f(K, N, A)$ dependent on the key, nonce and the AD, and then compute the (tagged) ciphertext as a function of the secret key, the message and the value V as $C = g(K, V, M)$, such that $|V| = v$ for constant v. If $|\mathcal{N}| \geq 2^{v/2}$, then it is possible to find a pair $(N_1, A_1), (N_2, A_2)$ such that $f(K, N_1, A_1) = f(K, N_2, A_2)$ in a nonce-respecting birthday attack, and then use it to forge for M (hence semi-universal forgery):

1: Initialize empty table T, pick arbitrary $\hat{M} \in \{0,1\}^v$.
2: **for** $i \leftarrow 1$ **to** $2^{v/2}$ **do**
3: Pick (N', A') with a fresh N' randomly.
4: Query $C' \leftarrow \mathcal{E}_K(N', A', \hat{M})$, then insert $(C', (N', A'))$ to T.
5: **end for**
6: Find entries $(C', (N_1, A_1))$, $(C', (N_2, A_2))$ (with collision on C') in T.
7: Query $C \leftarrow \mathcal{E}_K(N_1, A_1, M)$ and forge with (N_2, A_2, C).

The attack succeeds with a probability close to $1/2$, in particular choosing $\hat{M} \in \{0,1\}^{2v}$ ensures that a C' collision implies a V collision with overwhelming probability (thanks to the ciphertext expansion). It is reusable with the same $(N_1, A_1), (N_2, A_2)$, and uses every nonce no more than $1 + q$ times, with q the number of desired forgeries.

The attack applies with 64-bit-bound complexity (as $v = 128$) to, **AEZ**, **CLOC**, **SILC**, **COLM** and with some care to **CCM**.[4] This attack applies with complexity above 64-bit-bound (as $v = 192$) to **JAMBU**.

Semi-universal Forgery: Sponges (Nonce Reuse, Varying Complexity). In sponge-based modes, the processing can again be expressed with two functions f and g but nonce reuse allows the attacker to force arbitrary values to the outer r bits of the sponge state after processing the first message block. Using this, the previous attack can be adapted to work with complexity $2^{c/2}$ (where c is the capacity of the sponge-based scheme) to forge for arbitrary (A, M):

1: Initialize empty tables T, pick arbitrary $\hat{M} \in \{0,1\}^c$.
2: **for** $i \leftarrow 1$ **to** $2^{c/2}$ **do**
3: Pick a fresh N' randomly.

[4] With $\tau = 128$, we must use A' of 240 bits to make sure that the encoding of the nonce and AD for the CBC MAC is block-aligned.

4: Query $C'\|T' \leftarrow \mathcal{E}_K(N', A, 0^r)$, then query $C''\|T'' \leftarrow \mathcal{E}_(N', A, C'\|\hat{M})$.
5: Compute $C''_1, \ldots, C''_\ell \xleftarrow{r} C''$, then insert $(C''_2\| \ldots \|C''_\ell\|T'', N')$ to T.
6: **end for**
7: Find entries $(C''\|T'', N_1)$, $(C''\|T'', N_2)$ (with collision on $C''\|T''$) in T.
8: Query $C\|T \leftarrow \mathcal{E}_K(N_1, A, M)$ and forge with $(N_2, A, C\|T)$.

The success probability is close to 1/2. The second query in the attacks forces the internal state of the sponge to become $0^r\|S$ for some $S \in \{0,1\}^c$, hence the birthday complexity in c. The attack is reusable with the same $(N_1, A), (N_2, A)$,[5] and uses every nonce no more than $2+q$ times, with q the number of desired forgeries. The attack applies with 64-bit-bound complexity (as $c = 128$) to **NORX** and with above-64-bit-bound complexity (as $c = 256$) to **Keyak** and **Ascon**. We note that for Keyak and Ascon, the exhaustive key search has the same time complexity as this attack, but needs only a single query.

Universal Forgery and CCA Decryption: Ciphertext Translation (Nonce Misuse, Tiny Complexity). Some candidates use so called *ciphertext translation* [37] to incorporate the authentication of AD with a message-only encryption core $\bar{\mathcal{E}}$. These schemes compute the tagged ciphertext as $\mathcal{E}_K(N, A, M) = \bar{\mathcal{E}}_K(N, M) \oplus 0^{|M|}\|H_K(A)$ where $\bar{\mathcal{E}}_K(N, M)$ returns a core-ciphertext and a τ-bit tag and H is an AXU hash with τ-bit output. To forge for (N, A, M), we pick arbitrary $\hat{N} \neq N$, $\hat{M} \neq M$ and $A' \neq A$ and we do:

1: Query $C^1\|T^1 \leftarrow \mathcal{E}_K(\hat{N}, A, \hat{M})$ and $C^2\|T^2 \leftarrow \mathcal{E}_K(\hat{N}, A', \hat{M})$.
2: Compute $\Delta \leftarrow T^1 \oplus T^2$.
3: Query $C'\|T' \leftarrow \mathcal{E}_K(N, A', M)$ and forge with $(N, A, C'\|(T' \oplus \Delta))$.

It is easily verified that the forgery is correct. This attack can be modified to decrypt a ciphertext $N, A, C\|T$; knowing Δ, we query $N, A', C\|(T \oplus \Delta)$ and learn the message M. This attack applies to **OCB**, **AES-OTR** and **Deoxys-I**.

4 AES-GCM

Universal Forgery (Nonce Misuse, Tiny Complexity). This attack has been first described by Joux as the "forbidden attack" [28]. The main idea is that recovering the derived key L makes forging very easy. We assume that $\tau = 128$. To forge for N, A, M, we pick random $\hat{N}$ and $M^1 \neq M^2 \in \{0,1\}^{128}$ and do:

1: Query $C^1\|T^1 \leftarrow \mathcal{E}_K(N, \varepsilon, M^1)$ and $C^2\|T^2 \leftarrow \mathcal{E}_K(N, \varepsilon, M^2)$.
2: Compute L as root of $P(\Lambda) = (C^1_1 \oplus C^2_1) \cdot \Lambda^2 \oplus (T^1 \oplus T^2)$ over $\mathsf{GF}(2^{128})$.
3: Query $C'\|T' \leftarrow \mathcal{E}_K(N, A', M')$ with arbitrary A' and M' s.t. $|M'| = |M|$.
4: Forge with $(N, A, (C' \oplus M' \oplus M)\|(T' \oplus \mathrm{GHASH}_L(A', C') \oplus \mathrm{GHASH}_L(A, C)))$.

We note that L will be the only root of $P(\Lambda)$ as squaring yields a bijection over $\mathsf{GF}(2^{128})$. Once L is computed, forgeries become easy.

Universal Forgery (Nonce Respecting, 64-bit-Bound, $|N| > 128$). If nonces longer than 128 bits are allowed, it is possible to recover L in a nonce-respecting birthday attack. We note, however, that the use of nonce length other

[5] For Keyak, the attack attack can be reused with arbitrary AD, because AD and message are being processed simultaneously.

than 96 bits is uncommon and discouraged [25]. Assuming that $\tau = 128$, for each i we use distinct N^i of 256 bits and $M^i = B \| M_2^i$ for a fixed $B \in \{0,1\}^{128}$ and distinct $M_2^i \in \{0,1\}^{128}$, and do:

1: **for** $i \leftarrow 1$ **to** 2^{64} **do** query $C^i \| T^i \leftarrow \mathcal{E}_K(N^i, \varepsilon, M^i)$.
2: For $i \neq j$ s.t. $C_1^i = C_1^j$ find L as root of $P(\Lambda) = (C_2^i \oplus C_2^j) \cdot \Lambda^2 \oplus (T^1 \oplus T^2)$.
3: Forge using L.

Note that the collision in line 2 must imply $\mathrm{GHASH}_L(\varepsilon, N^i) = \mathrm{GHASH}_L(\varepsilon, N^{i'})$, so if it occurs, the attack succeeds. We note that a forgery allows to mount a CCA decryption attack (by changing AD).

5 AEZ v5

We present three nonce-respecting attacks that respectively recover the subkeys I, J and L, each at the 64-bit-bound complexity.

***J*-Recovery Attack.** The Chaigneau-Gilbert attack [12] on AEZ v4.1 can be applied to AEZ v5 to extract J by a nonce-respecting chosen message attack at the birthday bound. When N and A are single blocks, then based on the AEZ v5 specification [22] H becomes

$$\begin{aligned} h_k(\tau, N, A) &= E_K^{3,1}(\tau) \oplus E_K^{4,1}(N) \oplus E_K^{5,1}(A) \\ &= E_K^{3,1}(\tau) \oplus \mathsf{AES4}_k(N \oplus 4J \oplus 2I \oplus L) \oplus \mathsf{AES4}_k(A \oplus 5J \oplus 2I \oplus L). \end{aligned}$$

If we limit ourselves to queries with $A = N \oplus c$ for a fixed block c and variable nonces, a ciphertext collision with the pair (N, N') will mean that $N' = N \oplus c \oplus J$. The attack runs as follows:

1: Initialize an empty table T.
2: Pick an arbitrary block $c \in \{0,1\}^{128}$ and message $M \in \{0,1\}^{2 \cdot 128}$.
3: **for** $i \leftarrow 1$ **to** 2^{64} **do**
4: Pick a fresh N randomly, set $A \leftarrow N \oplus c$.
5: Query $C \leftarrow \mathcal{E}_K(N, A, \tau, M)$, store (C, N) in T.
6: **end for**
7: Find $(C, N), (C', N')$ in T with $C = C'$, compute $J = N \oplus N' \oplus c$.

The Chaigneau-Gilbert attack requires a little effort to be adapted to AEZ v5 but it can recover I and L with nonce-misuse. A nonce respecting recovery of I and L is possible if we can use nonces of several blocks (a feature of AEZ [22]), to have a similar attack as the one above.

***L*-Recovery Attack.** If $|N|_{128} = 2$ and $A = \varepsilon$, then following the AEZ v5 specifications H becomes

$$\begin{aligned} h_k(\tau, (N_1, N_2)) &= E_K^{3,1}(\tau) \oplus E_K^{4,1}(N_1) \oplus E_K^{4,2}(N_2) \\ &= E_K^{3,1}(\tau) \oplus \mathsf{AES4}_k(N_1 \oplus 4J \oplus 2I \oplus L) \oplus \mathsf{AES4}_k(N_2 \oplus 4J \oplus 2I \oplus 2L). \end{aligned}$$

We modify the J-recovery attack to use 2-block nonces with $N_2 = N_1 \oplus c$ for a fixed block c. A ciphertext collision with N and N' will then

***I*-Recovery Attack.** Next, we see that when $|N|_{128} = 9$, the hash function H becomes

$$\begin{aligned} h_k(\tau, (N_1, \ldots, N_9)) &= E_K^{3,1}(\tau) \oplus E_K^{4,1}(N_1) \oplus \cdots \oplus E_K^{4,9}(N_9) \\ &= E_K^{3,1}(\tau) \oplus \mathsf{AES4}_k(N_1 \oplus 4J \oplus 2I \oplus L) \oplus \cdots \oplus \\ &\quad \mathsf{AES4}_k(N_7 \oplus 4J \oplus 2I \oplus 7L) \oplus \mathsf{AES4}_k(N_8 \oplus 4J \oplus 2I) \oplus \\ &\quad \mathsf{AES4}_k(N_9 \oplus 4J \oplus 4I \oplus L). \end{aligned}$$

We again modify the J-recovery attack to use 9-block nonces with $N_2, \ldots, N_8$ constant and $N_9 = N_1 \oplus c$ for a fixed block c. A ciphertext collision with N and N' yields $6I = N_1 \oplus N_1' \oplus c$. So, we recover I, J, L with a nonce-respecting chosen message attack 64-bit-bound.

6 OCB3 (OCB v1.1)

***L*-Recovery Attack.** An attack by Ferguson [15] allows to recover the derived key L at 64-bit-bound using a single huge query. In the nonce-misuse setting, we can make many queries with empty message and two-block AD:

1: **for** $i \leftarrow 1$ **to** 2^{64} **do** query $T^i \leftarrow \mathcal{E}_K(N, A^i \| A^i, \varepsilon)$ with fresh $A^i \in \{0,1\}^{128}$.
2: Find $i \neq j$ with $T^i = T^j$, compute $L = (A^i \oplus A_j) \cdot (\gamma_1 \oplus \gamma_2)^{-1}$.

If tag collision occurs, we must have $A_1^i = A_1^j \oplus (\gamma_1 \oplus \gamma_2) \cdot L$. We need to reuse the nonce 2^{64} times.

Universal Forgery (Tiny Complexity, Using *L*). Using L, we can make a universal forgery for (N, A, M'). If $|M'|_{128} = \ell > 1$, we do:

1: Define a permutation $\pi : \{1, \ldots, \ell\} \to \{1, \ldots, \ell\}$ as $\pi(i) = (i + 1 \bmod \ell) + 1$.
2: **for** $i \leftarrow 1$ **to** ℓ **do** $M_i \leftarrow M'_{\pi(i)} \oplus \gamma_i \cdot L \oplus \gamma_{\pi(i)} \cdot L$.
3: Query $C \| T \leftarrow \mathcal{E}_K(N, A, M)$.
4: **for** $i \leftarrow 1$ **to** ℓ **do** $C'_i = C_{\pi^{-1}(i)} \oplus (\gamma_i \oplus \gamma_{\pi^{-1}(i)}) \cdot L$.
5: Forge with $(N, A, C' \| T)$.

If $|M'|_{128} = 1$, we construct $M = M' \| (\gamma_1 \oplus \gamma_2) \cdot L$, make a query with (N, A, M) to get $C \| T$, and take $C' = C_1$, which again gives a valid encryption $C' \| T$ of (N, A, M').

E_K Oracle (Tiny Complexity, Using *L*). We can also implement an E_K oracle. To compute $y_i = E_K(x_i)$ for arbitrary $x_1, \ldots, x_s \in \{0,1\}^{128}$ set $\ell = 2^{14}$, and do:

1: Pick $M \in \{0,1\}^{\ell \cdot 128}$ with $\bigoplus_{i>1} M_i = (2^{-1} \oplus \gamma_1 \oplus \gamma_\ell) \cdot L$ randomly.
2: Query $C \| T \leftarrow \mathcal{E}_K(N, \varepsilon, M)$, compute $R \leftarrow C_1 \oplus T \oplus \gamma_1 \cdot L$.
3: Find i s.t. $M_i \oplus R \oplus \gamma_i \cdot L = 0^7 \| 1 \| N'' \| 0^6$ for $N' \in \{0,1\}^{114}$.
4: Set $N' \leftarrow N'' \| 0^6$, compute $R' = C_i \oplus R \oplus \gamma_i \cdot L$.
5: **for** $i \leftarrow 1$ **to** s **do** set $M'_i \leftarrow x_i \oplus R' \oplus \gamma_i \cdot L$.
6: Query $C' \| T' \leftarrow \mathcal{E}_K(N', \varepsilon, M')$.
7: **for** $i \leftarrow 1$ **to** s **do** compute $y_i \leftarrow C'_i \oplus R' \oplus \gamma_i \cdot L$.

The R computed on line 2 is correct as $T = E_K(M_1 \oplus R \oplus \gamma_1 \cdot L) = C_1 \oplus R \oplus \gamma_1 \cdot L$. We can also add an unused nonce to the list of x_i-s to avoid making the $2^{14} \cdot$

128bit= 256 KB query more than once. Then the attack uses a single encryption query per list of blocks $x_1, \ldots, x_s$, of size $s+1$ blocks.

CCA Decryption Attack For Messages Of Odd Length (Tiny Complexity, Using L). Assume that we want to decrypt (N, A, C, T) (let M be its decryption). We can first compute R associated with N with the above E_K oracle, as well as some fresh N' and its associated R' with tiny complexity. The message M' defined by $M'_i = M_i \oplus R \oplus R'$ encrypts into (C', T') such that $C'_i = C_i \oplus R \oplus R'$ and $T' = T$ when ℓ is odd. So, a CCA decryption query with (N', A, C', T) gives M' from which we deduce M.

7 AES-OTR v3.1

***L*-Recovery Attack.** If we use the same nonce N 2^{64} times, we can recover L:

1: **for** $i \leftarrow 1$ **to** 2^{64} **do** query $C\|T \leftarrow \mathcal{E}_K(N, \varepsilon, M^i)$ with fresh $M^i \in \{0,1\}^{4\cdot 128}$.
2: Find $i \neq j$ s.t. $C_1^i \oplus M_2^i = C_3^j \oplus M_4^j$, compute $L = (M_1^i \oplus M_3^j) \cdot (1 \oplus 2)^{-1}$.

In a nonce respecting attack, we can encrypt a huge random message (with $|M|_{128} \approx 2^{64}$) with a nonce N and look for an internal collision with $i \neq j$

$$C_{2i} \oplus M_{2i-1} = C_{2j} \oplus M_{2j-1} \text{ implying } C_{2i-1} \oplus 2^{i-1} \cdot 2 \cdot L = C_{2j-1} \oplus 2^{j-1} \cdot 2 \cdot L,$$

revealing L for this N. We further expect to find many values of $1 \leq i \leq |M|_{128}/2$ for which $2^{i-1} \cdot L \oplus M_{2i-1}$ (or $2^{i-1} \cdot 3 \cdot L \oplus C_{2i-1}$) will be a string of the form $\epsilon(\tau)\|1\|N'$. For any such N' we can use $L' = C_{2i-1}$ (or $L' = C_{2i}$) to bootstrap the following attack.

E_K Oracle (Using (N, L) Pair). Assuming that we know an (N, L) pair $E_K(x_1), \ldots, E_K(x_r)$ for a list $x_1, \ldots, x_r$ as follows:

1: **for** $i \leftarrow 1$ **to** r **do** set $M_{2i-1} \leftarrow x_i \oplus 2^{2i-1} \cdot L$ and pick M_i arbitrarily.
2: Query $C\|T \leftarrow \mathcal{E}_K(N, \varepsilon, M)$.
3: **for** $i \leftarrow 1$ **to** r **do** compute $E_K(x_i) = M_{2i} \oplus C_{2i-1}$.

In each execution of this attack, we can add one block to the list of x_i-s to prepare a fresh pair N', L' for the next execution of the attack, allowing for its nonce respecting repetition.

8 CLOC

E_K Oracle in CLOC (Nonce-Respecting, Above 64-bit-Bound). In CLOC, the processing of AD and nonce has the form $V = f_1(f_2(K, A) \oplus \mathsf{ozp}(\mathsf{param}\|N))$ where the function f_1 is easy to invert. To compute $E_K(x)$ for an $x \in \{0,1\}^{128}$, we pick fixed AD A and do:

1: **for** $i \leftarrow 1$ **to** 2^{64} **do** query $C^i\|T^i \leftarrow \mathcal{E}_K(N^i, A, M^i)$ with random $M^i \in \{0,1\}^{2\cdot 128}$.
2: Find $i \neq j$ s.t. $M_1^i \oplus C_1^i = M_2^j \oplus C_2^j$, compute $W \leftarrow f_1^{-1}(\mathsf{fix1}(C_1^j)) \oplus \mathsf{ozp}(\mathsf{param}\|N^i)$.
3: **if** $f_1^{-1}(x) \oplus W$ of the form $\mathsf{ozp}(\mathsf{param}\|\bar{N})$ query $E_K(x)\|T \leftarrow \mathcal{E}_K(\bar{N}, A, 0^{128})$.
4: **else** abort.

The attack works as the collision on line 2 implies that $V^i = \mathsf{fix1}(C_1^j)$ so we deduce the V^i value for a random nonce N^i with A. This allows us to recover $W = f_2(K, A)$. If x is not of the correct form, it is bad luck. When using nonces of 112 bits, which is the maximum, the probability to have the correct form is 2^{-16}. But we can run this attack 2^{16} times to get many $W^i = f_2(K, A^i)$ with complexity 2^{80}. Then at least one is W^i will be such that $f_1^{-1}(x) \oplus W_i$ is of the correct format for any x.

This attack does not work on SILC, in which W depends on both N and A.

Universal Forgery and CPA Decryption Attack in CLOC (Nonce-Respecting, Above 64-bit-Bound). With the previous E_K oracle, we can simulate the encryption or the decryption process and thus mount universal forgeries and CPA decryption.

9 Deoxys v1.41

Semi-universal Forgery, CCA Decryption Attack: Deoxys-II (Reusable, Nonce-Misuse). The encryption algorithm of Deoxys-II can be expressed as $\mathcal{E}_K(N, A, M) = \bar{\mathcal{E}}(K, N, f_2(f_1(K, A), M), M)$ where $\bar{\mathcal{E}}$ produces a (stretched) ciphertext and f_1 and f_2 are keyed functions with constant-size output. The attacks are based on finding a collision on f_1. Assuming each nonce can be used up to 2^m times, to forge for (N, M) we use $N^1, \ldots, N^{2^{128-2m}} \neq N$ all distinct and $M' \neq M$ of 2 blocks, and do:

1: **for** $i \leftarrow 1$ **to** 2^{128-2m} **do**
2: **for** $j \leftarrow 1$ **to** 2^m **do** query $C^{i,j} \| T^{i,j} \leftarrow \mathcal{E}_K(N^i, A^{i,j}, M')$ with random $A^{i,j}$.
3: **end for**
4: Find $i, j \neq j'$ s.t. $A^{i,j} \neq A^{i,j'}$ and $T^{i,j} = T^{i,j'}$.
5: Query $C \| T \leftarrow \mathcal{E}_K(N, A^{i,j}, M)$ and forge with $(N, A^{i,j'}, C \| T)$.

We can modify this attack to decrypt $(N, A^{i,j}, C \| T)$ by making a CCA decryption query on $(N, A^{i,j'}, C, T)$. This can only decrypt messages using $A^{i,j}$ as associated data. The total complexity of the attack is 2^{128-m} queries. Note that if $m = 64$, the complexity becomes birthday bounded.

10 Tiaoxin-346

Nonce-Misuse Key Recovery. [6] We pick $M, \bar{M}, \tilde{M} \in \{0,1\}^{4 \cdot 128}$ such that $M_i \oplus \bar{M}_i = \Delta$ and $M_i \oplus \tilde{M}_i = \tilde{\Delta}$ for $i = 0, 1, 2, 3$ and $\Delta \neq \tilde{\Delta}$. We pick arbitrary N and A and recover two 128 bit words $T'[4]_0$ and $T'[3]_0$ of the internal state right after processing of N, A and the first two blocks of M by:

1: Query $C \| T \leftarrow \mathcal{E}_K(N, A, M)$, $\bar{C} \| \bar{T} \leftarrow \mathcal{E}_K(N, A, \bar{M})$ and $\tilde{C} \| \tilde{T} \leftarrow \mathcal{E}_K(N, A, \tilde{M})$.
2: **for** $i \leftarrow 2, 3$ **do** set $\gamma_i \leftarrow \mathsf{ShiftRows}^{-1}(\mathsf{MixColumns}^{-1}(\bar{C}_i \oplus C_i))$.
3: **for** $i \leftarrow 2, 3$ **do** set $\tilde{\gamma}_i \leftarrow \mathsf{ShiftRows}^{-1}(\mathsf{MixColumns}^{-1}(\tilde{C}_i \oplus C_i))$.

[6] Note that we change the meaning of subscript and square brackets compared to the original Tiaoxin description [35].

4: **for** byte index $j \leftarrow 0$ **to** 15 **do**
5: **for** $i \leftarrow 2, 3$ **do** Find $X_{i,j} = \{\gamma_{i,j} \mid \gamma_{i,j} = \mathsf{SubBytes}(x) \oplus \mathsf{SubBytes}(x \oplus \Delta)\}$.
6: **for** $i \leftarrow 2, 3$ **do** Find $\tilde{X}_{i,j} = \{\tilde{\gamma}_{i,j} \mid \tilde{\gamma}_{i,j} = \mathsf{SubBytes}(x) \oplus \mathsf{SubBytes}(x \oplus \tilde{\Delta})\}$.
7: Set $T'[4]_{0,j} \leftarrow X_{2,j} \cap \tilde{X}_{2,j}$ and $T'[3]_{0,j} \leftarrow X_{3,j} \cap \tilde{X}_{3,j}$.
8: **end for**

The above works, as we can verify that in the encryption of M we have

1. $T'[3] = R(T[3], M_0)$,
2. $T'[4] = R(T[4], M_1)$,
3. $T'[6] = R(T[6], M_0 \oplus M_1)$,
4. $C_0 = T'[3]_0 \oplus T'[3]_2 \oplus T'[4]_1 \oplus (T'[6]_3 \& T'[4]_3)$,
5. $C_1 = T'[6]_0 \oplus T'[4]_2 \oplus T'[3]_1 \oplus (T'[6]_5 \& T'[3]_2)$,
6. $T''[3] = R(T'[3], M_2)$,
7. $T''[4] = R(T'[4], M_3)$,
8. $T''[6] = R(T'[6], M_2 \oplus M_3)$,
9. $C_2 = T''[3]_0 \oplus T''[3]_2 \oplus T''[4]_1 \oplus (T''[6]_3 \& T''[4]_3)$,
10. $C_3 = T''[6]_0 \oplus T''[4]_2 \oplus T''[3]_1 \oplus (T''[6]_5 \& T''[3]_2)$.

In the encryption of $\bar{M}$ we have the following (and similar for $\tilde{M}$ and $\tilde{\Delta}$)

1. $\bar{T}'[3] = R(T[3], M_0 \oplus \Delta)$,
2. $\bar{T}'[4] = R(T[4], M_1 \oplus \Delta)$,
3. $T'[6] = R(T[6], M_0 \oplus M_1)$,
4. $\bar{C}_0 = \bar{T}'[3]_0 \oplus \bar{T}'[3]_2 \oplus \bar{T}'[4]_1 \oplus (T'[6]_3 \& \bar{T}'[4]_3)$,
5. $\bar{C}_1 = T'[6]_0 \oplus \bar{T}'[4]_2 \oplus \bar{T}'[3]_{10} \oplus (T'[6]_5 \& \bar{T}'[3]_2)$,
6. $\bar{T}''[3] = R(\bar{T}'[3], M_2 \oplus \Delta)$,
7. $\bar{T}''[4] = R(\bar{T}'[4], M_3 \oplus \Delta)$,
8. $T''[6] = R(T'[6], M_2 \oplus M_3)$,
9. $\bar{C}_2 = \bar{T}''[3]_0 \oplus \bar{T}''[3]_2 \oplus \bar{T}''[4]_1 \oplus (T''[6]_3 \& \bar{T}''[4]_3)$,
10. $\bar{C}_3 = T''[6]_0 \oplus \bar{T}''[4]_2 \oplus \bar{T}''[3]_1 \oplus (T''[6]_5 \& \bar{T}''[3]_2)$.

We can easily see that

$$\bar{T}'[3] \oplus T'[3] = (\Delta, 0, 0) \text{ and } \bar{T}''[3] \oplus T''[3] = (0, A(T'[3]_0) \oplus A(T'[3]_0 \oplus \Delta), 0),$$

$$\bar{T}'[4] \oplus T'[4] = (\Delta, 0, 0, 0) \text{ and } \bar{T}''[4] \oplus T''[4] = (0, A(T'[4]_0) \oplus A(T'[4]_0 \oplus \Delta), 0, 0).$$

It follows that the differences of ciphertext blocks used in the lines 5 and 6 are a result of a differential equation for a single round of AES. This can be reduced to a collection of 16 differential equations for AES Sbox, allowing to recover the parts of the secret state as intersections of solutions found in the said lines (we can check that we always have $|S_{i,j} \cap \tilde{S}_{i,j}| = 1$).

We can then repeat this process with longer messages to obtain $T[3]$ and $T[4]$ and we recover $T'[4]$ and $T'[3]$ with 12 queries (3 queries per 128-bit word of $T[4]$). The state $T[6]$ follows in a similar method using 18 queries. Once the state $(T[3], T[4], T[6])$ is recovered, we invert the initialization and obtain K.

11 AEGIS v1.1

Universal Forgery, Decryption Attack (Tiny Complexity, Nonce-Misuse). To forge for (N, A, M) or to decrypt (N, A, C, T), we only need to

recover the secret state S after processing A with nonce N, the rest of encryption/decryption can then be reconstructed.

We pick three messages $M', \bar{M}, \tilde{M} \in \{0,1\}^{3\cdot 128}$ with the same criteria as for Tiaoxin (with $\Delta \neq \tilde{\Delta}$). To recover a part S'_0 of the state A' right after processing M'_1 with N and A, we:

1: Query $C'\|T' \leftarrow \mathcal{E}_K(N, A, M')$, $\bar{C}\|\bar{T} \leftarrow \mathcal{E}_K(N, A, \bar{M})$ and $\tilde{C}\|\tilde{T} \leftarrow \mathcal{E}_K(N, A, \tilde{M})$.
2: Set $\gamma \leftarrow \mathsf{ShiftRows}^{-1}(\mathsf{MixColumns}^{-1}(\bar{C}_3 \oplus \bar{M}_3 \oplus C'_3 \oplus M'_3))$.
3: Set $\tilde{\gamma} \leftarrow \mathsf{ShiftRows}^{-1}(\mathsf{MixColumns}^{-1}(\tilde{C}_3 \oplus \tilde{M}_3 \oplus C'_3 \oplus M'_3))$.
4: Recover bytes of S'_0 using $\gamma, \tilde{\gamma}, \Delta, \tilde{\Delta}$ in differential equations as with Tiaoxin.

The attack works because the difference $(C'_3 \oplus M'_3) \oplus (\bar{C}_3 \oplus \bar{M}_3)$ (associated to $M'_1 \neq \bar{M}_1$) is equal to the difference $R(R(S_4) \oplus S_0 \oplus M'_1) \oplus R(R(S_4) \oplus S_0 \oplus \bar{M}_1)$ (where $R(S_4) \oplus S_0 = S'_0$), with R just a single AES round. We can repeat this strategy to recover the remaining four 128-bit words of $S'_1, \ldots, S'_4$ with 3 queries each. Then we can recover S, having done 15 nonce reusing queries. The possibility of a low-complexity nonce reusing attack is mentioned in the AEGIS v1.1 specifications [47].

12 ACORN v3

Universal Forgery, Decryption Attack (Tiny Complexity, Nonce-Misuse). To forge the encryption of (N, A, M) or to decrypt (N, A, C, T), we only need to recover the internal state S_o after processing N, A, which allows to finish the rest of encryption/decryption. We sketch the main idea of the attack.

We make two encryption queries $C^1\|T^1 \leftarrow \mathcal{E}_K(N, A, 0\|B)$ and $C^2\|T^2 \leftarrow \mathcal{E}_K(N, A, 1\|B)$ for any $B \in \{0,1\}^{58}$. We can see that ks^j_{i+o} is constant for $j = 1, 2$ and $i = 0, \ldots, 57$ and that $\mathsf{ks}^1_{58+o} \oplus \mathsf{ks}^2_{58+o} = S_{58+o,61} \oplus S_{58+o,193}$, which is a linear equation in the bits of S_o. We recover 292 more equations by making 292 pairs of (longer) queries that differ only in a single bit, and solve the system for S_o. The knowledge of S_o allows arbitrary forgeries and decryptions with N, A.

13 Ketje

Key Recovery (Tiny Complexity, Nonce-Misusing). The authors of Ketje themselves point at the possibility of this attack. Because Ketje uses only a single round of the Keccak$-f$ function [9], the diffusion between two consecutive sponge states is low. In addition, the algebraic degree of a single round of Keccak$-f$ is only 2. We use this to recover the internal state S after processing of N and A, and then the secret key K by inverting the processing of N, A. We sketch the main idea of the attack.

We make queries $C^i\|T^i \leftarrow \mathcal{E}_K(N, A, M^i)$ with some fixed (N, A) and $M^i \in \{0,1\}^{2\cdot(r-4)}$ s.t. $M^i_2 = 0^r$ for $i = 1, \ldots, \theta$. For each i we can use M^i_1 and C^i_2 to derive degree-2 polynomial equations with the bits in the inner (capacity) part of S as unknowns. Each bit in C^i_2 depends on 31 bits of the previous state on average [8], so we expect an overwhelming majority of the bits of the attacked

state to be covered by the derived equations. We need the number of nonce misusing queries θ to be a small multiple of $\frac{b-r+4}{r-4} = 11,5$ in order to fully determine the system. Moreover, no more than a single unique monomial of degree 2 per every bit of the state appears in the system, so with $\theta = 60$, we should be able to linearize the system and solve it for S.

14 Morus

Nonce-Misuse Universal Forgery and CPA Decryption. If we recover the state S right after the initialization with N, we can forge ciphertexts with this N and decrypt any ciphertext using this N. We sketch the S recovery attack.

We first recover S_2 and S_3 by querying $C^i \| T^i \leftarrow \mathcal{E}_K(N, \varepsilon, M^i)$ with $M^i \in \{0,1\}^{256}$ for $i = 1, \ldots, 4$. Letting $\delta_i = M_0^1 \oplus M_0^i$ with $i \neq 1$, we have that

$$\begin{aligned}(C^1 \oplus M^1) \oplus (C^i \oplus M^i) = &(\mathsf{Rotl}(\delta_i, b_1) \lll (w_3 + 96)) \oplus S_2 \& \mathsf{Rotl}(\delta_i \oplus \mathsf{Rotl}(\delta_i, b_1), b_3) \\ &\oplus S_3 \& (\mathsf{Rotl}(\delta_i, b_2) \lll w_4) \\ &\oplus (\mathsf{Rotl}(\delta_i, b_2) \lll w_4) \& \mathsf{Rotl}(\delta_i \oplus \mathsf{Rotl}(\delta_i, b_1), b_3),\end{aligned}$$

where Rotl is a linear function, $\lll$ denotes a circular rotation, and all b_r-s and w_t-s are constants. Each δ_i provides 128 linear equations in 256 binary unknowns, so with $\delta_1, \delta_2, \delta_3$, we are able to recover the values of S_2 and S_3 with high probability. Once S_2 and S_3 are known, $C_1^1 \oplus M_1^1$ can be expressed as a linear function of S_0 and S_1 and we learn their xor-difference.

We still need to recover S_0, S_1, S_4, i.e. 384 bits, and have 128 linear equations (so 256 unknown bits). We query $\bar{C}^j \| \bar{T}^j \leftarrow \mathcal{E}_K(N, \varepsilon, \bar{M}^j)$ with $\bar{M}^j = M_0^1 \| \bar{M}_1^j \| 0^{128}$ and $\bar{M}_1^j \in \{0,1\}^{128}$ for $j = 1, \ldots, \theta$. Each $\bar{C}_2^j$ will supply 128 polynomial equations in S_0, S_1, S_4 of degree at most 3. By examining the $\mathsf{StateUpdate}$ and the keystream generation functions of Morus, we verify that there will be no more than $19 \cdot 128$ unique monomials of degree higher than 1 present in all equations in the worst case and only $9.25 \cdot 128$ on average. Thus by taking $\theta = 16$, we should be able to linearise the system and recover S_0, S_1 and S_4 with high probability, using 20 queries for the entire attack.

Acknowledgements. We would like to thank all CAESAR designers who provided us with their feedback. We would like to thank the Ascon team for pointing out that generic attacks with the same time but much lower data complexity than our forgery exist, and the Deoxys team for suggesting a better way to measure adversarial resources for nonce misuse. We would also like to thank the attendants of the Dagstuhl seminar 2018, and the anonymous reviewers for constructive comments.

References

1. Andreeva, E., Bogdanov, A., Datta, N., Luykx, A., Mennink, B., Nandi, M., Tischhauser, E., Yasuda, K.: COLM v1 (2016). https://competitions.cr.yp.to/round3/colmv1.pdf
2. Aumasson, J., Jovanovic, P., Neves, S.: NORX v3.0 (2016). https://competitions.cr.yp.to/round3/norxv30.pdf

3. Bay, A., Ersoy, O., Karakoç, F.: Universal forgery and key recovery attacks on ELmD authenticated encryption algorithm. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 354–368. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_13
4. Bellare, M., Rogaway, P.: Encode-then-encipher encryption: how to exploit nonces or redundancy in plaintexts for efficient cryptography. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 317–330. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44448-3_24
5. Bernstein, D.J.: Cryptographic competitions: CAESAR submissions. http://competitions.cr.yp.to/caesar-submissions.html
6. Bernstein, D.J.: Cryptographic competitions: CAESAR (2014). https://competitions.cr.yp.to/caesar-call.html
7. Bernstein, D.J.: Cryptographic competitions: disasters (2014)
8. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak sponge function family main document. Submission NIST (Round 2) **3**(30) (2009)
9. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., Keer, R.V.: CAESAR submission: Ketje v2 (2016). https://competitions.cr.yp.to/round3/ketjev2.pdf
10. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., Keer, R.V.: CAESAR submission: Keyak v2 (2016). https://competitions.cr.yp.to/round3/keyakv22.pdf
11. Bost, R., Sanders, O.: Trick or tweak: on the (in)security of OTR's tweaks. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 333–353. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_12
12. Chaigneau, C., Gilbert, H.: Is AEZ v4.1 sufficiently resilient against key-recovery attacks? IACR Trans. Symmetric Cryptol. **2016**(1), 114–133 (2016). https://doi.org/10.13154/tosc.v2016.i1.114-133
13. Daemen, J., Mennink, B., Van Assche, G.: Full-state keyed duplex with built-in multi-user support. IACR Cryptology ePrint Archive 2017/498 (2017). http://eprint.iacr.org/2017/498
14. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.2 (2016). https://competitions.cr.yp.to/round3/asconv12.pdf
15. Ferguson, N.: Collision attacks on OCB. NIST CSRC website (2002)
16. Ferguson, N.: Authentication weaknesses in GCM (2005)
17. Fleischmann, E., Forler, C., Lucks, S.: McOE: a family of almost foolproof on-line authenticated encryption schemes. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 196–215. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34047-5_12
18. Forler, C., List, E., Lucks, S., Wenzel, J.: Reforgeability of authenticated encryption schemes. In: Pieprzyk, J., Suriadi, S. (eds.) ACISP 2017. LNCS, vol. 10343, pp. 19–37. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59870-3_2
19. Fuhr, T., Leurent, G., Suder, V.: Collision attacks against CAESAR candidates. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9453, pp. 510–532. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48800-3_21
20. Handschuh, H., Preneel, B.: Key-recovery attacks on universal hash function based MAC algorithms. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 144–161. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_9
21. Hoang, V.T., Krovetz, T., Rogaway, P.: Robust authenticated-encryption AEZ and the problem that it solves. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 15–44. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_2
22. Hoang, V.T., Krovetz, T., Rogaway, P.: AEZ v5: authenticated encryption by enciphering (2017). https://competitions.cr.yp.to/round3/aezv5.pdf

23. Huang, T., Wu, H.: Attack on AES-OTR. https://groups.google.com/forum/#!topic/crypto-competitions/upaRX2jdVCQ
24. Iwata, T., Minematsu, K., Guo, J., Morioka, S., Kobayashi, E.: CLOC and SILC (2016). https://competitions.cr.yp.to/round3/clocsilcv3.pdf
25. Iwata, T., Ohashi, K., Minematsu, K.: Breaking and repairing GCM security proofs. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 31–49. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_3
26. Jean, J., Nikolić, I., Peyrin, T.: Deoxys v1.41 (2016). https://competitions.cr.yp.to/round3/deoxysv141.pdf
27. Jonsson, J.: On the security of CTR + CBC-MAC. In: Nyberg, K., Heys, H. (eds.) SAC 2002. LNCS, vol. 2595, pp. 76–93. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36492-7_7
28. Joux, A.: Authentication failures in NIST version of GCM (2006)
29. Katz, J., Yung, M.: Unforgeable encryption and chosen ciphertext secure modes of operation. In: Goos, G., Hartmanis, J., van Leeuwen, J., Schneier, B. (eds.) FSE 2000. LNCS, vol. 1978, pp. 284–299. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44706-7_20
30. Krovetz, T., Rogaway, P.: OCB (v1.1) (2016). https://competitions.cr.yp.to/round3/ocbv11.pdf
31. Lu, J.: Almost universal forgery attacks on the COPA and marble authenticated encryption algorithms. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, pp. 789–799. ACM (2017)
32. McGrew, D.A., Viega, J.: The security and performance of the galois/counter mode (GCM) of operation. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT 2004. LNCS, vol. 3348, pp. 343–355. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30556-9_27
33. Mileva, A., Dimitrova, V., Velichkov, V.: Analysis of the authenticated cipher MORUS (v1). In: Pasalic, E., Knudsen, L.R. (eds.) BalkanCryptSec 2015. LNCS, vol. 9540, pp. 45–59. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29172-7_4
34. Minematsu, K.: AES-OTR v3.1 (2016). https://competitions.cr.yp.to/round3/aesotrv31.pdf
35. Nikolić, I.: Tiaoxin - 346 (2016). https://competitions.cr.yp.to/round3/tiaoxinv21.pdf
36. Procter, G., Cid, C.: On weak keys and forgery attacks against polynomial-based MAC schemes. J. Cryptology **28**(4), 769–795 (2015). https://doi.org/10.1007/s00145-014-9178-9
37. Rogaway, P.: Authenticated-encryption with associated-data. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, 18–22 November 2002, pp. 98–107 (2002)
38. Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 373–390. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_23
39. Rogaway, P., Wagner, D.A.: A critique of CCM. IACR Cryptology ePrint Archive 2003/70 (2003)
40. Saarinen, M.-J.O.: Cycling attacks on GCM, GHASH and other polynomial MACs and hashes. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 216–225. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34047-5_13

41. Sun, Z., Wang, P., Zhang, L.: Collision attacks on variant of OCB mode and its series. In: Kutyłowski, M., Yung, M. (eds.) Inscrypt 2012. LNCS, vol. 7763, pp. 216–224. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38519-3_14
42. Vaudenay, S., Vizár, D.: Under pressure: security of caesar candidates beyond their guarantees. Cryptology ePrint Archive, Report 2017/1147 (2017). https://eprint.iacr.org/2017/1147
43. Whiting, D., Ferguson, N., Housley, R.: Counter with CBC-MAC (CCM) (2003)
44. Wu, H.: ACORN: A lightweight authenticated cipher (v3) (2016). https://competitions.cr.yp.to/round2/acornv2.pdf
45. Wu, H., Huang, T.: The authenticated cipher MORUS (v2) (2016). https://competitions.cr.yp.to/round3/morusv2.pdf
46. Wu, H., Huang, T.: The JAMBU lightweight authentication encryption mode (v2.1) (2016). https://competitions.cr.yp.to/round3/jambuv21.pdf
47. Wu, H., Preneel, B.: AEGIS: a fast authenticated encryption algorithm (v1.1) (2016). https://competitions.cr.yp.to/round3/aegisv11.pdf