Weak Keys in the Rekeying Paradigm: Attacks on COMET-128 and mixFeed

Mustafa Khairallah

School of Physical and Mathematical Sciences
Nanyang Technological University
mustafam001@e.ntu.edu.sg

Abstract. In this article, we analyze two of the NIST Round 1 Candidates for the Lightweight Cryptography Standardization Process: COMET and mixFeed. We show how AEAD modes that are based on rekeying can be modelled as modes without rekeying in the multi-key setting, where every nonce is treated as a different user. Then we show that the security degradation due to weak keys in the multi-key setting will affect these modes in the single key setting. We show how the weak key analysis of both these modes may be applied.

Keywords: weak keys · authenticated encryption · comet · mixfeed · nist · forgery · AEAD

1 Introduction

Lightweight Symmetric Key Cryptography has been a growing research area in the past 10 years or more, with applications varying between block cipher design, autheticated encryption, hash functions and much more. This has lead the NIST to release a call for proposals for a new lightweight cryptography standard [nis18]. In this preprint, we study two of these proposals: COMET-128 [GJN19] and mixFeed [CN19], analyzing the existence and behaviour of weak keys for these two primitives.

2 Background

2.1 Weak Key Analysis of Authenticated Enryption

Weak keys are defined as keys that behave in a non-expected manner compared to how the encryption algorithm was intended and detecting whether a secret key belongs to the set of weak keys is easy [HP08]. For example, if an algorithm requires that every call to the primitive used in that algorithm uses a different key, but for some keys the key is fixed for all primitive calls, that is an unexpected behaviour. In many cases (as the cases we target in this article) this behaviour is detectable with one verification query, since if the weak key occurs, it immediately leads to forgery. Usually, the security bounds are calculated on average over the whole key space, which limits the vulnerability of the weak keys in the single key setting.

Whether the weak keys are exploitable or not is another issue. For example, several NIST candidates including: Remus [IKMP19], mixFeed [CN19], GIFT-COFB [BCI+19], HyENA [CDJN19] and COMET [GJN19] and potentially others have keys or masks that can behave in unexpected ways. However, it is not clear how these keys are exploitable. An example of a case where the weak keys were exploitable in the AEAD forgery context is the attack on POET by Abdelraheem et al. [ABT14].

2.2 Multi-Key Analysis

In [LMP17], Luykx et al. show that the effect of analyzing the security of a symmetric key primitive in the multi-key setting can lead to drastic security degradations when the same primitive is used by a huge number of users using different keys. They gave a formula for the adevrsarial advantage gain against a scheme that has weak keys when used by many users at the same time. If the probability that any key is a weak key is p, the probability that all the μ keys (where μ is the number of users) are not weak is given by

$$(1-p)^{\mu} \tag{1}$$

and the probability that at least one of them is weak is given by

$$1-(1-p)^{\mu} \tag{2}$$

This probability increases almost linearly when μ is small and approaches 1 when $\mu > p^{-1}$. For example, if $p = 2^{-32}$, the advantage is 0.98 at $\mu = 2^{34}$, 0.63 at $\mu = 2^{32}$, 0.39 at $\mu = 2^{31}$, 2^{-16} at $\mu = 2^{16}$ and 2^{-31} at $\mu = 2$.

NIST Candidates for the new lightweight cryptography standard has been required so far to be secure only in the single key setting [nis18]. Usually, considering only the single key setting reduces the vulnerabilities related to weak keys compared to the multi-key setting [LMP17]. However, due to the way COMET-128 and mixFeed apply a rekeying function to every new message and the way that key is used internally, it allows us to apply the multi-key-weak-key analysis, by viewing every message as a different user and the rekeying function as an authority that assigns a key to every user based on that user's id (nonce) and the master key. In this setting, we divide the algorithm into two parts:

1. $Z = \texttt{KDF}(N, K)$: the rekeying function.
2. $(C, T) = \texttt{Enc}(A, M, N, K, Z)$: the AEAD portion after initialization.

Moreover, the multi-key setting is interesting on its own for practical uses, specially in the field of ubiquitous computing and IoT where an astronomical number of devices is expected to use the upcoming standard. However, this article only focuses on the application of the multi-key analysis in the single key setting. We do not make any claims on the security in the multi-key setting, but we predict that the degradation will grow further.

2.3 Forgery Attacks on nonce based CBC-like AEAD with repeated keys

Before we move on to the weak key analysis of COMET-128 and mixFeed we show simple forgery attacks on schemes that use a CBC-like structure with block keys that come from a permutation and use a linear feedback function. Table 1 represents a wide class of serial block cipher based AEAD modes, where K is the block key, S_1, S represent the internal state after the cipher call and after the linear feedback, respectively, M, C are the plaintext and ciphertext strings, respectively, of length m each. ρ is a linear invertible operation over two blocks, and P is a permutation over the space of K.

Given this representation it is easy to see that if K is constant or is P has a short cycle, that is a very undesirable behaviour and can lead to many types of forgery. Let's assume that M consists of two blocks. The adversary can apply the following attack:

1. Ask for the encryption of M.
2. Calculate the internal state values for S_1, S at different points in the execution.

Table 1: Simplistic View of the Encryption Phase CBC-Like AEAD Modes

$S[-1] \leftarrow \text{rand}()$
$K \leftarrow \text{rand}()$
$i = 0$
for $i < m$:
  $S_1[i] \leftarrow E_K(S[i-1])$
  $(S[i], C[i]) \leftarrow \rho(S_1[i], M[i])$
  $K \leftarrow P(K)$

3. Find a ciphertext block C_x and a plaintext block M_x such that $(S[1], C_x) = \rho(S_1[0], M_x)$. This is very easy due to the properties of ρ.

4. Use the same tag from the encryption query with the ciphertext C_x to build a verification query.

If K is not fixed, but P has a short cycle of length l, then the same attack is applicable, but modified to

1. Ask for the encryption of M of legnth $2l$.

2. Calculate the internal state values for S_1, S at different points in the execution.

3. Find a ciphertext block C_x and a plaintext block M_x such that $(S[l], C_x) = \rho(S_1[0], M_x)$. This is very easy due to the properties of ρ.

4. Use the same tag from the encryption query with the ciphertext $C_X = C_x|C[l+1]|C[l+2]|\cdots|C[2l-1]$ to build a verification query.

Another possible forgery attack is

1. Ask for the encryption of M of legnth $l+1$.

2. Calculate the internal state values for S_1, S at different points in the execution.

3. Find a ciphertext block C_x and a plaintext block M_x such that $(S[0], C_x) = \rho(S_1[l], M_x)$. This is very easy due to the properties of ρ.

4. Use the same tag from the encryption query with the ciphertext $C_X = C[0]|C[1]|\cdots|C[l-1]|C_x|C[1]|C[2]|\cdots|C[l]$ to build a verification query.

3 Attack on COMET-128

3.1 Brief Description of COMET-128

COMET [GJN19] is a block cipher based AEAD algorithm submitted to the NIST Standardization Process for Lightweight Cryptography. The schematic of the algorithm provided in the specification is given in Figure 1. It is based on related-key security and the ideal cipher model, where first, the key derivation function (`KDF`) is applied on the nonce N and the master key K to get the session key Z_0. Afterwards, Z_i is used as the key to the block cipher call with index i, where Z_i is the output of a permutation applied on Z_{i-1}.

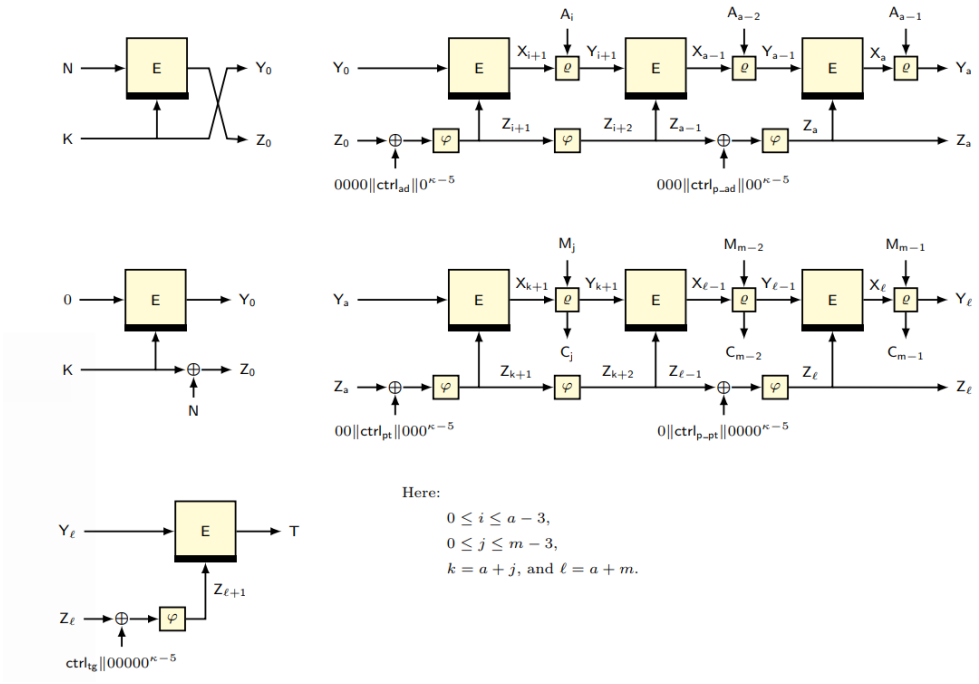

Figure 1: The schematic of COMET captured from the specification document [GJN19]

3.2 Multi-Key Representation and Summary of Results

Following the setting in Section 2.2, we show that for every pair $(N|A, K)$ there are 2^{64} weak keys, with probability 2^{-64} that the KDF outputs one of these keys. For simplicity of description of the attack, we will drop A, w.l.o.g., considering authenticated encryption without associated data. However, while this simpler setting is enough to show our claims, the results are equally applicable in the case of a constant A.

The existence of these weak keys and the applicability of the multi-key analysis leads to a set of interesting results:

1. After one query of length at least 32 bytes, forgery is successful with adverserial advantage 2^{-64}.
2. With 2^{63} online queries of 32 bytes each, forgery is successful with adverserial advantage $\sim 2^{-1}$. This is acheived with 0 offline queries, which contradicts the claims made by the designers.
3. With 2^{64} online queries of 32 bytes each, forgery is successful with adverserial advantage approaching ~ 1.
4. If the the forgery is successful, the master key is identified easily with high probability with 2^{65} offline queries.

3.3 Existence of Weak Keys

The specification of COMET-128 does not include any discussion on the weak key analysis. COMET-128 uses a CBC-like structure, where the master key is used as an IV, and the session key Z is used as the block cipher key and is updated after each block cipher call:

1. At 5 points during the algorithm execution, the least significant 5 bits of Z are XORed with one of 5 constants. This is done for domain separation.

2. After each block cipher call, the least significant 64 bits of Z are multiplied by a constant $\alpha = x$ over GF(2^{64}).

The second update is what we target in our attacks. If the value of the least significant 64 bits of Z during one stage of the algorithm is 0^{64}, this means that Z is constant over all the blocks of that stage. Since this event is defined over 64 bits of Z, which is chosen uniformally at random from the KDF (a permutation over GF(2^{128})), there are 2^{64} weak values of Z for each stage of the algorithm and the probability that Z at a certain stage is weak is equal to $\frac{2^{64}}{2^{128}} = 2^{-64}$. Since COMET-128 applies the KDF with a different N for every different message and since the KDF is a permutation, given μ online queries, we get μ messages encrypted with μ different values Z_i.

3.4 Existensial Forgery Attack with Weak Keys

Given we have established how the weak keys behave and their probability, we describe how to forge a ciphertext once a weak key has been sampled by the KDF. Let M be the known message encrypted with a weak key $Z_a \oplus 0010^{125} = 0^{64}|Z_{a_t}$ (refer to Figure 1), where $|M| \geq 256$. Let M_1 and M_2 be the first two message blocks after parsing M, with C_1 and C_2 as the corresponding ciphertexts. Since the attacker knows M and C, he can retreive the internal state values S_1 and S_2, where S_1 is the state before the absorption of M_1 and S_2 is the state after the absorption of M_2. Hence, we have

$$S_1 = Shuffle^{-1}(M_1 \oplus C_1) \tag{3}$$

and

$$S_2 = M_2 \oplus Shuffle^{-1}(M_2 \oplus C_2) \tag{4}$$

The attacker wants to find M_x and C_x, such that

$$S_1 = Shuffle^{-1}(M_x \oplus C_x) \tag{5}$$

and

$$S_2 = M_x \oplus Shuffle^{-1}(M_x \oplus C_x) \tag{6}$$

Which is a simple well-defined Linear system of equations defined over 256 boolean variables and easily solvable. The attacker removes C_1 and C_2 from the ciphertext, and inserts C_x in the location of C_1, while shifting the rest of the ciphertext 16 bytes backwards and reducing the ciphertext size by 16, leading to a successful forgery. This attack has been verified by modifying the reference implementation of COMET-128 to use some weak keys Z_i. The overall complexity of the attack is 2^{64} online queries, 0 offline queries and 2^{64} offline time complexity to find M_x and C_x and succeeds with probability close to 1.

While this attack is powerful in its own regard, as it disproves the claims made by the designers that forgery requires D online queries and T offline queries, such that $DT = 2^{187.5}$ or $D \sim 2^{64}$ and $T \sim 2^{119}$, in order to get a near 1 advantage, and makes the security of COMET-128 questionable in both the single key and multi-key settings, in the next section we show that because of another feature of COMET-128, all the security of the scheme collapses once this event happens.

Table 2: Intergrity Claims Made by the Designers of COMET-128 vs. Our Attack: D_e is the number of encryption queries, D_v is the number of verification queries and T is the number of offline primitive queries

Method	D_e	D_v	T
Tag Guessing [GJN19]	-	2^{128}	-
Decryption-Encryption Matching [GJN19]	2^{121}	2^{128}	-
Decryption-Offline Matching 1 [GJN19]	-	2^{121}	2^{128}
Decryption-Offline Matching 2 [GJN19]	-	2^{65}	$2^{122.5}$
Weak Key Analysis [Ours]	2^{65}	2^{64}	-

Table 3: Privacy Claims Made by the Designers of COMET-128 vs. Our Attack: D_e is the number of encryption queries, D_v is the number of verification queries and T is the number of offline primitive queries

Method	D_e	D_v	T
Online-Online Matching [GJN19]	2^{65}	-	2^{191}
Online-Offline Matching [GJN19]	2^{65}	-	2^{183}
Key Guessing [GJN19]	1	-	2^{128}
Weak Key Analysis [Ours]	2^{65}	2^{64}	2^{65}

3.5 Key Recovery Attack

The previous existential forgery attack can be used as a filter to discover the occurence of a weak key. Once the forgery succeeds, we know that Z during the message encryption phase of the algorithm has one of the weak key values, which are 2^{64} values. The attacker can then choose a message that has been previously encrypted with a weak key, and reverse the algorithm with each of these values. Since the master key K is used as an IV in COMET-128, this will lead to 2^{64} possible key candidates. For each of these key candidates, the attacker can apply $\texttt{KDF}(N, K)$ and verify whether the `KDF` generates the corresponding Z. Since the probability that $E_K(N) = Z$ is 2^{-128}, we expect to be able to uniquely identify the master key at this point, which completely breaks the system. The complexity is $\mathcal{O}(1) \cdot 2^{64}$.

3.6 Summary of Results

The designers make several claims in their document. First, they claim that the `permute` function has a period of 2^{64} and hence the only way for the key to repeat is to encrypt more than 2^{64} blocks. We have shown that both claims are false, since there are 2^{64} keys with period 1 and consequently this is another way for the key to repeat, and it is easily detectable with a single verification query.

The designers also mention four ways to attack the integrity of COMET-128, we interpret them as suggested by the designers in section 5.1.4 of [GJN19]. When subsituting in the bounds they give for the values of n = the block size and κ = the key size, we get the results in Table 2 for interity and Table 3 for privacy. Our weak key analysis shows that these claims are not true in both cases.

4 Attack on mixFeed

4.1 Brief Description of mixFeed

mixFeed [CN19] is an AES-based AEAD algorithm submitted to round 1 of the NIST Lightweight Cryptography Standardization Process. It uses a hybrid feedback structure,

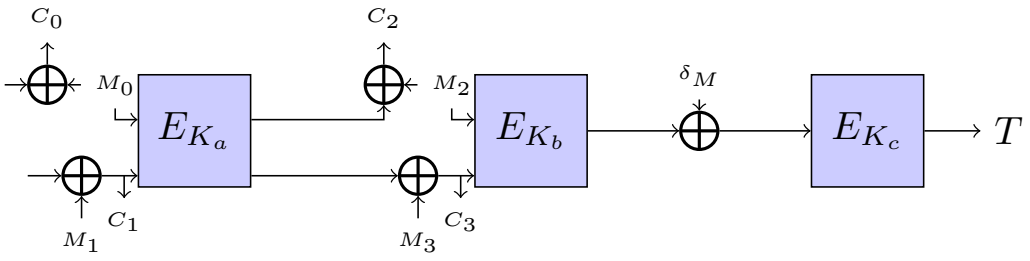

Figure 2: Part of the Encryption Phase of mixFeed

where half the input to the block cipher comes directly from the plaintext, while the other half is generated from the previous block cipher call and the plaintext in a CBC-like manner. The initial session key is generated using a `KDF` that depends on the master key K and the nonce N, then each block key is the output of applying a permutation P to the previous block key. The permutation P is defined as 11 rounds of the AES key Scheduling Algorithm [DR13]. The encryption part is shown in Figure 2. It was shown in a previous attack [Kha19] that given a known plaintext/ciphertext pair, the attacker can force the input to the block cipher to a certain value during the forgery challenge, so we consider this part of the analysis as a given and we do not include it in this article. Forcing the input to the block cipher is not enough to lead to an attack and is applicable to many block cipher based schemes. The question is whether the attacker can get enough information about the session key to increase the adversarial advantage.

4.2 Weak Key Analysis of mixFeed

The designers of mixFeed discuss the multi-key analysis in a brief statement in the specification. However, they do not mention the weak-key analysis. At first, it is not obvious why the weak key analysis is relevant to mixFeed. However, when we study how the mode operates, it is quite similar to modes like COMET, except that the key update function between blocks is not a multiplication by constant over a finite field, but it is the key schedule permutation of AES itself. In other words, every block cipher call takes as a key $K_i = P(K_{i-1})$, where K_{i-1} is the key used in the previous block and P is the permutation that applies 11 rounds of the AES key schedule. As explained in Section 2.3, insertion or omition forgeries succeed if the key is repeated, i.e. if the permutation cycle used to update the key is smaller than the message length. If the permutation is well designed, e.g. maximal length LFSR or arithemtic counter, the probability of this event should be very low. Also, if the permuation is an ideal permutation picked uniformally at random, it should have n cycles whose lengths follow a Poisson distribution.

The AES key schedule permutation is not designed to be an ideal permutation and it should not be used as one. It can be described as a permutation over four 32-bit words, which consists of 11 rounds. The rounds differ only in the round constant. We define the permutation f_c over a 32 bit word as the feedback function in round c as:

$$W \rightarrow SubWord(W \ggg 8) + rcon[c] \tag{7}$$

where $W \ggg r$ represents bitwise right rotation of W by r bits and $rcon[c]$ is defined as $x^{c\%11}|0^{24}$ such that x is defined over GF(2^8). Given this permutation, a single round of the AES key schedule can be defined as

$$W_0, W_1, W_2, W_3 \rightarrow W_0 \oplus f_c, W_1 \oplus W_0 \oplus f_c, W_2 \oplus W_1 \oplus W_0 \oplus f_c, W_3 \oplus W_2 \oplus W_1 \oplus W_0 \oplus f_c \tag{8}$$

Table 4: 8 round unrolling of the AES key schedule

Round 0	W_0	W_1	W_2	W_3
Round 1	$W_0 \oplus f_0$	$W_1 \oplus W_0 \oplus f_0$	$W_2 \oplus W_1 \oplus W_0 \oplus f_0$	$W_3 \oplus W_2 \oplus W_1 \oplus W_0 \oplus f_0$
Round 2	$W_0 \oplus f_0 \oplus f_1$	$W_1 \oplus f_1$	$W_2 \oplus W_0 \oplus f_0 \oplus f_1$	$W_3 \oplus W_1 \oplus f_1$
Round 3	$W_0 \oplus f_0 \oplus f_1 \oplus f_2$	$W_1 \oplus W_0 \oplus f_0 \oplus f_2$	$W_2 \oplus W_1 \oplus f_1 \oplus f_2$	$W_3 \oplus W_2 \oplus f_2$
Round 4	$W_0 \oplus f_0 \oplus f_1 \oplus f_2 \oplus f_3$	$W_1 \oplus f_1 \oplus f_3$	$W_2 \oplus f_2 \oplus f_3$	$W_3 \oplus f_3$
Round 8	$W_0 \oplus \bigoplus_{i=0}^{7} f_i$	$W_1 \oplus \bigoplus_{i=0}^{3} f_{2*i+1}$	$W_2 \oplus f_2 \oplus f_3 \oplus f_6 \oplus f_7$	$W_3 \oplus f_3 \oplus f_7$

where f_c is applied to W_3 and eight unrolled rounds can be defined as in Table 4.

In fact, there is an iterative structure over 4 rounds, where we can write the value of any key word after 4 rounds in terms of the initial value of this word and a certain set of feedback functions. If a key is a fixed point over R rounds, where R is a multiple of 4, then the involved feedback functions must add up to 0. If the feedback function is ideal, we expect this to happen with probability 2^{-32} for each word and 2^{-128} in total, but of course, this is not the case. It is trivial to see that there is only 1 value which is a fixed point after 4 rounds, and in general the conditions for a fixed point after R rounds are

$$\bigoplus_{i=0}^{R/4-1} f_{3+4i} = 0 \tag{9}$$

$$\bigoplus_{i=0}^{R/4-1} f_{2+4i} = 0 \tag{10}$$

$$\bigoplus_{i=0}^{R/4-1} f_{1+4i} = 0 \tag{11}$$

$$\bigoplus_{i=0}^{R/4-1} f_{4i} = 0 \tag{12}$$

$$\tag{13}$$

For example, fixed points over 8 rounds must satisfy

$$f_3 \oplus f_7 = 0 \tag{14}$$

$$f_2 \oplus f_6 = 0 \tag{15}$$

$$f_1 \oplus f_5 = 0 \tag{16}$$

$$f_0 \oplus f_4 = 0 \tag{17}$$

$$\tag{18}$$

The last condition can be written as $f_0(W_3) \oplus f_4(W_3 \oplus f_3) = 0$. Since f_0 and f_4 differ only in the constant value, we can rewrite the condition as

$$SubWord(W \ggg 8) \oplus SubWord(W \ggg 8 \oplus \Delta) = \delta \tag{19}$$

where $\Delta = f_3$ and $\delta = rcon[4] \oplus rcon[0]$. Clearly, this a non-linear equation. What is interesting, is that this equation is defined over the Sbox of AES and can be divided into three equations on the form $Sbox(x) \oplus Sbox(x \oplus y) = 0$ and one equation of the form $Sbox(x) \oplus Sbox(x \oplus y) = a$. For the first three equations, $y = 0$ since the AES Sbox is bijective, while for the last one y has 127 possible values that can be retreived from the Difference Distribution Table of the AES Sbox. Hence, we reduce the possibilities of f_3 to 127 values. Then,

Table 5: Representatives of 20 Cycles of length= 1133759136 for the AES Key Schedule 11 Round Permutation Used in mixFeed

000102030405060708090a0b0c0d0e0f
00020406080a0c0e10121416181a1c1e
0004080c1014181c2024282c3034383c
000810182028303840485058606870 78
0010203040506070 8090a0b0c0d0e0f0
101112131415161718191a1b1c1d1e1f
20222426282a2c2e30323436383a3c3e
4044484c5054585c6064686c7074787c
80889098a0a8b0b8c0c8d0d8e0e8f0f8
303132333435363738393a3b3c3d3e3f
707172737475767778797a7b7c7d7e7f
000306090c0f1215181b1e2124272a2d
00050a0f14191e23282d32373c41464b
00070e151c232a31383f464d545b6269
000d1a2734414e5b6875828f9ca9b6c3
00152a3f54697e93a8bdd2e7fc11263b
00172e455c738aa1b8cfe6fd142b4259
00183048607890a8c0d8f00820385068
001c3854708ca8c4e0fc1834506c88a4
001f3e5d7c9bbad9f81736557493b2d1

$$f_3(W_2 \oplus W_3 \oplus f_2) \oplus f_7(W_2 \oplus W_3 \oplus f_2 \oplus f_6) = f_3(W_2 \oplus W_3 \oplus f_2) \oplus f_7(W_2 \oplus W_3) = 0 \quad (20)$$

Hence, similar arguments can be made about f_2 and similarily f_1 and f_0. By such argument, one expects roughly about $2^{27.9}$ (to be verified) fixed points for the reduced-round AES key schedule of 8 rounds. One can go analyzing more rounds. While this problem is interesting on its own regard, it is not the scope of our result and we leave it to future work. We just mention the analysis to show that the AES Key Schedule is far from an ideal permutation and also because the cycle length we have found is a multiple of 4.

We have run a simple cycle finding script using brute force and found at least 20 cycles of length $1133759136 \sim 2^{30.08}$, out of 33 seeds we have tried. We give a representative of each of those cycles in Table 5, in case the reader want to verify the results. We will also make our simple script available online. It is not clear to us why this number is special. However, this means that there are at least $2^{34.4}$ weak keys which allow forgery of messages of legth $2^{30.08} + 1$ blocks into messages of length $2^{31.08} + 1$. Finding each of these cycles takes around 1 hour on a single-core personal computer using brute force, hence we do not know how many cycles are there of these structure or of different length. We found a large set of values that do not belong to cycles of that length or smaller. However, our findings are good enough to show that the security bounds claimed by the designers of mixFeed are not true. The designers claim that the adversarial advantage is $\frac{\sigma T}{2^{192}}$. However, as we explained in the analysis of COMET-128, by applying multi-key-weak-key analysis, we see that the advantage increases drasctically. According to the designers, after encrypting a message of $2^{30.08} + 1$ blocks and decrypting a message of length $2^{31.08} + 1$ blocks, the adversarial advantage should be $2^{-160.92}$. However, the weak key analysis show that forgery is successful with probability *at least* $2^{-93.59}$. After encrypting $2^{18.92}$ such messages, with overall online complexity of 2^{50} blocks, the adverserial advantage is $2^{-74.68}$.

While this result does not make mixFeed insecure, it shows a huge gap in the security analysis of mixFeed and calls for further cryptanalytic efforts if mixFeed is to be used in the

real world. As mentioned earlier, our result is just a lower bound found by brute forcing some key values. For example, there is a big question mark on the special cycle length we found, which may potentially be related to more cycles, further increasing the advantage. Since we cannot do a full characterization of the AES Key Schedule Permutation to find all the cycles and given that the experiments show that a certain cycle length is highly probable, we use statistical inference to argue about the average cycle length of the permutation. We assume that the cycle length of a random permutation follows a Poisson distributions [Gra06], such that

$$L(l = k) = \frac{e^{-\lambda}\lambda^k}{k!} \tag{21}$$

where λ in this case is the average cycle length, then it is easy to see that the Maximum Likelihood Estimate of λ in our case is 1133759136, while a hypothesis that the average cycle length is in the order of 2^{50} can be easily rejected. None of these results are conclusive. Nevertheless, without proper understanding of the underlying permutation, and given our experimental results which show a huge gap between the security claims and reality, it is hard to argue for the security of mixFeed.

It is worth noting that the weak key analysis does not lead to the master key recovery, like in the case of COMET-128, since mixFeed does not use the master key during the main part of the encryption.

4.3 Summary of Results

The designers of mixFeed make a security claim that the adversarial advantage is bounded by $\frac{DT}{2^{192}}$. We believe a better characterization would have been $max(\frac{DT}{2^{192}}, \frac{1}{2^{128}})$, since the designers should have included key guessing and tag guessing attacks. It is not clear whether this bound is for privacy or integrity or both. We have shown that the integrity bound for the adversarial advantage cannot be lower than 2^{-75} at $(D, T) = (2^{50}, 1)$, with the results making it very uncovincing that this is a tight lower bound.

5 Conclusion

We have applied the multi-key-weak-key analysis to two AEAD modes: COMET-128 and mixFeed, showing a huge gap between the security claims and reality for both of them. In the case of COMET-128, the whole security of the system breaks when the number of queries approaches the bound of 2^{64}, which contradicts the security bounds specified by the designers, and there should be no security claims beyond 64 bits. This may be fixed by choosing a different key permutation P, which will probably impact the performance of the scheme. In case of mixFeed, we enhanced the adverserial advantage by a factor of at least $2^{67.32}$ compared to the designers claim and at least $2^{53.32}$ compared to tag/key guessing attacks. We do not see how can this be fixed for mixFeed, since the choice of P is inherit in the mode design and even if instantiated with a different cipher, we expect the security degradation to be even more drastic, as AES has quite a strong key scheduling permutation compared to other ciphers. However, Whether it can be further enhanced is inconclusive. While our results invalidate many of the claims made by the designers, we do not make any claims on the practical security of these modes in real life lightweight applications as it is not the scope of this article and we leave this to future work, for third party evaluation of our results and for future cryptanalysis efforts.

Acknowlegement

I would like to thank Thomas Peyrin, Tetsu Iwata and Kazuhiko Minematsu on many fruitful discussions on topics related to this analysis, inluding weak keys, birthday bound security and the AES Key Scheduling.

I would like to thank Mridul Nandi, Shay Gueron, Ashwin Jha and Bishwajit Chakraborty for going through this draft and discussing the findings.

Statement

This document is in the state of a very early draft, I apologize for any typos or errors. I welcome constructive comments to my email.

References

[ABT14] Mohamed Ahmed Abdelraheem, Andrey Bogdanov, and Elmar Tischhauser. Weak-key analysis of poet. *IACR Cryptology ePrint Archive*, 2014:226, 2014.

[BCI+19] Subhadeep Banik, Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, Mridul Nandi, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT-COFB. NIST Lightweight Cryptography Project, 2019. https://csrc.nist.gov/Projects/Lightweight-Cryptography/Round-1-Candidates.

[CDJN19] Avik Chakraborti, Nilanjan Datta, Ashwin Jha, and Mridul Nandi. HyENA. NIST Lightweight Cryptography Project, 2019. https://csrc.nist.gov/Projects/Lightweight-Cryptography/Round-1-Candidates.

[CN19] Bishwajit Chakraborty and Mridul Nandi. mixFeed. NIST Lightweight Cryptography Project, 2019. https://csrc.nist.gov/Projects/Lightweight-Cryptography/Round-1-Candidates.

[DR13] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[GJN19] Shay Gueron, Ashwin Jha, and Mridul Nandi. COMET: COunter Mode Encryption with authentication Tag. NIST Lightweight Cryptography Project, 2019. https://csrc.nist.gov/Projects/Lightweight-Cryptography/Round-1-Candidates.

[Gra06] Andrew Granville. Cycle lengths in a permutation are typically poisson. *the electronic journal of combinatorics*, 13(1):107, 2006.

[HP08] Helena Handschuh and Bart Preneel. Key-recovery attacks on universal hash function based mac algorithms. In *Annual International Cryptology Conference*, pages 144–161. Springer, 2008.

[IKMP19] Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, and Thomas Peyrin. REMUS. NIST Lightweight Cryptography Project, 2019. https://csrc.nist.gov/Projects/Lightweight-Cryptography/Round-1-Candidates.

[Kha19] Mustafa Khairallah. Forgery attack on mixfeed in the nonce-misuse scenario. *IACR Cryptology ePrint Archive*, 2019:457, 2019.

[LMP17] Atul Luykx, Bart Mennink, and Kenneth G Paterson. Analyzing multi-key security degradation. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 575–605. Springer, 2017.

[nis18] Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process , 2018. https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/final-lwc-submission-requirements-august2018.pdf.