Forgery Attack on mixFeed in the Nonce-Misuse Scenario

Mustafa Khairallah

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

Abstract. mixFeed [CN19] is a round 1 candidate for the NIST Lightweight Cryptography Standardization Project. It is a single-pass, nonce-based, AES-based authenticated encryption algorithms. The authors claim that while there are no guarantees for security in terms of confidentiality in case of nonce-misuse (repetition), the integrity security still holds up to 2^{32} data complexity. In this report, this claim is not true in case the plaintext length is non-zero ($\geq$ 16 bytes to be exact). We show a forgery attack that requires only two encryption queries with the same nonce and 34 bytes of data.

Keywords: AEAD · forgery · mixFeed · Nonce Misuse · collision

1 Introduction

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, 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. On page 4, section 3, of [CN19], the authors make the claim that there is no conventional privacy security in case of nonce misuse. However, the integrity security remains until 2^{32} data in case of nonce misuse.

While it is not clear in the brief submission document how this bound was calculated, we believe through our analysis that is should be derived through a similar analysis of the integrity of the encrypted CBC-MAC [Vau00, PR00] (with 64 bits of random feedback between every two consecutive block-cipher calls). However, our analysis show that this claim may only be true for the case when the plaintext size is less than 16 bytes, which is a very restrictive scenario. In the next section, we show a simple forgery attack that requires only 32 bytes of plaintext and succeeds with probability 1 after only 1 nonce repetition.

2 Attack on the mixFeed AEAD mode in the Nonce-Misuse model

1. Generate an associated data string A and a plaintext string M of 32 bytes, divided into 4 words of 8 bytes each: M_0, M_1, M_2, M_3.
2. Generate a plaintext string $M^{'}$ of 32 bytes, divided into 4 words of 8 bytes each: $M^{'}_0, M^{'}_1, M^{'}_2, M^{'}_3$.
3. Send the following query to the encryption oracle: (N, A, M), storing the ciphertext/tag pair (C, T), where C consists of 4 words of 8 bytes each.

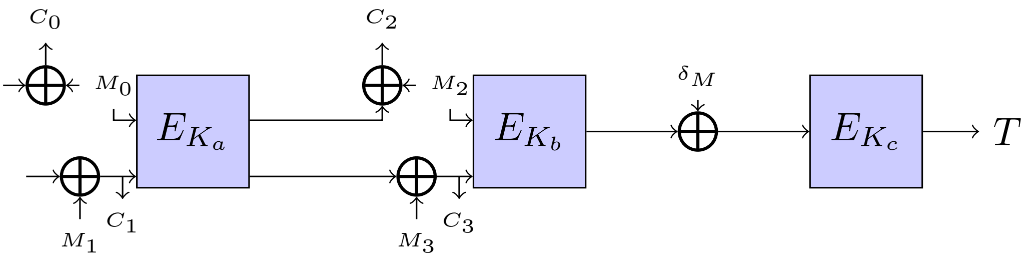

Figure 1: Trace of the first encryption query

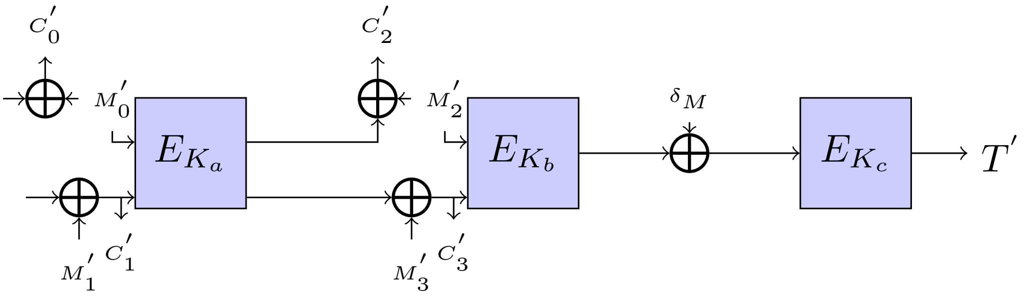

Figure 2: Trace of the second encryption query

4. Send the following query to the encryption oracle: $(N, A, M^{'})$, storing the ciphertext/tag pair $(C^{'}, T^{'})$, where $C^{'}$ consists of 4 words of 8 bytes each.

5. Calculate a ciphertext string $C^{''} = (C_0, C_1, C_2 \oplus M_2 \oplus M_2^{'}, C_3^{'})$.

6. Send the following challenge query to the decryption oracle: $(N, A, C^{''}, T')$. The decryption succeed with probability $p = 1$.

2.1 Attack Details

In order to understand why the attack works, we trace the intermediate values in the targeted part of the execution for the encryption and decryption queries. In Figures 1 and 2, we show the encryption calls for M and $M^{'}$. The goal on the attacker is to match the chaining values at the input of the second encryption in the challenge query. Due to the hybrid feedback structure, different strategies need to be used for different words of the ciphertext. For the ciphertext feedback branch (bottom branch of Figure 3), we simply change C_3 to $C_3^{'}$, which directly decides the imput to the block cipher in the decryption process. For the plaintext feedback branch (top branch of Figure 3), using $C_2^{''} = C_2 \oplus M_2 \oplus M_2^{'}$ as the ciphertext word leads $M_2^{'}$ at the input of the block cipher, since $C_2 \oplus M_2$ is the output of the block cipher in the previous call (1). Hence, the second encryption call matches the second encryption call from 2. Since all the calls before this call match 1 and all the calls afterwards match 2, using the same Tag $T^{'}$ from 2 leads to successful forgery attack.

2.2 Example

We have verified our attack using the reference implementation of mixFeed [CN19]. We generated the example forgery shown below.

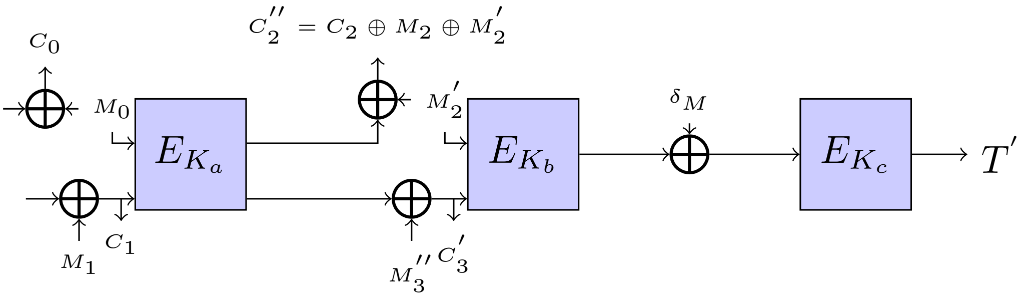

Figure 3: Trace of the challenge decryption query

The two encryption queries are:

```
Count = 1
Key = 000102030405060708090A0B0C0D0E0F
Nonce = 000102030405060708090A0B0C0D0E
PT = 000102030405060708090A0B0C0D0E0F
     101112131415161718191A1B1C1D1E1F
AD = 000102030405060708090A0B0C0D0E0F
CT = F4C757EEC527CAF2083A4E0E3548EB46
     83EA28AB2C68D70AA9A90EF42CA6451E
     324946C94446C53C5C77E661FCE80750

Count = 2
Key = 000102030405060708090A0B0C0D0E0F
Nonce = 000102030405060708090A0B0C0D0E
PT = 00081018202830384048505860687078
     80889098A0A8B0B8C0C8D0D8E0E8F0F8
AD = 000102030405060708090A0B0C0D0E0F
CT = F4CE45F5E10AFCCD407B145D592D9531
     4E21C4BB0B694B376CC43C361BA8B89A
     2C55A84A127C07C611B2E35175B7E28C
```

And the challenge ciphertext is

```
CT = F4C757EEC527CAF2083A4E0E3548EB46
     4E21C4BB0B694B377178C437D053ABF9
     2C55A84A127C07C611B2E35175B7E28C
```

where the decryption oracle outputs

```
PT = 000102030405060708090A0B0C0D0E0F
     DDDAFE0333148A2AC0C8D0D8E0E8F0F8
```

3 Instantiating the attack with different Associated data strings

The attack can be also be instantiated using only 16 bytes of plaintext, where the encryption queries have different associated data strings of equal number of bytes. The attacker can then select the AD from one query with 8 bytes of the ciphertext and 8 bytes of the plaintext taken from the other query to forge a decryption query.

3.1 Example

Count = 1
Key = 000102030405060708090A0B0C0D0E0F
Nonce = 000102030405060708090A0B0C0D0E
PT = 000102030405060708090A0B0C0D0E0F
AD = 000102030405060708090A0B0C0D0E0F
CT = F4C757EEC527CAF2083A4E0E3548EB46
89E7DB42C6777B7BBAFE1ABB4022AF28

Count = 2
Key = 000102030405060708090A0B0C0D0E0F
Nonce = 000102030405060708090A0B0C0D0E
PT = 00081018202830384048505860687078
AD = 00081018202830384048505860687078
CT = BCBA409676B0679FB27F7F70D1A0A6D9
84AE15E2E3347E8886E59A759E43A0D9

CT = BCBA409676B0679F407B145D592D9531
84AE15E2E3347E8886E59A759E43A0D9
PT = 487C157BB792AB6A4048505860687078

4 Conclusion

In this report we showed that the claims of integrity of mixFeed in the nonce misuse case are not true in general. In fact, it can only be true in case of empty (or potentially very small) plaintext. This does not affect the security of mixFeed in the nonce respecting case.

References

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

[PR00] Erez Petrank and Charles Rackoff. Cbc mac for real-time data sources. *Journal of Cryptology*, 13(3):315–338, 2000.

[Vau00] Serge Vaudenay. Decorrelation over infinite domains: the encrypted cbc-mac case. In *International Workshop on Selected Areas in Cryptography*, pages 189–201. Springer, 2000.