Forgery Attacks on FlexAE and FlexAEAD

Maria Eichlseder, Daniel Kales, and Markus Schofnegger

Graz University of Technology, Austria
{firstname.lastname}@iaik.tugraz.at

Abstract. FlexAEAD is one of the round-1 candidates in the ongoing NIST Lightweight Cryptography standardization project. In this note, we show several forgery attacks on FlexAEAD with complexity less than the security bound given by the designers, such as a block reordering attack on full FlexAEAD-128 with estimated success probability about 2^{-54}. Additionally, we show some trivial forgeries and point out domain separation issues.

Keywords: authenticated encryption · forgery attack · NIST LWC

1 Introduction

FlexAEAD [11] is one of the round-1 candidate algorithms of the ongoing NIST Lightweight Cryptography (LWC) standardization project [12]. The FlexAEAD family of authenticated encryption (AEAD) algorithms is based on the previously published authenticated encryption design FlexAE [8,9,10]. Compared to FlexAE, FlexAEAD was modified to also handle associated data blocks, and the generation of the ciphertext blocks was amended by an additional call to their internal keyed permutation to better resist reordering attacks.

In this note, we show that all proposed variants of FlexAEAD are vulnerable against different forgery attacks with complexity below the claimed security level:

1. Forgery based on reordering the associated data blocks.
2. Forgery based on truncating the ciphertext.
3. Forgery based on reordering the ciphertext blocks.

These attacks work despite several countermeasures included by the designers against such reordering attacks when developing FlexAEAD based on FlexAE. Our two main observations are (1) that the designers' rationale to combine the mode with a non-ideal primitive and rely on multiple encryption to hide distinguishing properties is not sound, and (2) that the attacks can be made significantly more efficient by taking advantage of the strong clustering effect of the sparse differential characteristics permitted by the non-ideal primitive. We also discuss applicability to the original FlexAE design, as well as several domain separation problems in FlexAEAD. Where possible, we propose fixes.

This note was originally posted on NIST's LWC mailing list [3]. Independent observations on this list include an iterated truncated differential attack and Yoyo distinguisher on the underlying block cipher [13,14] and a trivial padding domain separation attack for associated data [7] also mentioned in Section 4.1.

2 Description of FlexAE and FlexAEAD

In this section, we summarize the construction of FlexAEAD [11].

The main building block of FlexAE and FlexAEAD is a keyed permutation PF_K. PF_K is an Even-Mansour construction with whitening keys K_A, K_B, where the master key is $K = K_A \mid\mid K_B$. The inner permutation is built using a nibble shuffling layer reminiscent of Tree-Structured SPNs [6] and several parallel applications of the 8-bit AES S-Box [1]. The full construction of PF_K is given in Figure 1. For a more detailed description of the building blocks, we refer to the NIST submission document [11].

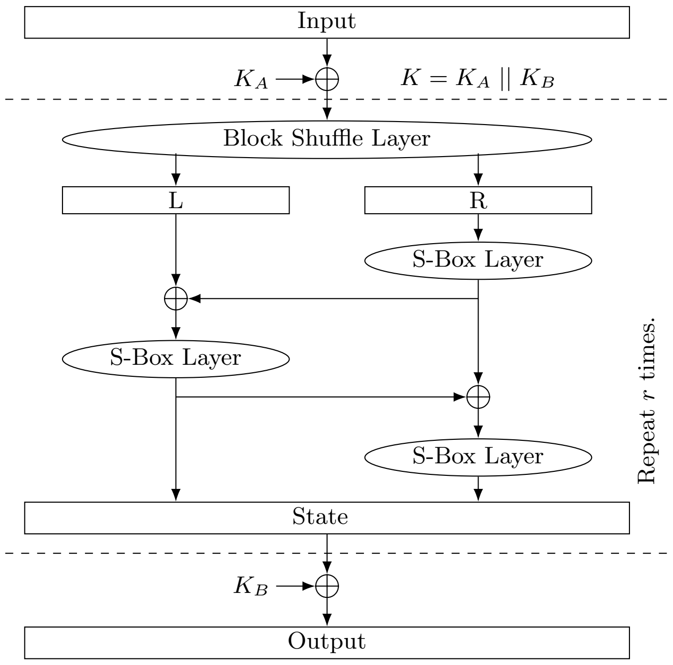
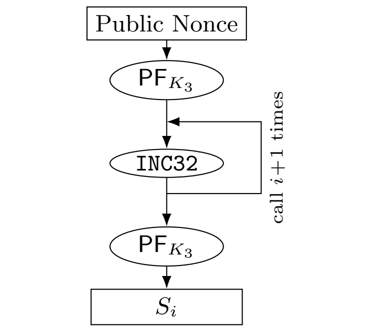
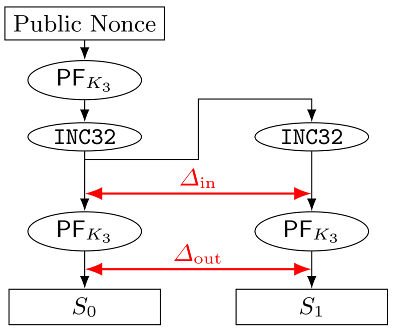


Fig. 1: Keyed permutation PF_K [11], with $r \in \{5, 6, 7\}$ for FlexAEAD-$\{64, 128, 256\}$.

PF_K is used with four different keys in the FlexAEAD construction. These four keys are derived from a master key K^* by applying PF_{K^*} three times to an initial state of 0^n, iterating this process to generate enough bits for the four subkeys K_0, K_1, K_2, K_3. A base counter is generated by applying PF_{K_3} to the nonce. This base counter is then used to generate the sequence $(S_0, \ldots, S_{n+m-1})$ by repeatedly applying the increment step `INC32` to the base counter, which treats each 32-bit block of the base counter as an 32-bit little-endian integer and increases it by one. PF_{K_3} is then again applied to the result of `INC32`, yielding the block S_0. Further blocks S_i can be generated by calling INC32 on the base counter $i + 1$ times before finally applying PF_{K_3}, as shown in Figure 2a.



(a) Generation of the sequence S.



(b) Differential $\Delta_{\text{in}} \rightarrow \Delta_{\text{out}}$.

Fig. 2: Counter-based differences in the generation of the sequence S.

The sequence $S_0, \dots, S_{n-1}, S_n, \dots, S_{n+m-1}$ is then used to mask the associated data blocks $A_0, \dots, A_{n-1}$ and plaintext blocks $P_0, \dots, P_{m-1}$, as well as intermediate results of the ciphertext generation process. This construction is inspired by the IAPM mode of operation [4,5]. To compute the tag T, PF_{K_0} is applied to the XOR of the intermediate results after the first application of PF_{K_2} to each masked block, plus a constant indicating whether the last plaintext block was a full or a partial block. The full construction is illustrated in Figure 3.

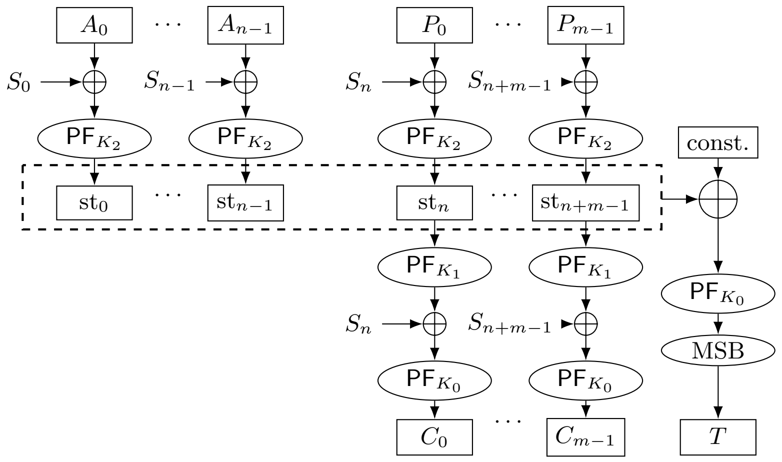


Fig. 3: The $\mathsf{FlexAEAD}$ mode for authenticated encryption (simplified, from [11]).

3 Forgery Attacks on FlexAE and FlexAEAD

In this section, we first propose a differential characteristic for PF_{K_3}. Then, we show how to apply the resulting counter differential to obtain several forgery attacks on FlexAEAD-64, FlexAEAD-128, FlexAEAD-256, and FlexAE.

3.1 Differential Characteristic for the Counter Sequence

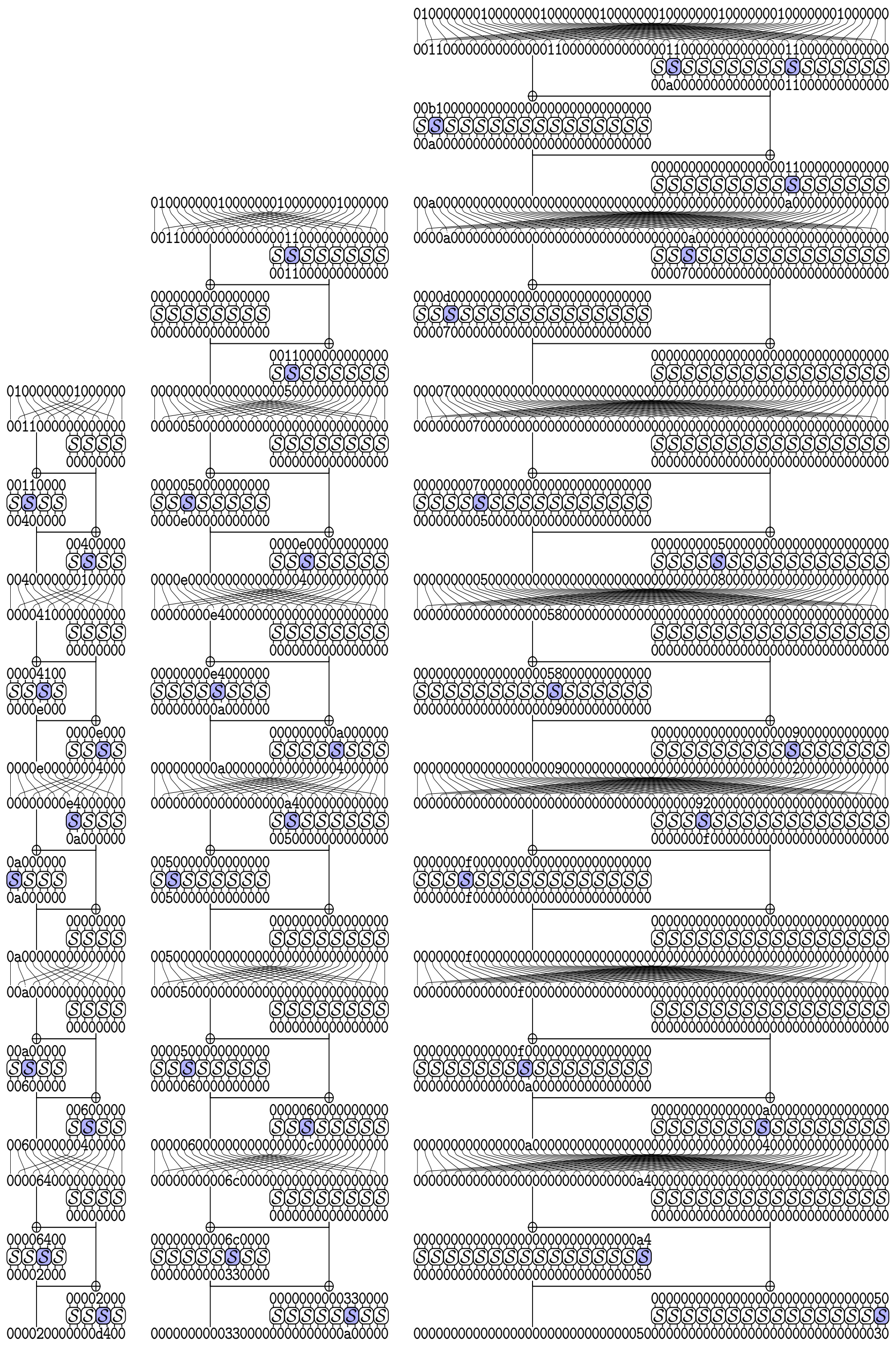
Recall the generation of the sequence S, as shown in Figure 2a. The intermediate state is updated by calling `INC32`, incrementing each 32-bit block of the state. Consider the difference between two states S_i and S_{i+1}: The only difference between the input to the final call to PF_{K_3} is one additional call to `INC32`. A little-endian addition by 1 behaves like an XOR operation with probability $\frac{1}{2}$ (exactly when the least significant bit of the state is zero). Therefore, the call `INC32` behaves like an XOR with a probability of 2^{-2} (2^{-4}, 2^{-8}) for FlexAEAD-64 (FlexAEAD-128, FlexAEAD-256). This process is shown in Figure 2b.

Using an input difference of $\Delta_{\text{in}} = \texttt{01000000 01000000}$ for FlexAEAD-64 (repeated twice for FlexAEAD-128 and four times for FlexAEAD-256), we consider the following differential characteristics for PF given in Figure 4. In the following, we always denote the input difference of this characteristic as Δ_{in} and the output difference as Δ_{out}. Under the Markov assumption, the probability of these differential characteristics is 2^{-66} (Figure 4a for FlexAEAD-64), 2^{-79} (Figure 4b for FlexAEAD-128), and 2^{-108} (Figure 4c for FlexAEAD-256), respectively. Note that the Markov assumption is clearly not well-suited for this keyless construction with limited diffusion.

The round function construction is very prone to clustering of characteristics, as illustrated by the corresponding partially truncated characteristics in Figure 5, where `*` denotes a nibble with an unspecified difference, `xx` is an arbitrary fixed nonzero difference, and `zz` is the high-probability S-box output difference with $\mathbb{P}[\texttt{xx} \to \texttt{zz}] = 2^{-6}$. Using the estimate that $\mathbb{P}[\texttt{11} \to \texttt{**}] = 1$, $\mathbb{P}[\{\texttt{**}, \texttt{*0}, \texttt{0*}\} \to \texttt{*0}] = \mathbb{P}[\{\texttt{**}, \texttt{*0}, \texttt{0*}\} \to \texttt{0*}] = 2^{-4}$, and $\mathbb{P}[\texttt{**} \to \texttt{xx}] = 2^{-8}$, we obtain estimated probabilities of 2^{-46} (FlexAEAD-64), 2^{-54} (FlexAEAD-128), and 2^{-70} (FlexAEAD-256). Furthermore, these clusters work not only for a fixed starting difference of `01` in the least significant byte of each counter, but for any single-nibble differences $\{\texttt{01}, \texttt{03}, \texttt{07}, \texttt{0f}\}$. As a consequence, the transition from modular difference $+1$ in each counter to a suitable XOR-difference works with high success probability close to 1. A more precise probability estimate could be obtained using tools such as semi-truncated characteristics [2] and exploiting the lack of round keys, but we expect a very similar result. Practical experiments on up to 3 rounds confirm this estimate, see Section 4.4.

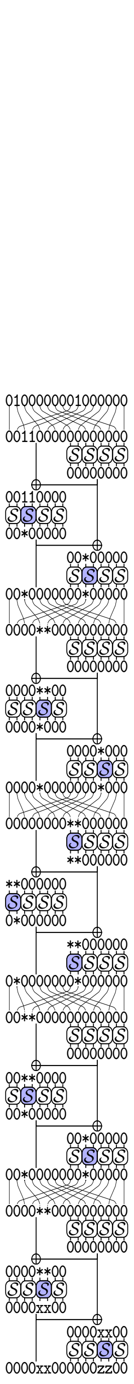
3.2 Forgery Attacks for FlexAEAD using the Counter Difference

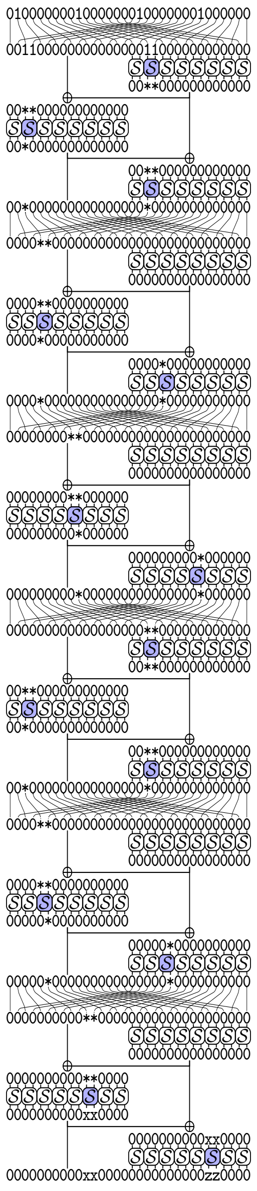
We can now use these differentials $\Delta_{\text{in}} \to \Delta_{\text{out}}$ in the counter sequence to mount forgery attacks on the full FlexAEAD-64, FlexAEAD-128, and FlexAEAD-256 schemes. In the following, we describe several different approaches.



(a) F-64: 2^{-66} (b) FlexAEAD-128: 2^{-79} (c) FlexAEAD-256: 2^{-108}

Fig. 4: Differential characteristics for full-round PF_K in FlexAEAD variants.


(a) F-64: 2^{-46}


(b) FlexAEAD-128: 2^{-54}

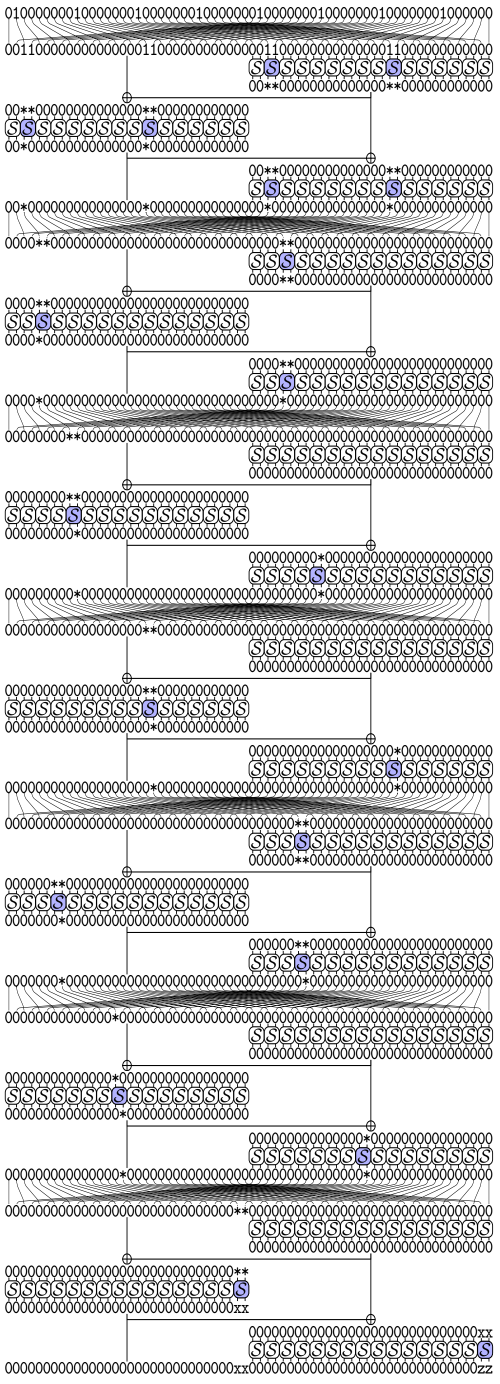

(c) FlexAEAD-256: 2^{-70}

Fig. 5: Clustered characteristics for full-round PF_K in FlexAEAD variants.

Changing Associated Data. We query a tag for some plaintext P with associated data $A = A_0 \mid\mid A_1$, where $A_0 \oplus A_1 = \mathit{\Delta}_{\text{out}}$. With a probability of about 2^{-46} (FlexAEAD-64), 2^{-54} (FlexAEAD-128), or 2^{-70} (FlexAEAD-256), the sequence blocks S_0 and S_1 follow the cluster of differential characteristics, and therefore also fulfill $S_0 \oplus S_1 = \mathit{\Delta}_{\text{out}}$. Then, $A_0 \oplus A_1 \oplus S_0 \oplus S_1 = 0$, so $S_0 \oplus A_0 = S_1 \oplus A_1$, resulting in a contribution of the two associated data blocks to the checksum of $\mathsf{PF}_{K_2}(S_0 \oplus A_0) \oplus \mathsf{PF}_{K_2}(S_1 \oplus A_1) = 0$.

Now, if we swap A_0 and A_1, with the same reasoning, the contribution to the checksum will again be 0, so the original tag is valid for the modified associated data with swapped blocks.

Although the example above assumes a distance of 1 between associated data blocks, we can generalize this property and also find similar differential characteristics for higher distances j. Distances with lower hamming weight and with several suitable XOR-differences following the same truncated difference generally result in a better probability. In practical experiments on round-reduced FlexAEAD, we observed an even higher success probability than expected when swapping associated data blocks, such as examples with a non-zero, but constant contribution to the checksum.

Truncating Ciphertext. In a similar fashion to the previous attack, we can also use this strategy to create a forgery targeting the plaintext.

Again, consider the generation of the sequence S, using the same strategy and differential characteristics as in Section 3.1. Now query a tag with a plaintext $P = P_0 \mid\mid \cdots \mid\mid P_{m-2} \mid\mid P_{m-1}$, where $P_{m-2} \oplus P_{m-1} = \mathit{\Delta}_{\text{out}}$. With the same reasoning and success probability as before, the combined contribution to the checksum of P_{m-2} and P_{m-1} is 0, since, like in the previous attack, $P_{m-2} \oplus S_{m-2} = P_{m-1} \oplus S_{m-1}$, and therefore $\mathsf{PF}_{K_2}(S_0 \oplus A_0) \oplus \mathsf{PF}_{K_2}(S_1 \oplus A_1) = 0$.

We can now produce a forgery by truncating the last two ciphertext blocks, since the contribution of the corresponding plaintext blocks to the checksum and therefore the tag is 0, and the number of blocks does not influence the tag.

Reordering Ciphertext. For their submission to the NIST Lightweight Cryptography standardization project, the designers of FlexAEAD updated their design from the previous version FlexAE in order to include associated data and prevent trivial reordering attacks. In this section, we show a forgery based on reordering ciphertexts of a chosen-plaintext query. Again, this attack is based on the same property of the sequence S as the two previous attacks.

Consider a chosen plaintext $P = P_0 \mid\mid P_1$, where $P_0 \oplus P_1 = \mathit{\Delta}_{\text{out}}$, and the corresponding ciphertext $C = C_0 \mid\mid C_1$ and tag T. As before, this difference of 1 in the block index results in the differential characteristics depicted in Figure 5. In FlexAEAD, the sequence values S_0 and S_1 are added at two points during the encryption process, so that the internal difference $\mathit{\Delta}_{\text{out}}$ propagates as shown in Figure 6. By now swapping the ciphertext blocks C_0 and C_1, we have a valid forgery using the original tag T. If the sequence generation followed the chosen characteristic, the two swapped ciphertext blocks will again have a checksum contribution of 0 during the decryption process. However, the resulting plaintext blocks are unpredictable.

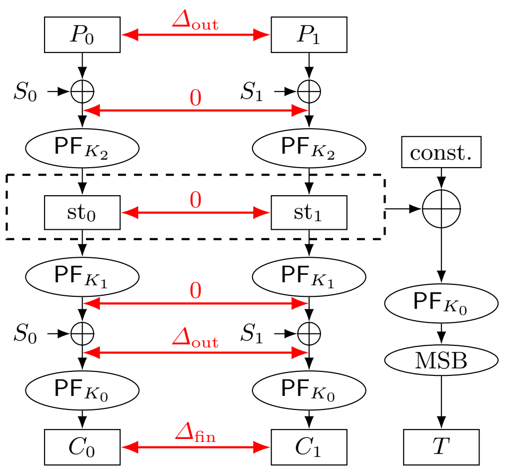


Fig. 6: Propagation of differences in the FlexAEAD encryption function, assuming $S_0 \oplus S_1 = \Delta_{\mathrm{out}}$.

4 Discussion and Further Observations on the Mode

The forgery attacks proposed in Section 3 exploit high-probability differential characteristics for the primitive PF, and are best prevented by increasing the number of rounds or replacing this primitive entirely. We remark that the designers were aware of the low bounds for PF, but argued that only the bounds for the multiple application $\mathsf{PF}^3 = \mathsf{PF} \circ \mathsf{PF} \circ \mathsf{PF}$ as used to compute C_i from P_i are relevant. As demonstrated in Section 3, this is not the case.

In the following, we discuss additional issues with the FlexAEAD mode of operation. These issues are independent of the underlying primitive PF and can be fixed with small tweaks to the mode or by phrasing the security claim for more restrictive message lengths. Finally, we discuss the applicability of our results to FlexAE and their practical verification.

4.1 Domain Separation and Length Issues

Domain Separation between Associated Data and Plaintext. In Figure 3, observe that the first step of the encryption (i.e., producing part of the checksum) is exactly the same for associated data and the plaintext. Using this observation, we can create a trivial forgery with probability 1 by redeclaring some part of the plaintext to be associated data instead. As an example, given a nonce-associated data-ciphertext-tag tuple (N, A, C, T) with a known plaintext $P = P_0 \mid\mid P_1$ (and a corresponding ciphertext $C = C_0 \mid\mid C_1$), we can craft a second valid tuple (N^*, A^*, C^*, T^*) with probability 1 by setting

$$N^* = N\,, \quad A^* = A \mid\mid P_0\,, \quad C^* = C_1\,, \quad T^* = T\,.$$

This forgery attack works for all versions of FlexAEAD.

Zero-Length Associated Data and Plaintext. During encryption, the nonce is used to generate the sequence S for each block of associated data and plaintext. If the combined length of associated data and plaintext is 0, the sequence is never used during encryption at all and the final tag does not depend on the nonce. Thus, a forgery is obtained by querying the static tag for empty associated data and plaintext under an arbitrary nonce and then combining it with a different nonce.

To fix this issue, the nonce needs to be included in the computation of the tag, for example by prepending the nonce N before the first associated data block A_0 in the associated data processing phase.

Padding of Associated Data. The associated data is padded using zeros, but the original length of the last associated data block does not influence the tag. Thus, there is no way to distinguish between valid associated data ending in 0 and a padded associated data block. In contrast, the padding of the plaintext has an influence on the final tag value; it appears this was omitted for associated data by mistake, and can easily be fixed. This issue was also observed by Mège [7].

4.2 Other Observations

Overflow of the Internal Counter. During the generation of the sequence S, an internal state is updated repeatedly using the `INC32` function. The internal state repeats after 2^{32} calls, therefore limiting the size of the encrypted payload to 2^{32} blocks. Otherwise, if the associated data or plaintext length is larger than 2^{32}, any two blocks $(A_i, A_{i+2^{32}})$ or $(C_i, C_{i+2^{32}})$ can be swapped to produce a forgery.

This can be addressed by either explicitly imposing a corresponding data length limit as part of the security claim, or by choosing a counter size that does not overflow within the data length limit.

4.3 Applicability to FlexAE

FlexAE, published at IEEE ICC 2017 [9], is the predecessor design of FlexAEAD. It features a slightly simpler mode that omits the step $\mathsf{PF}_{K_1}(\cdot) \oplus S_j$ in the computation of ciphertext block C_j and does not support associated data A. The primitive also shows minor differences, such as 3 slightly different S-boxes, which have no significant impact on the security analysis. The additional steps in FlexAEAD were added by the designers to fix problems in FlexAE; in particular, the ciphertext reordering attack of Section 3.2 works with probability 1 for FlexAE.

FlexAE even permits forgeries with zero encryption queries, as illustrated in Figure 7. The following is a forgery with probability about 2^{-54} for FlexAE-64-128 (or, with similar characteristics, 2^{-86} for FlexAE-128-256, 2^{-150} for FlexAE-256-512, or 2^{-278} for FlexAE-512-1024): Take an arbitrary nonce N and single-block ciphertext C, and select $T = C \oplus \Delta_{\text{out}}$ with $\Delta_{\text{out}} = \texttt{xx000000 zz000000}$, where xx is an arbitrary nonzero difference and zz is the high-probability S-box output difference with $\mathbb{P}[\texttt{xx} \rightarrow \texttt{zz}] = 2^{-6}$. This works based on the differential with input difference $\Delta_{\text{in}} = \texttt{10101010 10101010}$ for PF_{K_0} due to the constant addition between the computation of tag T and a single-block ciphertext C. The actual success probability is likely higher; there are several similar clusters contributing to the same differential (see Figure 7), and the alternative padding constant $(\texttt{01})^*$ instead of $(\texttt{10})^*$ gives an alternative compatible Δ_{in} to double the probability.

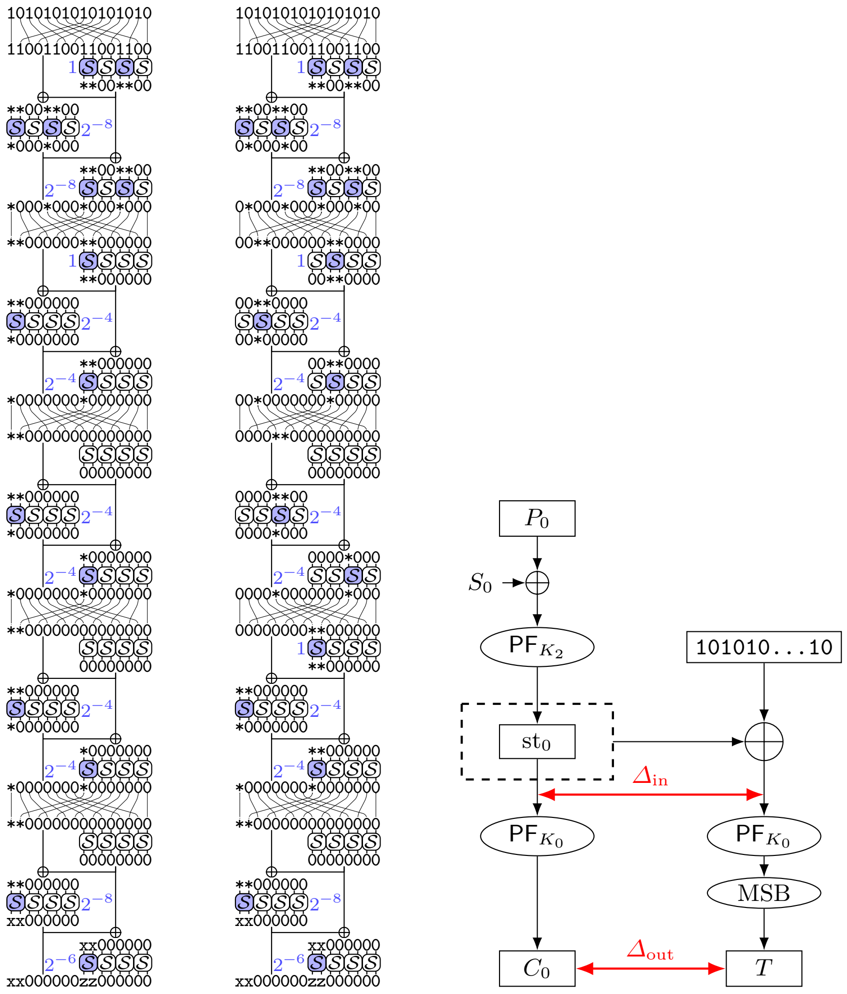


Fig. 7: Zero-query forgery for full FlexAE-64-128 with clusters of probability 2^{-54}.

4.4 Practical Verification

All our practical tests use the reference implementation of FlexAEAD. We successfully verified the domain separation issue (Section 4.1) using the full-round version, and we could also confirm that for inputs with zero-length associated data and plaintext blocks, the final tag does not depend on the nonce (Section 4.2).

Moreover, we practically verified the estimated probabilities of our differential characteristics for reduced-round versions of both FlexAEAD-128 and FlexAEAD-256. More specifically, on average we are able to find the required output difference for FlexAEAD-128 with 3-round PF_K using about 2^{30} samples, and that for FlexAEAD-256 with 2-round PF_K using about 2^{24} samples, as expected.

5 Conclusion

In this short note, we showed several forgery attacks against FlexAEAD (also applying to its predecessor FlexAE). Except the trivial forgery based on domain separation issues, these forgery variants are based on high-probability clusters of differential characteristics in the generation of the internal sequence S. The resulting success probabilities per forgery attempt are summarized as follows. Using a single encryption query with a fixed difference between two consecutive associated data or plaintext blocks, a forgery attempt with swapped or truncated blocks is successful with probability about 2^{-46} (FlexAEAD-64), 2^{-54} (FlexAEAD-128), or 2^{-70} (FlexAEAD-256). Furthermore, we proposed forgery attacks on FlexAE with zero encryption queries and arbitrary single-block ciphertexts with success probability about 2^{-54} (FlexAE-64-128), 2^{-86} (FlexAE-128-256), 2^{-150} (FlexAE-256-512), or 2^{-278} (FlexAE-512-1024). While increasing the number of rounds of the permutation PF or the number of calls to PF during sequence generation would combat these attacks, this also comes with a corresponding performance impact. Additionally, due to the fragile nature of the tag generation procedure, similar forgery attacks might still exist.

References

1. Daemen, J., Rijmen, V.: The Design of Rijndael: AES – The Advanced Encryption Standard. Information Security and Cryptography, Springer (2002), `https://doi.org/10.1007/978-3-662-04722-4`
2. Eichlseder, M., Kales, D.: Clustering related-tweak characteristics: Application to MANTIS-6. IACR Transactions on Symmetric Cryptology **2018**(2), 111–132 (2018), `https://doi.org/10.13154/tosc.v2018.i2.111-132`
3. Eichlseder, M., Kales, D., Schofnegger, M.: OFFICIAL COMMENT: FlexAEAD. Posting on the NIST LWC mailing list, `https://groups.google.com/a/list.nist.gov/d/msg/lwc-forum/cRjs9x43G2I/KsBQLdDODAAJ`
4. Jutla, C.S.: Encryption modes with almost free message integrity. In: Pfitzmann, B. (ed.) Advances in Cryptology – EUROCRYPT 2001. LNCS, vol. 2045, pp. 529–544. Springer (2001), `https://doi.org/10.1007/3-540-44987-6_32`

5. Jutla, C.S.: Encryption modes with almost free message integrity. Journal of Cryptology **21**(4), 547–578 (2008), `https://doi.org/10.1007/s00145-008-9024-z`
6. Kam, J.B., Davida, G.I.: Structured design of substitution-permutation encryption networks. IEEE Transactions on Computers **28**(10), 747–753 (1979), `https://doi.org/10.1109/TC.1979.1675242`
7. Mège, A.: OFFICIAL COMMENT: FlexAEAD. Posting on the NIST LWC mailing list, `https://groups.google.com/a/list.nist.gov/d/msg/lwc-forum/DPQVEJ5oBeU/YXW0QjfjBQAJ`
8. do Nascimento, E.M.: Algoritmo de Criptografia Leve com Utilização de Autenticação. Ph.D. thesis, Instituto Militar de Engenharia, Rio de Janeiro (2017), `http://www.comp.ime.eb.br/pos/arquivos/publicacoes/dissertacoes/2017/2017-Eduardo.pdf`
9. do Nascimento, E.M., Xexéo, J.A.M.: A flexible authenticated lightweight cipher using Even-Mansour construction. In: IEEE International Conference on Communications – ICC 2017. pp. 1–6. IEEE (2017), `https://doi.org/10.1109/ICC.2017.7996734`
10. do Nascimento, E.M., Xexéo, J.A.M.: A lightweight cipher with integrated authentication. In: Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais – SBSEG. pp. 25–32. Sociedade Brasileira de Computação (2018), `https://portaldeconteudo.sbc.org.br/index.php/sbseg_estendido/article/view/4138`
11. do Nascimento, E.M., Xexéo, J.A.M.: FlexAEAD. Submission to Round 1 of the NIST Lightweight Cryptography Standardization process (2019), `https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/FlexAEAD-spec.pdf`
12. National Institute of Standards and Technology (NIST): Lightweight cryptography standardization process (2019), `https://csrc.nist.gov/projects/lightweight-cryptography`
13. Rahman, M., Saha, D., Paul, G.: Attacks against FlexAEAD. Posting on the NIST LWC mailing list, `https://groups.google.com/a/list.nist.gov/d/msg/lwc-forum/VLWtGnJStew/X3Fxexg1AQAJ`
14. Rahman, M., Saha, D., Paul, G.: Interated truncated differential for internal keyed permutation of FlexAEAD. IACR Cryptology ePrint Archive, Report 2019/539 (2019), `https://eprint.iacr.org/2019/539`