Forgery and Subkey Recovery on CAESAR candidate iFeed

Willem Schroé, Bart Mennink, Elena Andreeva, and Bart Preneel

Dept. Electrical Engineering, ESAT/COSIC, KU Leuven, and iMinds, Belgium
firstname.lastname@esat.kuleuven.be

Abstract. iFeed is a blockcipher-based authenticated encryption design by Zhang, Wu, Sui, and Wang and a first round candidate to the CAESAR competition. iFeed is claimed to achieve confidentiality and authenticity in the nonce-respecting setting, and confidentiality in the nonce-reuse setting. Recently, Chakraborti et al. published forgeries on iFeed in the RUP and nonce-reuse settings. The latter attacks, however, do not invalidate the iFeed designers' security claims. In this work, we consider the security of iFeed in the nonce-respecting setting, and show that a valid forgery can be constructed after only one encryption query. Even more, the forgery leaks both subkeys $E_K(0^{128})$ and $E_K(PMN\|1)$, where K is the secret key and PMN the nonce used for the authenticated encryption. Furthermore, we show how at the price of just one additional forgery one can learn $E_K(P^*)$ for any freely chosen plaintext P^*. These design weaknesses allow one to decrypt earlier iFeed encryptions under the respective nonces, breaking the forward secrecy of iFeed, and leading to a total security compromise of the iFeed design.

Keywords: CAESAR, iFeed, Forgery, Subkey Recovery, Breaking Forward Secrecy.

1 Introduction

The CAESAR [3] competition was launched in 2014 to select a portfolio of recommended robust and efficient authenticated encryption algorithms by 2018. CAESAR is much in the spirit of the Advanced Encryption Standard [1] (AES) and SHA-3 hash [5] algorithm competitions, which were organized by the American institute of standards and technology (NIST). The main goal of this type of competitions is that they allow for public discussions, analysis, and therefore more comprehensive, secure and collectively agreed upon choices of cryptographic algorithms. In March, 2014, CAESAR received 57 submissions. In July 2015, 30 of them have advanced to the second round.

In this work we analyze the security of the first round CAESAR submission iFeed[AES] v1 – iFeed for short – by Zhang et al. [6]. iFeed is an AES blockcipher-based design which combines PMAC-style authentication with dedicated encryption. Below we will treat iFeed generically, and consider it to be based on an arbitrary blockcipher E. iFeed processes the data in an on-line manner and it is inverse-free, meaning that both encryption and decryption only use

the block cipher in forward direction. The design is inherently nonce-based: the authors claim *and* prove confidentiality and authenticity of iFeed in a setting where the adversary is not allowed to make repeated queries under the same nonce. We refer to this setting and type of adversary as *nonce-respecting*. The design is moreover claimed to achieve confidentiality under some conditions also in the *nonce-reuse* setting.

iFeed uses the secret key K to derive two subkeys: a nonce-independent $Z_0 = E_K(0^{128})$ and its multiples $Z_i = 2^i \cdot Z_0$, and a nonce-dependent $U = E_K(PMN \| 10^*)$ where PMN is a variable public message number (nonce) and 10^* is the padding to a full 128-bit block with one 1 bit and appropriate number of 0 bits. The processing of the associated data and of the message are done independently of each other, both resulting in a subtag. The XOR of the two subtags produces the final tag. The general encryption and decryption procedures of iFeed are given in Figures 1 and 2, respectively. The processing of the associated data is distinctively independent of the nonce: the inputs to the blockcipher E are masked only using Z_i. The data is encrypted using both the Z_i and U. One design choice of iFeed is that the computation of the subtags is performed using the same subkeys: Z_1 or Z_2 for the associated data, and $Z_1 \oplus U$ and $Z_2 \oplus U$ for the plaintext.[1]

Chakraborti et al. [4] recently presented forgeries on iFeed both in the nonce-reuse setting, and in a setting where the adversary is granted access to ciphertext decryptions irrespective of the verification result, also known as the *release of unverified plaintext* (RUP) setting from [2]. The iFeed designers, however, do not claim any security against these properties, and the attacks from Chakraborti et al. do not invalidate the security of iFeed.

Our Contribution

In this paper, we present a simple forgery attack on iFeed in the nonce-respecting model. Our attack exploits the unfortunate repetition of the Z_1 (respectively Z_2) subkeys at the finalization of the associated data and plaintext. Our attack on iFeed leads to a total security break and also invalidates the security claims and proofs posited by the designers of iFeed. The attack uses only one encryption query with no associated data and an arbitrary n-bit (or single block) plaintext. Then, we show how to construct a valid forgery with n-bit associated data and $2n$-bit ciphertext.

As a consequence of our attack the values of the subkeys Z_0 and U are also leaked. These subkeys are valuable information as they can be used to recover plaintext from old encryptions, even though they were performed under a different nonce. Namely, we show how at the price of just a single additional forgery, one learns $E_K(P^*)$ for any plaintext P^*. More concretely, the extra forgery allows an attacker to learn the $U' = E_K(PMN' \| 10^*)$ values for some earlier used nonce PMN'. Once U' is obtained, the ciphertext with PMN' for

[1] Z_1 is used if the last associated data block is fractional; Z_2 is used otherwise. Similarly for $Z_1 \oplus U$ and $Z_2 \oplus U$ with respect to the last plaintext block.

any P^* can be retrieved: the inputs to E_K in the decryption algorithm can be computed offline (using Z_i, U', and the ciphertext), and forgeries can be performed to find out the plaintexts one by one.

Our forgeries not only invalidate the security claim and proof of iFeed [6], but they also break its forward secrecy. To gain a better understanding of the presented iFeed security weaknesses, we conclude by revisiting and pointing out the errata in the security proof of iFeed.

Outline

We introduce iFeed in Section 2. Our forgery and subkey recovery attack on iFeed is given in Section 3. Next, we show how the subkeys can be used to learn $E_K(P^*)$ for any P^* in Section 4. We elaborate on the key properties that caused this attack to work and look back at the proof of iFeed in Section 5. The paper is concluded in Section 6.

2 iFeed

iFeed [6] is a blockcipher-based authenticated encryption scheme. It is originally specified using the AES-128 blockcipher. Our attacks are generic and do not exploit any weakness of AES-128. Therefore, from now we simply describe iFeed based on any blockcipher $E : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$, where $n = 128$. iFeed is on-line, it only uses E in forward direction (inverse-free), and it allows for parallelized encryption.

The scheme is keyed via a key $K \in \{0,1\}^n$. It operates on public message numbers PMN of size between 1 and 127 bits, associated data A in $\{0,1\}^{\leq |A|_{\max}}$, and plaintexts/ciphertexts P/C from $\{0,1\}^{\leq |P|_{\max}}$, where $|A|_{\max}$ and $|P|_{\max}$ are some large values which sum to at most $2^{71} - 512$. The tag T is of size $32 \leq \tau \leq n = 128$. In the nonce-respecting setting, the public message number is required to be unique for every query to the iFeed encryption function $\mathcal{E}$. The iFeed encryption $\mathcal{E}$ and decryption $\mathcal{D}$ functions are depicted in Figure 1 and 2, respectively. Here, $\mathrm{Pad}(X)$ equals X if $|X| = n$ and $X \| 10^{n-1-|X|}$ if $|X| < n$, and the usage of Z_1 versus Z_2 (resp. Z'_1 versus Z'_2) depends on the last block of A (resp. P): Z_2/Z'_2 is used if the last data block is of size n bits, Z_1/Z'_1 is used if the last block is fractional. We remark that our attacks use integral blocks only, for which $\mathrm{Pad}(X) = X$ and Z_2 is used instead of Z_1.

For the presentation of the attacks, however, it suffices to only discuss a simplified version of iFeed. In more detail, in our attacks we will only query $\mathcal{E}$ on input of n-bit plaintext and no associated data. We also make the forgery queries to $\mathcal{D}$ for either n or $2n$-bit associated data and $2n$-bit ciphertext. All queries consider 127-bit PMN and tag size $\tau = n = 128$. In Algorithms 1 and 2, we give a formal description of iFeed's $\mathcal{E}$ and $\mathcal{D}$, respectively, for the input sizes relevant for our attacks. We refer to [6] for the general description of iFeed, and stress that our attacks easily translate to the general case.

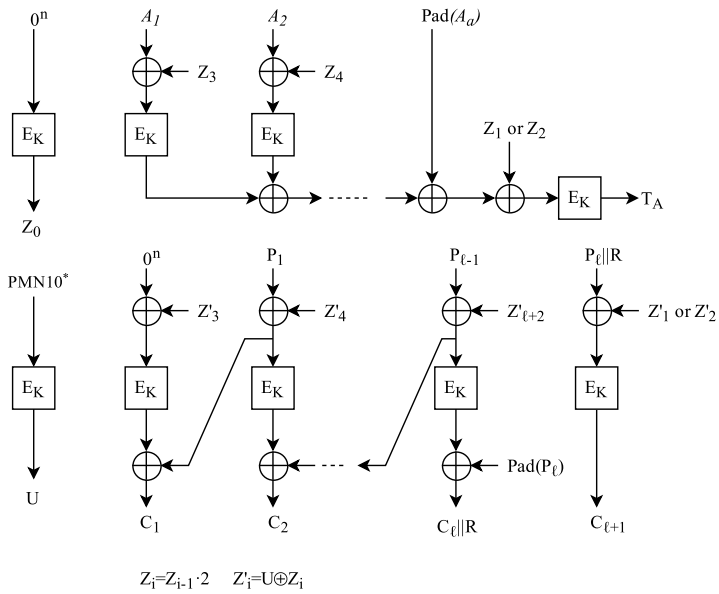

Fig. 1: iFeed encryption $\mathcal{E}$. All wires represent n-bit values. The output is the ciphertext $C_1 \cdots C_\ell$ and the tag $T = \mathrm{left}_\tau(T_A \oplus C_{\ell+1})$

The iFeed mode makes use of secret subkey $Z_0 = E_K(0^{128})$ which is used to derive additional subkeys $Z_i = 2 \cdot Z_{i-1}$. Here, the multiplication is performed in the binary Galois Field $\mathrm{GF}(2^{128})$ defined by the primitive polynomial $x^{128} + x^7 + x^2 + x + 1$.

3 Forgery and Subkey Recovery Attack on iFeed

Let $K \xleftarrow{\$} \{0,1\}^n$ be the secret key and consider $\tau = n$. We present our forgery attack on iFeed. It consists of one encryption query and the forgery itself. Upon successful verification, the forgery will disclose the subkeys $Z_0 = E_K(0^{128})$ and $U = E_K(PMN\|1)$.

- Fix arbitrary $PMN \in \{0,1\}^{127}$ and arbitrary $P_1 \in \{0,1\}^{128}$, and make **encryption query** with no associated data $A = \varepsilon$:

$$(C_1, T) = \mathcal{E}_K(PMN, \varepsilon, P_1)\,;$$

- Write $C_1' = C_1 \oplus P_1 \oplus PMN\|1$, and fix an arbitrary $C_2' \in \{0,1\}^{128}$. Set $A = C_2'$, $T' = 0^{128}$, and output **forgery**:

$$\mathcal{D}_K(PMN, A, C_1'C_2', T')\,.$$

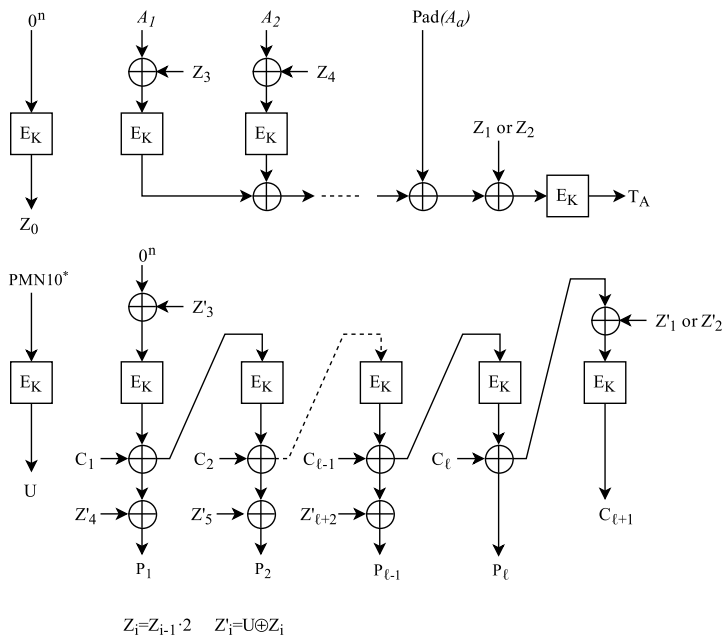

Fig. 2: iFeed decryption $\mathcal{D}$. All wires represent n-bit values. The output is the plaintext $P_1 \cdots P_\ell$ when T is $\mathrm{left}_\tau(T_A \oplus C_{\ell+1})$

We next demonstrate that the forgery attempt is successful. First, regarding the encryption query, note that

$$C_1 = E_K(Z_3 \oplus U) \oplus P_1 \,. \tag{1}$$

Now, verification of the forgery (cf. Algorithm 2) succeeds if $T' = T'_A \oplus C'_3 = 0^{128}$. Note that we have

$$\begin{aligned}
P'_1 &= E_K(Z_3 \oplus U) \oplus C'_1 \oplus Z_4 \oplus U \\
&= PMN\|1 \oplus Z_4 \oplus U \,, \\
P'_2 &= E_K(P'_1 \oplus Z_4 \oplus U) \oplus C'_2 \\
&= E_K(PMN\|1) \oplus C'_2 = U \oplus C'_2 \,, \\
C'_3 &= E_K(P'_2 \oplus Z_2 \oplus U) \\
&= E_K(C'_2 \oplus Z_2) \,.
\end{aligned}$$

As we defined $A = C'_2$, this yields successful verification:

$$T'_A = E_K(A \oplus Z_2) = E_K(C'_2 \oplus Z_2) = C'_3 \,.$$

As verification is successful, $\mathcal{D}$ returns $P'_1 P'_2$. The values $U = P'_2 \oplus C'_2$ and $Z_0 = 2^{-4}(P'_1 \oplus PMN\|1 \oplus U)$ are directly obtained.

Algorithm 1 iFeed $\mathcal{E}$ for $|PMN| = 127$, $|A| = 0$, and $|P_1| = n$

$Z_0 = E_K(0^{128})$
for $i = 1, \ldots, 3$ **do**
    $Z_i = 2 \cdot Z_{i-1}$
$U = E_K(PMN \| 1)$
$T_A = 0^{128}$
$C_1 = E_K(Z_3 \oplus U) \oplus P_1$
$C_2 = E_K(P_1 \oplus Z_2 \oplus U)$
return $(C, T) = (C_1, T_A \oplus C_2)$

Algorithm 2 iFeed $\mathcal{D}$ for $|PMN| = 127$, $|A'| = n$ or $2n$, and $|C'| = 2n$

$Z_0 = E_K(0^{128})$
for $i = 1, \ldots, 4$ **do**
    $Z_i = 2 \cdot Z_{i-1}$
$U = E_K(PMN \| 1)$
if $|A'| = n$ **then**
    define $A'_1 = A'$
    $T'_A = E_K(A'_1 \oplus Z_2)$
else
    parse $A'_1 A'_2 = A'$
    $T'_A = E_K(E_K(A'_1 \oplus Z_3) \oplus A'_2 \oplus Z_2)$
parse $C'_1 C'_2 = C'$
$P'_1 = E_K(Z_3 \oplus U) \oplus C'_1 \oplus Z_4 \oplus U$
$P'_2 = E_K(P'_1 \oplus Z_4 \oplus U) \oplus C'_2$
$C'_3 = E_K(P'_2 \oplus Z_2 \oplus U)$
if $T' = T'_A \oplus C'_3$ **then**
    return $P' = P'_1 P'_2$
else
    return $\perp$

4 Finding $E_K(P^*)$ for Any Plaintext P^*

Below, we construct an additional forgery, which is based on the information gained from the main forgery attack. Namely, the goal is to use the knowledge of the subkeys U, Z_0, and the value $E_K(Z_3 \oplus U) = C_1 \oplus P_1$ from (1), to produce another forgery, which would allow one to learn $E_K(P^*)$ for any plaintext data $P^* \in \{0,1\}^n$.

- Let PMN be as before, define $(A''_1, A''_2) = (P^* \oplus Z_3, U)$, $(C''_1, C''_2) = (E_K(Z_3 \oplus U) \oplus P^*, 0^{128})$, and $T = 0^{128}$, and output **forgery**:

$$\mathcal{D}_K(PMN, A''_1 A''_2, C''_1 C''_2, T'') \,.$$

The verification (cf. Algorithm 2) is successful if $T'' = T''_A \oplus C''_3 = 0^{128}$. Note that we have

$$\begin{aligned} P''_1 &= E_K(Z_3 \oplus U) \oplus C''_1 \oplus Z_4 \oplus U \\ &= P^* \oplus Z_4 \oplus U\,, \\ P''_2 &= E_K(P''_1 \oplus Z_4 \oplus U) \oplus C''_2 \\ &= E_K(P^*)\,, \\ C''_3 &= E_K(P''_2 \oplus Z_2 \oplus U) \\ &= E_K(E_K(P^*) \oplus Z_2 \oplus U)\,. \end{aligned}$$

On the other hand, T''_A is computed from the two-block $(A''_1 A''_2)$ as follows

$$\begin{aligned} T''_A &= E_K(E_K(A''_1 \oplus Z_3) \oplus A''_2 \oplus Z_2) \\ &= E_K(E_K(P^*) \oplus Z_2 \oplus U)\,, \end{aligned}$$

thus $C''_3 = T''_A$, and the verification is successful. The resulting plaintext satisfies $P''_2 = E_K(P^*)$. This attack works for any n-bit data block P^*, it can for instance be used to recover the subkeys of older iFeed encryptions (by putting $P^* = PMN' \| 10^* \neq PMN \| 1$), and indirectly to decrypt earlier encryptions without having possession of the key K.

5 Why the Attack Works

The attack of Section 3 is possible due to two main iFeed properties:

1. iFeed uses Z_2 as masking in both the T'_A and C'_{l+1};
2. The second last ciphertext block (C_1 in our encryption example) is not masked with $Z_4 \oplus U$ as other ciphertext blocks.

Using these properties, the forgery is constructed in such a way upon decryption, $PMN \| 1$ is directly fed into E_K and the term U in the final mask $Z_2 \oplus U$ is canceled out.

The original submission document of iFeed [6] comes with an authenticity proof of security in the nonce-respecting setting. Our attack of Section 3, however, shows that this security claims is invalid. At a high level, the flaw is caused by an oversight that the subkeys for the associated data and the plaintext are dependent. Indeed, the associated data is masked with $Z_1, \ldots, Z_{a+2}$ and the plaintext with $Z_1 \oplus U, \ldots, Z_{\ell+2} \oplus U$ (where a and ℓ denote the number of associated data and plaintext blocks). For the decryption query, the proof claims that both T_A and $C_{\ell+1}$ is randomly generated except with a small probability.[2] These two cases independently of each other rely on the randomness of Z_0. In our attacks, T_A and $C_{\ell+1}$ are, indeed, both newly and randomly generated. However,

[2] In fact, it is claimed that P_{w+1} is random, where w is the first block in the forgery that is different from the older encryption query with the same nonce, and that all subsequent values $P_{w+2}, \ldots, P_\ell, C_{\ell+1}$ are random.

their drawing is not independent, in fact, they satisfy $T_A = C_{\ell+1}$. Unfortunately, this is what in our analysis indicates a security problem which as exemplified leads to serious security problems.

6 Conclusion

Our attacks completely disprove and break the authenticity of the iFeed authenticated encryption scheme. In more detail, if an adversary has access to $\mathcal{D}$, it can forge *and* learn the secret subkeys. We furthermore show that with the knowledge of these data, an attacker can use $\mathcal{D}$ to decrypt earlier ciphertexts, even if they were encrypted under a different nonce (because of Sect. 4). On the other hand, it appears that in the absence of a decryption mechanism, the confidentiality (CPA security) of iFeed still stands.

As a possible remedy to the design of iFeed and future work we suggest the exploration of the possibilities for applying different masks. One option may be to use $Z_i \oplus U$ as is for plaintext encryption but $3 \cdot Z_i$ for the associated data. We, however, do not recommend the replacement of nonce encryptions with encryptions of nonces masked with Z values since the same type of weaknesses surfaces.

Acknowledgments. The authors would like to thank Liting Zhang for verifying the attack. This work was supported in part by European Union's Horizon 2020 research and innovation programme under grant agreement No 644052 HECTOR and grant agreement No H2020-MSCA-ITN-2014-643161 ECRYPT-NET, and in part by the Research Council KU Leuven: GOA TENSE (GOA/11/007). Elena Andreeva and Bart Mennink are Postdoctoral Fellows of the Research Foundation – Flanders (FWO).

References

1. Announcing the Advanced Encryption Standard (AES) (October 2001), Federal Information Processing Standards Publication 197. United States National Institute of Standards and Technology (NIST)
2. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., Yasuda, K.: How to Securely Release Unverified Plaintext in Authenticated Encryption. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part I. Lecture Notes in Computer Science, vol. 8873, pp. 105–125. Springer (2014)
3. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness (May 2014), `http://competitions.cr.yp.to/caesar.html`
4. Chakraborti, A., Datta, N., Minematsu, K., Gupta, S.S.: Forgery on iFeed[AES] in RUP and Nonce-Misuse Settings (2015), CAESAR mailing list
5. SHA-3 Competition (2007-2012), `http://csrc.nist.gov/groups/ST/hash/sha-3/`, United States National Institute of Standards and Technology (NIST)
6. Zhang, L., Wu, W., Sui, H., Wang, P.: iFeed[AES] v1 (2014), submission to CAESAR competition