Cryptanalysis of the Authenticated Encryption Algorithm COFFE

Ivan Tjuawinata(✉), Tao Huang, and Hongjun Wu

Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore, Singapore
S120015@e.ntu.edu.sg, {huangtao,wuhj}@ntu.edu.sg

Abstract. COFFE is a hash-based authenticated encryption scheme. In the original paper, it was claimed to have IND-CPA security and also ciphertext integrity even in nonce-misuse scenario. In this paper, we analyse the security of COFFE. Our attack shows that even under the assumption that the primitive hash function is ideal, a valid ciphertext can be forged with 2 enquiries with success probability close to 1. The motivation of the attack is to find a collision on the input of each of the hash calls in the COFFE instantiation. It can be done in two ways.

The first way is by modifying nonce and last message block size. Chosen appropriately, we can ensure two COFFE instantiations with different nonce and different last message block size can have exactly the same intermediate state value. This hence leads to a valid ciphertext to be generated. Another way is by considering two different COFFE instantiations with different message block size despite same key. In this case, we will use the existence of consecutive zero in the binary representation of π to achieve identical intermediate state value on two different COFFE instantiations. Having the state collisions, the forgery attack is then conducted by choosing two different plaintexts with appropriate nonce and tag size to query. Having this fact, without knowing the secret key, we can then validly encrypt another plaintext with probability equal to 1.

Keywords: COFFE · Authenticated cipher · Forgery attack

1 Introduction

Authenticated encryption is a symmetric encryption scheme aiming to provide authenticity at the same time as confidentiality to the message. Initially, Bellare and Namprempre proposed the authenticated encryption(AE) schemes by integrating an encryption scheme with an authentication scheme in 2000, [1]. In 2001, Krawczyk published a paper [8] that studies the possibility to solve this problem by applying the existing symmetric key cryptosystem and hash function one after another.

The difficulty of the general composition approach is although the security of the parts individually is well-studied, the application of one function may affect the security of the other. Furthermore, in implementation point of view, it is

O. Dunkelman and L. Keliher (Eds.): SAC 2015, LNCS 9566, pp. 510–526, 2016.
DOI: 10.1007/978-3-319-31301-6_29

not very efficient and error-prone considering it is required to have two different primitives, one for encryption, one for plaintext integrity.

To tackle the first difficulty, a lot of dedicated designs to simultaneously encrypt and authenticate the message have been proposed, among which the authenticated encryption mode is a commonly used design approach. Some examples of these mode of operations are IAPM [7], OCB [11], Jambu [13], GCM [5], CCM [4] and ELmD [3].

The consideration for the efficiency comes from the fact that encryption and authentication is done independently with each of their own primitive. So one way to solve this is to consider using the same primitive for both purposes. The initial direction that research goes was to construct a block-cipher based hash function for the authentication purpose such as the ones found in [9,10].

Another way to solve this problem is to purely use a hash function for both encryption and authentication purposes. Some of AE modes that is based on hash functions are OMD [2] and COFFE [6].

COFFE is a hash-function-based authenticated encryption scheme designed by Forler *et al.* It was published in ESC 2013 [6]. COFFE is designed to be secure for computationally constraint environment. As mentioned above, COFFE utilises a hash function for both encryption and authentication without introducing any block cipher primitive. According to [6], COFFE is one of the first authenticated encryption that is purely based on hash function. This alternative direction of constructing an authenticated encryption system is interesting for constructing a secure authenticated encryption.

The designers claim that COFFE is secure against chosen plaintext attack in nonce-respecting scenario. It is also claimed to have ciphertext-integrity even in nonce-misuse scenario. In particular, it is claimed that the ciphertext integrity of COFFE is at least strong as the indistinguishability of the hash function used. That is, forging a ciphertext with a valid tag should be as hard as finding collision in the underlying hash function. Furthermore, it also provides additional features. Firstly, it provides failure-friendly authenticity, that is, COFFE provides reasonable authenticity in the case of weaker underlying hash function. Secondly, it also provides side channel resistance under nonce-respecting scenario.

In this paper, we first analyse the design of COFFE. During the analysis we consider the scheme firstly under the nonce-repeating scenario. Instead of using any specific hash function for the underlying primitive, we analyse it on the generic construction case with an ideal underlying hash function. We show that under these settings, some instances of COFFE with particular parameters are vulnerable to distinguishing attack, ciphertext forgery attack, or related key recovery attack. Thus, the security claim of COFFE for these parameters does not hold.

The attacks come from the consideration that intermediate state values of two different COFFE instantiations can be made the same while having different inputs. The vulnerability comes from the fact that having most of the parameters to be variables, different set of parameters can be chosen and combined to create the collision. The attack starts by first trying to find a specific value for the parameters where this can happen. Having found these parameters, different approaches are made to exploit this discovery to launch either distinguishing attack, forgery attack, or key recovery attack. In this paper, we found that for the distinguishing and forgery attack, if we use the same secret key for all the instantiations, the success probability is close to 1.

The rest of this paper is structured as follows: The generic specification of COFFE is given in Sect. 2. Section 3 provides some analysis and observation of COFFE. Section 4 introduces the distinguishing attack. Section 5 provides two variants of ciphertext forgery attack. We proposed a related key recovery attack on Sect. 6. Lastly, Sect. 7 concludes the paper.

2 The COFFE Authenticated Cipher

The COFFE family of authenticated ciphers uses six parameters: key length, nonce length, block size, hash function input and output size, and tag length. We will briefly describe the specification of COFFE authenticated cipher. The full specification can be found in [6]. An overview of COFFE is provided in Fig. 1.

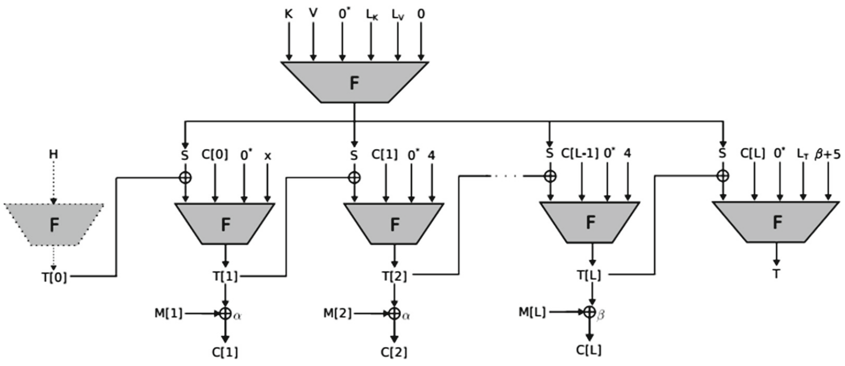

Fig. 1. General scheme of COFFE encryption and authentication (Fig. 2 of [6])

2.1 Notations

Throughout this paper, we will be using the following notations:

- $\mathcal{F}$: Underlying Hash function
  - γ: Input size for $\mathcal{F}$ assuming "one compression function invocation per hash function call"
  - δ: Output size for $\mathcal{F}$
- $\mathcal{L}_K$: Secret key length expressed in bits. The length of this string should be a multiple of a byte

- $\mathcal{L}_V$: Nonce length expressed in bits. The length of this string should be a multiple of a byte
- $\mathcal{L}_T$: Tag length expressed in bits. The length of this string should be a multiple of a byte, $\mathcal{L}_T \leq \delta$
- α: Message block size
- β: Last message block size, $\beta \leq \alpha \leq \delta$
- x: Bits for domain value. The number of byte for used for x follows the number of bytes needed to express $\beta + 5$ in bits.
- Let v be a binary string and b be a positive integer.
  - $|v|$: The length of v in bits.
  - $|v|_b$: A $b-$bit binary representation of v.
  - $[v]$: The length of v in byte.
  - $[v]_b$: A $b-$byte binary representation of v.
  - $\mathfrak{b}$: $[\beta + 5]$.
- $\mathcal{S}_1||\mathcal{S}_2$: Concatenation of string $\mathcal{S}_1$ followed by $\mathcal{S}_2$.
- $\mathcal{S}_1 \bigoplus_\ell \mathcal{S}_2$: The ℓ- bit string obtained by XOR-ing the ℓ least significant bits of $\mathcal{S}_1$ and $\mathcal{S}_2$.
  - $\mathcal{S}_1 =_b \mathcal{S}_2$: The last b bits of both $\mathcal{S}_1$ and $\mathcal{S}_2$ is the same.
- $\mathcal{S}_1||0^\star||\mathcal{S}_2$: When clear the total length should be, say a bits, concatenate $\mathcal{S}_1$ with 0-bits then with $\mathcal{S}_2$ with the number of 0-bits being the difference between a and the total length of $\mathcal{S}_1$ and $\mathcal{S}_2$.
- $\mathcal{K}$: Secret key string
- $\mathcal{V}$: Nonce
- $\mathcal{L}$: The number of message blocks for the encryption
- $\mathcal{S}$: Session Key with length δ bits
- $\mathcal{H}$: Associated Data
- $\mathcal{M}[i], 1 \leq i \leq \mathcal{L}$: The i-th message block, an α bit string except for $\mathcal{M}[\mathcal{L}]$ having length β bits.
- $\mathcal{C}[0]$: The initial vector
- $\mathcal{C}[i], 1 \leq i \leq \mathcal{L}$: The i-th ciphertext block with the same length as $\mathcal{M}[i]$
- $\mathcal{T}[i], 0 \leq i \leq L$: Chaining values for the scheme each of which having length δ bits
- $\mathcal{T}$: Message Tag.

2.2 Associated Data Processing

The method of processing the associated data, $\mathcal{H}$, can be divided into three cases based on the length of the associated data.

- If the length of $\mathcal{H}$ is less than δ bits, it is appended by 1 followed by appropriate number of zeros to reach δ bits. This is defined as $\mathcal{T}[0]$ and a domain value x is defined to be 1.
- If the length of $\mathcal{H}$ is exactly δ bits, this is directly defined as $\mathcal{T}[0]$ while the domain value x is set to be 2.
- If the length of $\mathcal{H}$ is more than δ bits, feed $\mathcal{H}$ to $\mathcal{F}$ and the resulting hash output is used as the value of $\mathcal{T}[0]$ and x is defined as 3.

2.3 Initialization

There are two values that need to be computed in the initialization phase, $\mathcal{S}$ and $\mathcal{C}[0]$. Firstly, the session key, $\mathcal{S}$ which is defined based on $\mathcal{K}, \mathcal{V}, \mathcal{L}_K, \mathcal{L}_V$, and $\mathfrak{b}$. The value of $\mathcal{S}$ is defined to be $\mathcal{F}(\mathcal{K}||\mathcal{V}||0^\star||\mathcal{L}_K||\mathcal{L}_V||[\mathbf{0}]_\mathfrak{b})$. Note that here $0^\star$ is used to pad the string to make the length equals to γ.

Next, the constant $\mathcal{C}[0]$ which depends only on the message block size α. $\mathcal{C}[0]$ is defined to be the first $\frac{\alpha}{4}$ post-decimal values of π interpreted as a hexadecimal string. So for example, since the decimal values of π is $.14159\ldots$, if $\alpha = 16$, Then $\mathcal{C}[0] = 0 \times 1415 = 0001010000010101$.

2.4 Processing Plaintext

Plaintext is encrypted to obtain the ciphertext after the generation of session key $\mathcal{S}$, the initialization vector $\mathcal{C}[0]$, initial chain value $\mathcal{T}[0]$ and the domain value, x. The plaintext blocks are processed as follow:

```
    $\mathcal{T}[1] = \mathcal{F}((\mathcal{S} \bigoplus \mathcal{T}[0]) \mid\mid \mathcal{C}[0] \mid\mid 0^\star \mid\mid [x]_\mathfrak{b})$
    $\mathcal{C}[1] = \mathcal{M}[1] \bigoplus_\alpha \mathcal{T}[1]$
for all blocks $\mathcal{M}[i], 2 \le i \le \mathcal{L} - 1$ {
        $\mathcal{T}[i] = \mathcal{F}((\mathcal{S} \bigoplus \mathcal{T}[i-1]) \mid\mid \mathcal{C}[i-1] \mid\mid 0^\star \mid\mid [4]_\mathfrak{b})$
        $\mathcal{C}[i] = \mathcal{M}[i] \bigoplus_\alpha \mathcal{T}[i]$
}
    $\mathcal{T}[\mathcal{L}] = \mathcal{F}((\mathcal{S} \bigoplus \mathcal{T}[\mathcal{L}-1]) \mid\mid \mathcal{C}[\mathcal{L}-1] \mid\mid 0^\star \mid\mid [4]_\mathfrak{b})$
    $\mathcal{C}[\mathcal{L}] = \mathcal{M}[\mathcal{L}] \bigoplus_\beta \mathcal{T}[\mathcal{L}]$.
```

2.5 Tag Generation

After the associated data and plaintext are processed, the $\mathcal{L}_T$-bit tag $\mathcal{T}$ is derived:

$$\mathcal{T} = \mathcal{F}((\mathcal{S} \bigoplus \mathcal{T}[\mathcal{L}]) \mid\mid \mathcal{C}[\mathcal{L}] \mid\mid 0^\star \mid\mid \mathcal{L}_T \mid\mid \beta + 5)$$

The decryption is trivial and we omit it here. For the verification, only the $\mathcal{L}_T$ least significant bits of the tags are checked.

2.6 Security Goals of COFFE

COFFE is claimed to have the INT-CTXT (ciphertext integrity) and IND-CPA(indistinguishable under chosen plaintext attack) property under nonce-respecting scenario.

In particular, in Lemma 1 of [6], we have:

Lemma 1. *Let Π be a COFFE scheme as defined above with $\mathcal{F}$ as its underlying hash function. Then the advantage of adversary $\mathcal{A}$ under nonce-respecting* *scenario with q queries and ℓ message blocks to the encryption oracle with time bounded by t can be upper bounded by:*

$$\mathbf{Adv}_{\Pi}^{CPA}(q,\ell,t) \leq \frac{8\ell^2+3q^2}{2^n} + 2.\mathbf{Adv}_{\mathcal{F}}^{PRF\text{-}XRK}(q,\ell,t).$$

In other words, distinguishing COFFE from a random function with chosen input under the bound of (q,ℓ,t) should be at least as hard as distinguishing $\mathcal{F}$ from a random function $\$: \{0,1\}^\gamma \Rightarrow \{0,1\}^\delta$.

Additionally, COFFE has some other security claim under different circumstances. Firstly, under the nonce-misuse scenario, it claimed that

– "*..., the integrity of the ciphertext does not depend on a nonce, but only on the security of $\mathcal{F}$*".

In particular, in Lemma 2 of [6], we have:

Lemma 2. *Let Π be a* ***COFFE*** *scheme as defined above with $\mathcal{F}$ as its underlying hash function. Then in the nonce-ignoring adversary scenario with q queries for ℓ message blocks and t times, we have*

$$\mathbf{Adv}_{\Pi}^{INT\text{-}CTXT}(q,\ell,t) \leq \frac{3\ell^2+2q^2}{2^\delta} + \frac{q}{2^{\mathcal{L}_T}} + \mathbf{Adv}_{\mathcal{F}}^{PRF}(q+\ell, O(t)).$$

This implies that the hardness of forging a ciphertext with a valid tag should be at least as hard as distinguishing $\mathcal{F}$ from a random function from $\{0,1\}^\gamma$ to $\{0,1\}^\delta$.

Secondly, COFFE also provides a failure-friendly authenticity. That is, under a weaker assumption on the security of the underlying hash function $\mathcal{F}$, the authenticity of the message is still kept.

Lastly, COFFE also provides a reasonable resistance against side channel attack. This is so because *"for each encryption process, a new short term key is derived from a nonce and the long term key"* [6].

3 Analysis on the COFFE Scheme

In our analysis, we will assume $\mathcal{F} : \{0,1\}^\star \Rightarrow \{0,1\}^\delta$ to be an arbitrary ideal hash function with γ being the largest possible length of the input to ensure exactly one compression function invocation per hash function call. Here we are assuming the possibility of the parameters to have length more than 255 bits. In other words, it is possible that it requires more than 1 byte to represent $\mathcal{L}_K, \mathcal{L}_V, \mathcal{L}_T$ in their binary format.

The first observation is about the input for the hash function call. Note that since we are only considering concatenation, there is not always a way, given the concatenated string, to uniquely determine the value for each strings before the concatenation. For example, if $a||b = 11011$, it is possible for $a = 110, b = 11$ or $a = 1, b = 1011$. This leads to the possibility that two different sets of strings to be concatenated to the same string.

Observation 1. *For the input of any hash function call, due to the absence of separator between substrings and changeable elements lengths, it is possible to have two different sets of strings to be concatenated to the same string.*

On the following subsections, we analyse this observation further to find whether it is possible to utilise this to cause a collision in the intermediate state value of the COFFE. We first consider the case when we fix the message block size while allowing two different last message block sizes, β_1 and β_2, to be used. The analysis is focused on the case when $|\beta_1 - \beta_2|$ is a multiple of 256. The analysis on this can be found on Sect. 3.1. Next we also consider the possibility of having identical intermediate state values when we change the message block size, α, while keeping β fixed. The analysis is focused on how α should be chosen in such a way for the first message block encryption of both instantiations to have identical hash value output. This is discussed in Section 3.2.

3.1 Modification of β

We fix α and consider different values of β. In our next observation, with large enough α, it is possible to have $\beta_1 < \beta_2 \leq \alpha$ such that $\beta_2 - \beta_1$ is a multiple of 256. This implies that the last byte of the input of $\mathcal{F}$ in the tag generation for the two different plaintexts can be the same. As discussed above, however, we want the collision to happen in the whole input string for any $\mathcal{F}$ input. If both $\beta_1 + 5$ and $\beta_2 + 5$ require 2 bytes to represent in binary format, the second to last byte will never agree. So for collision to happen, we need $\beta_1 + 5 < 256, 256 \leq \beta_2 + 5 < 65536$ and $\beta_2 = \beta_1 + 256\rho$ for some integer $1 \leq \rho \leq 255$.

To further analyse this observation, we consider the note by the designers regarding the increase of number of byte required for the binary representations of the domain. In [6], it is stated that if $\beta + 5$ exceeds one byte, all domain representations in the current COFFE will be encoded as two-byte values instead of one. So this is important in our analysis on the possibility of exploring this observation to introduce a successful attack.

Note that in the message processing, assuming that γ is big enough, there are enough bits of the zero padding between $\mathcal{C}[i]$ and the domain values for the encryption to absorb the additional byte for the domain values in case $\beta + 5$ is increased from one byte to two bytes value. So the parts that need to be taken care of for this to happen are the session key generation and tag generation.

In the session key generation, we consider the last several bytes of the input of $\mathcal{F}$. Here we have the input to be $\ldots \mid\mid a \mid\mid \mathcal{L}_K \mid\mid \mathcal{L}_V \mid\mid 0$. Note that when we expand the domain value from 1- to 2-byte value, the domain value should still have the same value. So the second to last byte for the input must be 0. This gives us our next observation.

Observation 2. *To ensure that collision can occur when extending the domain from 1- to 2-byte value, the initial value of $\mathcal{L}_V$ must be a multiple of 256. This means that if the initial $\mathcal{L}_V$ is a 1-byte value, it must be* 0, *that is, no nonce in the first instance.*

Our primary goal in this section is to investigate the possibilities to have two different (key, nonce) pairs, $(\mathcal{K}_1, \mathcal{V}_1)$ and $(\mathcal{K}_2, \mathcal{V}_2)$ with lengths $\mathcal{L}_{K_1}, \mathcal{L}_{V_1}, \mathcal{L}_{K_2}$, $\mathcal{L}_{V_2}$ respectively such that

$$(\mathcal{K}_1 \mid\mid \mathcal{V}_1 \mid\mid 0^\star \mid\mid \mathcal{L}_{K_1} \mid\mid \mathcal{L}_{V_1} \mid\mid [0]_1) = (\mathcal{K}_2 \mid\mid \mathcal{V}_2 \mid\mid 0^\star \mid\mid \mathcal{L}_{K_2} \mid\mid \mathcal{L}_{V_2} \mid\mid [0]_2).$$

For simplicity, let

$$\mathcal{S}_1 = (\mathcal{K}_1 \mid\mid \mathcal{V}_1 \mid\mid 0^\star \mid\mid \mathcal{L}_{K_1} \mid\mid \mathcal{L}_{V_1} \mid\mid [0]_1),$$
$$\mathcal{S}_2 = (\mathcal{K}_2 \mid\mid \mathcal{V}_2 \mid\mid 0^\star \mid\mid \mathcal{L}_{K_2} \mid\mid \mathcal{L}_{V_2} \mid\mid [0]_2).$$

Here, we note that collision is indeed possible as illustrated by the following example: Let SHA-512 be our hash function. We can choose any 256-bit string, say $\mathcal{K}$, and set $\mathcal{K}_1 = \mathcal{K}_2 = \mathcal{K}$. Now we use any one bit value (0 or 1) as our $\mathcal{V}_2$ while $\mathcal{V}_1$ is set to be $\mathcal{V}_2$ appended by 255 zeros. Now if we use 472 zero paddings on the first string while using 727 bits for the second string we will have

$$\mathcal{S}_1 = \mathcal{S}_2 = (\mathcal{K} \mid\mid 1 \mid\mid 0^{255} \mid\mid 0^{472} \mid\mid [1]_1 \mid\mid [0]_1 \mid\mid [1]_1 \mid\mid [0]_1 \mid\mid [0]_1).$$

In the remaining of this section, we will try to analyse whether such collision is possible for other instances of COFFE. Here we analyse different cases of $\mathcal{S}_1$ on the possibility of having $\mathcal{S}_1 = \mathcal{S}_2$. The factors that we need to consider are the number of bytes required for $\mathcal{L}_{K_i}, \mathcal{L}_{V_i}$ and whether there is any zero paddings required. Note that if $\mathcal{L}_{K_i}$ or $\mathcal{L}_{V_i}$ is a 3-byte value, the value will be at least 65536 which is too big. To simplify our discussion, for this paper, we will only consider the key and nonce to have length whose binary format can be represented as at most a 2-byte value. Due to the big number of cases we need to consider and the similarity of the cases, we will just discuss one case as example and a full analysis of the other cases can be found in the appendix of the full version paper [12] while the Table 1 containing the conclusion is provided for reference.

I.3.c Case I.3.c.: $\mathcal{S}_1$ has no zero paddings, $[\mathcal{L}_{K_1}] = 1, [\mathcal{L}_{V_1}] = 2, [\mathcal{L}_{K_2}] = 1$, $[\mathcal{L}_{V_2}] = 2$.

By Observation 2, $\mathcal{L}_{V_1} = 256\mathbf{b}$ and $\mathcal{L}_{K_1} = \mathbf{a}$ where $1 \leq \mathbf{a}, \mathbf{b} \leq 255$. Both $\mathbf{a}$ and $\mathbf{b}$ are nonzero because of the following reasons. First of all, since $\mathcal{L}_{K_1} = \mathbf{a}$, if $\mathbf{a} = 0$, then there is no secret key, in which case, no confidentiality for the message. So we can disregard the case when $\mathcal{L}_K = 0$. Next, since $[\mathcal{L}_{V_1}] = 2$, this should mean that $\mathcal{L}_{V_1} \geq 256$ since otherwise, $[\mathcal{L}_{V_1}] = 1$. So if $\mathbf{b} = 0$, this implies $\mathcal{L}_{V_1} = 0$ which violates the requirement $\mathcal{L}_{V_1} \geq 256$. Hence

$$\mathcal{S}_1 = (\mathcal{K}_1 \mid\mid \mathcal{V}_1 \mid\mid \mathbf{a} \mid\mid \mathbf{b} \mid\mid [0]_2).$$

Let $\mathcal{V}_1 = \mathcal{V}_1' \mid\mid \mathbf{d}$ where $\mathbf{d}$ is the last byte of $\mathcal{V}_1$. So

$$\mathcal{S}_1 = (\mathcal{K}_1 \mid\mid \mathcal{V}_1' \mid\mid \mathbf{d} \mid\mid \mathbf{a} \mid\mid \mathbf{b} \mid\mid [0]_2).$$

Consider the alternative string $\mathcal{S}_2$. Recall that here we want $\mathcal{S}_1 = \mathcal{S}_2$ where $\mathcal{S}_2$ has its domain value represented as a 2-bytes value. This implies that the $[0]_2$ in the last 2 bytes of $\mathcal{S}_1$ must appear as the domain for $\mathcal{S}_2$. So this implies that $\mathcal{L}_{K_2} = \mathbf{d}$ and $\mathcal{L}_{V_2} = 256\mathbf{a} + \mathbf{b}$. Let t' be the number of zero padding in $\mathcal{S}_2$ where $t' \geq 0$. Equating $\mathcal{S}_1$ with $\mathcal{S}_2$, we have $\mathcal{K}_1 \mid\mid \mathcal{V}_1' = \mathcal{K}_2 \mid\mid \mathcal{V}_2 \mid\mid 0^{t'}$. Now comparing the length of these substrings, we have $\mathbf{a}+256\mathbf{b}-8 = \mathbf{d}+256\mathbf{a}+\mathbf{b}+t'$ or equivalently, $255(\mathbf{b}-\mathbf{a}) = \mathbf{d}+8+t'$. Consider the family:

$$\mathcal{F}_{(1,2),(1,2)} = \{(\mathbf{a}, \mathbf{b}, \mathbf{d}, t') : 1 \leq \mathbf{a}, \mathbf{b}, \mathbf{d} \leq 255, t' \geq 0, 255(\mathbf{b} - \mathbf{a}) = \mathbf{d} + 8 + t'\}.$$

Now we consider the feasibility of each element of $\mathcal{F}_{(1,2),(1,2)}$. Feasibility here means the possibilities of using these values as the parameters to have the collision. Let $(\mathbf{a}, \mathbf{b}, \mathbf{d}, t') \in \mathcal{F}_{(1,2),(1,2)}$. Note that the collision may not happen with probability 1 due to the case when $\mathcal{K}_1 \neq \mathcal{K}_2$. Note that since key is the first part of the collided string, this can only happen when $\mathcal{L}_{K_1} \neq \mathcal{L}_{K_2}$.

Before going on to the analysis, we have an assumption first. Suppose that $\mathcal{L}_{K_1} > \mathcal{L}_{K_2}$. Since $\mathcal{K}_1$ is the first $\mathcal{L}_{K_1}$ bits of $\mathcal{S}_1, \mathcal{K}_2$ is the first $\mathcal{L}_{K_2}$ bits of $\mathcal{S}_2$ and we need $\mathcal{S}_1 = \mathcal{S}_2$, the first $\mathcal{L}_{K_2}$ bits of $\mathcal{K}_1$ must be $\mathcal{K}_2$. Instead of assuming that this happens by chance, we will assume the following: The user has 2 different instantiations of COFFE scheme with different parameter and different key length. However, the keys chosen by the user are not independent. The longer key is an extension of the shorter key by a random secret string. We note that this assumption is only made for the related key setting attack and not for the general attack.

Based on this assumption, we then have the probability of $\mathcal{K}_1$ to have its first $\mathcal{L}_{K_2}$ bits to be the same as $\mathcal{K}_2$ is exactly 1.

Now back to our case, we have that $\mathcal{L}_{K_1} = \mathbf{a}$ and $\mathcal{L}_{K_2} = \mathbf{d}$. So the length difference of the two keys is $|\mathbf{a} - \mathbf{d}|$ bits. Now if $\mathbf{a} = \mathbf{d}$, then we have $\mathcal{K}_1 = \mathcal{K}_2$. Now the rest of the two strings are $\mathcal{V}_1' \mid\mid \mathbf{d} \mid\mid \mathbf{a} \mid\mid \mathbf{b} \mid\mid [0]_2$ and $\mathcal{V}_2 \mid\mid 0^{t'} \mid\mid \mathbf{d} \mid\mid \mathbf{a} \mid\mid \mathbf{b} \mid\mid [0]_2$. So we have $\mathcal{V}_1 = \mathcal{V}_2 \mid\mid 0^{t'} \mid\mid \mathbf{d}$. Now since $\mathcal{V}_1$ can be controlled by the attacker, we can easily set this to be true. So the probability of the two strings to collide is 1 if $\mathbf{a} = \mathbf{d}$.

Now consider when $\mathbf{a} \neq \mathbf{d}$, specifically, $\mathbf{a} > \mathbf{d}$. The other case can be analysed using exactly the same way. Now let $\mathcal{K}_1 = \mathcal{K}_2 \mid\mid \mathcal{K}_1'$ where $\mathcal{K}_1'$ is the last $\mathbf{a} - \mathbf{d}$ bits of $\mathcal{K}_1$. We have $\mathcal{K}_1' \mid\mid \mathcal{V}_1' \mid\mid \mathbf{d} \mid\mid \mathbf{a} \mid\mid \mathbf{b} \mid\mid [0]_2$ and $\mathcal{V}_2 \mid\mid 0^{t'} \mid\mid \mathbf{d} \mid\mid \mathbf{a} \mid\mid \mathbf{b} \mid\mid [0]_2$ as the remaining part of the two strings truncating the first $\mathbf{d}$ bits. Thus, $\mathcal{K}_1' \mid\mid \mathcal{V}_1' = \mathcal{V}_2 \mid\mid 0^{t'}$. Note that since $1 \leq \mathbf{a}, \mathbf{d} \leq 255, \mathbf{a} - \mathbf{d} \leq 255, \mathcal{V}_2$ has length $256\mathbf{a} + \mathbf{b} \geq 256$. So the entire $\mathcal{K}_1'$ is in $\mathcal{V}_2$. In other words, for the two strings to collide, we need the last $\mathbf{a} - \mathbf{d}$ bits of $\mathcal{V}_2$ must be equal to $\mathcal{K}_1'$. Since $\mathcal{K}_1'$ is supposed to be unknown, the probability of this collision is $2^{\mathbf{a}-\mathbf{d}}$. It is easy to see that the remaining substring can be set to collide with probability 1. So the probability of

Table 1. Session key generation input collision

$[\mathcal{L}_{\mathbf{K_1}}]$	$[\mathcal{L}_{\mathbf{V_1}}]$	**t**	$[\mathcal{L}_{\mathbf{K_2}}]$	$[\mathcal{L}_{\mathbf{V_2}}]$	$\mathbf{t'}$	**Collision?**	**Probability**	**Key restriction**
1	1	Any	Any	Any	Any	No	0	Not Applicable
2	1	Any	1	1	Any	No	≈ 0	Not Applicable
2	1	0	2	1	Any	$\mathcal{F}_{(2,1),(2,1)}$	$2^{-(\mathcal{L}_{K_1}-\mathcal{L}_{K_2})}$	$\mathcal{K}_1 =_{(t'+8)} 0^{t'} \mid\mid \lfloor \frac{\mathcal{L}_{K_2}}{256} \rfloor$
2	1	$0 < t < 8$	2	1	Any	$\mathcal{F}_{\mathfrak{p},(2,1),(2,1)}$	$2^{-(\mathcal{L}_{K_1}-\mathcal{L}_{K_2})}$	$\mathcal{K}_1 =_{(t'+8-t)} 0^{t'} \mid\mid \lfloor \frac{\mathcal{L}_{K_2}}{2^{8+t}} \rfloor$
2	1	Any	Any	2	Any	No	0	Not Applicable
1	2	Any	1	1	$255\mathcal{L}_{V_2} + t$	Yes	1	No
1	2	Any	2	1	Any	No	≈ 0	Not Applicable
1	2	0	1	2	Any	$\mathcal{F}_{(1,2),(1,2)}$	$2^{-\lvert\mathcal{L}_{K_1}-\mathcal{L}_{K_2}\rvert}$	No
1	2	$0 < t < 8$	1	2	Any	$\mathcal{F}_{\mathfrak{p},(1,2),(1,2)}$	$2^{-\lvert\mathcal{L}_{K_1}-\mathcal{L}_{K_2}\rvert}$	No
1	2	0	2	2	Any	$\mathcal{F}_{(1,2),(2,2)}$	$2^{-(\mathcal{L}_{K_2}-\mathcal{L}_{K_1})}$	No
1	2	$0 < t < 16$	2	2	Any	$\mathcal{F}_{\mathfrak{p},(1,2),(2,2)}$	$2^{-(\mathcal{L}_{K_2}-\mathcal{L}_{K_1})}$	No
2	2	Any	1	1	Any	No	0	Not Applicable
2	2	Any	2	1	$255\mathcal{L}_{V_2} + t$	Yes	1	No
2	2	Any	1	2	Any	No	≈ 0	Not Applicable
2	2	0	2	2	Any	$\mathcal{F}_{(2,2),(2,2)}$	$2^{-\lvert\mathcal{L}_{K_1}-\mathcal{L}_{K_2}\rvert}$	$\mathcal{K}_1 =_{\max(0,t'-(\mathcal{L}_{V_1}-8))} 0$
2	2	$0 < t < 8$	2	2	Any	$\mathcal{F}_{\mathfrak{p},(2,2),(2,2)}$	$2^{-\lvert\mathcal{L}_{K_1}-\mathcal{L}_{K_2}\rvert}$	$\mathcal{K}_1 =_{\max(0,t'-(\mathcal{L}_{V_1}-(8-t)))} 0$

collision to happen is $2^{-(\mathbf{a}-\mathbf{d})}$. Using exactly the same analysis, we will see that when $\mathbf{d} > \mathbf{a}$, the probability of collision to happen is $2^{-(\mathbf{d}-\mathbf{a})}$.Hence, for any non-negative integer k, we can define a subfamily of $\mathcal{F}_{(1,2),(1,2)}$,

$$\mathcal{F}_{(1,2),(1,2),k} = \{(\mathbf{a}, \mathbf{b}, \mathbf{d}, t') \in \mathcal{F}_{(1,2),(1,2)} : |\mathbf{a} - \mathbf{d}| \leq k\}.$$

Then for any quadruplet $(\mathbf{a}, \mathbf{b}, \mathbf{d}, t') \in \mathcal{F}_{(1,2),(1,2),k}$ we take as parameter, the collision probability is at least 2^{-k}.

We remark that this probability is applicable for any choices of $\mathcal{K}_1$ and $\mathcal{K}_2$. This observation is essential in our attacks later to decide whether the attacks are only applicable to a family of key to any value of key with the given length.

We include in Table 1 the full list of conclusion of the session key generation analysis. Here we will use t to represent the zero padding for $\mathcal{S}_1$ and t' for $\mathcal{S}_2$. In the Collision column, No means a collision in this case is impossible, yes means a collision will always happen on any choices of the parameter values (with appropriate choice of key and nonce). Lastly, $\mathcal{F}_{(a,b),(c,d)}$ or $\mathcal{F}_{\mathfrak{p},(a,b),(c,d)}$ is the family of parameter values that belongs to the respected case that collision is possible. The definition of each of the family can be found in the complete analysis of each case that is either can be found above or in the appendix of the full version paper [12].

Recall that in the tag generation, given $\mathcal{T}[\mathcal{L}], \mathcal{S}$, and $\mathcal{C}[\mathcal{L}]$, the input of $\mathcal{F}$ is

$$\left(\left(\mathcal{S} \bigoplus \mathcal{T}[\mathcal{L}]\right) \mid\mid \mathcal{C}[\mathcal{L}] \mid\mid 0^{\star} \mid\mid \mathcal{L}_T \mid\mid \beta + 5\right).$$

We note here that here we do not use the byte-aligned assumption in our analysis. The analysis can be restricted to a byte-aligned one by adding a restriction on the families to have some of the values to be divisible by 8. Here the values that are related to the remainder of any value divided by 256 must be divisible by 8. So for example, in the case when $[\mathcal{L}_K] = 2$, if $\mathcal{L}_K = (\mathbf{a} \mid\mid \mathbf{b})$, then we do not need $\mathbf{a}$ to be divisible by 8. We just need $\mathbf{b}$ to be divisible by 8. This will not change the existence of any of the families. However, it will certainly requires a bigger parameter value. For example, for the case when $[\mathcal{L}_{K_1}] = [\mathcal{L}_{V_1}] = [\mathcal{L}_{K_2}] = 2$ and $[\mathcal{L}_{V_2}] = 1$, if we want all the values to be byte aligned, the smallest parameters we can use is when $\mathcal{L}_{K_1} = \mathcal{L}_{K_2} = 256, \mathcal{L}_{V_2} = 8$, and $\mathcal{L}_{V_1} = 2048$. This leads to the input size for the hash function to be at least $2048 + 256 + 5 \times 8 = 2394$ bits. Here we set $\mathcal{V}_2 = 128$ and $\mathcal{V}_1 = \mathcal{V}_2 \mid\mid 0^{2040}$ and the other settings to be the same as the previous example.

We move on to the tag generation when $\beta + 5$ changes from 1-byte value to 2-byte value. Note that the only possible source of this 1-byte value is from $\mathcal{L}_T$. So, in the second instantiation where $\beta + 5$ is changed to a 2-byte value, the tag length will be different from the initial one. In fact, the first tag length needs to be a 2-bytes value, say $a \mid\mid b$ and the second tag length needs to be a while the difference between the two βs needs to be $256 \times b$.

3.2 Modification of $\boldsymbol{\alpha}$

This section discusses a special case of the analysis in which the user has at least two instantiations of COFFE where they have different message block sizes but the same (or related) key. Since we are considering changing α, the one we really need to take care of is just the generation of $\mathcal{C}[0]$. This is because for any other place where α affects the system, it is generated by the previous chain in which we can truncate easily.

Recall that $\mathcal{C}[0]$ is the first $\frac{\alpha}{4}$ post decimal values of π interpreted as hexadecimal values. Suppose that we want the difference of the two block sizes, α_1 and α_2, to be k with α_2 being the larger value. Since we are assuming the ideality of $\mathcal{F}$, we want the input of $\mathcal{F}$ in this point for both instantiation to coincide. So in other words, if the initial vector of the first instantiation is the α_1 bit $\mathcal{C}_1$ and the second one to be the α_2 bit $\mathcal{C}_2$, the additional k bits of α_2 should be absorbed by the next substring of the input, which is the zero padding. Hence, the last k bits of $\mathcal{C}_2$ should all be zeros. In other words, the value of α_1 so that it can coincide with the positions in the post decimal values of π to have a consecutive 0^k bits. So for example, if we want $\alpha_2 = \alpha_1 + 8$, and α_1 and α_2 to be a multiple of 8, then we will need to wait until the 306-th decimal place to get the 8 bits of consecutive zeros. In this case, $\alpha_1 = 1224$ and $\alpha_2 = 1232$. The requirement that α_1 and α_2 are divisible by 8 comes if we are assuming that the design is byte-aligned. Note that different αs can be found along the places where we can find k consecutive zeros in the binary representation of π.

4 Distinguishing Attack

In this section, to form a distinguishing attack, we use the session key collision discussed in the previous section and the appendix of the full version paper [12].

Assuming that we have the same secret key, different nonce, and different number of byte of domain value but the same session key, as before, we assume that now the session key for each instantiation is the same, each uses the proper number of byte of domain value.

We set the parameters $\alpha, \beta_1, \beta_2, \mathcal{L}_{T_1}$ and $\mathcal{L}_{T_2}$ as follows:

1. $\beta_1 + 5 < 256 < \beta_2 + 5 \leq \alpha + 5 \leq \delta + 5$
2. $\beta_2 - \beta_1 = 256\rho$ for some positive integer ρ
3. $\mathcal{L}_{T_1} < 256 \leq \mathcal{L}_{T_2}$ where $\mathcal{L}_{T_2} = 256\mathcal{L}_{T_1} + \rho$.

Set the first plaintext to be a two-blocks message, $\mathcal{M}_1 = (\mathcal{MB}_1 \mid\mid \mathcal{MB}_2)$ such that $\mathcal{MB}_1$ has α bits and $\mathcal{MB}_2$ has β_1 bits with tag length set to be $\mathcal{L}_{T_2}$. Assume the ciphertext is $\mathcal{C}_1 = (\mathcal{CB}_1 \mid\mid \mathcal{CB}_2)$ with $\mathcal{T}_1$ as the tag.

The second message block, is then chosen to be $\mathcal{M}_2 = (\mathcal{MB}_1 \mid\mid \mathcal{MB}'_2)$ such that $\mathcal{MB}'_2$ has β_2 bits. Here we will use the tag length to be $\mathcal{L}_{T_1}$. We also assume the ciphertext is $\mathcal{C}_2 = (\mathcal{CB}'_1 \mid\mid \mathcal{CB}'_2)$ with $\mathcal{T}_2$ as the tag.

As discussed above, since the first block of both message are the same, $\mathcal{MB}_1$, we should have $\mathcal{CB}_1 = \mathcal{CB}'_1$ and $\mathcal{T}[1]$ and $\mathcal{T}[2]$ should also be the same. Now remember that $\mathcal{MB}_2 \bigoplus \mathcal{CB}_2$ and $\mathcal{MB}'_2 \bigoplus \mathcal{CB}'_2$ tells us the last β_1 and β_2 bits of $\mathcal{T}[2]$ respectively. So if $\mathcal{C}_1$ and $\mathcal{C}_2$ are both from COFFE instantiation, we must have $\mathcal{CB}_1 = \mathcal{CB}'_1$ and the last β_1 bits of $\mathcal{CB}_2$ and the last β_1 bits of $\mathcal{CB}'_2$ should agree. So we will guess that it is a COFFE instantiation instead of a random function if these requirements are met. Note that this can happen if it is a random function with probability $2^{-(\alpha+\beta_1)}$.

Recall that a distinguishing attack works as follows. An oracle randomly chooses whether it uses a random function or a COFFE instantiation with the given parameter. Then as an attacker, we can request for encryption for some plaintext. Then an adversary tries to decide whether the oracle uses a random function or a COFFE instantiation. The distinguishing attack described above has error probability 0 if we conclude that the oracle uses a random function. However, if we guess that the oracle uses a COFFE instantiation, there is a probability of $2^{-(\alpha+\beta_1)}$ of the function is actually a random function instead of COFFE. Note that since $\alpha \geq 256$ in our attack, the failure probability is at most 2^{-256} which is very small. Therefore, with 2 enquiries to the oracle with 4 message blocks, COFFE with ideal underlying hash function in nonce-respecting scenario can be distinguished with probability close to 1. So in these instantiations of COFFE, the security claim given in Lemma 1 is not satisfied.

5 Ciphertext Forgery Attack

In this section, we will propose two different ciphertext forgery attacks. The first attack is based on the observation on Subsect. 3.1. It exploits the possibility of having an identical intermediate state value for two different instantiations when we fix α while using different values of β. The detail of the attack can be found in Sect. 5.1. Similarly, Sect. 5.2 discusses the forgery attack based on the discussion on Subsect. 3.2. Here we try to forge a valid ciphertext in the case when there exists two different COFFE instantiations with same key for different message block size. Here both attacks require 2 enquiries and can forge a valid ciphertext with probability one. The success probability 1 is applicable whenever we assume for both instantiations, the secret key used is the same instead of one key being an extension of the other. Lastly, we will also discuss the possibility of combining the two forgery attacks. This can be found in Subsect. 5.3.

5.1 Forgery Attack with Constant Message Block Size

Take any $(\mathcal{K}_1, \mathcal{V}_1), (\mathcal{K}_2, \mathcal{V}_2)$ (key, nonce) pairs from the discussion session such that they generate the same session key, one with 1-byte domain value, the other with two. Let $\mathcal{S}_1$ be the input for session key generation with 1-byte domain value and $\mathcal{S}_2$ be the input for the session key generation with $2-$bytes domain value. Here we assume that the input for session key generation is chosen accordingly based on the number of bytes of domain value. Hence, after this point, we can ignore the secret key and nonce and we can just assume that for each instantiation, we are using the same session key and associated data.

Note that any full block plaintext-ciphertext pair leaks α least significant bits of the output of the hash function for a fixed input, while any β-bit block plaintext-ciphertext pair leaks only β least significant bits of it. So since $\beta \leq \alpha$, it is always more desirable to get a full-block plaintext-ciphertext pairs since they leak the output value more.

Here we set the parameters $\alpha, \beta_1, \beta_2, \mathcal{L}_{T_1}$ and $\mathcal{L}_{T_2}$ as described before in Sect. 4.

Next we define the first message $\mathcal{M}_1$, a $3-$block message $(\mathcal{MB}_1 \mid\mid \mathcal{MB}_2 \mid\mid \mathcal{MB}_3)$ such that $|\mathcal{MB}_1| = |\mathcal{MB}_2| = \alpha, |\mathcal{MB}_3| = \beta_2$. Let the ciphertext of this message be $\mathcal{C}_1 = (\mathcal{CB}_1 \mid\mid \mathcal{CB}_2 \mid\mid \mathcal{CB}_3)$ with tag $\mathcal{T}_1$ with $\mathcal{L}_T$ set to any value. Here we can compute the values of $\mathcal{CB}_1$ and $\mathcal{CB}_2$ since $\mathcal{MB}_1 \bigoplus \mathcal{CB}_1$ gives us the last α bits of $\mathcal{T}[1]$ and $\mathcal{M}_2 \bigoplus \mathcal{C}_2$ gives us the last α bits of $\mathcal{T}[2]$ which are essential in the attack.

We define our second message $\mathcal{M}_2$, a $2-$block message $(\mathcal{MB}_1 \mid\mid \mathcal{MB}'_2)$ with the length of $\mathcal{MB}'_2$ to be β_1 bits and tag length to be $\mathcal{L}_{T_2}$. The first block is chosen to be exactly the same as before to ensure the value of $\mathcal{T}[1]$ and $\mathcal{T}[2]$ can be kept constant. Based on the previous message, the least α bits of both values are known. Suppose that the ciphertext of this plaintext is $\mathcal{C}_2 = (\mathcal{CB}_1 \mid\mid \mathcal{CB}'_2)$ with tag $\mathcal{T}_2$.

Using the information we obtain, we generate the following valid ciphertext. Define another 2-block message $\mathcal{M}_3 = (\mathcal{MB}_1 \mid\mid \mathcal{MB}''_2)$. Here we set $\mathcal{MB}_1$ to be the same as the first block from the previous message blocks. This is again to ensure the value of $\mathcal{T}[1]$ and $\mathcal{T}[2]$ can be kept constant. We let the length of $\mathcal{MB}''_2$ to be β_2 and choose $\mathcal{MB}''_2$ such that $\mathcal{MB}''_2 \bigoplus_{\beta_2} \mathcal{T}[2] = \mathcal{C}_2 \mid\mid 0^{\beta_2 - \beta_1}$. Here, $\mathcal{MB}''_2$ can be calculated since we know the last α bits of $\mathcal{T}[2]$ and $\alpha > \beta_2$. Using this message, it is easy to see that the tag generation will have the same input as before, although $\mathcal{L}_T$ is now $\mathcal{L}_{T_1}$. So the tag for this ciphertext will be the last β_1 bits of $\mathcal{T}_2$.

This attack has success probability equal to the probability of the two strings used as the input session key generation to be the same. As we have discussed before, for some parameters such as the ones in case I.3 and case II.3, this can even be 1. In other words, in the case when the success probability is one, the attack above proves that the ciphertext integrity of this cipher does not satisfy the bound given in Lemma 2 even in an ideal hash function situation.

Note that here we use three COFFE instantiations for each attack (2 for enquiry and 1 for the guess), while in our discussion on session key generation collision, we only consider the collision for two (key, nonce) pairs. So the same attack cannot directly work for nonce-respecting scenario unless we can find three (key, nonce) pairs that collide to the same session key.

5.2 Forgery Attack with Dynamic Message Block Size

In this section, we are assuming the existence of two different instantiations of COFFE with different message block size but the same secret key and constant last message block size β. Now we pick $\alpha_1 < \alpha_2$ such that $\alpha_2 - \alpha_1 = k$. Next we find the valid size of α_1 and α_2 based on our discussion in the discussion section. Here since we assume constant last message block size, β, we assume $\beta \leq \alpha_1$. Since we are using constant last message block size, to get the same session key, we can consider the nonce-misuse scenario where we use the same key and nonce for both instantiations. Note that this means the tag length should still be kept the same.

First, we generate message, $\mathcal{M}_1 = (\mathcal{MB}_1 \mid\mid \mathcal{MB}_2)$ with $|\mathcal{MB}_1| = \alpha_2$ and $|\mathcal{MB}_2| = \beta$. Now assume that we get the ciphertext $\mathcal{C}_1 = (\mathcal{CB}_1 \mid\mid \mathcal{CB}_2)$ with tag $\mathcal{T}_1$. In this pair, our objective is to find the last α_2 bits of $\mathcal{T}[1]$ which can be obtained by calculating $\mathcal{MB}_1 \bigoplus \mathcal{CB}_1$.

We then consider the following message: $\mathcal{M}_2 = (\mathcal{MB}'_1 \mid\mid \mathcal{MB}'_2)$ with $|\mathcal{MB}'_1| = \alpha_2$ and $|\mathcal{MB}'_2| = \beta$. We further require the last k bits of $\mathcal{MB}'_2 \bigoplus_{\alpha_1} \mathcal{T}[1]$ are all zeros. Note that $\mathcal{MB}'_2$ can be generated easily with the knowledge of the last α_1 bits of $\mathcal{T}[1]$. Assume that the ciphertext is $\mathcal{C}_2 = (\mathcal{CB}'_1 \mid\mid \mathcal{CB}'_2)$ with tag $\mathcal{T}_2$. Here the last k bits of $\mathcal{CB}'_1$ are all zero and $\mathcal{CB}'_2 \bigoplus \mathcal{MB}'_2$ tells us the last β bits of $\mathcal{T}[2]$ when the first block of the message is $\mathcal{MB}'_1$.

Now having this information, we will proceed to our forgery attack. The message block that we will use is $\mathcal{M}_3 = (\mathcal{MB}''_1 \mid\mid \mathcal{MB}'_2)$. Here we have $|\mathcal{MB}''_1| = \alpha_1$ and $\mathcal{MB}''_1$ is chosen such that

$$\left(\mathcal{MB}''_1 \bigoplus{}_{\alpha_1} \mathcal{T}[1]\right) \mid\mid 0^k = \mathcal{CB}'_1.$$

We also note that the second block is exactly the same used in $\mathcal{M}_2$. Then it is easy to see that the ciphertext is $\mathcal{C}_3 = (\mathcal{CB}''_1 \mid\mid \mathcal{CB}'_2)$ where $\mathcal{CB}''_1 = \mathcal{MB}''_1 \bigoplus_{\alpha_1} \mathcal{T}[1]$. Furthermore, the tag is exactly $\mathcal{T}_2$.

This forgery attack requires 2 enquiries to the oracle with 4 message blocks. So this attack provides a family of instances of COFFE that cannot provide ciphertext integrity as claimed in Lemma 2 under nonce-misuse scenario.

5.3 Combination of the Existing Attacks

In our previous two subsections, we change one of the parameters (α, β) while letting the other constant. This is done to simplify the analysis. However, it is possible for us to combine both attacks to generate new attack, that is, we change α and β in the same time. Notice that by combining the two attacks, the "nonce-misuse" requirement is not a must anymore. As discussed in the constant message block size subsection, as long as we can find a triple of (key,nonce) pairs that generate the same session key, we can launch the attack in the nonce-respecting scenario.

6 Related Key Recovery Attack

Note that, in most of the attacks we have mentioned, we are assuming same secret key. In this section, we will discuss the case with two different instances of COFFE with different key length. As discussed in our observations, in this case we assume that the longer key is obtained by extending the shorter key with secret string. As we have discussed in the appendix of the full version paper [12], there are $(\mathcal{K}_1, \mathcal{V}_1), (\mathcal{K}_2, \mathcal{V}_2)$ pairs that leads to the same session key (with one of them using one-byte domain value while the other using two-byte value) with different key length. Here we will use the pairs with k-bits key length difference and all of the difference are all in the nonce of the corresponding shorter length key. Now assume that $|\mathcal{L}_{K_1}| > |\mathcal{L}_{K_2}|$.

We again choose the parameters $\alpha, \beta_1, \beta_2, \mathcal{L}_{T_1}$, and $\mathcal{L}_{T_2}$ as in Sect. 4. The attack here is an adaptation of the distinguishing attack we proposed earlier. We use the two messages $\mathcal{M}_1$ and $\mathcal{M}_2$ as described in Sect. 4. The difference here is that for $\mathcal{M}_2$ with two bytes domain value and shorter key length, we will enquire 2^k different blocks of it with different k most significant bits of $\mathcal{V}_2$. Note that if the k most significant bits of $\mathcal{V}_2$ coincide with the k-bit extension of the secret key, then $\mathcal{CB}_1 = \mathcal{CB}_1'$ and the last β_1 bits of $\mathcal{CB}_2$ and the last β_1 bits of $\mathcal{CB}_2'$ should agree. So by using this approach, we can guess the k-bits extension of the secret key with the same complexity as exhaustive search for a k-bits secret key.

As discussed in the distinguishing attack section, when we decide that the guessed k-bits is wrong, the probability that the k-bits is actually the correct extension key is 0. So there will not be a false negative. However, when the k-bits we guess is wrong, the probability of false positive is, as discussed in the distinguishing attack, $2^{\alpha+\beta_1}$ which is at least 2^{-255} which is negligible.

So for the related key recovery attack, to recover the k-bit extension of the secret key in nonce-respecting scenario, we will need $2^k + 1$ plaintext-ciphertext pairs with success probability approximately 1. Note that the exact same attack can be adapted to the case when $|\mathcal{K}_1| < |\mathcal{K}_2|$.

7 Conclusion

7.1 Attack Summary

From the discussion above, we see that the security claim for the nonce-misusing scenario is not met for many different parameters. The same attack can be adapted to give a distinguishing attack for the nonce-respecting scenario for some subfamilies of the parameters mentioned above.

Lastly, having two different instances of COFFE with different key length with the longer key being the extension of the shorter key may not be a good idea. This is because if the parameter used belongs to the family we have found earlier, the extension of the key can be recovered with exhaustive search in the same way as if the secret key is just k bits.

In conclusion, COFFE does not satisfy any of the two security claims for some of the parameters that we have discussed before. The problem arises from the fact that concatenation of strings cannot be inverted uniquely and hence giving the opportunity of having two different set of strings concatenated to the same resulting strings.

7.2 Lesson Learned

Here we see that the the forgery and distinguishing attacks are feasible due to the possibility to have different (key,nonce) pairs to generate the same session key. This can be fixed by fixing the space for every given parameters. If some parameters are variables (such as the message block size in COFFE), we should ensure that the values of the variables get authenticated so as to prevent the forgery attack.

References

1. Bellare, M., Namprempre, C.: Authenticated encryption: relations among notions and analysis of the generic composition paradigm. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 531–545. Springer, Heidelberg (2000)
2. Cogliani, S., Maimut, D.S., Naccache, D., do Canto, R.P., Reyhanitabar, R., Vaudenay, S., Vizźar, D.: Offset Merkle-Damgård (OMD) version 1.0 A CAESAR proposal. In: CAESAR (2014). http://competitions.cr.yp.to/caesar-submissions.html
3. Datta, N., Nandi, M.: ELmD v1.0. In: CAESAR (2014). http://competitions.cr.yp.to/caesar-submissions.html
4. Dworkin, M.: Recommendation for BlockCipher Modes of Operation: The CCM Mode for Authentication and Confidentiality. NIST Special Publication 800–38C, May 2004
5. Dworkin, M.: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode(GCM) and GMAC. NIST Special Publication 800–38D, November 2007
6. Forler, C., McGrew, D., Lucks, S., Wenzel, J.: COFFE: Ciphertext Output Feedback Faithful Encryption authenticated encryption without a block cipher. In: Early Symmetric Crypto ESC (2013). https://eprint.iacr.org/2014/1003.pdf

7. Jutla, C.S.: Encryption modes with almost free message integrity. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 529–544. Springer, Heidelberg (2001)
8. Krawczyk, H.: The order of encryption and authentication for protecting communications (or: how secure is SSL?). In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, p. 310. Springer, Heidelberg (2001)
9. Meyer. C., Matyas. S.: Secure program load with manipulation detection code. In: Proceedings of Securicom, vol. 88, pp. 111–130 (1988)
10. Preneel, B., Govaerts, R., Vandewalle, J.: Hash functions based on block ciphers: a synthetic approach. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 368–378. Springer, Heidelberg (1994)
11. Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB: a block-cipher mode of operation for efficient authenticated encryption. In: Proceedings of the 8th ACM Conference on Computer and Communications Security, pp. 196–205 (2001)
12. Tjuawinata, I., Huang, T., Wu. H.: Cryptanalysis of the Authenticated Encryption Algorithm COFFE. Cryptology ePrint Archive, Report 2015/783 (2015). http://eprint.iacr.org/
13. Wu, H., Huang, T.: JAMBU Lightweight Authenticated Encryption Mode and AES-JAMBU. In: CAESAR (2014). http://competitions.cr.yp.to/caesar-submissions.html