On the Security of the LAC Authenticated Encryption Algorithm

Jiqiang Lu(✉)

Infocomm Security Department,
Institute for Infocomm Research,
Agency for Science, Technology and Research, 1 Fusionopolis Way,
Singapore 138632, Singapore
jlu@i2r.a-star.edu.sg, lvjiqiang@hotmail.com

Abstract. The LAC authenticated encryption algorithm was a candidate to the CAESAR competition on authenticated encryption, which follows the design of the ALE authenticated encryption algorithm. In this paper, we show that the security of LAC depends greatly on the parameter of the maximum message length and the order of padding the last message block, by cryptanalysing its variants that differ from the original LAC only in the above-mentioned two points. For the LAC variants, we present a structural state recovery attack in the nonce-respecting scenario, which is independent from the underlying block cipher, which requires only chosen queries to their encryption and tag generation oracles and can recover an internal state of the initialization phase for one of some used Public Message Numbers (PMNs) more advantageously than exhaustive key search; and the recovered internal state can be used to make an existential forgery attack under this PMN. Besides, slightly inferior to exhaustive key search, the state recovery attack can apply to the LAC variant that differs from LAC only in the order of padding the last message block. Although the state recovery attack does not apply to the original LAC, it sheds some light on this type of interesting structures, and shows that an authenticated encryption algorithm with a such or similar structure may be weakened when it is misused deliberately or accidentally with the reverse message padding order and a different maximum message length, and users should be careful about the two points when employing such a structure in reality.

Keywords: Authenticated encryption algorithm · LAC · State recovery attack · Forgery attack

1 Introduction

A (symmetric) authenticated encryption algorithm is an algorithm that transforms an arbitrary-length data stream (below an upper bound generally), called a message or plaintext, into another data stream of the same length, called a ciphertext, and generates an authentication tag for the message at the same

J.K. Liu and R. Steinfeld (Eds.): ACISP 2016, Part II, LNCS 9723, pp. 395–408, 2016.
DOI: 10.1007/978-3-319-40367-0_25

time, under the control of a secret key. We refer the reader to Bellare and Namprempre's work [1] for an introduction to authenticated encryption.

LAC [9] is a block-cipher-based lightweight authenticated encryption algorithm, which has a similar structure to the ALE [3] authenticated encryption algorithm. Built on a variant called LBlock-s of the LBlock lightweight block cipher [8], LAC takes as input an 80-bit user key, a 64-bit public message number (nonce) and a plaintext as well as associated data, and outputs a ciphertext of the same length as the plaintext and a 64-bit authentication tag. In 2014, LAC was submitted to the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) [4], however, Leurent [5] described an (existential) forgery attack on the full LAC algorithm, by showing that there exist 16-round differentials [2] with a probability of $2^{-61.52}$ in the LBlock-s cipher, which is slightly larger than the expected bound 2^{-64}. It is worthy to mention that the full ALE algorithm was shown in 2013 by Wu et al. [7] to suffer from an (existential) forgery attack based on differential cryptanalysis.

Leurent's attack on LAC as well as Wu et al.'s attack on ALE is mainly due to a security weakness of the underlying round-reduced block cipher, specifically the number of rounds is too small to be sufficient; otherwise, the attack would not work. In this paper, we analyse the security of LAC from a structural perspective, by focusing on its structure without exploiting any security weakness of the underlying block cipher, that is, we treat the block cipher as a sound pseudo-random permutation. We find that the security of LAC (in the nonce-respecting scenario) depends greatly on the parameter of the maximum message length and the order of padding the last message block (or equivalently the order of the two halves of the leaked 48-bit output), by presenting a state recovery attack on the LAC variants that differ from the original LAC only in that a different value is used for the parameter of the maximum message length and that the reverse order for padding the last message block is used. The attack on the LAC variants requires only chosen queries to their encryption and tag generation oracles, and can recover an internal state of the initialization phase for one of some used Public Message Numbers (PMNs) more advantageously than exhaustive key search. The recovered internal state can be used to make an existential forgery attack on the LAC variants under this PMN. Besides, slightly inferior to exhaustive key search, the attack can apply to the LAC variant that differs from LAC only in the reverse order of padding the last message block.

Our attack may apply to other authenticated encryption algorithms with similar structures, for example, it may apply to similar variants of ALE [6]. In reality, particularly in industry, a cryptographic algorithm is sometimes misused deliberately or accidentally, due to various practical reasons, say, being slightly modified for a particular application requirement, being modified with a different block cipher in place of the underlying block cipher, being confused with big- and little-endian formats, etc. As a result, although our attack does not apply to the original LAC, it sheds some light on this type of interesting structures, and shows that an authenticated encryption algorithm with a such or similar structure may be weakened when it is misused with the reverse message padding order and a different maximum message length. Thus, users should be very careful about the two

points when employing such a structure in reality, even when the underlying round-reduced block cipher has a sufficient number of rounds in the sense of security.

The remainder of the paper is organised as follows. In the next section, we give the notation and describe the LAC algorithm and the variants for our attacks. We present our state recovery attack and existential forgery attack on the LAC variants in Sects. 3 and 4, respectively. Section 5 concludes this paper.

2 Preliminaries

In this section, we give the notation used throughout this paper, and briefly describe the LAC algorithm and the variants for our attacks.

2.1 Notation

In all descriptions we assume that the bits of a value are numbered from right to left (or sometimes from top to bottom), starting with 1, with the first bit being the least significant bit. We use the following notation.

$\oplus$	bitwise logical exclusive OR (XOR) operation of two bit strings of the same length
$\|\|$	string concatenation
$\|X\|$	the number of elements when X is a set, or the bit length when X is a value
$\mathtt{e}$	the base of the natural logarithm ($\mathtt{e} = 2.71828\cdots$)
$\lceil X \rceil$	the smallest integer that is larger than or equal to a value X
$\mathcal{O}(X)$	a value that is of the same order as a value X

2.2 The LAC Authenticated Encryption Algorithm

The message encryption and tag generation procedure of LAC [9] consists of four phases: initialization, processing associated data, message encryption, and tag generation, as depicted in Fig. 1, where

- PMN is a 64-bit Public Message Number (PMN), serving as a nonce. The designers require that a PMN should be used (at most) only once under the same key, that is, LAC works in a nonce-respecting scenario.
- $\mathbf{E}$ is a simplified version LBlock-s of the LBlock [8] block cipher, that has a 64-bit block length, an 80-bit user key K and a total of 32 rounds;
- $\mathbf{G}$ is a 16-round reduced version of the $\mathbf{E}$ block cipher, with the 16 round subkeys generated from the key schedule $\mathbf{KS}$;
- $\widehat{\mathbf{G}}$ is the version of $\mathbf{G}$ that not only outputs a normal 64-bit ciphertext but also outputs the most significant 24 bits of the left half X_9 of the output of the eighth round of the $\mathbf{G}$ cipher and the most significant 24 bits of the left half X_{17} of the output of the sixteenth round of the $\mathbf{G}$ cipher (i.e., the output of $\mathbf{G}$), (the 48 bits serve as a keystream block);
- 0^{16} is a binary string of 16 zeros;

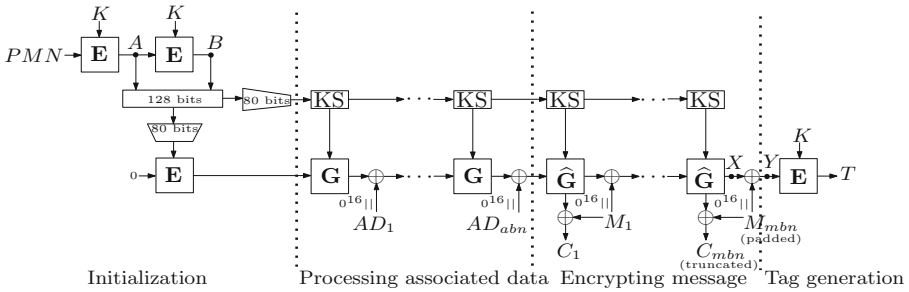

Fig. 1. The message encryption and tag generation procedure of LAC

- $(AD_1, AD_2, \cdots, AD_{abn})$ is an associated data of abn 48-bit blocks;
- $(M_1, M_2, \cdots, M_{mbn})$ is a message of mbn 48-bit blocks;
- $(C_1, C_2, \cdots, C_{mbn})$ is the ciphertext for $(M_1, M_2, \cdots, M_{mbn})$; and
- T is the tag for $(M_1, M_2, \cdots, M_{mbn})$.

During the initialization phase, a PMN passes through a cascade of two applications of the **E** block cipher with the user key K, and the concatenation of the outputs of the two applications of the **E** block cipher constitutes a 128-bit internal state. Then, the most significant 80 bits of the 128-bit internal state are used as the key for encrypting a 64-bit zero string with **E** to produce the initial data state; and the least significant 80 bits of the 128-bit internal state are to be used as the initial key state. During the phase of processing associated data, the key state is updated iteratively, and the data state is updated iteratively by first applying the **G** operation with the corresponding key state as the key, and then XORing with the corresponding block of associated data. During the phase of encrypting message, the key state is updated likewise, but the data state is updated iteratively by first applying the $\widehat{\mathbf{G}}$ operation with the corresponding key state as the key, and then XORing with the corresponding message block, and the 48-bit output leaked from $\widehat{\mathbf{G}}$ is XORed with the message block to produce the corresponding ciphertext block. Finally, the data state is encrypted with **E** under the user key K to generate an authentication tag. We refer the reader to [9] for the specification of LAC.

Denote the bit length of a message by msl. The message padding of LAC will append the smallest number u of zeros such that $(u + msl + 40) \bmod 48 = 0$ and then append the message length msl on the subsequent 40 bits (the length of a message is limited to be at most 2^{40} bits long). Accordingly, the $40 + u$ bits of the last one or two ciphertext blocks that correspond to the u bit positions of the appended u zeros and the 40 bit positions for message length msl will be truncated. In particular, when the last message block is full, LAC will make an additional 48-bit message block of the form $0^8||msl$, and the resulting ciphertext block will be discarded without transmission.

More specifically, suppose m is the last message block of a message, then the padding is of the form $m||\underbrace{0\cdots0}_{u}||msl$, such that $(u + |m| + 40) \bmod 48 = 0$.

Suppose the 64-bit output of the last $\widehat{\mathbf{G}}$ operation is $X_{17}||X_{16}$ and its 48-bit output is $X_9[9 \sim 32]||X_{17}[9 \sim 32]$, where X_9, X_{16}, X_{17} are 32-bit blocks, $X_9[9 \sim 32]$ represents bits $(9, 10, \cdots, 32)$ of X_9, and so on. If $|m| \leq 8$, then the last ciphertext block before truncation is $(X_9[9 \sim 32]||X_{17}[9 \sim 32]) \oplus (m||\underbrace{0 \cdots 0}_{u}|msl)$, and the bits corresponding to $(\underbrace{0 \cdots 0}_{u}|msl)$ will be truncated before transmission. Padding is similar if the bit length of the associated data is not a multiple of 48.

2.3 Variants of LAC

Denote by mml the bit number of the maximum message length. (CAESAR requires that a maximum message length must not be smaller than 65536 bytes [4]). Now we define some variants of LAC as follows:

- $24 \leq mml \leq 31$ is used. Thus, the resulting message padding is to append the smallest number u of zeros such that $(u + msl + mml) \bmod 48 = 0$ and the message length msl on the subsequent mml bits. ($mml = 40$ for LAC.)
- For the last block m of a message of msl bits long, the padding is of the form $msl||\underbrace{0 \cdots 0}_{u}||m$, such that $(u + msl + mml) \bmod 48 = 0$[1]. This is the reverse order of the original LAC.
- All the other specifications of the variants are exactly the same as LAC, (including that when the last message block is full, the variants will make an additional 48-bit message block and the resulting ciphertext block will be discarded without transmission).

The first two points may be easily made in reality, due to various reasons, for example, the first point may be deliberate to meet the different message length of a particular application, and the second point may be accidental due to a confusion with endianness, particularly when employing a different cipher.

We note that for our attacks, there are some trivial equivalents to the above variants, for example, a variant assuming that there is no message padding if the last message block is full — a popular manner for message padding for message authentication schemes, and another variant assuming that the last ciphertext block for the padded message block will be transmitted without truncation.

We denote by $\widehat{\text{LAC}}$ the variants of LAC, as well as their equivalents with respect to our attacks. To make it easier to describe our attacks, we define four 64-bit (secret) parameters A, B, X, Y to represent the values at the four internal states marked in Fig. 1, that is, A is the output of the first $\mathbf{E}$ operation; B is the output of the second $\mathbf{E}$ operation; X is the output of the last $\widehat{\mathbf{G}}$ operation; and Y is the input to the last $\mathbf{E}$ operation.

[1] An equivalent of this point under our attack is that the position of the most significant 24 bits of the output of the eighth round of the $\widehat{\mathbf{G}}$ operation is exchanged with the position of the most significant 24 bits of the output of the sixteenth round of the $\widehat{\mathbf{G}}$ operation, (without reversing the message padding order), that is $(X_{17}[9 \sim 32]||X_9[9 \sim 32])$.

3 State Recovery Attack on $\widehat{\text{LAC}}$

In this section, we present a state recovery attack on $\widehat{\text{LAC}}$ in a nonce-respecting scenario (under the same key). The attack requires only chosen queries to the encryption and tag generation oracle of $\widehat{\text{LAC}}$, and it can recover the 128-bit internal state immediately after the first two $\mathbf{E}$ operations for one of some used PMNs, more advantageously than exhaustive key search.

3.1 Attack Procedure

The attack procedure is made up of three phases, to be described in Subsects. 3.1.1, 3.1.2 and 3.1.3. Observe that for a message of msl bits long such that $msl \bmod 48 = 48 - mml$, the last-block message-ciphertext pair reveals $(48 - mml)$ bits of the 64-bit output of the last $\widehat{\mathbf{G}}$ operation, by the specification of $\widehat{\text{LAC}}$.

3.1.1 Phase I

This phase works in a chosen-message and known-nonce scenario with fixed associated data, which is illustrated in Fig. 2.

(a) Choose an arbitrary message M of msl bits long such that $msl \bmod 48 = 48 - mml$, and represent it as $(M_1, M_2, \cdots, M_{mbn})$, where the first $mbn - 1$ blocks are 48 bits long each, and $mbn = \lceil \frac{msl}{48} \rceil < \frac{2^{mml}}{48}$. Query the $\widehat{\text{LAC}}$ encryption and tag generation oracle with the message M and associated data $(AD_1, AD_2, \cdots, AD_{abn})$ of abn 48-bit blocks long for 2^{ϕ} times ($abn \geq 0$, and the last one or two blocks are padded ones), where ϕ meets the following Condition (1), and ϕ and mbn meet the following Condition (2):

$$2^{\phi + mml - 64} \ll 1 - \mathsf{e}^{-2^{2\phi - 64}}; \tag{1}$$

$$2^{2\phi + 2 \times mml - 48 \times mbn + 32} \ll 1 - \mathsf{e}^{-2^{2\phi - 64}}. \tag{2}$$

For the i-th query ($i = 1, 2, \cdots, 2^{\phi}$), we denote by:

- PMN_i the PMN used;
- $C^{(i)} = (C_1^{(i)}, C_2^{(i)}, \cdots, C_{mbn}^{(i)})$ the ciphertext, where the first $mbn - 1$ blocks are 48 bits long each, and the last block $C_{mbn}^{(i)}$ is $msl \bmod 48 = 48 - mml$ bits long;
- T_i the tag; and
- A_i, B_i, X_i, Y_i respectively for the four parameters A, B, C, D defined in Sect. 2.3.

Observe that

$$X_i[41 \sim (88 - mml)] = (M_{mbn} \oplus C_{mbn}^{(i)})[1 \sim (48 - mml)],$$
$$Y_i = X_i \oplus (0^{16 + mml} || M_{mbn}).$$

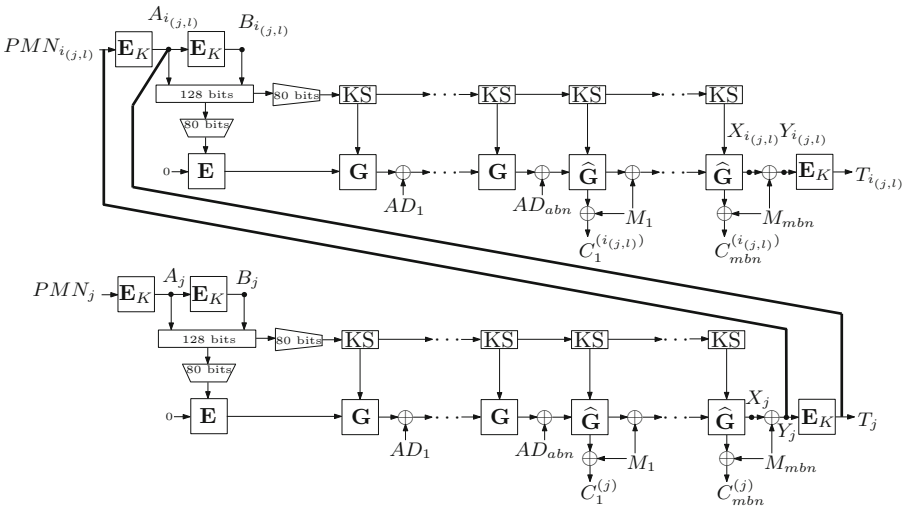

Fig. 2. Phase I of the state recovery attack on $\widehat{\text{LAC}}$

(b) For each permutation (PMN_i, PMN_j) of two PMNs PMN_i and PMN_j,[2] $(1 \leq i, j \leq 2^{\phi}, j \neq i)$, check whether $PMN_i = Y_j$ partially by checking whether

$$\begin{aligned} & PMN_i[41 \sim (88 - mml)] \\ = \ & Y_j[41 \sim (88 - mml)] \\ (= \ & X_j[41 \sim (88 - mml)] \oplus (0^{16+mml} || M_{mbn})[41 \sim (88 - mml)]) \\ (= \ & (M_{mbn} \oplus C_{mbn}^{(j)})[1 \sim (48 - mml)] \oplus \\ & (0^{16+mml} || M_{mbn})[41 \sim (88 - mml)]), \end{aligned} \tag{3}$$

which can be done efficiently by storing PMN_j in a table indexed by $Y_j[41 \sim (88 - mml)]$. Keep only the qualified permutations (PMN_i, PMN_j), and we denote by $PMN_{i_{(j,l)}}$ the qualified PMNs PMN_i for PMN_j, where l is the number of qualified PMNs PMN_i. Thus, we have $PMN_{i_{(j,l)}}[41 \sim (88 - mml)] = Y_j[41 \sim (88 - mml)]$. Furthermore, we have:

$$\text{if } PMN_{i_{(j,l)}} = Y_j, \text{then } A_{i_{(j,l)}} = T_j.$$

3.1.2 Phase II

This phase works in a chosen-message and chosen-nonce scenario with arbitrary associated data, which is illustrated in Fig. 3.

[2] Note that (PMN_i, PMN_j) is a permutation, rather than a combination. Thus, (PMN_i, PMN_j) and (PMN_j, PMN_i) are different.

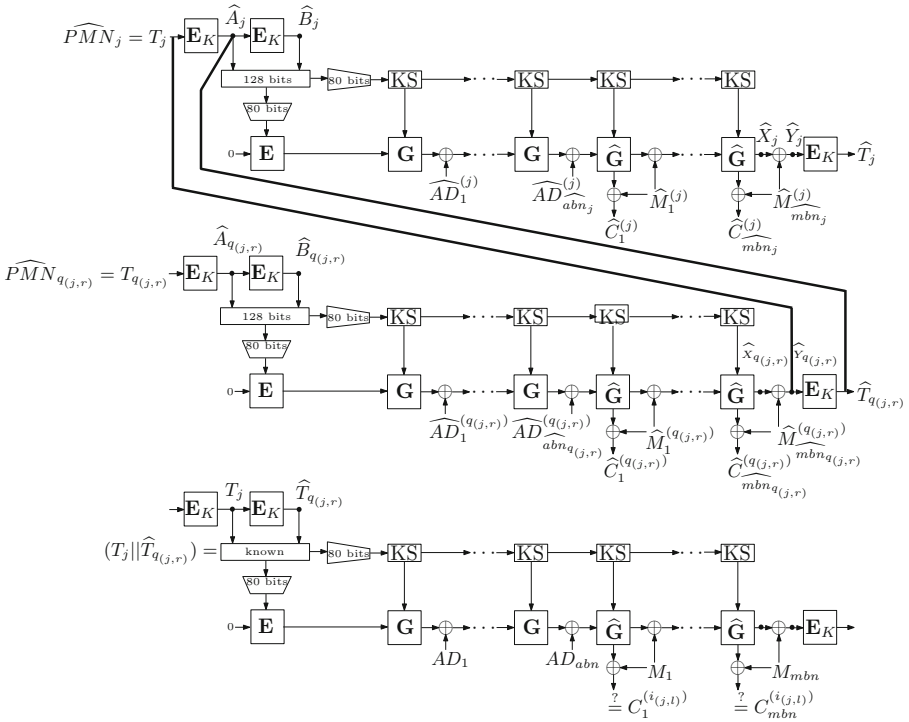

Fig. 3. Phase II of the state recovery attack on $\widehat{\text{LAC}}$

(a) Let $\mathcal{S} = \{T_j$ such that $|PMN_{i_{(j,l)}}| > 0, j = 1, 2, \cdots, 2^{\phi}\}$, and $|\mathcal{S}| = 2^{\beta}$. Query the $\widehat{\text{LAC}}$ encryption and tag generation oracle with an arbitrary associated data for any PMN $PMN = T_j \in \mathcal{S}$ in a chosen-nonce scenario. For $PMN = T_j$, we denote

- the PMN by $\widehat{PMN}_j (= T_j)$;
- the associated data by $(\widehat{AD}_1^{(j)}, \widehat{AD}_2^{(j)}, \cdots, \widehat{AD}_{\widehat{abn}_j}^{(j)})$, that is $\widehat{abn}_j$ 48-bit blocks long, ($\widehat{abn}_j \geq 0$, $\widehat{abn}_j$ can be different one another, and the last one or two blocks are padded ones);
- by $\widehat{M}^{(j)} = (\widehat{M}_1^{(j)}, \widehat{M}_2^{(j)}, \cdots, \widehat{M}_{\widehat{mbn}_j}^{(j)})$ the message of $\widehat{msl}_j$ bits long such that $\widehat{msl}_j \bmod 48 = 48 - mml$, where $\widehat{mbn}_j = \lceil \frac{\widehat{msl}_j}{48} \rceil < \frac{2^{mml}}{48}$, the first $\widehat{mbn}_j - 1$ blocks are 48 bits long each, ($\widehat{mbn}_j$ can be different one another);
- the ciphertext by $\widehat{C}^{(j)} = (\widehat{C}_1^{(j)}, \widehat{C}_2^{(j)}, \cdots, \widehat{C}_{\widehat{mbn}_j}^{(j)})$, where the first $\widehat{mbn}_j - 1$ blocks are 48 bits long each, and the last block $\widehat{C}_{\widehat{mbn}_j}^{(j)}$ is $\widehat{msl}_j \bmod 48 = 48 - mml$ bits long;

– the tag by $\widehat{T}_j$; and
– $\widehat{A}_j$, $\widehat{B}_j$, $\widehat{X}_j$, $\widehat{Y}_j$ respectively for the four parameters A, B, C, D defined in Sect. 2.3.

Note that if $\widehat{PMN}_j (= T_j)$ happens to appear in Step I-(a) we can reuse the corresponding associated data, message, ciphertext and tag, (without querying for $\widehat{PMN}_j$ here).

(b) For each permutation $(\widehat{PMN}_p, \widehat{PMN}_q)$ of two PMNs $\widehat{PMN}_p$ and $\widehat{PMN}_q$,[3] $(1 \le p \ne q \le 2^\beta)$, check whether $\widehat{PMN}_p = \widehat{Y}_q$ partially by checking whether

$$\begin{aligned}
&\widehat{PMN}_p[41 \sim (88 - mml)] \\
=\ &\widehat{Y}_q[41 \sim (88 - mml)] \\
(=\ &\widehat{X}_q[41 \sim (88 - mml)] \oplus (0^{16+mml} || \widehat{M}^{(q)}_{\widehat{mbn}_q})[41 \sim (88 - mml)]) \\
(=\ &(\widehat{M}^{(q)}_{\widehat{mbn}_q} \oplus \widehat{C}^{(q)}_{\widehat{mbn}_q})[1 \sim (48 - mml)] \oplus \\
&(0^{16+mml} || \widehat{M}^{(q)}_{\widehat{mbn}_q})[41 \sim (88 - mml)]),
\end{aligned}$$

which can be similarly done efficiently by storing $\widehat{PMN}_p$ in a table indexed by $\widehat{Y}_p[41 \sim (88 - mml)]$. Keep only the qualified permutations $(\widehat{PMN}_p, \widehat{PMN}_q)$. In particular, for $\widehat{PMN}_j (= T_j)$, we denote by $\widehat{PMN}_{q_{(j,r)}} (= T_{q_{(j,r)}})$ the qualified PMNs, where r is the number of qualified PMNs for $\widehat{PMN}_j$. Thus, we have

$$\begin{aligned}
&\widehat{PMN}_j[41 \sim (88 - mml)] \\
=\ &\widehat{Y}_{q_{(j,r)}}[41 \sim (88 - mml)] \qquad (4) \\
(=\ &\widehat{X}_{q_{(j,r)}}[41 \sim (88 - mml)] \oplus (0^{16+mml} || \widehat{M}^{(q_{(j,r)})}_{\widehat{mbn}_{q_{(j,r)}}})[41 \sim (88 - mml)]) \\
(=\ &(\widehat{M}^{(q_{(j,r)})}_{\widehat{mbn}_{q_{(j,r)}}} \oplus \widehat{C}^{(q_{(j,r)})}_{\widehat{mbn}_{q_{(j,r)}}})[1 \sim (48 - mml)] \oplus \\
&(0^{16+mml} || \widehat{M}^{(q_{(j,r)})}_{\widehat{mbn}_{q_{(j,r)}}})[41 \sim (88 - mml)]),
\end{aligned}$$

and if $\widehat{PMN}_j = \widehat{Y}_{q_{(j,r)}}$, then $\widehat{A}_j = \widehat{T}_{q_{(j,r)}}$.

(c) For any $(\widehat{PMN}_j (= T_j), \widehat{PMN}_{q_{(j,r)}} (= T_{q_{(j,r)}}))$, treat the corresponding value $(T_j || \widehat{T}_{q_{(j,r)}})$ as the 128-bit secret state immediately after the first two $\mathbf{E}$ operations of $\widehat{\text{LAC}}$, then compute the resulting ciphertext for message $M = (M_1, M_2, \cdots, M_{mbn})$ under the associated data $(AD_1, AD_2, \cdots, AD_{abn})$, and finally check whether the mbn ciphertext blocks respectively match the

[3] Likewise, $(\widehat{PMN}_p, \widehat{PMN}_q)$ is a permutation, so $(\widehat{PMN}_p, \widehat{PMN}_q)$ and $(\widehat{PMN}_q, \widehat{PMN}_p)$ are different.

mbn ciphertext blocks of some $C^{(i_{(j,l)})}$ such that $(PMN_{i_{(j,l)}}, PMN_j)$ is a qualified permutation in Phase I. Record the qualified triples $(PMN_{i_{(j,l)}}, \widehat{PMN}_j, \widehat{PMN}_{q_{(j,r)}})$ only.

3.1.3 Phase III

For a recorded $(PMN_{i_{(j,l)}}, \widehat{PMN}_j, \widehat{PMN}_{q_{(j,r)}})$ in Step II-(c), output $(T_j||\widehat{T}_{q_{(j,r)}})$ as the 128-bit secret state just after the first two $\mathbf{E}$ operations under $PMN = PMN_{i_{(j,l)}}$. As a consequence, we can generate all subsequent internal states except the last $\mathbf{E}_K$ operation under $PMN = PMN_{i_{(j,l)}}$.

3.2 Complexity Analysis

In Step I-(b), for every PMN PMN_j, it is expected that there are $\frac{2^\phi-1}{2^{64}} \approx 2^{\phi-64}$ qualified PMNs PMN_i such that $PMN_i = Y_j$, and there are $\frac{2^\phi-1}{2^{48-mml}} \approx 2^{\phi+mml-48}$ qualified PMNs PMN_j such that Eq. (3) holds, that is $l \approx 2^{\phi+mml-48}$ on average; and the $2^{\phi-64}$ qualified PMNs PMN_i must be among the $2^{\phi+mml-48}$ qualified PMNs $PMN_{i_{(j,l)}}$. Since $\phi \geq 24$ (from Condition (1)) generally, it is expected that $\beta = \phi$ in Phase II. The probability that there exists a qualified permutation (PMN_i, PMN_j) such that $PMN_i = Y_j$ holds is $1-(1-\frac{1}{2^{64}})^{2^\phi \times (2^\phi-1)} \approx 1 - \mathsf{e}^{-2^{2\phi-64}}$.

In Step II-(b), the expected number of distinct $\widehat{PMN}_p$ is 2^β; for every $\widehat{PMN}_p$, it is expected that there are approximately $\frac{2^\beta-1}{2^{64}} \approx 2^{\beta-64}$ qualified PMNs $\widehat{PMN}_q$ such that $\widehat{PMN}_p = \widehat{Y}_q$; and for every $\widehat{PMN}_j (= T_j)$, it is expected that there are approximately $\frac{2^\beta-1}{2^{48-mml}} \approx 2^{\beta+mml-48}$ qualified PMNs $\widehat{PMN}_{q_{(j,r)}} (= T_{q_{(j,r)}})$ such that Eq. (4) holds, that is $r \approx 2^{\beta+mml-48}$ on average; and the $2^{\beta-64}$ qualified PMNs $\widehat{PMN}_q$ must be among the $2^{\beta+mml-48}$ qualified PMNs $\widehat{PMN}_{q_{(j,r)}}$. Hence, the expected number of the set $\{(\widehat{PMN}_j (= T_j), \widehat{PMN}_{q_{(j,r)}} (= T_{q_{(j,r)}})) | j = 1, 2, \cdots, 2^\beta\}$ is $2^\beta \times 2^{\beta+mml-48} = 2^{2\beta+mml-48}$.

In Step II-(c), for a permutation $(\widehat{PMN}_j, \widehat{PMN}_{q_{(j,r)}})$, the resulting ciphertext matches the ciphertext of some $C^{(i_{(j,l)})}$ such that $(PMN_{i_{(j,l)}}, PMN_j)$ is a qualified permutation in Phase I is expected to be approximately $2^{\phi+mml-48} \times (2^{-48})^{mbn} = 2^{\phi+mml-48\times(mbn+1)}$. Hence, the expected number of the recorded $(PMN_{i_{(j,l)}}, \widehat{PMN}_j, \widehat{PMN}_{q_{(j,r)}})$ is

$$2^{2\beta+mml-48} \times 2^{\phi+mml-48\times(mbn+1)} = 2^{3\phi+2\times mml-48\times(mbn+2)}.$$

On the other hand, a permutation $(PMN_{i_{(j,l)}}, \widehat{PMN}_j, \widehat{PMN}_{q_{(j,r)}})$ such that both $PMN_{i_{(j,l)}} = Y_j$ and $\widehat{PMN}_j = \widehat{Y}_{q_{(j,r)}}$ hold is expected to be correct, and thus can pass the filtering condition with probability one. For a permutation $(PMN_{i_{(j,l)}}, PMN_j)$ such that $PMN_{i_{(j,l)}} = Y_j$ holds, the probability that there is a qualified permutation $(\widehat{PMN}_j (= T_j), \widehat{PMN}_{q_{(j,r)}} (= T_{q_{(j,r)}}))$ such that

$\widehat{PMN}_j = \widehat{Y}_{q_{(j,r)}}$ holds is $1-(1-\frac{1}{2^{64}})^{2^{\beta}-1} \approx 2^{\beta-64}$, as $\beta = \phi < 64$. Hence, the probability that there is a permutation $(PMN_{i_{(j,l)}}, \widehat{PMN}_j, \widehat{PMN}_{q_{(j,r)}})$ such that both $PMN_{i_{(j,l)}} = Y_j$ and $\widehat{PMN}_j = \widehat{Y}_{q_{(j,r)}}$ hold is $(1-\mathsf{e}^{-2^{2\phi-64}}) \times 2^{\beta-64} = (1-\mathsf{e}^{-2^{2\phi-64}}) \times 2^{\phi-64}$.

Step I-(a) requires 2^{ϕ} queries and a memory of approximately $2^{\phi} \times (mbn \times \frac{48}{8} + 8 \times 2 + 3 \times 2) = 2^{\phi} \times (6 \times mbn + 22)$ bytes, and Step I-(b) has a computational complexity of approximately $2^{\phi} \times 2^{\phi+mml-48} = 2^{2\phi+mml-48}$ memory accesses to retrieve $(PMN_{i_{(j,l)}}, PMN_j)$. Step II-(a) requires 2^{β} queries and a memory of $\sum_{i=1}^{2^{\beta}}(\widehat{abn}_i \times \frac{48}{8} + 2 \times \widehat{mbn}_i \times \frac{48}{8} + 8 \times 2 + 3 \times 2)$ bytes (it can be reduced by using the same set of associated data and message), and Step II-(b) has a computational complexity of approximately $2^{2\beta+mml-48}$ memory accesses to retrieve $(\widehat{PMN}_j, \widehat{PMN}_{q_{(j,r)}})$. Step II-(c) has a computational complexity of approximately $2^{2\beta+mml-48} = 2^{2\phi+mml-48}$ computations of $\widehat{\text{LAC}}$.

The attack is valid if:

1. the expected number of correct $(PMN_{i_{(j,l)}}, \widehat{PMN}_j, \widehat{PMN}_{q_{(j,r)}})$ recorded in Step II-(c) (that meets both $PMN_{i_{(j,l)}} = Y_j$ and $\widehat{PMN}_j = \widehat{Y}_{q_{(j,r)}}$) is much more than the expected number of wrong $(PMN_{i_{(j,l)}}, \widehat{PMN}_j, \widehat{PMN}_{q_{(j,r)}})$ recorded in Step II-(c), that is

$$2^{3\phi+2\times mml-48\times(mbn+2)} \ll (1-\mathsf{e}^{-2^{2\phi-64}}) \times 2^{\phi-64},$$

   which corresponds to Condition (2); and
2. the expected computational complexity for the attack to recover an internal state is less than the computational complexity of a generic attack (e.g. exhaustive key search) recovering an internal state, that is

$$\frac{2^{2\phi+mml-48}}{(1-\mathsf{e}^{-2^{2\phi-64}}) \times 2^{\phi-64}} \ll 2^{80},$$

   which corresponds to Condition (1).

A simple analysis of Conditions (1) and (2) reveals that there is a solution for ϕ and mbn when $(24 \leq) mml \leq 31$. Therefore, working in the nonce-respecting scenario, the state recovery attack requires $2^{\phi+1}$ queries on the $\widehat{\text{LAC}}$ encryption and tag generation oracle and a memory of $\mathcal{O}(2^{\phi+1})$ bytes, and has a computational complexity of approximately $2^{2\phi+mml-48}$ computations of $\widehat{\text{LAC}}$, with a success probability of $(1-\mathsf{e}^{-2^{2\phi-64}}) \times 2^{\phi-64}$. In particular, when $mml = 30$, we can set $\phi = 32$ and $mbn \geq 4$, and the attack requires 2^{33} queries and a memory of $\mathcal{O}(2^{33})$ bytes, and has a computational complexity of approximately 2^{46} computations of $\widehat{\text{LAC}}$, with a success probability of about $2^{-32.7}$. When $mml = 24$, we can set $\phi = 32$ and $mbn \geq 2$, and the attack requires 2^{33} queries and a memory of $\mathcal{O}(2^{33})$ bytes, and has a computational complexity of approximately 2^{40}

computations of $\widehat{\text{LAC}}$, with a success probability of about $2^{-32.7}$. Obviously, the memory can be reused if we want to repeat the attack with many times, e.g. for different keys.

Note that the attack can apply to the case with $mml = 40$ for the original LAC, but it is slightly inferior to exhaustive key search, for example, when we set $\phi = 32$ and $mbn \geq 4$, the attack requires 2^{33} queries and a memory of $\mathcal{O}(2^{33})$ bytes, and has a computational complexity of approximately 2^{56} computations, with a success probability of about $2^{-32.7}$.

4 Existential Forgery Attack on $\widehat{\text{LAC}}$

Once a concerned 128-bit internal state is recovered by the above state recovery attack, we can make an existential forgery attack on $\widehat{\text{LAC}}$ without any further queries, under the PMN corresponding to the recovered internal state.

For a permutation $(PMN_{i_{(j,l)}}, \widehat{PMN}_j(= T_j), \widehat{PMN}_{q_{(j,r)}}(= T_{q_{(j,r)}}))$ outputted in the above state recovery attack, the corresponding 128-bit secret state just after the first two $\mathbf{E}$ operations is $(T_j||\widehat{T}_{q_{(j,r)}})$. We can choose a message $(\widetilde{M}_1, \widetilde{M}_2, \cdots, \widetilde{M}_{\widetilde{mbn}})$ of $\widetilde{mbn}(\geq 3)$ 48-bit blocks long, and then produce its ciphertext $(\widetilde{C}_1, \widetilde{C}_2, \cdots, \widetilde{C}_{\widetilde{mbn}})$ and tag $\widetilde{T}$ under $PMN = PMN_{i_{(j,l)}}$, for example, in the following way as illustrated in Fig. 4:

1. Compute the first $(\widetilde{mbn} - 3)$ ciphertext blocks $(\widetilde{C}_1, \widetilde{C}_2, \cdots, \widetilde{C}_{\widetilde{mbn}-3})$ for the first $(\widetilde{mbn}-3)$ message blocks $(\widetilde{M}_1, \widetilde{M}_2, \cdots, \widetilde{M}_{\widetilde{mbn}-3})$ until immediately after the output of the last third $\widehat{\mathbf{G}}$ operation.
2. Let $\widetilde{Y} = PMN_{i_{(j,l)}}$ (or T_j). Choose and pad the last message block to form $\widetilde{M}_{\widetilde{mbn}}$, then compute $\widetilde{X} = (0^{16}||\widetilde{M}_{\widetilde{mbn}}) \oplus \widetilde{Y}$, decrypt $\widetilde{X}$ through the last $\widehat{\mathbf{G}}$ operation, and we denote the resulting value by $\widetilde{Z}$ (that is the input to the last $\widehat{\mathbf{G}}$ operation), finally compute the last ciphertext block $\widetilde{C}_{\widetilde{mbn}}$.
3. For $t = 1, 2, \cdots, 2^{16}$, choose randomly at uniform the last second message block $\widetilde{M}^{(t)}_{\widetilde{mbn}-1}$, and do as follows:
   (a) Decrypt $(0^{16}||\widetilde{M}^{(t)}_{\widetilde{mbn}-1}) \oplus \widetilde{Z}$ through the last second $\widehat{\mathbf{G}}$ operation, and check whether the most significant 16 bits of the resulting value correspond to the most significant 16 bits of the 64-bit output of the last third $\widehat{\mathbf{G}}$ operation that is generated in Step 1. If yes, go to the next sub-step; otherwise, repeat Step 3 with a different t.
   (b) Compute the last third message block $\widetilde{M}_{\widetilde{mbn}-2}$ from the output of the last third $\widehat{\mathbf{G}}$ operation and the input to the last second $\widehat{\mathbf{G}}$ operation, and compute the last second and third ciphertext blocks $\widetilde{C}_{\widetilde{mbn}-1}$ and $\widetilde{C}_{\widetilde{mbn}-2}$. The corresponding tag $\widetilde{T} = T_j$ (respectively $\widehat{T}_{q_{(j,r)}}$).

Observe that $\widetilde{Y}$ can be that value corresponding to a different permutation outputted in the above state recovery attack, and $\widetilde{T}$ will be the corresponding value as well.

The attack has a computational complexity of 2^{16} computations of the $\widehat{\mathbf{G}}$ operation to obtain a forgery for a message under $PMN = PMN_{i_{(j,l)}}$, with a success probability of $(1-2^{-16})^{2^{16}} \approx 63\,\%$, (a larger success probability can be achieved by using a larger t). Once made, the forgery can replace the original ciphertext-tag pair under $PMN = PMN_{i_{(j,l)}}$ during online communications, and thus will pass the decryption and tag verification phase of $\widehat{\text{LAC}}$, in the nonce-respecting scenario that does not allow to use a nonce twice in the decryption and tag verification phase.

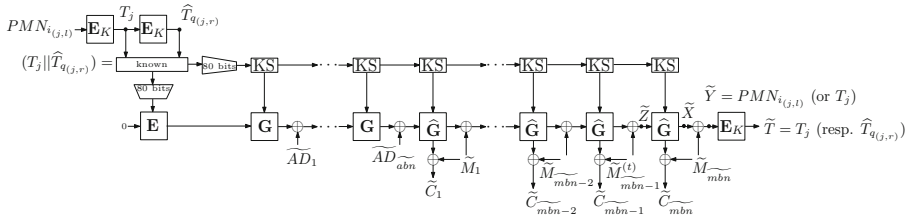

Fig. 4. Existential forgery attack on $\widehat{\text{LAC}}$

Note that the forgery attack that includes the phases of the state recovery attack as a step is meaningless if a forgery is the sole attack goal, because it is slower than a generic attack on a 64-bit tag size. It is a side result of the state recovery attack presented in Sect. 3.

5 Concluding Remarks

In this paper, we have shown that the security of the LAC authenticated encryption algorithm depends greatly on the parameter of the maximum message length and the order of padding the last message block (or equivalently the order of the two halves of the leaked 48-bit output), by presenting a structural state recovery attack on its variants that differ from the original LAC only in the two points. Furthermore, slightly inferior to exhaustive key search, the attack can apply to the LAC variant that differs from the original LAC only in the reverse order of padding the last message block. The attack is only based on the structure of the LAC variants, and may apply to other authenticated encryption algorithms with similar structures. Therefore, an authenticated encryption algorithm with a such or similar structure may be weakened when it is misused with the reverse message padding order (or equivalently the reverse order of the two halves of the leaked output) and different maximum message lengths, and thus users should be very careful about the two points when employing such a structure in reality.

Acknowledgments. The author is grateful to Prof. Wenling Wu and Lei Zhang for their discussions on an earlier version of this work, and to Prof. Yongzhuang Wei and the Natural Science Foundation of China (No. 61572148) for their support.

References

1. Bellare, M., Namprempre, C.: Authenticated encryption: relations among notions and analysis of the generic composition paradigm. J. Cryptology **21**(4), 469–491 (2008)
2. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. J. Cryptology **4**(1), 3–72 (1991). Springer
3. Bogdanov, A., Mendel, F., Regazzoni, F., Rijmen, V., Tischhauser, E.: ALE: AES-based lightweight authenticated encryption. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 447–466. Springer, Heidelberg (2014)
4. CAESAR – Competition for Authenticated Encryption: Security, Applicability, and Robustness. http://competitions.cr.yp.to/caesar.html
5. Leurent, G.: Differential forgery attack against LAC. In: Dunkelman, O., Keliher, L. (eds.) SAC 2015. LNCS, vol. 9566, pp. 217–224. Springer, Heidelberg (2016). doi:10.1007/978-3-319-31301-6_13
6. Lin, L.: Private communications (2014)
7. Wu, S., Wu, H., Huang, T., Wang, M., Wu, W.: Leaked-state-forgery attack against the authenticated encryption algorithm ALE. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 377–404. Springer, Heidelberg (2013)
8. Wu, W., Zhang, L.: LBlock: a lightweight block cipher. In: Lopez, J., Tsudik, G. (eds.) ACNS 2011. LNCS, vol. 6715, pp. 327–344. Springer, Heidelberg (2011)
9. Zhang, L., Wu, W., Wang, Y., Wu, S., Zhang, J.: LAC: a lightweifht authenticated encryption cipher version 1, Submission to the CAESAR competition, 15 March 2014. http://competitions.cr.yp.to/round1/lacv1.pdf