LAC: A Lightweight Authenticated Encryption Cipher

Version 1 — 15 March 2014

Designers and Submitters:

Lei Zhang, Wenling Wu, Yanfeng Wang, Shengbao Wu, Jian Zhang

{zhanglei, wwl, wangyanfeng, wushengbao, zhangjian}@tca.iscas.ac.cn

Trusted Computing and Information Assurance Laboratory
Institute of Software, Chinese Academy of Sciences

Contents

Chapter 1

Introduction

LAC is a lightweight authenticated encryption cipher based on a similar structure of ALE [2] and a simplified version of the lightweight block cipher LBlock as underlying primitive.

The recommended parameter set for LAC is as follows.

	Recommended Parameter (in bits)
Key	80
Public Message Number	64
Secret Message Number	0
Tag	64

Its encryption/authentication procedure accepts an 80-bit master key K, a 64-bit public message number (PMN), a message m, an associated data α, and outputs the ciphertext c of exactly the same length as message and a 64-bit authentication tag τ.

Its decryption/verification procedure accepts an 80-bit key K, a 64-bit public message number (PMN), a ciphertext c, an associated data α, a 64-bit tag τ, and returns the decrypted message m if tag is correct or $\perp$ otherwise.

LAC is a nonce-respecting design, and the public message number is used as nonce. We assume a public message number (PMN) is only used once with the same master key for encryption. Moreover, we restrict that an equivalent of at most 2^{40} bits are allowed to be authenticated or both authenticated and encrypted with the same master key.

Chapter 2

Specification

2.1 Symbols and Notations

Notations	Descriptions
K	80-bit master key
PMN	64-bit public message number
SMN	0-bit secret message number
m	message
α	associate data
c	ciphertext
τ	64-bit authentication tag
$\perp$	a special symbol denotes the verification failure of LAC
0^n	the binary string of n-bit successive "0"
$\|\|$	concatenation of two binary strings
$<<<$	left rotation operation
$[i]_2$	binary form of an integer i
$\|X\|$	the bit length of string X
LBlock-s	the simplified version of lightweight block cipher LBlock
E	the full 32-round LBlock-s
G	the reduced 16-round LBlock-s
$G/leak$	the reduced 16-round LBlock-s with data-leaked
KS	round-reduced key schedule algorithm of LBlock-s
$Trunc_{High}(X, l)$	truncate and output the higher l-bit of X
$Trunc_{Low}(X, l)$	truncate and output the lower l-bit of X

2.2 Specification of LAC

LAC is a lightweight authenticated encryption cipher based on a similar structure of ALE [2] and a simplified version of the lightweight block cipher LBlock. Its encryption/authentication procedure accepts an 80-bit master key K, a 64-bit public message number (PMN), a message m, and an associated data α. We do not support secret message number (SMN) here, namely the length of SMN is 0. Then the encryption/authentication procedure outputs the ciphertext c of exactly the same length as message and a 64-bit authentication tag τ. Its decryption/verification procedure accepts an 80-bit key K, a 64-bit public message number (PMN), a ciphertext c, an associated data α and a 64-bit tag τ. It returns the decrypted message m if tag is correct or $\perp$ otherwise. Meanwhile, we assume a public message number (PMN) is only used once with the same master key for encryption, and an equivalent of at most 2^{40} bits are allowed to be authenticated or both authenticated and encrypted with the

same master key.

The encryption/authentication procedure is shown in Figure 2.1. E is the full 32-round LBlock-s cipher, KS is the key schedule algorithm of 16-round LBlock-s and one more round key state update with the fixed constant `0x15`, G is 16-round LBlock-s, $G/leak$ is 16-round LBlock-s with 48 bits leaked from the data state. The detailed procedure can be described in five steps:

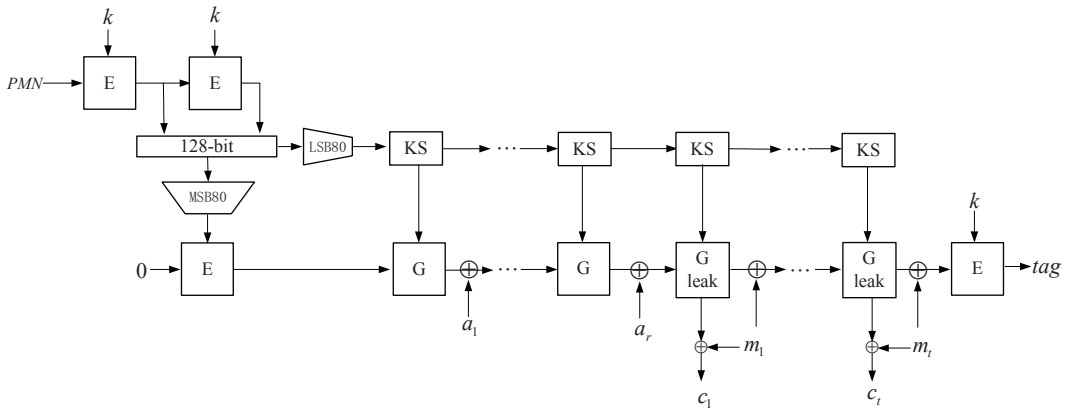

Figure 2.1: The encryption/authentication operation of LAC

Padding: Assume that the length of message m is len bits. It first appends the smallest number x (with $(x + len + 40)\ mod\ 48 = 0,\ 0 \leq x < 48$) of zeros to the message, and then appends the message length len coded on 40 bits such that the length of the result is a multiple of 48-bit. Finally, the resulting padded message is split into t blocks of 48-bit each, $M = m_1||\ldots||m_t$. Note that for associated data α the same padding method is used and the padded associated data is split into r blocks, $A = a_1||\ldots||a_r$.

Initialization: The internal state consists of an 80-bit key state and a 64-bit data state. Firstly, the PMN is encrypted with LBlock-s under the master key K and secondly the 64-bit output is also encrypted under the master key K for full rounds. The 64 bits of the first output and the 64 bits of the second output compose a new 128-bit key $UKey$. The lower 80 bits of $UKey$ are used as the initialized key state. Meanwhile, the data state is initialized by encrypting 0^{64} with LBlock-s under the key obtained from the higher 80-bit of $UKey$. Up to now, both states are initialized.

Processing associated data: Associated data A is processed block by block: The data state is encrypted with 16-round LBlock-s (G function) using the key state as key. The final round subkey is updated one more time using the LBlock-s round key schedule with round constant `0x15`. This value is stored in the key state. Finally, the current block of A is xored to the lower 48-bit of the internal state.

Processing message: Message M is also processed block by block: The data state is encrypted with data-leaked 16-round LBlock-s ($G/leak$ function) using the key state as key. At the same time, the higher 24 bits after the 8-round output and the higher 24 bits after 16-round output, 48 bits in all are leaked from the data state. The leaked bits are xored to the current message block m_i to produce the ciphertext block c_i. The final round subkey is updated one more time using the LBlock-s round key schedule with round constant `0x15`. The current message block m_i is also xored to the lower 48-bit of the internal state. Finally, the result data $C = c_1||\ldots||c_t$ is truncated to exactly the same length as message and output as ciphertext c. The last few bits corresponding to padding data will be processed as above but without outputting as the ciphertext.

Finalization: Finally, the data state is encrypted with LBlock-s using the master key K. The output of this encryption is returned as the authentication tag for the message and associated data.

The decryption/verification procedure of LAC can be defined correspondingly. The only two differences are that one works with the ciphertext $c = c_1||\ldots||c_t$ instead of the message m while xoring with the stream and that the supplied tag value is compared to the one computed by the algorithm. Moreover, only if the tag is correct the decrypted message is returned.

2.3 Description of LBlock-s

LBlock-s is a simplified version of lightweight block cipher LBlock. To reduce the cost in software and hardware implementation, 10 different 4-bit sboxes in LBlock are replaced with a same 4-bit sbox. Besides, the modified key schedule algorithm proposed in [5] are used to avoid the implementation restriction of $\lll$ 29 in 8-bit platform and to improve the diffusion effect of key schedule algorithm. Moreover, since LAC does not need the decryption procedure of LBlock-s, we can apply 32 identical rounds and do not have to omit the switch operation in the last round. This can avoid additional control operation in hardware implementation. Specifically, the round function of LBlock-s can be described as follows:

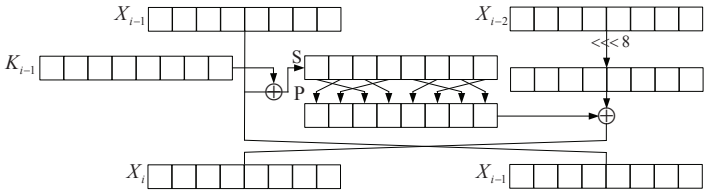

Figure 2.2: The round function of block cipher LBlock-s

Encryption Algorithm. The general structure of LBlock-s is a variant of Feistel Network, which is depicted in Figure 2.2. The number of iterative rounds is 32. The round function of LBlock-s includes three basic functions: round subkey addition AK, confusion function S and diffusion function P. The non-linear layer S consists of 8 identical 4-bit sbox (Table 2.1) in parallel. The diffusion function P is defined as a permutation of eight 4-bit nibbles.

Table 2.1: Contents of the 4-bit sbox

i	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(i)$	E	9	F	0	D	4	A	B	1	2	8	3	7	6	C	5

The 64-bit input is divided into two parts $X_1||X_0$. The same round function is iterated for 32 rounds and then obtains the 64-bit output $X_{33}||X_{32}$. The round function can be described as follows:

$$X_i = P(S(X_{i-1} \oplus K_{i-1})) \oplus (X_{i-2} \lll 8),\ i = 2, 3, ..., 33.$$

Key Schedule Algorithm. The 80-bit master key K is stored in a key register and denoted as $K = k_{79}k_{78}...k_0$. Output the leftmost 32 bits of register K as subkey K_1. For $i = 1, ..., 31$, update the key register K as follows:

1. $K \lll 24$

2. $[k_{55}k_{54}k_{53}k_{52}] = s[k_{79}k_{78}k_{77}k_{76}] \oplus [k_{55}k_{54}k_{53}k_{52}]$
   $[k_{31}k_{30}k_{29}k_{28}] = s[k_{75}k_{74}k_{73}k_{72}] \oplus [k_{31}k_{30}k_{29}k_{28}]$
   $[k_{67}k_{66}k_{65}k_{64}] = [k_{71}k_{70}k_{69}k_{68}] \oplus [k_{67}k_{66}k_{65}k_{64}]$
   $[k_{51}k_{50}k_{49}k_{48}] = [k_{11}k_{10}k_9k_8] \oplus [k_{51}k_{50}k_{49}k_{48}]$

3. $[k_{54}k_{53}k_{52}k_{51}k_{50}] = [k_{54}k_{53}k_{52}k_{51}k_{50}] \oplus [i]_2$

4. Output the leftmost 32 bits of current content of register K as round subkey K_{i+1}.

2.4 Round-reduced LBlock-s

Besides the full 32-round LBlock-s, the LAC algorithm also needs round-reduced LBlock-s including the 16-round LBlock-s named G function, 16-round LBlock-s with data-leaked named $G/leak$ and the round-reduced key schedule algorithm KS. The definitions are shown as :

G : For 64-bit input $X_1||X_0$ and subkeys $K_1, K_2, \ldots, K_{16}$, process as follows:

$$X_i = P(S(X_{i-1} \oplus K_{i-1})) \oplus (X_{i-2} \lll 8),\ i = 2, 3, ..., 17.$$

Finally, output the 64-bit $X_{17}||X_{16}$.

G/leak : For 64-bit input $X_1||X_0$ and subkeys $K_1, K_2, \ldots, K_{16}$, process as follows:

$$X_i = P(S(X_{i-1} \oplus K_{i-1})) \oplus (X_{i-2} \lll 8),\ i = 2, 3, ..., 9.$$

Output the higher 24 bits of X_9, denoted as $X_9^* = X_9[31, 30, ..., 8]$. Next continue the following operations:

$$X_i = P(S(X_{i-1} \oplus K_{i-1})) \oplus (X_{i-2} \lll 8),\ i = 10, 11, ..., 17.$$

Output the higher 24 bits of X_{17}, denoted as $X_{17}^* = X_{17}[31, 30, ..., 8]$. Finally output the 64-bit data state $X_{17}||X_{16}$, and concatenate $X_9^*||X_{17}^*$ as the 48-bit leaked output.

KS : For the 80-bit input state of the key register $K = k_{79}k_{78}...k_0$, output the leftmost 32 bits as subkey K_1. Then for $i = 1, 2, ..., 15$, update the key register as Step 1 to Step 4 of the key schedule of LBlock-s and output the round subkeys $K_2, K_3, \ldots, K_{16}$. Finally, update the key register one more time as Step 1 to Step 3 of the key schedule of LBlock-s, with the round constant $i =$ `0x15`.

2.5 Encryption/Authentication Procedure

The encryption/authentication procedure of LAC can be described with the pseudo-code as follows:

Input: an 80-bit master key K, a len-bit message m, an $alen$-bit associated data α and a 64-bit public message number (PMN).

Output: a len-bit ciphertext c and a 64-bit tag τ.

Begin Procedure
- Step 1: $M(m_1||\ldots||m_t) = Padding(m, len)$
- Step 2: $A(a_1||\ldots||a_r) = Padding(\alpha, alen)$
- Step 3: Initialize Procedure
  - Step 3.1: $O_1 = E(PMN, K)$
  - Step 3.2: $O_2 = E(O_1, K)$
  - Step 3.3: $UKey = O_1||O_2$
  - Step 3.4: $K_1 = Trunc_{High}(UKey, 80)$, $K_2 = Trunc_{Low}(UKey, 80)$
  - Step 3.5: $DataRegister = E(0^{64}, K_1)$
- Step 4: Associated Data Processing Procedure

  $KeyRegister = K_2$

  For $i = 1, 2, \ldots, r$, do the following:

  Step 4.1: $(K_1, K_2, \ldots, K_{16}) = KS(KeyRegister)$,

$DataRegister = G(DataRegister, (K_1, K_2, \ldots, K_{16})) \oplus (0^{16}||a_i)$

Step 5: Message Processing Procedure

For $i = 1, 2, \ldots, t$, do the following:

Step 5.1: $(K_1, K_2, \ldots, K_{16}) = KS(KeyRegister)$,

$DataRegister = G/leak(DataRegister, (K_1, K_2, \ldots, K_{16})) \oplus (0^{16}||m_i)$,

$c_i = LeakMessage \oplus m_i$

Step 6: Output Procedure

$c = Trunc_{High}(c_1|| \ldots ||c_t, len)$

$\tau = E(DataRegister, K)$

End Procedure

2.6 Decryption/Verification Procedure

Input: an 80-bit master key K, a len-bit ciphertext c, an $alen$-bit associated data α, a 64-bit public message number(PMN) and a 64-bit tag τ.

Output: a len-bit message m or $\perp$.

Begin Procedure

Step 1: Split c as 48-bit blocks $(c_1|| \ldots ||c_t)$ (note only $1 \leq |c_t| \leq 48$)

Step 2: $A(a_1|| \ldots ||a_r) = Padding(\alpha, alen)$

Step 3: Initialize Procedure

Step 3.1: $O_1 = E(PMN, K)$

Step 3.2: $O_2 = E(O_1, K)$

Step 3.3: $UKey = O_1||O_2$

Step 3.4: $K_1 = Trunc_{High}(UKey, 80)$, $K_2 = Trunc_{Low}(UKey, 80)$

Step 3.5: $DataRegister = E(0^{64}, K_1)$

Step 4: Associated Data Processing Procedure

$KeyRegister = K_2$

For $i = 1, 2, \ldots, r$, do the following:

Step 4.1: $(K_1, K_2, \ldots, K_{16}) = KS(KeyRegister)$,

$DataRegister = G(DataRegister, (K_1, K_2, \ldots, K_{16})) \oplus (0^{16}||a_i)$

Step 5: Message Processing Procedure

Step 5.1: For $i = 1, 2, \ldots, t-1$, do the following:

$(K_1, K_2, \ldots, K_{16}) = KS(KeyRegister)$,

$DataRegister = G/leak(DataRegister, (K_1, K_2, \ldots, K_{16}))$,

$m_i = LeakMessage \oplus c_i$,

$DataRegister = DataRegister \oplus (0^{16}||m_i)$

Step 5.2: $(K_1, K_2, \ldots, K_{16}) = KS(KeyRegister)$,

$DataRegister = G/leak(DataRegister, (K_1, K_2, \ldots, K_{16}))$,

$m_t = Trunc_{High}(LeakMessage, |c_t|) \oplus c_t$,

If($|m_t| \leq 8$)

$DataRegister = DataRegister \oplus (0^{16}||m_t||0^{48-|m_t|-|[len]_2|}||[len]_2)$

Else

$DataRegister = DataRegister \oplus (0^{16}||m_t||0^{48-|m_t|})$,

$(K_1, K_2, \ldots, K_{16}) = KS(KeyRegister)$,

$DataRegister = G/leak(DataRegister, (K_1, K_2, \ldots, K_{16}))$,

$DataRegister = DataRegister \oplus (0^{16}||0^{48-|[len]_2|}||[len]_2)$,

Step 6: Output Procedure

$\tau^* = E(DataRegister, K)$.

If($\tau^* == \tau$) Output $m = (m_1|| \ldots ||m_t)$;

Else Output $\perp$.

End Procedure

Chapter 3

Security Goals

First of all, we stress several assumptions of LAC, and then give the security claims for the cipher.

LAC does not contain secret message number and the public message number is used as a nonce. Like most of the nonce-based designs, a nonce value should be only used once with the same master key for encryption. Therefore, we assume LAC is a nonce-respecting setting, and users/adversaries are required to use the public message number as the non-reused nonce. Note that if the public message number is reused by some careless operation, LAC will lose all of its security claims. Moreover, we assume LAC will abort on verification failure. Namely, in the decryption/verification procedure, if the verification of tag is failed, then the algorithm returns no information beyond the verification failure. This can efficiently reduce the impact of chosen-ciphertext attack and help to provide the integrity security.

Under these assumptions, the security claims for LAC are given as follows:

Claim 1 (Confidentiality for the plaintext)

The security claim of confidentiality for the plaintext is that the expected number of key guesses is no less than 2^{80}.

Claim 2 (Integrity for the plaintext)

The security claim of integrity for the plaintext is that any forgery attack with an unused tuple $(PMN^*, \alpha^*, c^*, \tau^*)$ has a success probability at most 2^{-64}.

Claim 3 (Integrity for the associated data)

The security claim of integrity for the associated data is that any forgery attack with an unused tuple $(PMN^*, \alpha^*, c^*, \tau^*)$ has a success probability at most 2^{-64}.

Claim 4 (Integrity for the public message number)

The security claim of integrity for the public message number is that any forgery attack with an unused tuple $(PMN^*, \alpha^*, c^*, \tau^*)$ has a success probability at most 2^{-64}.

Table 3.1: Security claims of LAC

	plaintext	associated data	public message number
Confidentiality	80-bit security	–	–
Integrity	64-bit security	64-bit security	64-bit security

Chapter 4

Security Analysis

4.1 Extinguishing Differentials Analysis

We evaluated the minimal number of active sboxes for LBlock-s. Results show that 16-round LBlock-s guarantees at least 35 active sboxes. Since the maximum differential probability of the sbox used in LBlock-s is 2^{-2}, the probability of introducing and detecting an internal collision in LAC is less than or equal to 2^{-70}.

In Asiacrypt 2013, Wu et al [7] proposed a new kind of forgery attack named Leaked-State-Forgery attack, which may employ the leaked state of LAC to increase the probability of finding internal collisions. Notice that 6 leaked nibbles in $G/leak$ may be used to increase the probability. After taking these 6 leaked nibbles into account, results show that 16 rounds of LBlock-s provide at least 32 effective active sboxes. Moreover, we enumerate all differential patterns with exact 32 effective sboxes to check whether the differential probability of each effective sbox can reach the maximum probability 2^{-2}, and no result has been found. Thus, the probability of introducing and detecting an internal collision in $G/leak$ is less than 2^{-64}. Furthermore, the searching program illustrates that there is no practical differential characteristic corresponding to the maximum differential probability.

4.2 State Recovery

Subjected to the attacks on Pelican MAC [9], which mainly makes use of the small size of internal state to construct collisions due to birthday paradox, the newly proposed algorithms ALE [2], AEGIS [6], and FIDES [1] all have a big internal state size. In our algorithm the size of the internal state is 144-bit which can resist against the birthday attacks, and because of the assumption of nonce-respecting adversary, the round keys are ensured not to be reused, making the attacks on LEX [3] ineffective here.

However, an attacker can still inject a difference into the state in the tag verification process and obtain the decrypted plaintext if the forgery attack is allowed to be repeated for multiple times with the same key and nonce pair. Especially when the tag length is shorter than the key length as in our algorithm, a successful forgery can be obtained after enough attempts (such as 2^{64} times), but it is still hard to recover the whole state. Moreover, in our specification of LAC, we restrict that at most 2^{40} bits are allowed to be processed with the same master key, which may make this kind of attack more impractical.

4.3 Key Recovery

To recover the master key in LAC, an attacker needs to break the full (32 rounds) round of LBlock-s, which seems to be impossible. Even if the internal state has been completely recovered, the only information exposed to the adversary is just the input and output of the full LBlock-s, which does nothing to help recover the master key.

4.4 Forgery without State Recovery

The security of LAC against forgery attacks mainly depends on the probability of finding an internal collision. The higher the probability is, the weaker the security of LAC against forgery attacks is.

After collecting several known plaintext-ciphertext pairs, including the corresponding nonces and tags, an attacker may insert a difference into the state in the decryption and tag verification process by modifying the ciphertext, and makes a forgery attack if the introduced difference can be cancelled with a high probability.

However, the design of LAC provides enough security margins to avoid this kind of forgery attack by using internal collision. In summary, the probability of finding an internal collision and then making a successful forgery attack is less than 2^{-64}. Therefore, LAC is secure against forgery attacks.

4.5 Distinguishing Attack

If the *PMN* can be used to process different messages under the same key, we can easily obtain a distinguishing attack of LAC. Specifically, the differences of the ciphertexts are equal to that of plaintexts under the same key and same *PMN*. Thus, we can distinguish the cipher LAC from the random function immediately. As a result, the *PMN* is restricted to be used only once under the same key.

Moreover, we note that the length of the master key is 80-bit and the size of the state is 64-bit. According to the birthday attack, a collision on the 144-bit occurs with a high probability after querying the encryption processing with 2^{72} triple (PMN, K, m). The querying messages are fixed to be 5 blocks $m_0||m_1||m_2||m_3||m_4$ and the values of $m_2||m_3$ are fixed for all queries. After obtaining the corresponding ciphertexts and tags, we compute the differences of the message and the corresponding ciphertext to verify if the collision occurs on the first consecutive three blocks, that is $m_i \oplus c_i, 0 \leq i \leq 2$. If the 144 bits are equal for (PMN, K, m) and (PMN^*, K^*, m^*), we expect that the internal state generating from these two queries are equal after m_2 and m_2^*. Because $m_i = m_i^*, 2 \leq i \leq 3$, c_3 also equals to c_3^* for LAC. While, the probability of $c_3 = c_3^*$ is small for a random function. Therefore, we can distinguish the LAC from the random function with a time complexity of 2^{72}. If $K = K^*$, the tag for $(PMN, K, m_0||m_1||m_2||m_3||m_4^*)$ is equal to that of (PMN^*, K^*, m^*) and we denote the tag as τ. Based on the collision-pair, we can successfully show a forgery attack for $(PMN, K, c_0||c_1||c_2||c_3||c_4^*, \tau)$ with a time complexity 2^{72}. However, the security to resist the forgery attack is restricted to 64-bit and this attack makes no influence on the security against forgery attack.

To avoid the unknown potential threat, we restrict that an equivalent of at most 2^{40} bits are allowed to be authenticated or both authenticated and encrypted with the same master key.

4.6 Guess-and-Determine Attack

Since LAC employs a data-leaked module in the encryption procedure, the values of several internal state nibbles may be available to adversary. Hence the guess-and-determine attack may be a possible threat. We evaluate the security of LAC against guess-and-determine attack by the method proposed in [3]. After searching with the tool [10] we believe that when knowing 6 nibbles at both the beginning and the end of 8-middle-round of $G/leak$, it takes no less than 20 guessed-nibbles to recover all middle states.

4.7 Slide Attack

Slide attacks are quite common to the authenticated encryption scheme by exploiting the similarities of the algorithm. However, LAC uses the full 32-round LBlock-s in the initialization

and finalization, but only 16-round LBlock-s for the encryption. In the key schedule algorithm of LBlock-s, different round constants are used to generate subkeys. Moreover, in the padding procedure of LAC, we will append the length of message in the last padded block which can destroy the similarity efficiently. All of these features make it sufficient to break the similarities which can be exploited by slide attacks.

Chapter 5

Features

5.1 Main Features

Here we list some main features of LAC in respect to security and performance aspects. Since LAC is based on similar design structure as ALE, it also shares many strong properties of its design.

One-Pass LAC needs only one pass through the message to provide both encryption and authentication. This allows on-the-fly operation.

On-Line LAC is an online scheme since it does not have to know message length before the last message block is input in the encryption/authentication procedure.

Efficient Security Analysis Security analysis of LAC benefits from existing analysis on ALE, Pelican MAC and LEX which have similar design principles. Hence in the design process and parameter choices of LAC, we consider the previous known analytical results sufficiently and choose rounds number, position of leak-data etc carefully to avoid previous known attacks.

Secure Underlying Primitive LAC exploits a mature lightweight block cipher LBlock-s as its underlying primitive. Hence previous analytical results of LBlock can provide enough security confidence and avoid the suspicion of back-door or potential weakness. Moreover, the optimized hardware/software implementations of LBlock can be reused with only a few simple modifications.

Extremely Lightweight The design of LAC aims at lightweight authenticated cipher scheme. We try to provide enough confidentiality and integrity protections in extreme constraint application environment. Hence by choosing appropriate parameter such as key/public message number size and the underlying block cipher, we can obtain a hardware implementation of LAC in around 2000 GE which can satisfy most lightweight applications.

Flexible and Compatible The design of LAC including the basic module of LBlock-s only utilize basic instructions such as 4-bit table look-up, bit-wise xor and rotation etc, without any complex or special operations such as finite multiplication or integer addition. Hence the implementation of LAC can be very flexible and compatible for various platforms and do not need the support of any special instruction set.

5.2 Hardware Performance Evaluation

The design of LAC is targeted for lightweight hardware implementation. It is basically employing the module of LBlock-s together with a control unit for the other operations of initialization, xor of the leak state with message, and finalization etc.

First of all, load the 64-bit public message number (PMN) and 80-bit master key K in the module of LBlock-s. After the first execution of full round LBlock-s, store the 64-bit output. Then reload the 80-bit master key K and execute the second full round LBlock-s. Concatenate the resultant output to obtain the 128-bit $UKey$. Load the 64-bit null message as plaintext and the higher 80-bit of $UKey$ as master key, and execute the full round LBlock-s to obtain the beginning state of encryption procedure. Then load the lower 80-bit of $UKey$ as master

key, and execute the round-reduced LBlock-s functions $G(G/leak)$ and KS to process the associated data and message in order. Meanwhile, maintain the input and output values in the respective wires and synchronized with the correct order to perform the xor operations of state and message. Finally, reload the 80-bit master key K and execute the finalization full round LBlock-s to obtain the tag.

Therefore, except the basic module of LBlock-s, we still need 80-bit register to store the intermediate value of lower 80-bit of $UKey$, 48-bit xor operation, and a module of control unit to maintain the data in respective wires and choose data in correct order to perform the operation. For the module of LBlock-s, we apply a parallelization implementation which performs one round in one clock cycle. Compared with original LBlock, the main difference of the hardware implementation of LBlock-s employs the same 4-bit sbox in both encryption and key schedule, and the modified key schedule needs four additional 4-bit xor operations. Hence to perform one round LBlock-s occupies about 1360 GE and requires one clock cycle. Therefore, in the above trivial parallelization implementation, LAC occupies about 2030 GE, and requires 144 clock cycles to authenticate and encrypt one block of 48 bits. This cycle count includes the overhead of 128 cycles, which is caused by the four invocation of LBlock-s algorithm in initialization and finalization. Thus, only 16 clock cycles are needed to process each further 48-bit block of data.

Moreover, we can present a more compact implementation of LAC for extreme constraint application environment. Here we utilize a serialized implementation of LBlock-s module. It employs a 4-bit datapath and only one 4-bit sbox is implemented and reused in both encryption and key schedule. Then to perform one round LBlock-s occupies about 1060 GE, and requires 12 clock cycles in total (including encryption and subkey generation). Moreover, in the above LAC implementation, we can utilize external RAM to avoid the 80-bit register needed to store the intermediate result of $UKey$. Therefore, in a more compact serialized implementation, LAC occupies about 1300 GE, and requires 1728 clock cycles to authenticate and encrypt one block of 48 bits. This cycle count includes the overhead of 1536 cycles, which is caused by the four invocation of LBlock-s algorithm in initialization and finalization. Thus, only 192 clock cycles are needed to process each further 48-bit block of data.

5.3 Software Performance Evaluation

In this subsection, we evaluate the performance of LAC in various software platforms such as 8-bit, 32-bit and 64-bit processors in sensor network and high performance server etc. Notice that the overhead of LAC per message amounts to 4 full-round LBlock-s calls, and for long messages, LAC needs about 16-round LBlock-s to both encrypt and authenticate a block of message.

8-bit Platform

For the encryption procedure of one round LBlock-s, we can split 64-bit input block into 16 nibbles and each nibble is 4-bit long. Then one round encryption needs 8 sbox table look-up operations and 16 xor operations in total. For the key schedule procedure, we can store the 80-bit key in 20 nibbles, and to produce a new round subkey needs 2 sbox table look-ups and 6 xor operations. Therefore, one round of LBlock-s needs about 32 instructions in total. Then LAC requires about 4620 instructions to authenticate and encrypt one block of 48 bits. This count includes the overhead of 4096 instructions caused by the four invocation of LBlock-s algorithm needed for initialization and finalization. Then only 524 instructions (including the xor operation of state/leaked state and message) are needed to process each further 48-bit block of data. This count does not consider the overhead for loading and offloading of the data.

32/64-bit Platform

To perform high performance implementation, we utilize several 8×32 big tables to perform the round function of LBlock-s. Then we can finish one round of LBlock-s by simply using 4

table look-ups and 5 xor operations. We give our software implementation performances for LAC with different message lengths in Table 3. The processor used for the benchmarks is an Intel Core i7-3612QM @2.10GHz. Note that here we only give a standard implementation of LAC and do not need any special instruction sets or parallelized operation to optimize the performance.

Table 5.1: Software performance of LAC

message length (bytes)	12	16	32	64	128	256	512	1024	2048	4096
LAC (cycles/byte)	720	589	440	256	206	174	152	144	140	138

5.4 Potential Applications

Nowadays, with the development of wireless communicating and embedded system, many aspects of our modern life are all in need of lightweight applications. Because of its competitive footprint and time-efficient implementation, LAC is particularly suitable for resource-constrained environment, such as public transport, pay TV systems, smart electricity meters, anti-counterfeiting, access control, parking management, identification, goods tracking etc. Meanwhile, LAC can be a preferable choice as the basic cryptographic primitives of smart cards, RFID tags, and sensor nodes, which have similar features, weak computation ability, small storage space, and strict power constraints etc.

5.5 Security and Performance Comparison

Let us show the justification of the recommended parameter set. Firstly, LAC does not support the secret number as AES-GCM does. In LAC, the public message number is processed through the block cipher as the plaintext and plays the same role as the nonce in other nonce-based authenticated encryption ciphers. Meanwhile, the length of key and public message number are respectively equal to the key length and block length of LBlock-s, which is compatible to many previous lightweight block ciphers. Furthermore, the tag is restricted to the same length as the ciphertext without any truncation.

Previously, designers always proposed a provable secure authenticated encryption scheme assuming that the internal block cipher is pseudorandom. After determining the advance encryption standard(AES), the composite authenticated ciphers are recommended by the NIST and GCM-AES is famous for its various applications. Because the security goals of LAC and GCM-AES are different, the comparison of security is meaningless.

LAC is a specific lightweight authenticated encryption cipher. The lightweight property is the most important feature for LAC. That is to say, the area cost is the biggest advantage over other ciphers. As the cost of LBlock-s is much cheaper than that of AES, the area needed for authenticated encryption cipher with LBlock-s is also much less than that with AES block cipher. As shown in hardware performance, the cost of LAC is even less than the performance of AES block cipher. Thus, the hardware performance of LAC has an advantage over AES-GCM and ALE.

For software performance, the intel AES-NI instruction enables the high speed performance in GCM-AES. Obviously, LAC has no advantage on the software performance without using the block cipher instruction. However, the operations of LAC are all basic and easy to process in arbitrary computer and system. As a result, the performance of LAC is more flexible than that of GCM-AES.

Chapter 6

Design Rationale

6.1 Structure

The structure of our algorithm is directly inherited from the ALE. We choose it because of its so many merits. It is a nature and innovative way to enlarge the size of the internal state by a layer of key schedule on the basis of the structure of Pelican MAC. The structure seems quite general and flexible, just like an encryption mode, where the designers can choose the underlying block cipher with some fine tuning to ensure the security requirement. ALE adopts AES, while our algorithm uses the lightweight block cipher LBlock-s, making it more applicable in resource constraint environment.

6.2 The Underlying Encryption Cipher LBlock-s

LBlock-s is an improved version of LBlock [8], which is a lightweight block cipher proposed at ACNS 2011. Let us first revisit the design rationality of LBlock and then explain the reasons for improved parts. The general structure of LBlock is a variant of Feistel network, and its design decisions contain a lot of considerations about security and efficient implementations (such as area, cost and performance etc).

For the sbox layer, 4-bit sbox has much more advantage when implemented in hardware compared with the regular 8-bit sbox. Therefore, 10 different 4-bit sboxes are used in LBlock and all of them can be implemented in hardware with only about 22 GE. Furthermore, in the aspect of security, the sboxes used in LBlock are carefully chosen so that they all fulfill the following conditions: no fix point, completed, best nonlinearity, best differential probability, and good algebraic order etc. In LBlock-s, we only use the same 4-bit sbox (s_0 in LBlock) for ten positions to reduce the cost in hardware but with no influence on security. Thus, the cost in compact serialized implementation for sbox layer can be reduced from 220 GE to 22 GE.

For the diffusion layer, LBlock chooses to use two permutations which can be implemented with no cost in hardware. Instead of the bitwise permutation usually used, the 4-bit nibble-wise permutation in round function can be implemented cheaply not only in hardware but also in software environments such as 8-bit microprocessor platforms. Moreover, the 8-bit rotation of right half in each round can be omitted in 8-bit platform implementation. On the other hand, in the aspect of security requirement, the overall structure of LBlock satisfies that in both encryption and decryption directions it can achieve best diffusion in 8 rounds. Furthermore, the number of differential and linear active sboxes both increase quickly.

Similar to many other lightweight block ciphers, the key schedule algorithm of LBlock is also designed in a stream cipher way. A 29-bit left rotation and two 4-bit S-boxes as non-linear operations are applied to generate the round subkeys. In order to improve the security of LBlock-s, the modified key schedule algorithm [5] is used to get faster diffusion. Firstly, the bit-wise key schedule is revised to nibble-wise algorithm to get a faster implementation on software platform. Secondly, four nibble-wise XORs are added to improve the diffusion.

Finally, the security of the new algorithm LBlock-s against biclique cryptanalysis and related-key attack has been improved.

6.3 Parameter Choices

The choices of parameters in LAC are highly related to the goals of security and efficiency. We enhance the ability of LAC in processing associated data and message by reducing the encryption round of G and $G/leak$, with the precondition that LAC is secure against various attacks.

Thus, the number of encryption rounds in G and $G/leak$, the length of message blocks and the leaked positions in $G/leak$ are carefully chosen to protect LAC against forgery attacks and state recovery attacks, especially the leaked-state-forgery attack of ALE [7] and the guess-and-determine attack of FIDES [4]. In the initialization and finalization of LAC, a 32-round LBlock-s is used to protect LAC against key recovery attack. Since the key schedule algorithm of LBlock-s has a better diffusion property than that of LBlock, and LBlock (also has 32 rounds) has enough secure margin against known attacks, we expect that the initialization and finalization of LAC is strong. The key length of LAC is the same as that used in LBlock.

Moreover, the choice of constant `0x15` in processing associated data and massage has two aspects of consideration. First, it helps to distinguish the round functions of promoting associated data and massage from those of initialization and finalization, thus provides some security to protect LAC against slide attacks. Secondly, the representation of `0x15` in bits is "10101", which influences two nibbles of subkeys.

The designers have not hidden any weaknesses in this cipher.

Chapter 7

Intellectual Property

To the best of our knowledge, neither the LAC authenticated cipher, the LBlock and LBlock-s block cipher, nor any part of our design are covered by any patents or other intellectual-property constraints. We have not, and will not, apply for patents on any part of our design or anything in this document, and we are unaware of any other patents or intellectual-property constraints that cover this work.

If any of this information changes, the submitters will promptly (and within at most one month) announce these changes on the crypto-competitions mailing list.

Chapter 8

Consent

The submitters hereby consent to all decisions of the CAESAR selection committee regarding the selection or non-selection of this submission as a second-round candidate, a third-round candidate, a finalist, a member of the final portfolio, or any other designation provided by the committee. The submitters understand that the committee will not comment on the algorithms, except that for each selected algorithm the committee will simply cite the previously published analyses that led to the selection of the algorithm. The submitters understand that the selection of some algorithms is not a negative comment regarding other algorithms, and that an excellent algorithm might fail to be selected simply because not enough analysis was available at the time of the committee decision. The submitters acknowledge that the committee decisions reflect the collective expert judgments of the committee members and are not subject to appeal. The submitters understand that if they disagree with published analyses then they are expected to promptly and publicly respond to those analyses, not to wait for subsequent committee decisions. The submitters understand that this statement is required as a condition of consideration of this submission by the CAESAR selection committee.

Chapter 9

Test Vector

Key	01 23 45 67 89 AB CD EF FE DC
Public Message Number	FE DC BA 98 76 54 32 10
Message	01 23 45 67 89 AB CD EF FE DC BA 98 76 54 32 10
Associated Data	88 99 AA BB CC DD EE FF
Ciphertext	D2 F8 DC 9D D2 90 0C B2 09 76 CC FA 43 6C B0 9E
Tag	E8 72 F1 D8 5D 97 FE B9

Bibliography

[1] Bilgin, B., Bogdanov, A., Knezevic, M., Mendel, F., Wang, Q.: FIDES: Lightweight Authenticated Cipher with Side-Channel Resistance for Constrained Hardware. CHES 2013. LNCS vol. 8086, pp. 142-158. Springer, (2013)

[2] Bogdanov, A., Mendel, F., Regazzoni, F., Rijmen, V., Tischhauser, E.: ALE: AES-Based Lightweight Authenticated Encryption. Accepted by FSE 2013. `http://www2.compute.dtu.dk/~anbog/fse13-ale.pdf` (2013)

[3] Bouillaguet, C., Derbez, P., Fouque, P.A..: Automatic Search of Attacks on Round-Reduced AES and Applications. CRYPTO 2011. LNCS vol. 6841, pp. 169-187. Springer, (2011)

[4] Dinur, I., Jean, J.: Cryptanalysis of FIDES. accepted by FSE 2014. `https://eprint.iacr.org/2014/058.pdf` (2014)

[5] Wang, Y., Wu, W., Yu, X., Zhang, L.: Security on LBlock against Biclique Cryptanalysis. WISA 2012. LNCS, vol. 7690, pp. 1-14. Springer, (2012)

[6] Wu, H., Preneel, B.: AEGIS: A Fast Authenticated Encryption Algorithm (Full Version). `http://eprint.iacr.org/2013/695.pdf` (2013)

[7] Wu, S., Wu, H., Huang, T., Wang, M., Wu, W.: Leaked-State-Forgery Attack against the Authenticated Encryption Algorithm ALE. ASIACRYPT 2013. LNCS, vol. 8269, pp. 377-404. Springer, (2013)

[8] Wu, W., Zhang, L.: LBlock: A Lightweight Block Cipher. ACNS 2011. LNCS, vol. 6715, pp. 327-344. Springer, (2011)

[9] Yuan, Z., Wang, W., Jia, K., Xu, G., Wang, X.: New Birthday Attacks on Some MACs Based on Block Ciphers. CRYPTO 2009. LNCS, vol. 5677. pp. 209-230. Springer, (2009)

[10] `http://www.lifl.fr/~bouillag/implementation.html`