From: MEGE, Alexandre <alexandre.mege@airbus.com>
Sent: Monday, May 27, 2019 11:33 AM
To: lightweight-crypto
Cc: lwc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: CLAE

Dear All,
It seems CLAE doesn't support messages where PT is a single Byte of value 0xFF.
In that case, the last Byte is removed by CLAE, and the resulting CT is identical as the CT with an empty PT.

Example:

```
Key = 000102030405060708090A0B0C0D0E0F
Nonce = 000102030405060708090A0B
PT = FF
AD =
CT = 612717B0FC93B3FD00B364618D8E2B95
```

```
crypto_aead_decrypt returned bad 'mlen': Got <0>, expected <1> Key = 000102030405060708090A0B0C0D0E0F Nonce
= 000102030405060708090A0B PT = FF AD = FF CT = 2DB0DFDE21A85076A797552E26DD9DDC

crypto_aead_decrypt returned bad 'mlen': Got <0>, expected <1>
```

Same tag as with empty PT:

```
Key = 000102030405060708090A0B0C0D0E0F
Nonce = 000102030405060708090A0B
PT =
AD =
CT = 612717B0FC93B3FD00B364618D8E2B95
```

Best regards,
Alexandre Mege

From: MEGE, Alexandre <alexandre.mege@airbus.com>
Sent: Monday, June 3, 2019 5:24 AM
To: lwc-forum@list.nist.gov
Subject: [lwc-forum] OFFICIAL COMMENT: CLAE

Dear All,
It seems CLAE doesn't support messages where PT is a single Byte of value 0xFF.
In that case, the last Byte is removed by CLAE, and the resulting CT is identical as the CT with an empty PT.

Example:

Key = 000102030405060708090A0B0C0D0E0F
Nonce = 000102030405060708090A0B
PT = FF
AD =
CT = 612717B0FC93B3FD00B364618D8E2B95

crypto_aead_decrypt returned bad 'mlen': Got <0>, expected <1> Key = 000102030405060708090A0B0C0D0E0F Nonce = 000102030405060708090A0B PT = FF AD = FF CT = 2DB0DFDE21A85076A797552E26DD9DDC

crypto_aead_decrypt returned bad 'mlen': Got <0>, expected <1>

Same tag as with empty PT:

Key = 000102030405060708090A0B0C0D0E0F
Nonce = 000102030405060708090A0B
PT =
AD =
CT = 612717B0FC93B3FD00B364618D8E2B95

Best regards,
Alexandre Mege

From:	Liu, Dongxi (Data61, Marsfield) <Dongxi.Liu@data61.csiro.au>
Sent:	Tuesday, June 4, 2019 1:30 AM
To:	MEGE, Alexandre; lwc-forum@list.nist.gov
Subject:	[lwc-forum] RE: OFFICIAL COMMENT: CLAE

Dear Alexandre,

Thanks for interest.

CLAE does not support any too-short messages ending with one or more 0xFFs, due to its trivial padding method. This limitation is described in Section 5 of CLAE specification. For short messages, users are supposed to pre-process the messages to avoiding 0xFF at the tail. CLAE is open to any other padding methods to pre-process messages; for a long-enough message, the trivial padding method of CLAE is not invoked in both encryption and decryption.

Regards,
Dongxi Liu

-----Original Message-----
From: MEGE, Alexandre <alexandre.mege@airbus.com>
Sent: Monday, 3 June 2019 7:24 PM
To: lwc-forum@list.nist.gov
Subject: [lwc-forum] OFFICIAL COMMENT: CLAE

Dear All,
It seems CLAE doesn't support messages where PT is a single Byte of value 0xFF.
In that case, the last Byte is removed by CLAE, and the resulting CT is identical as the CT with an empty PT.

Example:

```
Key = 000102030405060708090A0B0C0D0E0F
Nonce = 000102030405060708090A0B
PT = FF
AD =
CT = 612717B0FC93B3FD00B364618D8E2B95
```

crypto_aead_decrypt returned bad 'mlen': Got <0>, expected <1> Key = 000102030405060708090A0B0C0D0E0F Nonce = 000102030405060708090A0B PT = FF AD = FF CT = 2DB0DFDE21A85076A797552E26DD9DDC

crypto_aead_decrypt returned bad 'mlen': Got <0>, expected <1>

Same tag as with empty PT:

```
Key = 000102030405060708090A0B0C0D0E0F
Nonce = 000102030405060708090A0B
PT =
AD =
CT = 612717B0FC93B3FD00B364618D8E2B95
```

From: MEGE, Alexandre <alexandre.mege@airbus.com>
Sent: Wednesday, June 5, 2019 3:23 AM
To: Liu, Dongxi (Data61, Marsfield); lwc-forum@list.nist.gov
Subject: [lwc-forum] RE: OFFICIAL COMMENT: CLAE

Dear Dongxi,

Thank you for the quick feedback.
I confirm the limitation with 0xFF at the tail in CLAE only applies to short messages as described in section 5 of CLAE specification.
Messages longer than 8 Bytes are not affected by this limitation..

Regards,
Alexandre Mège

-----Original Message-----
From: Liu, Dongxi (Data61, Marsfield) [mailto:Dongxi.Liu@data61.csiro.au]
Sent: Tuesday, June 04, 2019 7:30 AM
To: MEGE, Alexandre; lwc-forum@list.nist.gov
Subject: [lwc-forum] RE: OFFICIAL COMMENT: CLAE

Dear Alexandre,

Thanks for interest.
CLAE does not support any too-short messages ending with one or more 0xFFs, due to its trivial padding method. This limitation is described in Section 5 of CLAE specification. For short messages, users are supposed to pre-process the messages to avoiding 0xFF at the tail. CLAE is open to any other padding methods to pre-process messages; for a long-enough message, the trivial padding method of CLAE is not invoked in both encryption and decryption.

Regards,
Dongxi Liu

-----Original Message-----
From: MEGE, Alexandre <alexandre.mege@airbus.com>
Sent: Monday, 3 June 2019 7:24 PM
To: lwc-forum@list.nist.gov
Subject: [lwc-forum] OFFICIAL COMMENT: CLAE

From:	Andre Schrottenloher <andre.schrottenloher@inria.fr>
Sent:	Tuesday, June 11, 2019 3:55 AM
To:	lightweight-crypto
Cc:	lwc-forum@list.nist.gov
Subject:	OFFICIAL COMMENT: CLAE
Attachments:	clae_collisions.c; clae_collisions.py

Dear all,

It seems that CLAE is vulnerable to practical forgery attacks, due to ciphertext / tag collisions occurring with a high probability:
- AD forgeries: with the same key, nonce and message, collisions of the ciphertext (and tag) between an AD "0" (a single byte) and an empty AD occur with high probability
- with the same key, same message, same AD, collisions of the ciphertext (and tag) between two nonces (of a certain form, with a small difference) occur with high probability

Below are the details and examples.
Attached to this email is a Python implementation of CLAE and some code to generate such collisions, as well as some C code that simply verifies these examples with the implementation provided in the submission.

I would like to thank Dongxi Liu for his earlier response and validation.

=================================
1. Same message, different AD
Take a random message m and nonce n, and a random key. Encrypt m,n under the key with empty AD on the one hand, with AD 0 (one byte) on the other hand.
Then the ciphertexts/tags collide with high probability. Here is an example:

key = C0C87CB1A40E388B5E1357F9B07C1116
nonce = AD098AE68BC10AFCF5870593
Message = 429632DB47
Ciphertext is:
F914FFC0BB055B86FF44D4ADDD91AA1B

Explanation

The only difference between encrypting n,m, AD = [] and n,m,AD = [0] lies in the function spillA which is applied only once (as there is a single AD block). The function first computes two indices r0 and l0, then updates two bytes in the ciphertext c[l0] and c[r0] accordingly.
However, there is (seemingly little more than) a 1/256 chance that these two indices are equal, in which case there is (seemingly little more than) a
1/256 chance that the updates of the ciphertext cancel out. At this point, the internal state becomes the same in both cases, so is the output.

==================================
2. Same message, same AD, two nonces

For simplicity, consider a key of the form 00 01 * (it starts with a byte
0 and a byte 1). For a random key, this happens with probability 2^{-16}.
Consider the encryption of an empty message and an empty AD with nonces n1
and n2 of this form:

n1 = 01******01*****
n2 = 00******00*****

Then the ciphertexts/tags collide with high probability. Here is an example:

key = 00012224A200AE805638793A94345087
nonce1 = 00A9B0B8FD7B0055E72C0F0D
nonce2 = 10A9B0B8FD7B1055E72C0F0D
Ciphertext is:
9465D2BD2488D417B2BA7D7EC1D84C2F

Explanation attempt

Following the specification: in the case of an empty message / AD, the
function "vecIV" is first called once with the 12-byte nonce to initialize
a two-byte internal state: we can suppose that it collides between n1 and
n2 with probability 2^{-16}. Afterwards, the function "enc_byte" is called
8 times, and produces 8 pairs of bytes for the ciphertext.

A call of "enc_byte" calls "vecIV" twice, using both 6-byte halves of
the IV.
This is the only nonce-dependent computation. We only have to make sure
that the outputs of these 12 "vecIV" calls will be equal when n2 is
replaced
by n1.

So we consider "vecIV" with inputs k (which does not change), a half of the
IV (which changes), s0 and s1 (which are two internal state bytes that
we can
suppose random, and do not change).

If we change the 4 msbs of the first byte of the (half) IV, then only the
last iteration of the loop is affected, precisely the line:
o[1] = (o[1]+k[o[0]>>4]+k[o[1]&0x0F]+k[b1])^b0^b1;

where b1 is changed. If we consider that (o[1]+k[o[0]>>4]+k[o[1]&0x0F]) is
a random value s at this point, we wish to obtain:
((s + k[b1]))^b0^b1 = ((s + k[b1']))^b0^b1'

For simplicity, we can take the initial b1 to be zero.
Basically, obtaining (s + k[0]) = (s + k[b1'])^b1' is not difficult.

For example if k[0] = 0 , b1' = 1 and k[1] = 1, this happens for a random
s with probability (at least) 1/2.

This condition must be verified 16 times for random values s, with a total
probability >= 2^{-16}. In practice, with empty messages, random keys and
nonces of the form above, a tag collision occurs with higher probability
than this.

With keys and nonces of the same form, and random messages of length
greater than 8 bytes, collisions occur even more often, although I have
no clear explanation. Here is an example:

key = 000128FA684D7C6C383414BAF2CE0E51
Message = 6BBAF8FF1C29F54D7335
nonce1 = 00C6845D50220089D56DE6AD
nonce2 = 10C6845D50221089D56DE6AD
Ciphertext is:
5A568636746261FFB1D0481FFEEF9DEE2427

Adding an AD is possible since the collision on the internal state has
already occurred at this point. Here is an example with non-empty AD:

key = 000141BAACAB4FDF86783AB21C44412F
Message = AA190375CA4D895BBEAA
nonce1 = 0045EDB589CE00FA31E93181
nonce2 = 1045EDB589CE10FA31E93181
Ad = 68EE9224DD
Ciphertext is:
14BA1DC650ECAB2D2ECA557BB7B975A69A52
14BA1DC650ECAB2D2ECA557BB7B975A69A52

Best regards,

André Schrottenloher

From: Liu, Dongxi (Data61, Marsfield) <Dongxi.Liu@data61.csiro.au>
Sent: Thursday, July 4, 2019 1:16 AM
To: Andre Schrottenloher; lightweight-crypto
Cc: lwc-forum@list.nist.gov
Subject: RE: [lwc-forum] OFFICIAL COMMENT: CLAE
Attachments: clae_collisions_revised.py; encrypt.c

Dear Andre and All,

Andre reported two forgery attacks to CLAE. Thanks for the analysis.
In the first forgery attack, AD can be forged with higher probability for some values. In the second forgery attack, two nonces lead to the same ciphertexts, but not compromising the integrity of messages and AD.

Attached please find our revised CLAE to address such vulnerabilities. For the first one, the rounds of processing AD is increased from 1 round (the submitted version) to 4 rounds (determined by tag length); additionally, the lowest 4 bits of adlen is added into the initial state. For the second one, 12-byte nonces are further processed by regarding them as extra AD. Changes are marked with "Revised CLAE" in encrypt.c.

Thanks.

Best regards,
Dongxi Liu

-----Original Message-----
From: Andre Schrottenloher <andre.schrottenloher@inria.fr>
Sent: Tuesday, 11 June 2019 5:55 PM
To: lightweight-crypto@nist.gov
Cc: lwc-forum@list.nist.gov
Subject: [lwc-forum] OFFICIAL COMMENT: CLAE

Dear all,

It seems that CLAE is vulnerable to practical forgery attacks, due to ciphertext / tag collisions occurring with a high probability:
- AD forgeries: with the same key, nonce and message, collisions of the ciphertext (and tag) between an AD "0" (a single byte) and an empty AD occur with high probability
- with the same key, same message, same AD, collisions of the ciphertext (and tag) between two nonces (of a certain form, with a small difference) occur with high probability

Below are the details and examples.
Attached to this email is a Python implementation of CLAE and some code to generate such collisions, as well as some C code that simply verifies these examples with the implementation provided in the submission.

I would like to thank Dongxi Liu for his earlier response and validation.

================================
1. Same message, different AD