From: Miguel Montes <miguel.montes@gmail.com>
Sent: Sunday, April 28, 2019 9:33 AM
To: lightweight-crypto
Subject: OFFICIAL COMMENT: ORANGE

Dear all:
Rereading my last email, I've noticed that I made a mistake. The byte that is always even is not in the ciphertext, but in the keystream. The last bit of that byte in the ciphertext is always equal to the last bit of the corresponding byte of the plaintext.
Best regards

Miguel Montes

On Sat, Apr 27, 2019 at 5:15 PM Miguel Montes <miguel.montes@gmail.com> wrote:

> Dear all:
> There is a problem with the implementation of the finite field multiplication used in ORANGE-Zest. The first byte of the result is always even.
> Because of this, when the length of the last block of plaintext is greater than 16, byte 16 of the last block of ciphertext is always even.
>
> Also I think the implementation of mult does not comply with the specs. The function mult, as specified, seems to swap both halves of its input. It receives V^b, V^t and returns V^t || α^c . V^b
> As implemented, it simply multiplies it second half of its input.
> Am I misunderstanding the specs?.
> Best regards
> Miguel Montes

From: Bishwajit Chakraborty <bishu.math.ynwa@gmail.com>
Sent: Wednesday, May 01, 2019 12:47 AM
To: Miguel Montes
Cc: lightweight-crypto; lwc-forum@list.nist.gov
Subject: Re: [lwc-forum] OFFICIAL COMMENT: ORANGE
Attachments: errata_ORANGE.pdf; Orange.tar.gz

Dear Miguel,

Thank you very much for pointing out the typos. We have also additionally found few typos in the specification.

Dear all,
The errata for Orange and the revised reference implementation are attached.

Thanks and regards,
ORANGE Team

On Sun, Apr 28, 2019 at 1:45 AM Miguel Montes <miguel.montes@gmail.com> wrote:

> Dear all:
> There is a problem with the implementation of the finite field multiplication used in ORANGE-Zest. The first byte of the result is always even.
> Because of this, when the length of the last block of plaintext is greater than 16, byte 16 of the last block of ciphertext is always even.
>
> Also I think the implementation of mult does not comply with the specs. The function mult, as specified, seems to swap both halves of its input. It receives V^b, V^t and returns V^t || α^c . V^b
> As implemented, it simply multiplies it second half of its input.
> Am I misunderstanding the specs?.
> Best regards
> Miguel Montes
> --
> To unsubscribe from this group, send email to lwc-forum+unsubscribe@list.nist.gov
> Visit this group at https://groups.google.com/a/list.nist.gov/d/forum/lwc-forum
> ---
> You received this message because you are subscribed to the Google Groups "lwc-forum" group.
> To unsubscribe from this group and stop receiving emails from it, send an email to lwc-forum+unsubscribe@list.nist.gov.

--
To unsubscribe from this group, send email to lwc-forum+unsubscribe@list.nist.gov
Visit this group at https://groups.google.com/a/list.nist.gov/d/forum/lwc-forum

Errata of ORANGE

Designers/Submitters:
Bishwajit Chakraborty - Indian Statistical Institute,Kolkata
Mridul Nandi - Indian Statistical Institute, Kolkata, India

bishu.math.ynwa@gmail.com
mridul.nandi@gmail.com

May 1, 2019

In spec_orange.pdf: We make the following corrections on the specification document. No corresponding change is required in the reference implementations.

1. $\mathsf{ORANGE\text{-}Zest}_{[\mathsf{P}]}.\mathsf{enc}$ **line 9**: return value $(C, \mathsf{proc\_tg}(U))$ and not $\mathsf{proc\_tg}(U)$.

2. function ORANGISH **line 21**: There is no α multiplication in Hash. So $Z \leftarrow \mathsf{proc\_hash}(X, (A_{d-1}\| \ldots \|, A_0), 0, 0)$ and not $Z \leftarrow \mathsf{proc\_hash}(X, (A_{d-1}\| \ldots \|, A_0), 1, 1)$.

3. function proc_txt **line 37**: return value is (D', U_d) and not (D', U_a).

4. function $\mathsf{ORANGE\text{-}Zest}_{[\mathsf{P}]}.\mathsf{dec}$:

   (a) **line 5** : $a = 0$ is changed to $a = 0, m \neq 0$.

   (b) **line 9** : $m \neq 0$ is changed to $a \neq 0, m \neq 0$.

5. function mult **line 32** : return value is $\alpha^c \cdot V^b \parallel V^t$ and not $V^t \parallel \alpha^c \cdot V^b$.

In crypto_aead/orangezestv1/ref/orangemodule.h: The primitive polynomial $alpha\_128$ was getting reset to x^{128} instead of $x^{128} + x^7 + x^2 + x + 1$. (in **lines 84-90** of orangemodule.h). This is corrected by using an **else** argument in **line 88** of orangemodule.h.

The revised **Test vectors for ORANGE-Zest** in **Appendix B** can be found bellow :

Test vectors for ORANGE-Zest

Test vector 1:
$Key = 000102030405060708090A0B0C0D0E0F$
$Nonce = 000102030405060708090A0B0C0D0E0F$
$PT =$
$AD = 00010203$
$CT = 84A4C553119EA342C50CCCCE43782567$

Test vector 2:
$Key = 000102030405060708090A0B0C0D0E0F$
$Nonce = 000102030405060708090A0B0C0D0E0F$
$PT =$
$AD =$
$CT = 5A65624E01D1349D2211EFBD52217976$

Test vector 3:
$Key = 000102030405060708090A0B0C0D0E0F$

$Nonce = 000102030405060708090A0B0C0D0E0F$
$PT = 000102030405060708090A0B0C0D0E0F101112131415161718191A$
$AD = 000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D$
$CT = 06C8617CFB5C8CACA64F1F2B9460EADE7776AB0F814F4CFB0E561C621AB9EB080D6CE$
$0D200E80EE74E8C00$

Acknowledgment: We would like to thank Miguel Montes for pointing out the mismatch between reference implementation and pseudocode specification.

From: Bart Mennink <b.mennink@cs.ru.nl>
Sent: Friday, September 6, 2019 3:55 AM
To: lightweight-crypto
Cc: b.mennink@cs.ru.nl; lwc-forum@list.nist.gov; Christoph Dobraunig; Florian Mendel
Subject: [lwc-forum] OFFICIAL COMMENT: ORANGE
Attachments: attack.c

Dear all,

We think that we found a practical forgery for ORANGE. The trick is in the fact that for one-block messages, the input to the second permutation is fully determined by M0, C0, and K, and it is independent of half of the state Y0. In particular, due to how rho works, the bottom part of Y0 gets replaced by 2K+M0b, where M0b is the bottom half of M0.
The other half of the state is known to an attacker who knows the message and can be modified with the ciphertext. Hence, an attacker can change it to a value of its choice.

Please find the implementation of the attack attached.

The authors of ORANGE have also provided a security proof [2]. For this proof, however, the authors consider a slightly modified version of the scheme, where line 12 of the original proposal, "if m\neq0 then (C,U) <- proc_txt(K,U,M,+)", is replaced with "if m\neq0 then (C,U) <- proc_txt(S,U,M,+)", where S comes from the internal state after processing the associated data.

This change precisely pinpoints the oversight of the scheme that we exploit, and our attack seems not to apply to this modified scheme anymore. Nevertheless, our attack demonstrates that the proof of [2] does not convey to the actual LWC submission [1].

Best regards,
Christoph, Florian, and Bart.

[1] Bishwajit Chakraborty and Mridul Nandi. ORANGE. NIST LWC submission (2019).
[2] Bishwajit Chakraborty and Mridul Nandi. Security Proof of ORANGE-Zest (2019).