Updates on COMET

Shay Gueron[1,2], Ashwin Jha[3], Mridul Nandi[3]
{shay.gueron,ashwin.jha1991,mridul.nandi}@gmail.com

[1]University of Haifa, Israel
[2]Amazon Web Services Inc., Seattle, USA
[3]Indian Statistical Institute Kolkata, India

September 18, 2020

1 Security Proof

A detailed security proof for COMET has been recently submitted [2] to NIST Lightweight Cryptography (LwC) Workshop 2020. Here, we briefly discuss the implications of the formal security bounds proved in [2].

Let D denote the total no. of bytes in all queries to the COMET mode of operation, and T denote the number of direct invocations of the underlying block cipher. Traditionally, D is called the data complexity or online query limit and T is called the time complexity or offline query limit.

In [2, Section 7.1], we show that the attack advantage of any adversary $\mathcal{A}$ against COMET-128 and COMET-64 are bounded as follows:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{aead}}_{\mathsf{COMET\text{-}128}}(\mathcal{A}) &\leq \frac{D}{2^{63.75}} + \frac{T}{2^{125.19}} + \frac{DT}{2^{184.24}},\\ \mathsf{Adv}^{\mathsf{aead}}_{\mathsf{COMET\text{-}64}}(\mathcal{A}) &\leq \frac{D}{2^{58.98}} + \frac{T}{2^{121.58}} + \frac{DT}{2^{152.24}} + \frac{D^2T}{2^{193.67}},\end{aligned} \tag{1}$$

Eq. (1) justifies our security claims for COMET-128 (primary version) as given in [1, Table 3], i.e., secure while $D < 2^{64}$ and $T < 2^{119}$. For COMET-64, Eq. (1) justifies security while $D < 2^{40}$ and $T < 2^{112}$. Our data complexity bounds for COMET-64 (secondary version) is slightly less than the claimed complexity ($D < 2^{45}$). However, we note that there is no matching attack, and it is possible that the data complexity can be improved further.

2 Third-party Security Analysis

We briefly discuss two cryptanalysis results on COMET. Although these results do not threaten the security claims of COMET, they help in establishing additional confidence on the security claims of COMET.

Khairallah's Result [3]: Khairallah [3] studied COMET-128 under the weak key model. While the author also presents a multi-user analysis, here we only concentrate on the single key analysis. The main observation of Khairallah's attack is the fact that the key updation function permute only applies to the least significant (LS) 64 bits of the key. So, if the LS 64 bits are all zeroes than the key remains unchanged throughout an encryption query, which can be used to construct forgery and key recover attack. However, we note that the probability that the LS 64 bits are all zeroes is roughly $D/2^{68}$ (D is in bytes), i.e., *the attack requires approx.* 2^{68} *bytes of data which breaches the prescribed data limits by a factor of* 2^4.

Observations of Bernstein, Gilbert and Turan [4]: In a private letter, Bernstein, Gilbert and Turan, proposed two more observations on the security of COMET-64. The first observation builds upon Khairallah's observation, by constructing an encryption query only version of the previous attack. However, *the data complexity of their attack is worse than the previous attack. Specifically, it requires about* 2^{96} *data blocks.*

The second attack is a slide attack that tries to match two separate encryption query states (key and input). *This strategy requires data complexity about* 2^{64} *blocks. Hence, again, their observations do not threaten the security claims of COMET.*

We emphasize that *all of the above given attack strategies are handled in the detailed security proof submitted [2] to the NIST LwC workshop 2020.* In summary, the above given third-party analysis and the security bound given in Eq. (1) helps in improving the confidence on the security claims of COMET.

3 Third-party Implementation Results

Hardware implementation results: In a recent work [5], Rezvani et al. presented hardware implementation and performance results for several second round candidates, including COMET-128_AES-128/128 and COMET-128_CHAM-128/128, on multiple platforms including Artix-7, Spartan-6. In the following, we highlight some of their results:

- COMET-128_AES-128/128 and Ascon-AEAD achieve the *highest throughput* when compared to other implemented ciphers such as SpoC, GIFT-COFB and Sparkle.
- COMET-128_AES-128/128 and Ascon-AEAD are the *most energy efficient* ciphers among the above mentioned candidates, with COMET-128_AES-128/128 performing slightly better than Ascon-AEAD.
- COMET-128_CHAM-128/128, SpoC and GIFT-COFB have the *lowest increase in dynamic power with increasing frequency.*

Apart from standalone implementation, in a recent paper, Rezvani et al. [6] study the simultaneous deployment of COMET-128_AES-128/128 and COMET-128_CHAM-128/128 with GIFT-COFB. They show that on Artix-7 FPGA a simultaneous deployment of the three ciphers uses only 55% of the combined area of separate implementation of each cipher.

Software implementation results: In [7, 8], Weatherley has given an extensive implementation and benchmarking of round 2 candidates, including COMET-128_CHAM-128/128, COMET-64_CHAM-64/128 and COMET-64_SPECK-64/128, on 8-bit and 32-bit microcontrollers. As per the results available in [7, 8], the *COMET implementations outperform all other candidates with a significant margin.*

4 Target Area and Comparison with NIST Standard

Target Area of Application: Based on third-party hardware and software implementations and our design goals, we recommend COMET for both hardware oriented applications like RFID-based applications and IOT sensors, and software oriented applications like low-end microcontroller-based embedded systems. Specific variants of COMET to use depends upon the exact requirements.

Comparison with Current NIST Standard: We compare the performance characteristic of COMET with AES-GCM (the relevant NIST standard).
Hardware: On Artix-7 FPGA, COMET outperforms AES-GCM in the following parameters:

- Area: As per [5], COMET-128_AES-128/128 and COMET-128_CHAM-128/128 require 2753 and 2214 LUTs, while AES-GCM requires 3268 LUTs. In fact, in a recent private communication, Reznavi et al. state that COMET-128_CHAM-128/128's area has been further improved to 1664 LUTs, almost twice as low as AES-GCM.
- Frequency: As per [5], COMET-128_AES-128/128 achieves higher frequency than AES-GCM (251 vs 222 MHz).
- Energy: As per [5, 9], COMET-128_AES-128/128 and COMET-128_CHAM-128/128 consume lesser energy as compared to AES-GCM at comparable frequencies. Specifically, COMET-128_AES-128/128 and COMET-128_CHAM-128/128 consume 0.16 and 0.57 nJ/bit at 40 MHz, whereas AES-GCM consumes 1.15 nJ/bit at 50 MHz.

Software: While we could not find comparable implementation and benchmarking values for COMET and AES-GCM in literature, we expect COMET to perform better than AES-GCM in terms of both speed and memory usage.

5 Proposed Tweak in the Design

We propose two small tweaks in the COMET mode of operation (resulting design is called COMETv2), that reduce memory usage by n bits and avoid the attack strategies given in [3, 4]. These changes are color coded (additions/modifications are colored blue and deletions are colored red) in Algorithm 1-3. Note that, we reuse the notations and definitions exactly as in the specification file [1]. To summarize, we have made following changes:

Algorithm 1 Encryption/Decryption algorithm in COMETv2.

```
function COMETv2-n[|C|].E(K, N, A, M)
    C ← ⊥
    (Y_0, Z_0, a, m, ℓ) ← init(K, N, A, M)
    if a ≠ 0 then
        (Y_a, Z_a) ← proc_ad(Y_0, Z_0, A)
    if m ≠ 0 then
        (Y_ℓ, Z_ℓ, C) ← proc_pt(Y_a, Z_a, M)
    T ← proc_tg(Y_ℓ, Z_ℓ)
    return (C, T)
```

```
function COMETv2-n[|C|].D(K, N, A, C, T)
    M ← ⊥
    is_auth ← 0
    (Y_0, Z_0, a, m, ℓ) ← init(K, N, A, C)
    if a ≠ 0 then
        (Y_a, Z_a) ← proc_ad(Y_0, Z_0, A)
    if m ≠ 0 then
        (Y_ℓ, Z_ℓ, M) ← proc_ct(Y_a, Z_a, C)
    T' ← proc_tg(Y_ℓ, Z_ℓ)
    if T' = T then
        is_auth ← 1
    else
        M ← ⊥
    return (is_auth, M)
```

1. **Change in control bit absorption:** In the current version, the control bit information for the current state requires knowledge of the next input (AD or PT) block. This requires an extra n-bit memory to store the next input block.

   In the proposed tweak, the control bits are XORed at the point where the first/last AD or PT blocks are absorbed (in v1 control bits are XORed one block earlier). This change was done based on our personal communication with William Diehl, one of the authors of [5]. This small change is expected to save the additional n-bit memory (or register) required earlier, as now we do not need the information about next block.

2. **Change in permute Function:** In the current version, the permute function only updates the least significant $\kappa/2 = 64$ bits of the input key. This is done via powering up-based masking in the field $\mathbb{F}_2^{64}$ using the primitive polynomial $x^{64} + x^4 + x^3 + x + 1$.

   In the proposed tweak, the permute function updates the least significant $\kappa - 8 = 120$ bit of the input key. This is done via powering up-based masking over $\mathbb{F}_2^{120}$ (instead of $\mathbb{F}_2^{64}$) using the primitive polynomial $x^{120} + x^9 + x^6 + x^2 + 1$. This small change increases the data complexity of the attack strategy in [3, 4] (see section 2 for attack details), from $D \approx 2^{68}$ to $D \approx 2^{124}$ (in case of COMET-128).

5.1 Security of COMETv2

In [2], we show the following security bounds for COMETv2-128 and COMETv2-64:

$$\begin{aligned} \mathsf{Adv}^{\mathsf{aead}}_{\mathsf{COMETv2\text{-}128}}(\mathcal{A}) &\leq \frac{D}{2^{119.53}} + \frac{T}{2^{125.19}} + \frac{DT}{2^{184.24}}, \\ \mathsf{Adv}^{\mathsf{aead}}_{\mathsf{COMETv2\text{-}64}}(\mathcal{A}) &\leq \frac{D}{2^{67}} + \frac{T}{2^{121.58}} + \frac{DT}{2^{152.24}}, \end{aligned} \tag{2}$$

In other words, COMETv2-128 is secure while $D < 2^{119}$ and $T < 2^{125}$ and $DT < 2^{184}$, and COMET-64 is secure while $D < 2^{67}$ and $T < 2^{121}$ and $DT < 2^{152}$.

References

[1] Gueron, S., Jha A., Nandi M.: COMET: Counter Mode Encryption with Tag. Submission to NIST Lightweight Cryptography Standardization Process (round 2) (2019)

[2] Gueron, S., Jha A., Nandi M.: Revisiting the Security of COMET Authentication Scheme. In submission to NIST Lightweight Cryptography Workshop 2020 (2020)

[3] Khairallah, M.: Weak Keys in the Rekeying Paradigm: Application to COMET and mixFeed. In IACR ToSC **2019(4)**:272–289 (2019)

[4] Bernstein, D.J., Gilbert, H., Turan, M.S.: Observations on COMET-64. Personal communication (2020)

[5] Rezvani, B., Coleman, F., Sachin, S., Diehl, W.: Hardware Implementations of NIST Lightweight Cryptographic Candidates: A First Look. IACR Cryptology ePrint Archive **2019**:824 (2019)

Algorithm 2 Main modules of COMETv2.

function init(K, N, A, M)
  if $n = 64$ **then**
    $(Y_0, Z_0) \leftarrow$ init_state_64(K, N)
  else
    $(Y_0, Z_0) \leftarrow$ init_state_128(K, N)
  $a \leftarrow \lceil |A|/n \rceil$
  $m \leftarrow \lceil |M|/n \rceil$
  $\ell \leftarrow a + m$
  return (Y_0, Z_0, a, m, ℓ)

function round$(Y, Z, I,$ ctrl$, b)$
  $Z' \leftarrow$ get_blk_key(Z)
  $X \leftarrow \mathsf{IC}(Z', Y)$
  $Z \leftarrow Z' \oplus$ ctrl
  if $b = 0$ **then**
    $Y \leftarrow$ update$(X, I, 0)$
    return (Y, Z)
  else
    $(Y, O) \leftarrow$ update(X, I, b)
    return (Y, Z, O)

function proc_ad(Y_0, Z_0, A)
  $(A_{a-1}, \ldots, A_0) \leftarrow$ parse(A)
  ~~$Z_0 \leftarrow Z_0 \oplus 000010^{\kappa-5}$~~ ctrl $\leftarrow 000010^{\kappa-5}$
  for $i = 0$ **to** $a - 2$ **do**
    $(Y_{i+1}, Z_{i+1}) \leftarrow$ round$(Y_i, Z_i, A_i,$ ctrl$, 0)$
    ctrl $\leftarrow 0^{\kappa}$
  if $n \nmid |A_{a-1}|$ **then**
    ~~$Z_{a-1} \leftarrow Z_{a-1} \oplus 000100^{\kappa-5}$~~ ctrl $\leftarrow$ ctrl $\oplus\, 000100^{\kappa-5}$
  $(Y_a, Z_a) \leftarrow$ round$(Y_{a-1}, Z_{a-1}, A_{a-1},$ ctrl$, 0)$
  return (Y_a, Z_a)

function proc_pt(Y_a, Z_a, M)
  $(M_{m-1}, \ldots, M_0) \leftarrow$ parse(M)
  ~~$Z_a \leftarrow Z_a \oplus 001000^{\kappa-5}$~~ ctrl $\leftarrow 001000^{\kappa-5}$
  for $j = 0$ **to** $m - 2$ **do**
    $k \leftarrow a + j$
    $(Y_{k+1}, Z_{k+1}, C_j) \leftarrow$ round$(Y_k, Z_k, M_j,$ ctrl$, 1)$
    ctrl $\leftarrow 0^{\kappa}$
  if $n \nmid |M_{m-1}|$ **then**
    ~~$Z_{\ell-1} \leftarrow Z_{\ell-1} \oplus 010000^{\kappa-5}$~~ ctrl $\leftarrow$ ctrl $\oplus\, 010000^{\kappa-5}$
  $(Y_\ell, Z_\ell, C_{m-1}) \leftarrow$ round$(Y_{\ell-1}, Z_{\ell-1}, M_{m-1},$ ctrl$, 1)$
  $C \leftarrow (C_{m-1}, \ldots, C_0)$
  return (Y_ℓ, Z_ℓ, C)

function proc_ct(Y_a, Z_a, C)
  $(C_{m-1}, \ldots, C_0) \leftarrow$ parse(C)
  ~~$Z_a \leftarrow Z_a \oplus 001000^{\kappa-5}$~~ ctrl $\leftarrow 001000^{\kappa-5}$
  for $j = 0$ **to** $m - 2$ **do**
    $k \leftarrow a + j$
    $(Y_{k+1}, Z_{k+1}, M_j) \leftarrow$ round$(Y_k, Z_k, C_j,$ ctrl$, 2)$
    ctrl $\leftarrow 0^{\kappa}$
  if $n \nmid |C_{m-1}|$ **then**
    ~~$Z_{\ell-1} \leftarrow Z_{\ell-1} \oplus 010000^{\kappa-5}$~~ ctrl $\leftarrow$ ctrl $\oplus\, 010000^{\kappa-5}$
  $(Y_\ell, Z_\ell, M_{m-1}) \leftarrow$ round$(Y_{\ell-1}, Z_{\ell-1}, C_{m-1},$ ctrl$, 2)$
  $M \leftarrow (M_{m-1}, \ldots, M_0)$
  return (Y_ℓ, Z_ℓ, M)

function proc_tg(Y_ℓ, Z_ℓ)
  $Z_\ell \leftarrow Z_\ell \oplus 100000^{\kappa-5}$
  $Z'_\ell \leftarrow$ get_blk_key(Z_ℓ)
  $T \leftarrow \mathsf{IC}(Z'_\ell, Y_\ell)$
  return T

[6] Rezvani, B., Conroy, T., Beckwith, L., Bozzay, M., Laffoon, T., McFeeters, D., Shi, Y., Vu, M., Diehl, W.: Efficient Simultaneous Deployment of Multiple Lightweight Authenticated Ciphers. IACR Cryptology ePrint Archive **2020**:609 (2020)

[7] Weatherley, R.: Lightweight Cryptography Primitives – Performance on 32-bit platforms – Overall group rankings. Online: https://rweather.github.io/lightweight-crypto/performance.html#perf_overall (2020)

[8] Weatherley, R.: Lightweight Cryptography Primitives – Performance on AVR – Overall group rankings. Online: https://rweather.github.io/lightweight-crypto/performance_avr.html#perf_avr_overall (2020)

[9] Abdulgadir, A., Diehl, W., Kaps, J.: An Open-Source Platform for Evaluation of Hardware Implementations of Lightweight Authenticated Ciphers. In proceedings of International Conference on ReConFigurable Computing and FPGAs – ReConFig 2019, pages 1–5 (2019)

Algorithm 3 Various sub-modules of COMETv2. Here, $\lceil x \rceil_r$ and $\lfloor x \rfloor_r$ respectively denote the most significant and least significant r bits of x. α denotes the primitive element of $\mathbb{F}_2^p$.

function chop(I, ℓ)
  if $\ell > n$ **then**
    return $\perp$
  else
    return $i_{\ell-1} \dots i_0$

function parse(I)
  $\ell = \lceil |I|/n \rceil$
  if $\ell = 0$ **then**
    return $\perp$
  else
    $(I_{\ell-1}, \dots, I_0) \xleftarrow{n} I$
    return $(I_{\ell-1}, \dots, I_0)$

function opt_pad0*1(I)
  if $|I| = 0$ **or** $n \nmid |I|$ **then**
    $\xi = n - (|I| \bmod n)$
    $I \leftarrow 0^{\xi-1}1 \| I$
  return I

function permute(Z')
  ~~$p \leftarrow \kappa/2$~~ $p \leftarrow \kappa - 8$
  ~~$(Z'_1, Z'_0) \xleftarrow{p} Z'$~~ $(Z'_1, Z'_0) \leftarrow (\lceil Z' \rceil_{\kappa-p}, \lfloor Z' \rfloor_p)$;
  $Z_0 \leftarrow Z'_0 \quad \alpha$
  $Z \leftarrow (Z'_1, Z_0)$
  return Z

function init_state_128(K, N)
  $Y \leftarrow K$
  $Z \leftarrow$ IC(K, N)
  return (Y, Z)

function init_state_64(K, N)
  $Y \leftarrow$ IC($K, 0$)
  $Z \leftarrow K \oplus 0^{\kappa-r} \| N$
  return (Y, Z)

function shuffle(X')
  $(X'_3, X'_2, X'_1, X'_0) \xleftarrow{n/4} X'$
  $X_2 \leftarrow X'_2 \ggg 1$
  $X \leftarrow (X'_1, X'_0, X_2, X'_3)$
  return X

function get_blk_key(Z')
  $Z \leftarrow$ permute(Z')
  return Z

function update(X, I, b)
  if $b = 0$ **then**
    $Y \leftarrow X \oplus$ opt_pad0*1(I)
    return Y
  else
    $X' \leftarrow$ shuffle(X)
    $O \leftarrow$ chop($X', |I|$) $\oplus I$
    if $b = 1$ **then**
      $Y \leftarrow X \oplus$ opt_pad0*1(I)
    else if $b = 2$ **then**
      $Y \leftarrow X \oplus$ opt_pad0*1(O)
    return (Y, O)