# Tweaks for Oribatida v1.3

Arghya Bhattacharjee[1], Eik List[2],
Cuauhtemoc Mancillas López[3] and Mridul Nandi[1]

[1]Indian Statistical Institute Kolkata, India
bhattacharjeearghya29(at)gmail.com, mridul.nandi(at)gmail.com
[2]Bauhaus-Universität Weimar, Germany
<firstname>.<lastname>(at)uni-weimar.de
[3]Computer Science Department, CINVESTAV-IPN, Mexico
cuauhtemoc.mancillas83(at)gmail.com

**Abstract.** This document describes two small tweaks to update the second-round candidate in the NIST Lightweight competition Oribatida v1.2 to a more secure Oribatida v1.3. The first tweak addresses the observation by Rohit and Sarkar [12] and increases the security of the 192-bit variant; the second tweak updates the constants for domain separation slightly only for the sake of simplicity. We give a brief comparison with other Int-RUP-secure candidates in the competition.

**Keywords:** Authenticated encryption · Permutation · Provable security

## 1 Introduction

**Oribatida** is a permutation-based authenticated encryption scheme a special feedback function that reuses parts of the hidden state to mask parts of the ciphertext for higher nAE security and Int-RUP security. This brief work announces a slight update from Oribatida (v1.2) from [3] to Oribatida v1.3. Prior, we briefly review the encryption and notions necessary to describe our update.

**Encryption with Oribatida.** Let $P, P' \in \mathsf{Perm}(\mathbb{F}_2^n)$ be public permutations. $P'$ is intended to be a round-reduced variant of $P$. As in the classical sponge [2], Oribatida splits the state $S_i = (U_i \,\|\, V_i)$ into a rate $U_i$ of $r$ bits, where inputs are XORed to, and a capacity $V_i$ of $c = n - r$ bits. A nonce $N$ and key $K$ are concatenated at the beginning before the associated data $A$ is processed in $r$-bit blocks $A_i$ and XORed to the rate part in between permutation calls. The final associated-data block $A_a$ is padded with a $10^*$ padding if the associated data is not empty. In between intermediate associated-data blocks, $P'$ is used to slightly boost the performance. At all other locations, $P$ is used as the primitive.
The message $M$ is processed in $r$-bit blocks $M_i$; similarly, the ciphertext is output as $r$-bit blocks $C_i$. The final message block $M_m$ is padded by a $10^*$-padding; the final ciphertext block is truncated to the $|M_m|$ most significant (msb) bits. Unlike the usual sponge, the $s$ least significant bits (lsb) of the capacity of the previous state, $\mathsf{lsb}_s(V_i)$, are used to mask the ciphertext block $C_i$. Thus, Oribatida ensures higher Int-RUP security than other NIST lightweight submissions. We assume that the key size is at most the capacity, $k \leq c$, and the tag size is at most $\tau \leq r$ bits. In the end, the rate is truncated to $\tau$ bits and a tag is returned with the ciphertext. At up to three points, domain values are XORed to the capacity to prevent trivial collisions: $d_N$ when processing nonce and key $(N \,\|\, K)$ at the initialization, $d_A$ when processing the final block of a nonempty associated data, and $d_E$ when processing the final message or ciphertext block. The domains depend on the bit lengths of the unpadded associated data $\ell_A$, and that of the message, $\ell_E$.

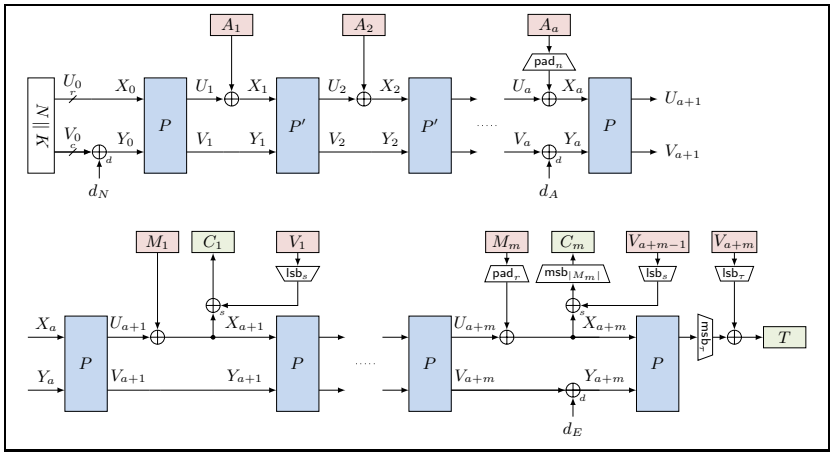
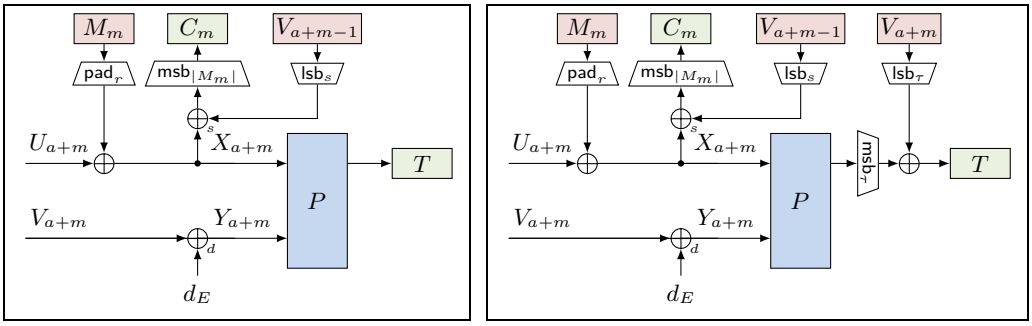



Figure 1: Processing $a$-block associated data $A$ and an $m$-block message $M$ with Oribatida.




(a) Oribatida v1.2 [3]. (b) Oribatida v1.3.

Figure 2: Tag generation of Oribatida.

**Security Notions.** The security notions are the standard nonce-based authenticated encryption (nAE) and integrity under release of unverified plaintext (Int-RUP) [1]. Since the primitive $P$ is an unkeyed permutation, the security is evaluated in the ideal-permutation model. Thus, distinguishers have encryption and decryption oracles to the construction but also to a primitive oracle for $P$ and $P'$. We denote the resources of distinguishers by $q_c$ construction queries of to $\sigma$ blocks in total, and $q_p$ primitive queries. We use $q = q_c + q_p$. The NIST requirements [9] state that a construction should provide nAE security for up to $2^{50}$ blocks encrypted under the same key and $2^{112}$ (offline) operations.

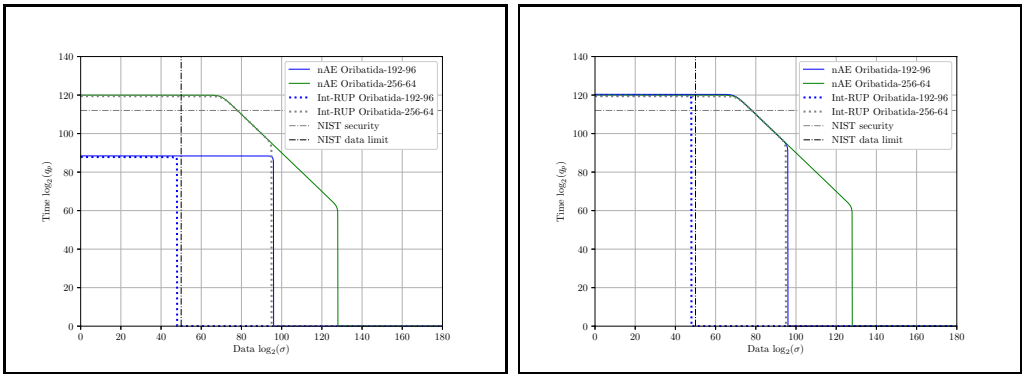
# 2 Updates to Oribatida v1.3

Oribatida v1.3 differs from Oribatida v1.2 in two small aspects:

**Aspect (1): Tag Masking.** Oribatida v1.2 released the tag without masking. As a consequence, Rohit and Sarkar [12] pointed out that an adversary could see the full rate and had to guess only the $(n-\tau)$-bit hidden part to be able to invert the encryption process. To succumb this attack, Oribatida v1.3 masks the tag such that the adversary can see $\tau - s$ bits if $s \leq \tau$, which restores the complexity from $q/2^{n-\tau}$ to $q/2^{c+s}$. Figure 2 illustrates both tag-generation processes for comparison. The masking of the authentication tag is performed exactly as for ciphertext blocks, which streamlines this process.

**Algorithm 1** Partial specification of Oribatida.

131: **function** GetDomainForA($\ell_A, \ell_E$)
132: ~~**if** $\ell_A = 0$ **then return** $\langle 4 \rangle_n$~~
133: **if** $\ell_E > 0 \wedge \ell_A \bmod r = 0$ **then return** $\langle 4 \rangle_n$
134: **if** $\ell_E > 0 \wedge \ell_A \bmod r \neq 0$ **then return** $\langle 6 \rangle_n$
135: **if** $\ell_E = 0 \wedge \ell_A \bmod r = 0$ **then return** $\langle 12 \rangle_n$
136: **if** $\ell_E = 0 \wedge \ell_A \bmod r \neq 0$ **then return** $\langle 14 \rangle_n$




(a) Oribatida v1.2 [3]. (b) Oribatida v1.3.

Figure 3: Security of Oribatida using $q_c = 2^{50}$.

**Aspect (2): Domains.** For domain separation, Oribatida defines $d$-bit constants $d_N$, $d_A$, and $d_E$ that are XORed with the least significant byte of the state at three stages. Domain constants are encoded as 4-bit strings $(t_3, t_2, t_1, t_0)$ that reflect inputs in the hardware API. Oribatida v1.3 updates the definition of the end-of-type (**EOT**) control bit $t_2$, and removes an unnecessary and confusing line from the algorithm of Oribatida v1.2. In the following, $\mathbf{EOT_{old}}$ represents **EOT** in Oribatida v1.2, and $\mathbf{EOT_{new}}$ that in the updated definition of **EOT** in Oribatida v1.3. Algorithm 1 reproduces the relevant part of the specification of Oribatida v1.2 and highlights the removal of line no. 132 to Oribatida v1.3.

- $\mathbf{EOT_{old}}$**:** $t_2$ is the **end-of-type** control bit. This bit is set to 1 iff the current data block is the final block of the same type, i.e., it is the last block of the message/associated data. Note that, if the associated data is empty, the nonce is treated as the final block of the associated data. So, $t_2$ is set to 1. For all other cases, $t_2$ is set to 0.

- $\mathbf{EOT_{new}}$**:** $t_2$ is the **end-of-type** control bit. This bit is set to 1 if the current data block is the final block of the same type, i.e., it is the last block of the nonce/associated data/message. Note that, if both the associated data and the message are empty, the nonce is treated as the final block of the associated data, i.e., $t_2 = 0$ in this case.

**Relevance.** Aspect (1) is crucial from a security point of view. The security effect of the additional tag masking is illustrated in Figure 3 for the maximum number of $q_c = 2^{50}$ construction queries as in the NIST guidelines. One can observe that it salvages the nAE security of the 192-bit version of Oribatida v1.3. Note that the figure cannot illustrate that many primitive (offline) queries to the permutation are in practice much easier to obtain than construction queries. Aspect (2) only simplifies the description.

Table 1: Comparison of Oribatida with further Int-RUP-security claiming submissions to the NIST lightweight competition. $n/t$ = block/tweak length of the primitive, $m$ = #message segments, sec. = security, IF = inverse-free, •/– = feature is present/absent.

| | Sizes (bits) | | | | | | | Security | | Features | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Construction | $\|N\|$ | $\|K\|$ | $\|T\|$ | $n$ | $t$ | State | Rate | nAE | Int-RUP | 1-pass | IF |
| Oribatida-192 (v1.2) [3] | 64 | 128 | 96 | 192 | 0 | 288 | 96 | 89 | 48 | • | • |
| Oribatida-256 (v1.2) [3] | 128 | 128 | 128 | 256 | 0 | 320 | 128 | 121 | 64 | • | • |
| Oribatida-192 (v1.3) **[This work]** | 64 | 128 | 96 | 192 | 0 | 288 | 96 | 121 | 48 | • | • |
| Oribatida-256 (v1.3) **[This work]** | 128 | 128 | 128 | 256 | 0 | 320 | 128 | 121 | 64 | • | • |
| ESTATE [6] | 128 | 128 | 128 | 128 | 4 | 260 | 64 | 64 | 64 | – | • |
| LOCUS-AEAD [5] | 128 | 128 | 64 | 64 | 4 | 324 | 32 | 64 | 64 | • | – |
| LOTUS-AEAD [5] | 128 | 128 | 64 | 64 | 4 | 384 | 32 | 64 | 64 | • | • |

# 3 Comparison with Lightweight Int-RUP-secure Schemes

Among the submissions to the NIST lightweight competition [9], ESTATE [6], LAEM [13], LOTUS-AEAD, and LOCUS-AEAD [5] claimed security in the Int-RUP model. Among these modes, ESTATE, LOTUS-AEAD and LOCUS-AEAD were elected into the second round. This section provides a brief comparison of our proposal to those. The individual properties are summarized in Table 1.

**Brief Description.** ESTATE follows SIV [11]: the associated data and message are authenticated using a variant of CBC-MAC with a tweakable block cipher before the tag is used as the initial vector of CBC-like encryption. The intermediate values are used as keystream and added to the message blocks. LOCUS-AEAD and LOTUS-AEAD employ a variant of PMAC [4] to process the associated data with the tweakable block cipher. For encryption, LOTUS-AEAD uses a variant of OTR [8], a two-round, two-branch Feistel structure to process the message in double blocks. LOCUS-AEAD employs an encryption similar to OCB [10] and EME/EME* [7]. LOCUS-AEAD and LOTUS-AEAD employ a single pass over the message for encryption, but two calls to the primitive per message block. The intermediate values are summed to the associated-data hash and the final message block; the encrypted sum yields the tag.

**Efficiency.** Oribatida processes 96- or 128-bit message blocks per primitive call, whereas the size of the message processed in one primitive call is 64 bits for ESTATE and 32 for LOTUS-AEAD and LOCUS-AEAD. Thus, Oribatida offers higher throughput; moreover, the state size of Oribatida (288 and 320 bits, respectively) is smaller than those of LOTUS-AEAD (388 bits) and LOCUS-AEAD (324 bits). ESTATE has a state size of 260 bits; all three require to process the message with two calls to the primitive. LOCUS-AEAD requires the inverse operation of the underlying block cipher to be available for the decryption. In sum, Oribatida possesses a smaller state size than LOCUS-AEAD and LOTUS-AEAD, and higher nAE security, as well as a higher rate, compared to its Int-RUP-secure competitors.

**Security.** All three competitors are based on tweakable block ciphers, with Int-RUP claims limited to the birthday bound of the internal primitive. ESTATE inherits Int-RUP security until the birthday bound from SIV, which has been considered in [1, Sect. 6.2]. While LOCUS-AEAD and LOTUS-AEAD are similar to OCB and OTR but use intermediate checksums as in EME designs for the tag generation.

# 4 Changelog

For transparency, we provide a summary of the proposed tweaks to a potential Oribatida v1.3 and previous minor updates.

**Changes from Version v1.2 (2019-03-29) to v1.3:**

- **Tag Masking:** The tag was masked with parts of the previous capacity to address the observation [12] for Oribatida-192.
- **Updated Domains:** We updated the domain values for simplicity.

**Changes from Version v1.1 (2019-03-29) to v1.2 (2019-09-27):**

- **Security Goals:** The goals have been clarified further.
- **Security Bounds:** Bounds for nonce-based authenticated encryption and integrity under the release of unverified plaintexts have been added.
- **Through the document:** Fixed typos (often $\oplus_s$ instead of $\oplus$) and reformulated a few sentences for easier readability. Added a short remark on the heuristic for the two-step permutation.

**Changes from Version v1.0 (2019-02-25) to v1.1 (2019-03-29):**

- **Specification:** The figure and the algorithm of the key schedule in SimP have been corrected to match that of Simon.
- **Implementation:** The reference implementation of Oribatida has been corrected to use 26 key-update rounds for SimP-192 and 34 key-update rounds for SimP-256 per step. The previous implementation used two rounds per step less since Simon directly uses the master key as subkeys of the first two rounds.

# References

[1] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to Securely Release Unverified Plaintext in Authenticated Encryption. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT I*, volume 8873 of *LNCS*, pages 105–125. Springer, 2014.

[2] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge functions. In *ECRYPT hash workshop*, volume 2007. Citeseer, 2007.

[3] Arghya Bhattacharjee, Eik List, Cuauhtemoc Mancillas López, and Mridul Nandi. The Oribatida Family of Lightweight Authenticated Encryption Schemes Version v1.2. Technical report, Sep 27 2019. Second-round submission to the NIST Lightweight Cryptography Competition. https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/oribatida-spec-round2.pdf.

[4] John Black and Phillip Rogaway. A Block-Cipher Mode of Operation for Parallelizable Message Authentication. In Lars R. Knudsen, editor, *EUROCRYPT*, volume 2332 of *LNCS*, pages 384–397. Springer, 2002.

[5] Avik Chakraborti, Nilanjan Datta, Ashwin Jha, Cuauhtemoc Mancillas Lopez, Mridul Nandi, and Yu Sasaki. LOTUS-AEAD and LOCUS-AEAD. Technical report, Feb 26 2019. First-round submission to the NIST Lightweight Cryptography Competition.

[6] Avik Chakraborti, Ashwin Jha, Cuauhtemoc Mancillas Lopez, Mridul Nandi, and Yu Sasaki. ESTATE. Technical report, Mar 29 2019. First-round submission to the NIST Lightweight Cryptography Competition.

[7] Shai Halevi. EME*: Extending EME to Handle Arbitrary-Length Messages with Associated Data. In Anne Canteaut and Kapalee Viswanathan, editors, *INDOCRYPT*, volume 3348 of *LNCS*, pages 315–327. Springer, 2004.

[8] Kazuhiko Minematsu. Parallelizable Rate-1 Authenticated Encryption from Pseudorandom Functions. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT*, volume 8441 of *LNCS*, pages 275–292. Springer, 2014. Full version at https://eprint.iacr.org/2013/628.pdf.

[9] NIST. Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process. https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/final-lwc-submission-requirements-august2018.pdf, August 27 2018.

[10] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: a block-cipher mode of operation for efficient authenticated encryption. In Michael K. Reiter and Pierangela Samarati, editors, *ACM-CCS*, pages 196–205. ACM, 2001.

[11] Phillip Rogaway and Thomas Shrimpton. A Provable-Security Treatment of the Key-Wrap Problem. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *LNCS*, pages 373–390. Springer, 2006.

[12] Raghvendra Rohit and Sumanta Sarkar. [lwc-forum] ROUND 2 OFFICIAL COMMENT: Oribatida. NIST lwc forum mailing list, 17 September 17:09 2019.

[13] Han Sui, Wenling Wu, Lei Zhang, and Danxia Zhang. LAEM (Lightweight Authentication Encryption Mode). Technical report, Mar 25 2019. First-round submission to the NIST Lightweight Cryptography Competition.