The Oribatida Family of Lightweight Authenticated Encryption Schemes

Version v1.2
Submission to the NIST Lightweight Competition
Version Sep 27, 2019

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

[1]Indian Statistical Institute Kolkata
203, B.T Road, Kolkata 700108, India
bhattacharjeearghya29(at)gmail.com
mridul.nandi(at)gmail.com

[2]Faculty of Media
Bauhaus-Universität Weimar
Bauhausstr. 11, D-99423 Weimar, Germany
<firstname>.<lastname>(at)uni-weimar.de

[3]Computer Science Department, CINVESTAV-IPN
Av. IPN No. 2508 Col. San Pedro Zacatenco
Mexico, D.F. 07360, MEXICO
cuauhtemoc.mancillas83(at)gmail.com

Contents

1 Introduction

This work proposes the Oribatida family of permutation-based authenticated-encryption schemes.

Oribatida Is Lightweight. As a keyed permutation-based mode of operation, Oribatida has to store neither the state of the permutation (plus a small overhead) nor subkeys or tweaks. Authentication and encryption can be performed fully online and impose no need on buffering the input. For instantiations, we employ a lightweight family of permutations SimP that is very close to the block-cipher variants Simon-96-96 or Simon-128-128 and their respective key schedules. As a result, the instances of SimP possess a low state size of only 192 and 256 bits, respectively.

Oribatida Is Performant. For a security level of close to 128 bits, Oribatida provides a rate of 1/2 (two calls to a permutation for processing n-bit message material) for authentication and encryption even with a small permutation size of only $n = 192$ or $n = 256$ bits due to the way it masks ciphertext blocks.

Oribatida Alleviates Its Usage Across Different Platforms. Our proposed instantiations of Oribatida with the SimP family of permutations can be implemented with only the basic operations AND, rotations, and XOR. Since SimP avoids the use of S-boxes, implementations can split the state flexibly according to the target platforms' needs.

Oribatida Is Based on Well-known Components. The design of Oribatida is based on the well-known duplex mode. Therefore, it founds on well-established results. Since SimP is very close to the design of Simon, it can profit from the existing cryptanalysis, and rely on its already well-understood permutation design.

Oribatida Is Secure. The design of Oribatida inherits the minimal security guarantees of the duplex mode. Moreover, Oribatida augments the usual sponge by a ciphertext masking that boosts the security. While this specification omits tedious proof details, all members of the Oribatida family are expected to provide 128-bit security for encryption and integrity.

Oribatida Is Robust under Release of Unverified Plaintexts. While authenticated encryption can be realized in an online manner, proper authenticated decryption must be offline. However, resource-constrained devices can hardly buffer long messages until the authentication tag is verified, which can lead to a complete loss of privacy and integrity. The ciphertext masking of Oribatida limits the security damage in such cases. In the case of accidental misuse, Oribatida provides integrity also in the case that plaintext material leaks from invalid ciphertexts.

2 Notations

General Notations. We use uppercase letters (e.g., X, Y) for functions and variables, lowercase letters (e.g., x, y) for indices and lengths, as well as calligraphic uppercase letters (e.g., $\mathcal{X}, \mathcal{Y}$) for sets and spaces. We write $\mathbb{F}_2$ for the field of characteristic 2 and $\mathbb{F}_2^n = \{0,1\}^n$ for the set of vectors over $\mathbb{F}_2$, i.e., strings of n bits. $|X|$ denotes the number of bits of X. Given $X \in \mathbb{F}_2^n$, we write $X[i]$ for the i-th (least significant) bit of X, and define the bit order by $X = (X[n-1] \,\|\, \ldots \,\|\, X[1] \,\|\, X[0])$. We write $\emptyset$ for the empty set and ε for the empty string.

We denote by $X[x..y]$ the range of $X[x], \ldots, X[y]$ for non-zero integers x and y. Given binary strings X and Y, we denote their concatenation by $X \,\|\, Y$ and their bitwise XOR by $X \oplus Y$ when $|X| = |Y|$. For positive integers x and y and bit strings of different lengths $X \in \mathbb{F}_2^x$ and $Y \in \mathbb{F}_2^y$ with $x \geq y$, we define $X \oplus_y Y =^{\text{def}} X \oplus (0^{x-y} \,\|\, Y)$.
We write $X \twoheadleftarrow \mathcal{X}$ to indicate that X is chosen uniformly at random and independent from other variables from a set $\mathcal{X}$. We consider $\mathsf{Func}(\mathcal{X}, \mathcal{Y})$ to be the set of all mappings $F : \mathcal{X} \to \mathcal{Y}$, and $\mathsf{Perm}(\mathcal{X})$ to be the set of all permutations over $\mathcal{X}$. Given an event E, we denote the probability of E by $\Pr[E]$. We denote the invalid symbol by $\perp$. Moreover, we denote by $(n)_k =^{\text{def}} \prod_{i=0}^{k-1}(n-i)$ the falling factorial.
For $X \in \mathbb{F}_2^*$, we denote by $(X_1, X_2, \ldots, X_x) \xleftarrow{n} X$ the splitting of X into n-bit strings X_1, $\ldots$, X_{x-1}, and $|X_x| \leq n$, in form of $X_1 \,\|\, \ldots \,\|\, X_x = X$. Moreover, for $Y \in \mathbb{F}_x$, we write $(X_1, X_2, \ldots, X_m) \xleftarrow{x_1, x_2, \ldots, x_m} Y$ to denote the splitting of Y into $X_1 = Y[x-1..x-x_1]$, $X_2 = Y[x-x_1-1..x-x_1-x_2]$, $\ldots$, $X_m = Y[x_m-1..0]$, where $x = x_1 + x_2 + \ldots + x_m$ holds. For a given set $\mathcal{X}$ and some non-negative integer x, we write $\mathcal{X}^{\leq x}$ for the union set $\cup_{i=0}^{x} \mathcal{X}^i$. Given a non-negative integer $x < 2^n$, we write $\langle x \rangle_n$ for its conversion into an n-bit binary string with the most significant bit left, e.g., $\langle 135 \rangle_8 = (10000111)$. We omit n if it is clear from the context.

Nonce-based Authenticated Encryption. Let $\mathcal{K}$ be a set of keys, $\mathcal{N}$ be a set of nonces, $\mathcal{A}$ a set of associated data, $\mathcal{M}$ a set of messages, $\mathcal{C}$ a set of ciphertexts, and $\mathcal{T}$ a set of authentication tags. A nonce $N \in \mathcal{N}$ is an input that must be unique for each authenticated encryption query.
A nonce-based authenticated encryption scheme with associated data $\Pi = (\mathcal{E}, \mathcal{D})$ is a tuple of deterministic encryption algorithm $\mathcal{E} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{C} \times \mathcal{T}$ and deterministic decryption algorithm $\mathcal{D} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{C} \times \mathcal{T} \to \mathcal{M} \times \{\perp\}$ with associated key space $\mathcal{K}$. The encryption algorithm $\mathcal{E}$ takes a tuple (K, N, A, M) and outputs (C, T), where C is a ciphertext and T an authentication tag. We assume that $|C| = |M|$ holds for all inputs (K, N, A, M) and their corresponding ciphertexts. The associated data is authenticated, but not encrypted. The decryption function $\mathcal{D}$ takes a tuple (K, N, A, C, T) and outputs either the unique plaintext M for which $\mathcal{E}_K(N, A, M) = (C, T)$ holds, or outputs $\perp$ if the input is invalid. We introduce $\mathcal{E}_K^{N,A}(M)$ as short form of $\mathcal{E}_K(N, A, M)$ and $\mathcal{D}_K^{N,A}(C, T)$ for $\mathcal{D}_K(N, A, C, T)$, respectively.
We assume that authenticated encryption schemes are both correct and tidy. Correct means that for all $(K, N, A, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M}$, it holds that $\mathcal{D}_K^{N,A}(\mathcal{E}_K^{N,A}(M)) = M$. Tidy means that for all $(K, N, A, C, T) \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{C} \times \mathcal{T}$, it holds that $\mathcal{E}_K^{N,A}(\mathcal{D}_K^{N,A}(C, T)) = (C, T)$ iff $\mathcal{D}_K^{N,A}(C, T) \neq \perp$.

Standard Notions. The ideal AE scheme provides two oracles $\$: \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{C} \times \mathcal{T}$ and $\perp : \mathcal{N} \times \mathcal{A} \times \mathcal{C} \times \mathcal{T} \to \mathcal{M} \times \{\perp\}$ that offer access to encryption and verification. Note that we overload the $\perp$ notation to mean the oracle and the symbol of an invalid decryption. Given a tuple (N, A, M), the ideal encryption oracle outputs ciphertext-tag tuples (C, T) that are random bits of the expected length, i.e., computes $(C', T') = \mathcal{E}_K^{N,A}(M)$ and samples $C \twoheadleftarrow \{0,1\}^{|C'|}$ and $T \twoheadleftarrow \{0,1\}^{\tau}$. The ideal decryption oracle ensures correctness. That is, given an input (N, A, C, T) where (C, T) had been the output to a previous encryption query (N, A, M), the decryption oracle outputs the corresponding message M. Otherwise, the decryption always returns the invalid symbol $\perp$ for every new decryption query that had not been the answer of an earlier encryption query.

Ideal-permutation Model. Since this work studies schemes based on public permutations, we employ the usual security notions in the ideal-permutation model. So, the adversary always has an additional oracle $\pi^{\pm}$ that provides access to the public permuta-

tion π in forward and backward direction. We write $\Pi[\pi]$ and $\mathcal{E}[\pi]$, $\mathcal{D}[\pi]$, etc. to indicate that an authenticated-encryption scheme Π and its algorithms are based on a primitive $\pi \twoheadleftarrow \mathsf{Perm}(\mathcal{B})$, where $\mathcal{B} = \{0,1\}^n$ is some block space.

Definition 1 (nAE Security)**.** Let $K \twoheadleftarrow \mathcal{K}$, $\pi \twoheadleftarrow \mathsf{Perm}(\mathcal{B})$, and let $\Pi[\pi] = (\mathcal{E}[\pi]_K, \mathcal{D}[\pi]_K)$ be a nonce-based authenticated scheme. Let $\mathbf{A}$ be a nonce-respecting adversary. Then, $\mathbf{Adv}^{\mathsf{nAE}}_{\Pi[\pi]}(\mathbf{A}) \stackrel{\text{def}}{=} \Delta_{\mathbf{A}}(\mathcal{E}[\pi]_K, \mathcal{D}[\pi]_K, \pi^{\pm}; \$, \bot, \pi^{\pm})$.

Notions under Release of Unverified Plaintext Material. In the RUP model by Andreeva et al. [ABL+14], the understanding of a nonce-based AE scheme differs slightly from the previous definition. To formulate the forgery goal, the oracles are adapted. A verification oracle outputs 1 iff the input is valid, and 0 otherwise. A nonce-based RUP authenticated encryption scheme $\widetilde{\Pi} = (\widetilde{\mathcal{E}}_K, \widetilde{\mathcal{D}}_K, \widetilde{\mathcal{V}}_K)$ is a 3-tuple of encryption algorithm $\widetilde{\mathcal{E}} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{C} \times \mathcal{T}$, decryption algorithm $\widetilde{\mathcal{D}} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{C} \times \mathcal{T} \to \mathcal{M}$, and verification algorithm $\widetilde{\mathcal{V}}_K : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{C} \times \mathcal{T} \to \{0,1\}$. The signature of the encryption and decryption algorithms are unchanged, but the decryption oracle always outputs the resulting would-be plaintext. We consider the security of integrity in this model, dubbed INT-RUP by [ABL+14].

Definition 2 (INT-RUP Security)**.** Let $K \twoheadleftarrow \mathcal{K}$, $\pi \twoheadleftarrow \mathsf{Perm}(\mathcal{B})$, and let $\widetilde{\Pi}[\pi] = (\widetilde{\mathcal{E}}[\pi]_K, \widetilde{\mathcal{D}}[\pi]_K, \widetilde{\mathcal{V}}[\pi]_K)$ be a nonce-based RUP authenticated scheme. Let $\mathbf{A}$ be a nonce-respecting adversary. Then

$$\mathbf{Adv}^{\mathsf{INT\text{-}RUP}}_{\Pi[\pi]}(\mathbf{A}) \stackrel{\text{def}}{=} \underset{\mathbf{A}}{\Delta}(\widetilde{\mathcal{E}}[\pi]_K, \widetilde{\mathcal{D}}[\pi]_K, \widetilde{\mathcal{V}}[\pi]_K, \pi^{\pm}; \widetilde{\mathcal{E}}[\pi]_K, \widetilde{\mathcal{D}}[\pi]_K, \bot, \pi^{\pm}).$$

3 Specification of Oribatida

This section defines the Oribatida authenticated-encryption scheme.

General Definitions. Let n denote the state size, k the key size, r the rate, c the capacity, s the mask size, ν the nonce size, d a domain size, and τ a tag size in bits, all of which are non-negative integers. We define:

- The key space $\mathcal{K} = \mathbb{F}_2^k$, with $k \leq n$.
- The state space $\mathcal{S} = \mathbb{F}_2^n$.
- We denote by r the rate and by c the capacity of the Oribatida mode, where $r+c = n$ bits. We define a block space $\mathcal{B} = \mathbb{F}_2^r$.
- The nonce space $\mathcal{N} = \mathbb{F}_2^\nu$, with $\nu \leq r$. Oribatida requires $\nu + k = n$.
- A finite set of domains $\mathcal{DO} = \mathbb{F}_2^d$ for $d = 4$ bits.
- We define positive integers $a_{\max}$ and $m_{\max}$ for the maximal length in bits of associated data message inputs, respectively.
- The associated-data space $\mathcal{A} = \mathbb{F}_2^{\leq a_{\max}}$.
- Message and ciphertext spaces $\mathcal{M} = \mathcal{C} = \mathbb{F}_2^{\leq m_{\max}}$.
- Moreover, we define the space of authentication tags $\mathcal{T} = \mathbb{F}_2^\tau$ with $\tau \leq r$.

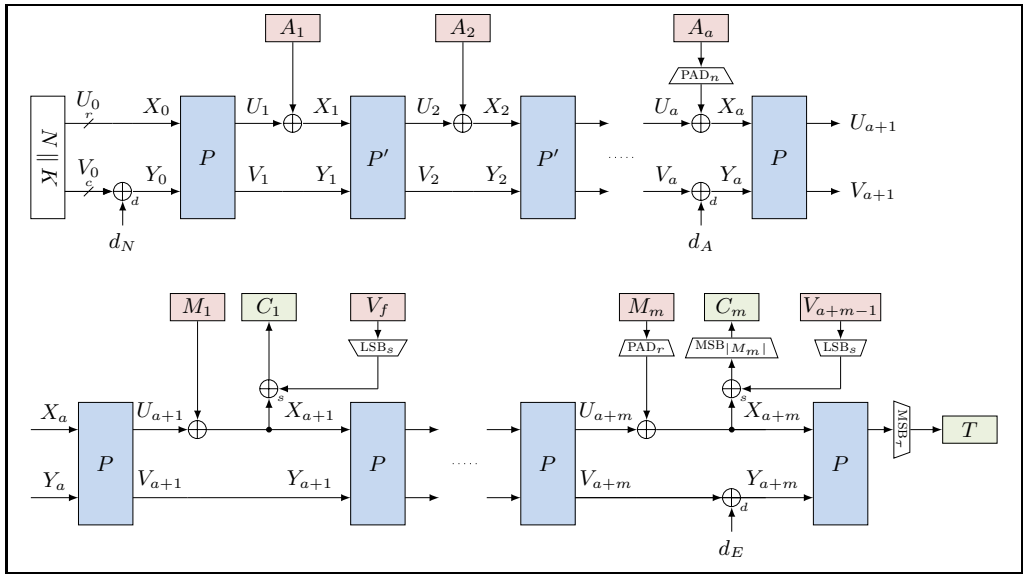

Figure 1: Authentication of an a-block associated data A and the encryption of an m-block plaintext M with Oribatida, for $a, m > 1$. P and P' are permutations, K the secret key, N the nonce, C the resulting ciphertext, and T the resulting authentication tag.

We define $s \leq c$ for the mask size in bits. We write two permutations $P, P' \in \mathsf{Perm}(\mathcal{S})$. We denote the state after the i-th call to the permutations by $S_i = (U_i \,\|\, V_i)$, and the state after XORing the subsequent associated-data block A_i or message block M_{i-a} to it by $(X_i \,\|\, Y_i)$, where a denotes the number of associated-data blocks after padding. We say that A is integral if its length is a multiple of r bits, and say that it is partial otherwise. Similarly, we say that M (or C) is integral if its length is a multiple of r bits, and call it partial otherwise.

The Core Idea. Oribatida is a variant of the monkey-wrap design [BDPVA12], as used before, e.g., in Ascon [DEMS16] or NORX [AJN14]. Oribatida extends previous designs by a ciphertext-block masking that increases the resilience against release of unverified plaintext material. We denote by (U_i, V_i) the outputs of and by (X_i, Y_i) the inputs to the permutation. As in the classical sponge, Oribatida considers the state $S_i = (U_i \,\|\, V_i)$ as a rate part U_i of r bits, where inputs are XORed to, and a capacity part V_i of $c = n - r$ bits. Unlike the usual sponge, an s-bit part of the capacity is used to mask the subsequent ciphertext block. The definition is given in Algorithm 1. In the following, explanations and details are presented.

3.1 Proposed Parameter Sets

Oribatida-n-s is proposed in two versions, parametrized by the state size of the permutation n, and a mask size s. Table 1 lists the proposed parameter sets. We define a security parameter $z = c + s$ that should be defined as 192 (the target for NIST security requirements). We briefly recall the parameters and the conditions satisfied by these parameters.

1. We always choose a key size of $k = 128$ bits.
2. n denotes the size of the permutation in bits, which is either 256 or 192.
3. The nonce length ν is chosen such that $\nu + k = n$ holds.

Table 1: Recommended schemes of Oribatida in the order of recommendation. The topmost is our primary recommendation. All integer values are given in bits. The state size is given by $r + c$. Rec. = Recommendation.

							State size		
		Permutations		Key	Nonce	Tag	Rate	Capacity	Mask
Rec.	Name	P	P'	(k)	(ν)	(τ)	(r)	(c)	(s)
1	Oribatida-256-64	SimP-256-4	SimP-256-2	128	128	128	128	128	64
2	Oribatida-192-96	SimP-192-4	SimP-192-2	128	64	96	96	96	96

4. The capacity of the permutation is chosen as $c = 192 - s$ bits (by the security requirement mentioned above).
5. Thus, the mask size s is at most the capacity: $s \leq c$.
6. Finally, the tag length is set to $\tau = n - c$ bits.

Oribatida-n-s employs two internal permutations $P, P' \in \mathsf{Perm}(\mathbb{F}_2^n)$, where P' is chosen as a round-reduced variant of P to process the associated data more efficiently. The following members of the Oribatida-n-s family are proposed, based on instantiations from the SimP family of permutations:

- Our **primary recommendation** is Oribatida-256-64. For P, this variant uses SimP-256-4 with $r_s = 34$ rounds per step and $\theta = 4$ steps. Moreover, for P', it employs SimP-256-2 with $r_s = 34$ rounds per step and $\theta = 2$ steps.
- Our **secondary recommendation** is Oribatida-192-96. For P, this variant uses SimP-192-4 with $r_s = 26$ and $\theta = 4$ steps. For P', it employs SimP-192-2 with $r_s = 26$ and $\theta = 2$ steps.

3.2 Limitations

The encryption of Oribatida produces a ciphertext of the same length as the plaintext, and a τ-bit authentication tag. Its decryption will – if the given tuple of key, nonce, associated data, ciphertext, and tag is valid – produce a plaintext of the same length as the ciphertext. The nonce must be unique for each encryption query, even if the message is empty. There is no secret message number for Oribatida.
At most $2^{50} - 1$ bytes, over all the summed lengths of all nonces, associated data after padding, and messages after padding of all queries, are allowed to be processed under the same secret key before the key must be changed. At most $2^{50} - 1$ bytes, over the summed length of nonce, associated data after padding, and message after padding, are allowed in a single query. In each encryption or verification query, the associated data can be empty or present; in each encryption query, the message can be empty or present; in each decryption query, the ciphertext can be empty or present. Oribatida demands that no information about would-be plaintexts is released to the outside if a decryption query is deemed invalid. The security guarantees in the INT-RUP model are a safety measure in temporarily exposed or exceptionally resource-constrained settings, but **must not** be interpreted as a recommendation to release unverified plaintexts.

Algorithm 1 Specification of Oribatida.

```
101: function E_K^{N,A}(M)
102:   ℓ_A ← |A|
103:   ℓ_E ← |M|
104:   d_N ← GetDomainForN(ℓ_A, ℓ_E)
105:   d_A ← GetDomainForA(ℓ_A, ℓ_E)
106:   d_E ← GetDomainForE(ℓ_E)
107:   A ← pad_r(A)
108:   M ← pad_r(M)
109:   (S_1, V_f) ← Init(K, N, d_N, ℓ_A)
110:   S_{a+1} ← ProcessAD(S_1, A, d_A)
111:   (C, T) ← Encrypt(S_{a+1}, M, V_f, d_E, ℓ_E)
112:   return (C, T)

121: function GetDomainForN(ℓ_A, ℓ_E)
122:   if ℓ_A = 0 ∧ ℓ_E = 0 then return ⟨9⟩_n
123:   return ⟨5⟩_n

131: function GetDomainForA(ℓ_A, ℓ_E)
132:   if ℓ_A = 0 then return ⟨4⟩_n
133:   if ℓ_E > 0 ∧ ℓ_A mod r = 0 then return ⟨4⟩_n
134:   if ℓ_E > 0 ∧ ℓ_A mod r ≠ 0 then return ⟨6⟩_n
135:   if ℓ_E = 0 ∧ ℓ_A mod r = 0 then return ⟨12⟩_n
136:   if ℓ_E = 0 ∧ ℓ_A mod r ≠ 0 then return ⟨14⟩_n

141: function GetDomainForE(ℓ_E)
142:   if ℓ_E = 0 then return ⟨0⟩_n
143:   if ℓ_E mod r = 0 then return ⟨13⟩_n
144:   if ℓ_E mod r ≠ 0 then return ⟨15⟩_n

151: function pad_x(X)
152:   if |X| mod x = 0 then return X
153:   return X ‖ 1 ‖ 0^{x−(|X| mod x)−1}

161: function Init(K, N, d_N, ℓ_A)
162:   V_0 ← lsb_s(N ‖ K)
163:   S_1 ← P((N ‖ K) ⊕_d d_N)
164:   V_1 ← lsb_s(S_1)
165:   if ℓ_A = 0 then return (S_1, V_0)
166:   if ℓ_A ≠ 0 then return (S_1, V_1)

171: function ProcessAD(S_1, A, d_A)
172:   (A_1, ··· , A_a) ←r A
173:   for i = 1..a − 1 do
174:     S_{i+1} ← P'(S_i ⊕ (A_i ‖ 0^c))
175:   S_{a+1} ← P(S_a ⊕ (A_a ‖ 0^c) ⊕_d d_A)
176:   return S_{a+1}

181: function lsb_x(X)
182:   if |X| ≤ x then return X
183:   return X[(|X| − x − 1)..0]

191: function msb_x(X)
192:   if |X| ≤ x then return X
193:   return X[(|X| − 1)..(|X| − x)]

201: function D_K^{N,A}(C, T)
202:   ℓ_A ← |A|
203:   ℓ_E ← |C|
204:   d_N ← GetDomainForN(ℓ_A, ℓ_E)
205:   d_A ← GetDomainForA(ℓ_A, ℓ_E)
206:   d_E ← GetDomainForE(ℓ_E)
207:   A ← pad_r(A)
208:   C ← pad_r(C)
209:   (S_1, V_f) ← Init(K, N, d_N, ℓ_A)
210:   S_{a+1} ← ProcessAD(S_1, A, d_A)
211:   (M, T') ← Decrypt(S_{a+1}, C, V_f, d_E, ℓ_E)
212:   if T = T' then return M
213:   else return ⊥

221: function Encrypt(S_{a+1}, M, V_f, d_E, ℓ_E)
222:   x ← ℓ_E mod r
223:   (M_1, ··· , M_m) ←r M
224:   V ← V_f
225:   for i = 1..m do
226:     (U_{a+i}, V_{a+i}) ←r,c S_{a+i}
227:     X_{a+i} ← M_i ⊕ U_{a+i}
228:     C_i ← X_{a+i} ⊕_s lsb_s(V)
229:     Y_{a+i} ← V_{a+i}
230:     if i = m then
231:       Y_{a+i} ← Y_{a+i} ⊕_d d_E
232:       C_m ← msb_x(C_m)
233:     V ← V_{a+i}
234:     S_{a+i+1} ← P(X_{a+i} ‖ Y_{a+i})
235:   C ← (C_1 ‖ C_2 ‖ ··· ‖ C_m)
236:   T ← msb_τ(S_{a+m+1})
237:   return (C, T)

241: function Decrypt(S_{a+1}, C, V_f, d_E, ℓ_E)
242:   x ← ℓ_E mod r
243:   if ℓ_E = 0 then
244:     T' ← msb_τ(S_{a+1})
245:     return (ε, T')
246:   (C_1, ··· , C_m) ←r C
247:   V ← V_f
248:   for i = 1..m do
249:     (U_{a+i}, V_{a+i}) ←r,c S_{a+i}
250:     X_{a+i} ← C_i ⊕_s lsb_s(V)
251:     Y_{a+i} ← V_{a+i}
252:     M_i ← U_{a+i} ⊕ X_{a+i}
253:     if i = m then
254:       Y_{a+i} ← Y_{a+i} ⊕_d d_E
255:       M_m ← msb_x(M_m)
256:     V ← V_{a+i}
257:     S_{a+i+1} ← P(X_{a+i} ‖ Y_{a+i})
258:   M ← (M_1 ‖ M_2 ‖ ··· ‖ M_m)
259:   T' ← msb_τ(S_{a+m+1})
260:   return (M, T')
```

3.3 Workflow of Oribatida

Initialization. Each variant of Oribatida uses a fixed-size nonce N whose length ν is chosen such that $k + \nu = n$ bits. N is concatenated with the key K to initialize the state: $N \,\|\, K$: $(U_0, V_0) \leftarrow (N \,\|\, K) \oplus_d d_N$. The domain d_N is XORed to the d least significant bits of the initial state. Then, the first state value S_1 results from a call to the permutation: $(U_1 \,\|\, V_1) \leftarrow P(U_0 \,\|\, V_0)$. Note that we store the value of V_0 or V_1 (aliased by V_f), depending on whether the associated data is empty or not, to mask the first block of ciphertext later.

Processing Associated Data. After the initialization, the associated data A is padded with a 10^*-padding if $|A| \bmod r \neq 0$ such that its length becomes the next highest multiple

of r bits. Thereupon, the padded associated data A is split into r-bit blocks $(A_1, \cdots, A_a)$. Given the state $(U_i, V_i) \xleftarrow{r,c} S_i$, A_i is XORed to the rate part of the state: $X_i \leftarrow U_i \oplus A_i$, for $1 \leq i < a$. For all non-final blocks of A, the capacity part of the permutation output, V_i, is simply forwarded to the capacity part of the subsequent input to the permutation P': $Y_i \leftarrow V_i$. The next state is computed by a call to the reduced permutation P' afterwards, for all indices $1 < i < a$ but the final a-th block of A: $S_i \leftarrow P'(X_i \,\|\, Y_i)$. When the final block A_a is processed, a domain d_A that depends on the lengths of A and M is XORed to the least significant byte of the capacity.

Encryption. After the associated data has been processed, the message M is encrypted. If the length of M is not a multiple of r bits, M is padded with a 10^*-padding such that its length after padding becomes the next highest multiple of r bits. Thereupon, M is split into r-bit blocks $(M_1, \cdots, M_m)$ after padding.
The blocks M_i are processed one after the other. Given the state value $(U_{a+i}, V_{a+i}) \xleftarrow{r,c} S_{a+i}$, the current block M_i is XORed to the rate part U_{a+i}: $X_{a+i} \leftarrow M_i \oplus U_{a+i}$. The capacity part is simply forwarded: $Y_{a+i} \leftarrow V_{a+i}$. Then, $(X_{a+i} \,\|\, Y_{a+i})$ is used as input to a call to P to derive the next state value $S_{a+i+1} \leftarrow P(X_{a+i} \,\|\, Y_{a+i})$.
The ciphertext blocks C_i are computed from a sum of the current rate, the current plaintext block, and a (partial) earlier value from the capacity. The first ciphertext block is computed from

$$C_1 \leftarrow X_{a+i} \oplus_s \textsc{lsb}_s(V_f).$$

If it is the final block, then, C_1 is computed from

$$C_1 \leftarrow \textsc{msb}_{\ell_E}(X_{a+i} \oplus_s \textsc{lsb}_s(V_f)),$$

where ℓ_E denotes the length of M before padding.
The further non-final ciphertext blocks C_i, $1 < i < m$ are computed from $C_i \leftarrow X_{a+i} \oplus_s \textsc{lsb}_s(V_{a+i-1})$, for $1 < i < m$. If $m > 1$, the final ciphertext block C_m is computed from

$$C_m \leftarrow \textsc{msb}_{\ell_E \bmod r}(X_{a+m} \oplus_s \textsc{lsb}_s(V_{a+m-1})).$$

For the final message block, a domain d_E is XORed to the least significant byte of the capacity: $Y_{a+m} \leftarrow V_{a+m} \oplus_d d_E$. Thereupon, P is called another time to derive $S_{a+m+1} \leftarrow P(X_{a+m} \,\|\, Y_{a+m})$. Its rate part – truncated to τ bits if necessary – is released as the authentication tag: $T \leftarrow \textsc{msb}_\tau(S_{a+m+1})$.

Decryption. The decryption algorithm takes a tuple (K, N, A, C, T). Again, the initialization with K and N as well as the processing of the associated data A is performed in the same manner as for encryption. If $|C| \bmod r \neq 0$, the decryption pads C with a 10^*-padding to the next multiple of r bits. In all cases, it splits C into r-bit blocks $(C_1, \cdots, C_{m-1})$ plus a final block C_m. If $m > 1$, the plaintext block is computed as

$$\begin{aligned} X_{a+i} &\leftarrow C_i \oplus_s \textsc{lsb}_s(V) \\ M_i &\leftarrow (U_{a+i} \oplus X_{a+i}), \end{aligned}$$

where $V = V_f$ for $i = 1$ and $V = V_{a+i-1}$ otherwise. The capacity is again simply forwarded to the next call of the permutation: $Y_{a+i} \leftarrow V_{a+i}$. The subsequent state is then computed by $(U_{a+i+1} \,\|\, V_{a+i+1}) = S_{a+i+1} \leftarrow P(X_{a+i} \,\|\, Y_{a+i})$.
For the final block m, the final plaintext block is computed from the padded ciphertext block C_m as

$$X_{a+m} \leftarrow C_m \oplus_s \textsc{lsb}_s(V)$$

$$M_m \leftarrow \text{LSB}_x(U_{a+m} \oplus X_{a+m}),$$

where $x =^{\text{def}} \ell_E \bmod r$. For the final block, the domain d_E is XORed to the least significant byte of the capacity: $Y_{a+m} \leftarrow V_{a+m} \oplus_d d_E$. The would-be tag T' is derived by computing $(T' \,\|\, Z) \leftarrow P(X_{a+m} \,\|\, Y_{a+m})$, and using only its most significant τ bits: $T' \leftarrow \text{MSB}_\tau(T' \,\|\, Z)$ as for the encryption If $T = T'$, the ciphertext is considered valid, and $M = (M_1 \,\|\, \cdots \,\|\, M_m)$ is released as plaintext. Otherwise, the ciphertext is deemed invalid, and $\perp$ is returned.

Domain Separation. For the purpose of domain separation, Oribatida defines a set of domain constants d_N, d_A and d_E. Note that $d = 4$ bits suffice in practice. The domains are XORed with the least significant byte of the state at three stages. Domains are encoded as bit strings, e.g., $\langle 12 \rangle_d = (1100)_2$. The value depends on the presence of A and M and whether their final blocks are absent, partial, or integral. This ensures that there exist no trivial collisions of inputs to P among blocks of A and M.
The constants are determined by the four control bits (t_3, t_2, t_1, t_0) that reflect inputs in the hardware API, similar to, e.g., [CDNY18]. The rationale behind them is the following:

- **EOI:** t_3 is the **end-of-input** control bit. This bit is set to 1 iff the current data block is the final block of the input. For all other cases, t_3 is set to 0.

- **EOT:** 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.

- **Partial:** t_1 is the **partial-control** bit. It is set to 1 if the current data block is partial, i.e. if its size is less than the required block size. For all other data blocks, t_1 is 0.

- **Type:** t_0 is called the **type-control** bit. It identifies the type of the current data block. For the nonce and the processing of the final message block, t_0 is set to 1. For all other cases, t_0 is set to 0.

While processing a data block, the domain values are set as the integer representation of $t_3 \,\|\, t_2 \,\|\, t_1 \,\|\, t_0$. For example, if we are processing the nonce (which is always a complete r-bit block), where the associated data is empty, and the message is not empty, it holds that $d_N = (t_3 t_2 t_1 t_0) = (0101)_2 = 5$.

4 Specification of The SimP Family of Permutations

This section specifies the permutation SimP. From a high-level view, SimP is a variant of the domain extender Ψ_r by Coron et al. [CDMS10]. We define SimP to use a round-reduced variant of the Simon [BSS+13] block cipher and its key schedule through four such steps. We briefly recall Ψ_r before we describe the details of Simon, provide an overview of existing cryptanalysis and close with a discussion of the implications on SimP.

4.1 The Ψ_r Domain Extender

The Ψ_r family is a two-branch Feistel-like network that consists of r calls to (pairwise independent) block ciphers. An illustration of Ψ_4 is given at the top of Figure 2. Let $\mathsf{BlockCipher}(\mathcal{K}, \mathcal{B})$ denote the set of all block ciphers with key space $\mathcal{K}$ and block space $\mathcal{B}$. For Ψ_r, $\pi_1, \pi_2, \ldots, \pi_r \in \mathsf{BlockCipher}(\mathbb{F}_2^n \times \mathbb{F}_2^n, \mathbb{F}_2^n)$ are independent block ciphers which use

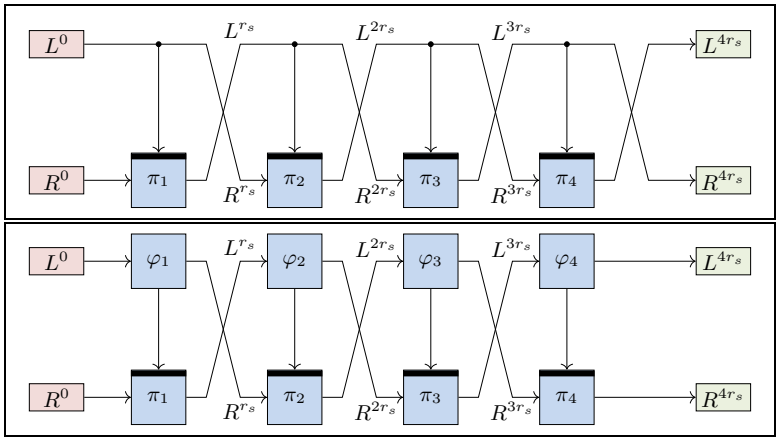

Figure 2: Top: The construction Ψ_4 [CDMS10]. The blocks π_i denote block ciphers over $\mathbb{F}_2^n$ with key space $\mathbb{F}_2^n$. **Bottom:** High-level view of the construction Φ_4 as a variant of Ψ_4. The blocks φ_i represent the key schedules that produce the subkeys and which are externalized from the block ciphers π_i in Φ_4. φ_i feeds the subkeys to π_i and outputs the final subkey K^{r_s} to become the next value R^{ir_s}.

one branch R^i as state input, and the other one, L^i, as secret key. Coron et al. provide statements on the indifferentiability of their constructions.

Theorem 1 ([CDMS10])**.** Let $\pi_1, \pi_2, \pi_3 \twoheadleftarrow \mathsf{BlockCipher}(\mathbb{F}_2^n)$ be pairwise independent permutations over $\mathbb{F}_2^n$. The three-step construction Ψ_3 with an ideal block cipher is (t_D, t_S, q, ϵ)-indifferentiable from an ideal cipher with $t_S = O(qn)$ and $\epsilon = 5q^2/2^n$.

Intuitively, it follows that a four-step construction with a fourth independent permutation $\pi_4 \twoheadleftarrow \mathsf{BlockCipher}(\mathcal{K}, \mathbb{F}_2^n)$ inherits at least the security of the three-step construction.

Definition 3 (Indifferentiability [MRH04])**.** Let C be a Turing machine with oracle access to either $(\Pi, \{\pi_1^{\pm}, \ldots, \pi_m^{\pm}\})$, where C is a construction and the π_i's are ideal primitives. C can employ any of the primitives internally. C is said to be (t_D, t_S, q, ϵ)-indifferentiable from an tuple $(\mathcal{P}, \{\pi_1^{\pm}, \ldots, \pi_m^{\pm}\})$, where $\mathcal{P}$ is an ideal primitive, if there exists a simulator S with oracle access to $\mathcal{P}$ that runs in time at most t_S, such that for any distinguisher $\mathbf{A}$ that runs in time at most t_D and makes at most q queries, it holds that

$$\left|\Pr\left[\mathbf{A}^{C,\pi_1,\ldots,\pi_m} = 1\right] - \Pr\left[\mathbf{A}^{\mathcal{P},\pi_1,\ldots,\pi_m} = 1\right]\right| < \epsilon.$$

C is said to be indifferentiable from $\mathcal{P}$ if ϵ is a negligible function of the security parameters for polynomially bounded q, t_D and t_S.

4.2 Φ_r: A Variant of Ψ_r That Includes The Key Schedule

The Ψ_r construction has to store the state that is transformed through the block cipher π_i's state transformation, plus the key of the current step. Internally, the block ciphers π_i also have to expand the secret key to subkeys that add to the total memory requirement. We propose a variant that avoids the need to store the current secret key input. For this purpose, we define the key-schedule permutation $\varphi_i : \mathbb{F}_2^n \to \mathbb{F}_2^n$ that takes an initial key K as input and outputs the subkeys $K^0, \ldots, K^{r_s}$ for fixed number of rounds r_s of π_i. An illustration is given at the bottom of Figure 2. Hereafter, we call the construction Φ_r when it consists of r steps in total. Note that Φ_r omits the final swap of the halves for simplicity and since it does not affect the security.

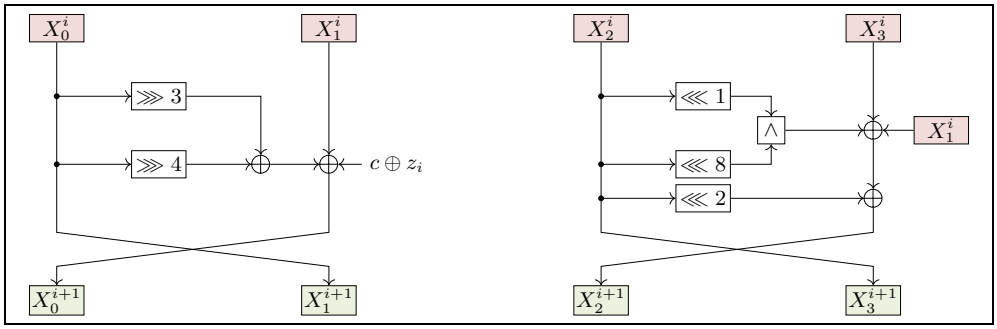

Figure 3: One iteration of the round function of SimP, which is equivalent to the key-update function (left) and the state-update function (right) of Simon-$2w/2w$, where w is the word size.

4.3 Simon

The Simon family of block ciphers [BSS+13] belongs to the lightest block ciphers in terms of hardware area and energy efficiency. Its round function consists of only an XOR, three bit-wise rotations, and a bit-wise AND, which renders it particularly lightweight and flexible. Moreover, Simon has been analyzed intensively since its proposal; among others, e.g., [ALLW14, CW16, LLW17a, Rad15, XZBL16] studied the security of Simon-96-96 and Simon-128-128. Considerably more works targeted the smaller-state variants of Simon, which has recently been standardized as part of ISO/IEC 29167-21:2018 [ISO18]. For concreteness, Simon-96-96 uses a word size $w = 48$ bits and employs 52 rounds, whereas Simon-128-128 uses $w = 64$ bits and 68 rounds.

4.4 The SimP-n-θ Family of Permutations

SimP is an instantiation of Φ_4 that tries to adhere to the standard as close as possible, SimP-192 employs the round-reduced Simon-96-96 as π and its key schedule as φ. To form a 256-bit permutation, SimP-256 uses Simon-128-128 with its key schedule. One iteration of the round function of Simon-$2w$-$2w$ and its key-update function side by side, as is used in SimP-n, is illustrated in Figure 3. Internally, the state of SimP-n-θ consists of four w-bit words $(X_0^i, X_1^i, X_2^i, X_3^i)$, where the superscript index i indicates the state after Round i. We denote by r_s the number of rounds per step, and index the steps from 1 to θ, and the rounds from 1 to $\theta \cdot r_s$. The plaintext is denoted as $(X_0^0, X_1^0, X_2^0, X_3^0)$; the ciphertext is given as $(X_0^{\theta r_s}, X_1^{\theta r_s}, X_2^{\theta r_s}, X_3^{\theta r_s})$.
After Round r_s, the state halves $(X_0^{r_s}, X_1^{r_s})$ and $(X_2^{r_s}, X_3^{r_s})$ are swapped; similarly, they are swapped also after Round $2r_s, \ldots, \theta r_s$. One round of the permutation is illustrated in Figure 3. Thus, SimP-192-θ uses Simon-96-96 and consists of four 48-bit words. SimP-256-θ employs the round function and the key-update function of Simon-128-128 as a 256-bit permutation. For SimP-256-θ, the state consists of four 64-bit words.

Round Function. Let w be a positive integer for the word size. for SimP-192, $w = 48$ bits; for SimP-256, $w = 64$ bits. Let $f : \mathbb{F}_{2^w} \to \mathbb{F}_{2^w}$ and $g : \mathbb{F}_{2^w} \to \mathbb{F}_{2^w}$ be defined as

$$f(x) \stackrel{\text{def}}{=} (x \lll 8) \wedge ((x \lll 1) \oplus (x \lll 2)) \text{ and}$$
$$g(x) \stackrel{\text{def}}{=} (x \ggg 3) \oplus (x \ggg 4)\,.$$

Key-update Function. Let $\varphi_j : (\mathbb{F}_{2^w})^2 \to (\mathbb{F}_{2^w})^2$, for $1 \le j \le \theta$ be key-update functions. Let $\ell = (j-1) \cdot r_s$. On input (X_0^ℓ, X_1^ℓ), it derives r_s keys $(X_0^{\ell+i}, X_1^{\ell+i})$, for $1 \le i \le r_s$, as

$$X_0^{\ell+i} \leftarrow X_1^{\ell+i-1} \oplus g(X_0^{\ell+i-1}) \oplus c \oplus z_{\ell+i-1} \qquad \text{and} \qquad X_1^{\ell+i} \leftarrow X_0^{\ell+i-1},$$

for $1 \le i \le r_s$. Note that $c = \texttt{0xff...ffc}$ is a w-bit constant.

State-update Function. We define the state-update function as $\pi : (\mathbb{F}_{2^w})^{r_s} \times (\mathbb{F}_{2^w})^2 \to (\mathbb{F}_{2^w})^2$, where the first input considers the expanded subkeys. Let $\ell = (j-1) \cdot r_s$. It takes r_s round keys $(X_0^\ell, \ldots, X_{r_s-1}^\ell)$ as key input, as well as (X_2^ℓ, X_3^ℓ) as state input, and computes $(X_2^{\ell+r_s}, X_3^{\ell+r_s})$ recursively as:

$$X_2^{\ell+i} \leftarrow f(X_2^{\ell+i-1}) \oplus X_3^{\ell+i-1} \oplus X_1^{\ell+i-1} \qquad \text{and} \qquad X_3^{\ell+i} \leftarrow X_2^{\ell+i-1},$$

for $1 \le i \le r_s$.

Step Function. Let $\rho_j : \mathbb{F}_{2^w}^4 \to \mathbb{F}_{2^w}^4$ denote the step function, for $1 \le j \le \theta$. Define $L^i = (X_0^i, X_1^i)$ and $R^i = (X_2^i, X_3^i)$. The step transforms $(L^i, R^i) = (X_0^i, X_1^i, X_2^i, X_3^i)$ into $(X_0^{i+r_s}, X_1^{i+r_s}, X_2^{i+r_s}, X_3^{i+r_s})$ as

$$(L^{r_s}, R^{r_s}) = (X_0^{i+r_s}, X_1^{i+r_s}, X_2^{i+r_s}, X_3^{i+r_s})$$
$$\rho_j(X_0^i, X_1^i, X_2^i, X_3^i) \stackrel{\text{def}}{=} (\pi_j(X_2^i, X_3^i), \varphi_j(X_0^i, X_1^i)),$$

for $1 \le j < \theta$. One exception is the final step ρ_θ, which omits the final swap of the halves:

$$\rho_\theta(X_0^i, X_1^i, X_2^i, X_3^i) \stackrel{\text{def}}{=} (\varphi_\theta(X_0^i, X_1^i), \pi_\theta(X_2^i, X_3^i)).$$

SimP-n-θ takes a plaintext $(X_0^0, X_1^0, X_2^0, X_3^0)$ and outputs $(L^r, R^r) = (X_0^r, X_1^r, X_2^r, X_3^r)$, with $r = \theta r_s$ as ciphertext.

Round Constants. The round constants are those of Simon-96-96 and Simon-128-128 [BSS+13], respectively. It holds that $c = \texttt{0xff...ffc}$, i.e., all w bits except for the least significant two bits are 1. More precisely, for $w = 48$, it holds that

$$c = (1111\,1111\,1111\,1111\,1111\,1111\,1111\,1111\,1111\,1111\,1111\,1100)_2.$$

For $w = 64$, it holds that

$$c = (1111\,1111\,1111\,1111\,1111\,1111\,1111\,1111\,1111\,1111\,1111\,1111\,1111\,1111\,1111\,1100)_2.$$

For both SimP-192 and SimP-256, the constants $z = z_0 z_1 \ldots z_{61}$ are defined as

$$z = (10\,1011\,1101\,1100\,0000\,1101\,0010\,0110\,0010\,1000\,0100\,0111\,1110\,0101\,1011\,0011)_2.$$

The sequence has a period of 62, so $z_i = z_{i \bmod 62}$, for non-negative integers i. Note that the order of the bits z_i is reversed.

Number of Steps θ. We consider only the choices of $\theta \in \{2, 4\}$. The case $\theta = 2$ is used only to process the intermediate associate data blocks. In all other cases, Oribatida uses $\theta = 4$. Figure 4 shows the step-reduced variant SimP-n-2.

Number of Rounds. SimP-192-4 consists of $r_s = 26$ rounds for each step, and therefore performs $r = 4 \cdot r_s = 104$ rounds in total. SimP-256-4 consists of $r_s = 34$ rounds for each block, and therefore performs $r = 4 \cdot r_s = 136$ rounds in total.
Similarly, SimP-192-2 consists of $r_s = 26$ rounds for each step, and performs $r = 2 \cdot r_s = 52$ rounds in total. SimP-256-2 consists of $r_s = 34$ rounds for each block, and performs $r = 2 \cdot r_s = 68$ rounds in total. For simplicity, we also denote SimP-n-4 as SimP-n and SimP-n-2 as SimP′-n. The algorithm for SimP-n-θ is given in Algorithm 2.

Table 2: Parameters of SimP.

Variant	Word size (w)	#Steps (θ)	#Rounds/Step (r_s)
SimP-192-2	48	2	26
SimP-192-4	48	4	26
SimP-256-2	64	2	34
SimP-256-4	64	4	34

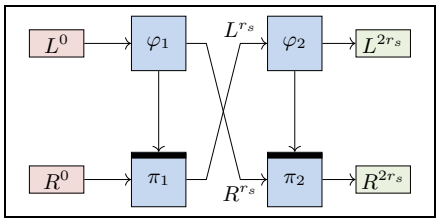

Figure 4: SimP-n-2.

Algorithm 2 Specification of the encryption and decryption algorithms of SimP-n-θ.

```
101: function SimP-n-θ(M)
102:   (X_0^0, X_1^0, X_2^0, X_3^0) ←^w M
103:   for i ← 1..θ do
104:     for j ← 1..r_s do
105:       ℓ ← (i − 1) · r_s + j
106:       X_0^ℓ ← X_1^{ℓ−1} ⊕ g(X_0^{ℓ−1}) ⊕ c ⊕ z_{ℓ−1}
107:       X_1^ℓ ← X_0^{ℓ−1}
108:       X_2^ℓ ← X_3^{ℓ−1} ⊕ f(X_2^{ℓ−1}) ⊕ X_1^{ℓ−1}
109:       X_3^ℓ ← X_2^{ℓ−1}
110:     if i ≠ θ then
111:       (X_0^ℓ, X_1^ℓ, X_2^ℓ, X_3^ℓ) ← SWAP(X_0^ℓ, X_1^ℓ, X_2^ℓ, X_3^ℓ)
112:   C ← (X_0^{θr_s} ‖ X_1^{θr_s} ‖ X_2^{θr_s} ‖ X_3^{θr_s})
113:   return C

121: function f(X)
122:   return ((X ⋘ 1) ∧ (X ⋘ 8))
123:           ⊕(X ⋘ 2)

131: function g(X)
132:   return (X ⋙ 3) ⊕ (X ⋙ 4)

141: function SWAP(X_0, X_1, X_2, X_3)
142:   return (X_2, X_3, X_0, X_1)
```

The Byte Order in Oribatida. For the sake of clarity, Figure 5 visualizes the byte and word order of the inputs. Let `SB` denote the state S in bytes; for more clarity, we further write this ordering in type-writer font. The rate consists of the first $r/8$ bytes of the state: `SB[0], ..., SB[r/8 - 1]`. The capacity represents the last $c/8$ bytes `SB[r/8], ..., SB[n/8 - 1]`. Similarly, the rate part of the state consists of the first words of the permutation input. If the state is interpreted as an n-bit value, the initial Byte 0 contains the most significant eight bits: `SB[0]` $= (S[n-1], S[n-2], \ldots, S[n-8])$. On the other side, the least significant eight bits are stored in Byte `SB[n/8 - 1]`: `SB[n/8 - 1]` $= (S[7], S[6], \ldots, S[0])$.
So, the rate part is used first as input to the key-update function; the capacity is used as input to the state-update function.

Remark 1. Instantiating a scheme proven in an idealized model such as indifferentiability with a symmetric-key primitive is almost always a heuristic: there simply exist few provably secure instantiations. Using the full Simon-$2w$-$2w$ for each step would be an option for a more secure, but considerably less performant scheme. Concerning SimP, our approach follows the prove-then-prune strategy from AEZ [HKR15]. However, after replacing each step by at least half of the number of rounds, and always using four steps, our approach is far less aggressive than it, as outlined above and seems to provide a sufficient security margin.

5 Security Arguments for Oribatida

This section provides arguments on the provable security of our mode. First, we briefly recall the necessary notions for nonce-based authenticated encryption. Thereupon, we provide an outline of its security, but omit proof details.

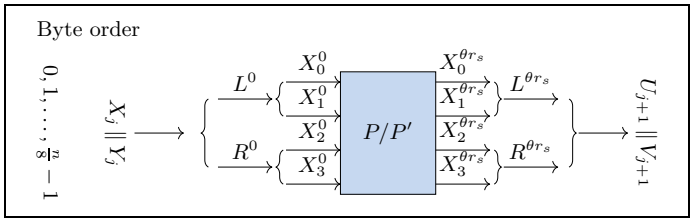

Figure 5: Byte and word orientation of inputs into and outputs from SimP as used in Oribatida.

Assume, we consider an information-theoretic nonce-respecting distinguisher $\mathbf{A}$ that has access to a construction oracle that is either Oribatida or random bits. Moreover, $\mathbf{A}$ has access to independent random permutations $P, P' \leftarsquigarrow \mathsf{Perm}(\mathbb{F}_2^n)$ in both worlds.

- As usual, we bound the number of primitive queries that $\mathbf{A}$ asks to the construction oracle by q_p.
- We further denote the number of encryption and verification queries by q_e and q_d, respectively.
- We use σ for the total number of r-bit blocks in associated data, plaintexts, and ciphertexts over all encryption and verification queries to Oribatida, respectively. We distinguish between σ_e and σ_d for the r-bit blocks used in the encryption and decryption queries to Oribatida, respectively.

The advantage of the nAE security of the full-stated keyed duplex, adapted to the nAE setting from [MRV15, Theorem 2], is dominated by

$$O\left(\frac{(q_c\ell)^2}{2^n} + \frac{(q_c\ell)^2}{2^c} + \frac{q_p}{2^c} + \frac{q_p}{2^k} + \frac{q_c + q_p}{2^\tau}\right), \tag{1}$$

which addresses the probabilities of collisions between internal states of the full state, between parts chosen, guessing the capacity with primitive queries, as well as the probability to find a valid verification query or the secret key.

Theorem 2 (nAE Security of Oribatida)**.** Let $\mathbf{A}$ be a nonce-respecting adversary w.r.t. $\Pi[\pi]_K$. Then

$$\mathbf{Adv}^{\mathsf{nAE}}_{\Pi[\pi]_K}(\mathbf{A}) \leq \frac{\binom{\sigma}{r} + 2\binom{q_p}{r}}{2^{r(r-1)}} + \frac{\sigma^2}{2^n} + \frac{r(q_d + \sigma_d) + 2\sigma_e q_p + q_p q_c + q_d(\sigma_e + q_p)}{2^{c+s}} + \frac{3q_p}{2^k} + \frac{2rq_p}{2^{n-\tau}} + \frac{q_d}{2^\tau}.$$

For integrity under release of unverified plaintexts, we have the following bound.

Theorem 3 (INT-RUP Security of Oribatida)**.** Let $\mathbf{A}$ be a nonce-respecting adversary w.r.t. $\Pi[\pi]_K$. Then

$$\mathbf{Adv}^{\mathsf{INT\text{-}RUP}}_{\Pi[\pi]_K}(\mathbf{A}) \leq \frac{\sigma_e^2}{2^n} + \frac{4\sigma_e\sigma_d + 4\sigma q_p + q_c q_p + q_p + r(\sigma_d + q_d)}{2^{c+s}} + \frac{q_d^2 + \binom{q_d+q_v}{2}}{2^c} + \frac{\binom{q_e}{r}}{2^{\tau(r-1)}} + \frac{3rq_p}{2^{n-\tau}} + \frac{3q_p}{2^k} + \frac{2\binom{q_p}{r}}{2^{r(r-1)}} + \frac{2q_v}{2^\tau}.$$

The proofs of Theorem 2 and 3 are deferred to the publication of Oribatida and are therefore not part of this submission document.

Table 3: Security claims for our recommended schemes. RUP = release of unverified plaintext material.

	nAE Security		INT-RUP Security	
Construction	Time (q_p)	Data (bytes)	Time (q_p)	Data (bytes)
Oribatida-256-64	2^{121}	2^{50}	2^{121}	2^{50}
Oribatida-192-96	2^{89}	2^{50}	2^{89}	2^{50}

Choice of Rate and Capacity Inputs to The Permutation. The omission of the final swap does not affect the security properties in the context of the permutation, but is an optimization. Note that the reduced permutation SimP-2 – that employs only two steps for processing the associated data in Oribatida – is no longer indistinguishable from a random permutation. It is easy to see that if the key input would be held constant and the state input would change, the state would remain a permutation, where the zero difference cannot occur. Since we choose the rate part as the key input, this distinguisher is not directly exploitable. In contrast, the step-reduced permutation that is used for processing the associated data needs only the differential probability of P' for security.

Security Level. The security level is expressed in bits. A security level of z bits means that in the single-key setting, the advantage of any adversary to distinguish it from the ideal primitive or to recover the key is negligible as long as its number of queries q and its total number of queried r-bit blocks σ over all messages satisfy $q, \sigma \ll O(2^z)$. Assume that P and P' are independent permutations. Table 3 illustrates the maximal advantages of a nonce-respecting adversary against the nAE or INT-RUP security, respectively, of our proposals with the given maximal number of primitive queries q_p and data σ in queries to the constructions. Note that the term of 89 bits of INT-RUP security for Oribatida-192-96 are due to a dominating term of $rq_p/2^{n-\tau}$ in our INT-RUP bound.

Remark 2 (Higher Security with A Simple Tweaks). Note that for our secondary proposal, Oribatida-192-96, higher nAE security of about 128 bits, and higher INT-RUP security of about 121 bits are easily possible when using tags of only $\tau = 64$ bits, which matches the NIST requirement. In general, Oribatida could be strengthened in a better way: by masking the tag output by the key, i.e. $T \leftarrow \text{MSB}_\tau(U_{a+m+1}) \oplus \text{MSB}_\tau(K)$. This would yield 121-bit nAE security for our secondary proposal while retaining the tag size of $\tau = 96$ bits.

6 Security of SimP

The number of steps and rounds of SimP was chosen to resist known cryptanalysis techniques. This section provides a rationale of our choices from the existing works.

6.1 Requirements

Oribatida with an random permutation aims at nAE security of $O(r\sigma_d/2^{c+s})$ and INT-RUP $O(q_d^2/2^c)$ in the ideal-permutation model. The advantage of those bounds should be much higher than the complexity to recover or predict the key. An instantiation of P must be free of distinguishing properties that allow to distinguish it from a random permutation with non-negligible advantage and considerably less than 2^n queries. This strengthens the adversary compared to the use of P in Oribatida. There, it can inject nonce, associated data, or message blocks only into the rate part and can observe ciphertext and tag outputs also only from that part, but masked. Concretely, we require from

P the absence of (truncated, higher-order) differential characteristics with probability $\geq 2^{-n}$, linear approximations with squared correlation $\geq 2^{-n}$, or component functions of degree $< n$ in SimP-4. Moreover, we require the absence of impossible-differential, zero-correlation, or integral distinguishers in SimP-4. However, we disregard rebound or other forms of inside-out attacks that are inapplicable in Oribatida, or splice-and-cut attacks when using SimP as a compression function.

6.2 Existing Cryptanalysis on Simon

Various works analyzed the Simon family of block ciphers since its proposal.

Differential Cryptanalysis. Cryptanalysis that appeared early after the proposal of Simon followed mainly heuristics for differential cryptanalysis: Abed et al. [ALLW14] followed a heuristic branch-and-bound approach that yielded differentials for up to 30 rounds of Simon-96. Biryukov et al. [BRV14] studied more efficient heuristics, but considered the small variants with state sizes up to 64 bits. Dinur et al. [DDGS15] showed that distinguishers on Simon with k key words can be extended by at least k rounds. Interestingly, boomerangs seemed to be less a threat to Simon-like ciphers than pure differentials.
Kölbl et al. [KLT15] redirected the research focus to the search for optimal characteristics. More recently, Liu et al. [LLW17a] employed a variant of Matsui's algorithm [Mat94] to find optimal differential characteristics. They found that characteristics with probability higher than 2^{-96} covered at most 27 rounds. Moreover, they found at best 31-round differentials with accumulated probability higher than 2^{-96}, i.e., of probability $2^{-95.34}$. For Simon-128, they showed that optimal differential characteristics covered at most 37 rounds and found 41-round differentials with cumulative probability of $2^{-123.74}$.

Linear Cryptanalysis is similarly effective for Simon-like ciphers as its differential counterpart. Alizadeh et al. [ABG+13, AAA+14] reported multi-trail linear distinguishers on all variants of Simon. For Simon-96-96, they proposed a distinguisher on up to 31 rounds that could be extended by two rounds in a key-recovery attack. Similarly, they reported a 37-round distinguisher for Simon-128-128 that could be extendable by two rounds. Chen and Wang [CW16] proposed improved key-recovery attacks with the help of dynamic key guessing. To the best of our knowledge, their attacks are the most effective ones for our considered variants in terms of the number of covered rounds, with up to 37 rounds of Simon-96-96 and up to 49 rounds of Simon-128-128 in theory.
Similar as for differentials, Liu et al. studied also optimal linear approximations [LLW17b]. They found that the optimal linear approximations can reach at most 28 rounds for Simon-96, and at most 37 rounds for Simon-128. Moreover, they determined linear hulls with potential of $2^{-93.8}$ for 31 rounds of Simon-96, and $2^{-123.15}$ for 41 rounds of Simon-128.

Integral, Impossible-differential, and Zero-correlation Distinguishers. Integral attacks cover at most 22 rounds for Simon-96-96 and 26 rounds of Simon-128-128. Initially, Zhang et al. [ZWW15] found integral distinguishers on up to 21 and 25 rounds for Simon-96 and Simon-128. Their results were extended by one round each by Xiang et al. [XZBL16], and later by Todo and Morii [TM16]. The latter could show the absence of integrals for 25-round Simon-96, which was confirmed by Kondo et al. [KSTI18].
The maximal number of rounds that impossible-differential and zero-correlation distinguishers can cover is given by at most twice the length of the maximal diffusion. From the results by Kölbl et al. [KLT15], full diffusion is achieved by 11 rounds for Simon-96 and 13 rounds for Simon-128-128. So, impossible-differential and zero-correlation distinguishers can cover at most 22 and 26 rounds in the single-key setting.

Table 4: Existing results of best distinguishers and best key-recovery attacks on Simon-96 in the single-key setting. – = not given; Pr. = probability; Pot. = linear potential; Deg. = degree.

Type	#Rounds	Time	Data	Pr./Pot./Deg.	Ref
Simon-96-96 Distinguishers					
Algebraic	14	–	20 CPs	–	[Rad15]
Integral	22	2^{95}	2^{95} CP	95	[XZBL16]
Differential	30	–	–	92.2	[ALLW14]
Differential	31	–	–	95.34	[LLW17a]
Linear	31	–	–	93.8	[LLW17b]
Simon-96-96 Key-recovery Attacks					
Multiple Linear	33	$2^{94.42}$	$2^{94.42}$ KP	94.42	[AAA+14]
Linear Hull	37	$2^{88.0}$	$2^{95.2}$ KP	95.2	[CW16]
Simon-128-128 Distinguishers					
Algebraic	16	–	20 CP	–	[Rad15]
Integral	26	2^{126}	2^{126} CP	126	[XZBL16]
Linear	37	–	–	128	[AAA+14]
Differential	41	–	–	123.74	[LLW17a]
Linear Hull	41	–	–	123.15	[LLW17b]
Simon-128-128 Key-recovery Attack					
Linear Hull	49	$2^{127.6}$	$2^{127.6}$ CP	126.6	[CW16]

Related-key Distinguishers. Kondo et al. [KSTI18] searched for iterative key differences in Simon. This allowed them to extend previous results by four to 15 rounds. For Simon-96-96, the authors found iterative key differentials for up to 20 rounds. It remains unclear if this yields an impossible differential; in the best case, a key-iterated 20-round distinguisher could be extended by $2+2+2$ wrapping rounds: two more blank rounds where one of the key words is not used, plus two rounds where the key difference can be canceled by the state differences, plus two outermost rounds since the result of the non-linear function is independent of the key and therefore predictable in Simon. So, an impossible-differential distinguisher could cover up to 26 rounds. Though, such an upper bound has not been formulated to an attack on the here-considered versions by Kondo et al.; therefore, it is not contained in the overview in Table 4.

Algebraic Cryptanalysis is unlikely to be a threat on Simon-like constructions for sufficiently many rounds. Raddum [Rad15] pointed out that the large number of rounds is necessary, by demonstrating that equation systems of up to 14 rounds of Simon-96-96 and up to 16 rounds of Simon-128 are solvable efficiently in a few minutes on an off-the-shelf laptop. Extensions to considerably more rounds are still unknown.

Meet-in-the-Middle Attacks are successful primarily on primitives that do not use parts of the key in sequences of several rounds. The Simon-$2w$-$2w$ versions use every key bit in each sequence of two subsequent rounds, which limits the chances of meet-in-the-middle attacks drastically. Considering 3-subset meet-in-the-middle attacks, together with an initial structure and partial matching, the length of an attack is limited to roughly that of twice the full diffusion plus four rounds plus the maximal length of an initial structure plus two rounds for a splice-and-cut part, which yields 30 rounds as a rough upper bound. It is unlikely that such attacks cover 30 or more rounds on Simon-$2w$-$2w$.

Table 5: Probabilities of optimal related-key differential characteristics for round-reduced variants of Simon-96-96 and Simon-128-128. p denotes the probability; SK = single-key model, RK = related-key model.

#Rounds	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
#Rounds	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	
Simon-96-96																		
$-\log_2(p)$ (SK)	4	6	8	12	14	18	20	26	30	36	38	44	48	54	56	62	64	66
	68	72	74	78	80	86	90	96										
$-\log_2(p)$ (RK)	0	2	4	10	12	18	20	26										
Simon-128-128																		
$-\log_2(p)$ (SK)	4	6	8	12	14	18	20	26	30	36	38	44	48	54	56	62	64	66
	68	72	74	78	80	86	90	96	98	104	108	114	116	122	124	126	128	
$-\log_2(p)$ (RK)	0	2	4	10	12	18	20	26										

Correlated Sequences. An interesting recent direction may be correlated sequences introduced by Rohit and Gong in [RG18]. Their technique requires only very few texts and claims to break 27 rounds of Simon-32 and Simeck-32; thus, it might outperformed all previous attacks by at least three rounds. Though, that approach needs further investigation and has seen application only to Simon-32-64 until now.

6.3 Implications to SimP

Since the key schedule of Simon is fully linear, the two state words that are transformed by the key schedule allow prediction of differences, linear and algebraic properties through a full step. In any case, SimP transforms each input word through at least $2r_s$ rounds of Simon.

Related-key Differential Cryptanalysis. SimP needs an analysis of related-key differential and linear characteristics. Existing methods such as the exhaustive search in [LLW17a] or SAT solvers [KLT15], render such studies difficult due to the large state size since the known tools cannot scale appropriately. There exist peer-reviewed related-key results on Simon, e.g., by Wang et al. [WWHL18]. For the sake of feasibility, they restricted their search to related-key trails for the small variants, i.e., Simon-32, Simon-48, and Simon-64. We conducted experiments using the SAT-based approach from [KLT15] as well as with the branch-and-bound approach from [LLW17a] to search for optimal differential characteristics on SimP. Though, the related-key analysis of Simon-like constructions is computationally difficult because of the large state size. We obtained improved trails for only for up to seven rounds of Simon-96; starting from eight rounds, the best found characteristics possessed a zero key difference for up to 10 rounds, which suggests that differences in the few key words do not improve the best single-key characteristics. It seems that the probabilities of the existing optimal differential characteristics and linear trails for Simon-96-96 and Simon-128-128 also hold for SimP-192-1 and SimP-256-1 beyond that point. Table 5 compares the probabilities of optimal single- and related-key differential characteristics.

Number of Steps and Rounds of SimP. SimP benefits from the intensive existing cryptanalysis of Simon. The usage of the key-update function of Simon seems to not promote considerably more effective differential or linear distinguishers compared to the single-key results on Simon. The usage of the $2w$-word key appears not exploitable neither by differ-

entials and linear characteristics, nor by techniques that try to exploit more available state such as meet-in-the-middle distinguishers. The reason seems to be mainly the diffusion in the key schedule together with the relatively large number of rounds.
The number of steps and the number of rounds in our employed instantiations of SimP have been chosen very conservatively, using the number of rounds per step r_s as half the number of rounds in Simon. This choice guarantees that each bit passes at least once through the full-round cipher, and therefore is expected to possess at least the algebraic degree of the full-round cipher. Moreover, the diffusion properties of Simon render impossible-differential, zero-correlation, or integral distinguishers implausible.
The design of SimP is very close to the original design of Simon. So, any considerable improvement in the cryptanalysis on SimP would most likely also be a higher threat on Simon-$2w$-$2w$. While such results are not impossible, the higher number of rounds in SimP provide an additional security margin.

7 Features

Oribatida . . .

- . . . is optimized for messages as short as 8 bytes and
- . . . is optimized for message lengths in full bytes.

As given by [NIS18]

Flexibility. All versions of the Oribatida family support. . .

- . . . nonce lengths of at least 64 bits,
- . . . tag lengths of at least 96 bits,
- . . . plaintext lengths of up to $2^{50} - 1$ bytes,
- . . . associated-data lengths of up to $2^{50} - 1$ bytes, and
- . . . processing $2^{50} - 1$ bytes under a single key.

In particular, our primary recommendation Oribatida-256-128 supports. . .

- . . . a nonce length of 128 bits,
- . . . a tag length of 128 bits, and
- . . . a key length of up to 256 bits.

Efficiency. As a keyed sponge mode that initializes the state from key and nonce, the key preprocessing is efficient and requires only a single call to the permutation.

Simplicity. The sponge mode is well-understood and has been analyzed intensely. It is easily adaptable to a hash function or a MAC. The implementation overhead for the decryption is low since the encryption can also be performed with the sole forward direction of the permutation. Moreover, a round-reduced permutation is used in between the associated-data blocks to further boost the performance.

8 Hardware Implementation

This section reports on hardware implementations of SimP and Oribatida.

Table 6: Implementation results for SimP-256 and Oribatida-256-64 encryption/decryption and only encryption on *Virtex 7 FPGA*. LUTs = lookup tables; AD = associated data; Enc. = encryption; Dec. = decryption.

				Frequency	Cycles		Throughput (Mbps)	
	LUTs	FF	#Slices	(MHz)	AD	Message	AD	Message
SimP								
SimP-256	495	340	148	580.51	69	137	1 076.88	542.37
SimP-192	383	259	122	581.98	53	105	1 054.15	532.10
Oribatida-256-64								
Enc. and Dec.	940	599	298	554.16	68	138	1 043.12	514.00
Enc. only	805	595	253	560.71	68	138	1 055.45	520.08

8.1 SimP

SimP is lightweight since its transformations are exactly the round function and the key-update function of Simon-96-96 or Simon-128-128, respectively. Both transformations are based on simple operations such as rotations, XORs, and ANDs that consume only routing resources and bit-wise logical operations. The area in GEs is approximately that of Simon-96 plus some overhead, which is caused from the need of an additional input to both transformations due to the swapping after r_s rounds.
Unprotected implementations of Simon are vulnerable against differential power analysis attacks using the leakage generated by the transitions in the state register; the Hamming-distance model captures such leakage. Masking – in particular, Boolean masking (XORing a random value to the output of the round function) – is one countermeasure that can be applied to Simon easily. The simple structure of Simon components allow to explore other countermeasures such as unrolling rounds to achieve higher-order side-channel resistance.

Latency. SimP can be implemented in different levels of serialization, from fully serial implementations that update solely a single bit per cycle up to round-based implementations that update the full state in one clock cycle. Depending on the choice, there is a broad implementation spectrum with a trade-off between throughput and area.

8.2 Oribatida

Hardware implementations of our proposed instance of Oribatida are relatively straightforward. It can be implemented efficiently with little extra cost compared to the duplex sponge. Additional costs result from the use of a module to generate the constants for the domain separation, which can be held in ROM. In modern FPGAs, this module takes only four look-up tables (LUTs). For domain separation, only a four-bit XOR is necessary at the input for capacity of the permutation. An additional 64-bit register to store a mask, and a 64-bit XOR to add the mask to the ciphertext is requires.
The use of SimP as its main building block allows to directly transfer the same strategy of using different data-path sizes to Oribatida. Thus, the implementer can choose among various trade-offs between throughput, latency, area, and power consumption.
In terms of side-channel resistance, the same aspects that hold for SimP also hold for Oribatida. Thus, Oribatida does not introduce additional weaknesses of side channels.
Table 6 lists our implementations results obtained from Xilinx Vivado 2018 optimizing for area. All results represent measurements after the place-and-route process.

In Table 6, we list two columns for the number of clock cycles and throughput, the former represents the results for the processing of associated data (with the step-reduced SimP), whereas the latter denotes the results for processing the message (with the non-reduced SimP). Our results leaves still room for further improvements in the close future.

9 Software Implementation

SimP Is Very Lightweight and Flexible. For SimP, only the round function and the key-update function of Simon have to be implemented, which can be realized using only rotations, logical ANDs and XORs. Since those operations treat individual bits separately without dependencies among the bits of the same words, the employed internal state size can be arbitrary. Therefore, SimP is well-suited for a variety of platforms independent of word-size limitations. There are no S-boxes or complex constants that must be stored, the RAM and ROM sizes are expected low. The round constants z_i can be implemented compactly using a five-bit Linear Feedback-shift Register [BSS+13].

SimP Alleviates Side-channel Countermeasures. The lack for S-boxes renders constant-time implementations straight-forward. Moreover, the low degree of the internal function alleviates protections with maskings or sharing-based countermeasures such as threshold implementations or consolidated masking schemes.

The Memory Footprint of Oribatida Is Also Low. The full implementation state is given by the n-bit state, the subsequent block, plus the overhead from the mask size, plus the result of initializing the key. Note that the key needs one single preprocessing call to the permutation P at initialization. Moreover, there is no overhead for the decryption operation of the primitive in Oribatida.
To minimize the memory requirements, e.g., Oribatida-192-96 needs 96-bits register for the block, 96 bits for the mask, plus 192 bits for the current state, and 128 bits for the key. Note that the state size of Oribatida is analogous to that of lightweight block ciphers with 128-bit security and small 64-bit state such as GIFT or LED. Though, such primitives either lead to birthday-bound security of at most 2^{32} blocks encrypted under the same key, or must be used in modes with security beyond the birthday bound that are usually slower. Therefore, they require further memory to store the previous state. To provide high security, block-cipher-based modes often have to occupy more memory.

10 Intellectual Property

The submitters are not aware of any patent involved in Oribatida. Furthermore, Oribatida will not be patented. If any of this information changes, the submitters will promptly (and within at most one month) announce these changes on the mailing list of the NIST lightweight competition. According to [BSS+18], "SIMON and SPECK are free from any intellectual property restrictions. [The work on SIMON and SPECK] was prepared by a United States Government employee and, therefore, is excluded from copyright by Section 105 of the Copyright Act of 1976. The algorithms [of SIMON and SPECK] are free for anyone to use. There are no patent or licensing restrictions. Copyright and related rights are expressly waived through the CC0 1.0 Universal License."

Acknowledgments. The authors thank Raghvendra Rohit and Sumanta Sarkar for pointing out an event in the tag-processing process of the Oribatida-192 version which was treated in the INT-RUP bound, but lacked a term in the nAE bound. We also treated it properly in the nAE bound in this version.

References

[AAA+14] Mohamed Ahmed Abdelraheem, Javad Alizadeh, Hoda AlKhzaimi, Mohammad Reza Aref, Nasour Bagheri, Praveen Gauravaram, and Martin M. Lauridsen. Improved Linear Cryptanalysis of Round Reduced SIMON. *IACR Cryptology ePrint Archive*, 2014:681, 2014.

[ABG+13] Javad Alizadeh, Nasour Bagheri, Praveen Gauravaram, Abhishek Kumar, and Somitra Kumar Sanadhya. Linear Cryptanalysis of Round Reduced SIMON. *IACR Cryptology ePrint Archive*, 2013:663, 2013.

[ABL+14] 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 *Lecture Notes in Computer Science*, pages 105–125. Springer, 2014.

[AJN14] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. NORX: Parallel and Scalable AEAD. In Miroslaw Kutylowski and Jaideep Vaidya, editors, *ESORICS II*, volume 8713 of *Lecture Notes in Computer Science*, pages 19–36. Springer, 2014.

[ALLW14] Farzaneh Abed, Eik List, Stefan Lucks, and Jakob Wenzel. Differential Cryptanalysis of Round-Reduced Simon and Speck. In Carlos Cid and Christian Rechberger, editors, *FSE*, volume 8540 of *Lecture Notes in Computer Science*, pages 525–545. Springer, 2014.

[BDPVA12] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Permutation-based encryption, authentication and authenticated encryption. *Directions in Authenticated Ciphers*, 2012.

[BRV14] Alex Biryukov, Arnab Roy, and Vesselin Velichkov. Differential Analysis of Block Ciphers SIMON and SPECK. In Carlos Cid and Christian Rechberger, editors, *FSE*, volume 8540 of *Lecture Notes in Computer Science*, pages 546–570. Springer, 2014.

[BSS+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK Families of Lightweight Block Ciphers. *IACR Cryptology ePrint Archive*, 2013:404, 2013.

[BSS+18] Ray Beaulieu, Doug Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK Families of Block Ciphers, 2018. https://github.com/nsacyber/simon-speck.

[CDMS10] Jean-Sébastien Coron, Yevgeniy Dodis, Avradip Mandal, and Yannick Seurin. A Domain Extender for the Ideal Cipher. In Daniele Micciancio, editor, *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 273–289. Springer, 2010.

[CDNY18] Avik Chakraborti, Nilanjan Datta, Mridul Nandi, and Kan Yasuda. Beetle Family of Lightweight and Secure Authenticated Encryption Ciphers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):218–241, 2018. https://doi.org/10.13154/tches.v2018.i2.218-241.

[CW16] Huaifeng Chen and Xiaoyun Wang. Improved Linear Hull Attack on Round-Reduced Simon with Dynamic Key-Guessing Techniques. In Thomas Peyrin, editor, *FSE*, volume 9783 of *Lecture Notes in Computer Science*, pages 428–449. Springer, 2016.

[DDGS15] Itai Dinur, Orr Dunkelman, Masha Gutman, and Adi Shamir. Improved Top-Down Techniques in Differential Cryptanalysis. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *LATINCRYPT*, volume 9230 of *Lecture Notes in Computer Science*, pages 139–156. Springer, 2015.

[DEMS16] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2. Submission to the CAESAR competition: http://competitions.cr.yp.to/round3/asconv12.pdf, 2016.

[HKR15] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. Robust Authenticated-Encryption – AEZ and the Problem That It Solves. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT I*, volume 9056 of *Lecture Notes in Computer Science*, pages 15–44. Springer, 2015.

[ISO18] ISO/IEC. Information technology – Automatic identification and data capture techniques – Part 21: Crypto Suite SIMON Security Services for Air Interface Communications, Oct 2018.

[KLT15] Stefan Kölbl, Gregor Leander, and Tyge Tiessen. Observations on the SIMON Block Cipher Family. In Rosario Gennaro and Matthew Robshaw, editors, *CRYPTO*, volume 9215 of *Lecture Notes in Computer Science*, pages 161–185. Springer, 2015.

[KSTI18] Kota Kondo, Yu Sasaki, Yosuke Todo, and Tetsu Iwata. On the Design Rationale of SIMON Block Cipher: Integral Attacks and Impossible Differential Attacks against SIMON Variants. *IEICE Transactions*, 101-A(1):88–98, 2018.

[LLW17a] Zhengbin Liu, Yongqiang Li, and Mingsheng Wang. Optimal Differential Trails in SIMON-like Ciphers. *IACR Trans. Symmetric Cryptol.*, 2017(1):358–379, 2017.

[LLW17b] Zhengbin Liu, Yongqiang Li, and Mingsheng Wang. The Security of SIMON-like Ciphers Against Linear Cryptanalysis. *IACR Cryptology ePrint Archive*, 2017:576, 2017.

[Mat94] Mitsuru Matsui. On Correlation Between the Order of S-boxes and the Strength of DES. In Alfredo De Santis, editor, *EUROCRYPT*, volume 950 of *Lecture Notes in Computer Science*, pages 366–375. Springer, 1994.

[MRH04] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2004.

[MRV15] Bart Mennink, Reza Reyhanitabar, and Damian Vizár. Security of Full-State Keyed Sponge and Duplex: Applications to Authenticated Encryption. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT II*, volume 9453 of *Lecture Notes in Computer Science*, pages 465–489. Springer, 2015.

[NIS18] NIST. Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process. https://csrc.nist.gov/projects/lightweight-cryptography, August 27 2018.

[Rad15] Håvard Raddum. Algebraic Analysis of the Simon Block Cipher Family. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *LATINCRYPT*, volume 9230 of *Lecture Notes in Computer Science*, pages 157–169. Springer, 2015.

[RG18] Raghvendra Rohit and Guang Gong. Correlated Sequence Attack on Reduced-Round Simon-32/64 and Simeck-32/64. *IACR Cryptology ePrint Archive*, 2018:699, 2018. https://eprint.iacr.org/2018/699.

[TM16] Yosuke Todo and Masakatu Morii. Bit-Based Division Property and Application to Simon Family. In Thomas Peyrin, editor, *FSE*, volume 9783 of *Lecture Notes in Computer Science*, pages 357–377. Springer, 2016.

[WWHL18] Xuzi Wang, Baofeng Wu, Lin Hou, and Dongdai Lin. Automatic Search for Related-Key Differential Trails in SIMON-like Block Ciphers Based on MILP. In Liqun Chen, Mark Manulis, and Steve Schneider, editors, *ISC*, volume 11060 of *Lecture Notes in Computer Science*, pages 116–131. Springer, 2018.

[XZBL16] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP Method to Searching Integral Distinguishers Based on Division Property for 6 Lightweight Block Ciphers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT I*, volume 10031 of *Lecture Notes in Computer Science*, pages 648–678, 2016.

[ZWW15] Huiling Zhang, Wenling Wu, and Yanfeng Wang. Integral Attack Against Bit-Oriented Block Ciphers. In Soonhak Kwon and Aaram Yun, editors, *ICISC*, volume 9558 of *Lecture Notes in Computer Science*, pages 102–118. Springer, 2015.