Research Article

Arghya Bhattacharjee*, Cuauhtemoc Mancillas López, Eik List, and Mridul Nandi

The Oribatida v1.3 Family of Lightweight Authenticated Encryption Schemes

https://doi.org/10.1515/jmc-2020-0018
Received Apr 06, 2020; accepted Oct 20, 2020

Abstract: Permutation-based modes have been established for lightweight authenticated encryption, as can be seen from the high interest in the ongoing NIST lightweight competition. However, their security is upper bounded by $O(\sigma^2/2^c)$ bits, where σ are the number of calls and c is the hidden capacity of the state. The development of more schemes that provide higher security bounds led to the CHES'18 proposal Beetle that raised the bound to $O(r\sigma/2^c)$, where r is the public rate of the state.
While authenticated encryption can be performed in an on-line manner, authenticated decryption assumes that the resulting plaintext is buffered and never released if the corresponding tag is incorrect. Since lightweight devices may lack the resources for buffering, additional robustness guarantees, such as integrity under release of unverified plaintexts (Int-RUP), are desirable. In this stronger setting, the security of the established schemes, including Beetle, is limited by $O(q_p q_d/2^c)$, where q_d is the maximal number of decryption queries, and q_p that of off-line primitive queries, which motivates novel approaches.
This work proposes Oribatida, a permutation-based AE scheme that derives s-bit masks from previous permutation outputs to mask ciphertext blocks. Oribatida can provide a security bound of $O(r\sigma^2/2^{c+s})$, which allows smaller permutations for the same level of security. It provides a security level dominated by $O(\sigma_d^2/2^c)$ under Int-RUP adversaries, which eliminates the dependency on primitive queries. We prove its security under nonce-respecting and Int-RUP adversaries. We show that our Int-RUP bound is tight and show general attacks on previous constructions.

Keywords: Authenticated encryption, permutation, provable security

2020 Mathematics Subject Classification: 11T06, 11T71, 11Y16, 94A60

1 Introduction

1.1 Permutation-based Modes

Permutation-based modes have been established for various applications of symmetric-key cryptography during the previous decade. Keyless modes have been standardized as the hash function SHA-3 and its derivative SHAKE for the extendable-output functions [36]. Keyed modes are used for authentication [51] or encryption [13]. Moreover, permutation-based schemes have found widespread adoption for authenticated encryption, as the CAESAR co-selection Ascon [34], or many more candidates have shown, e.g., PRIMATES [6], NORX [9], Ketje [19], or Keyak [20].

***Corresponding Author: Arghya Bhattacharjee:** Applied Statistics Unit, Indian Statistical Institute, Kolkata, India; Email: bhattacharjeearghya29@gmail.com
Cuauhtemoc Mancillas López: Computer Science Department, CINVESTAV-IPN, Mexico, Mexico; Email: cuauhtemoc.mancillas83@gmail.com
Eik List: Bauhaus-Universität Weimar, Weimar, Germany; Email: eik.list@uni-weimar.de
Mridul Nandi: Applied Statistics Unit, Indian Statistical Institute, Kolkata, India; Email: mridul.nandi@gmail.com

The sponge [17] and duplex [15] modes transform an internal n-bit state iteratively with a public permutation. Both modes absorb an input stream block-wise to generate a pseudo-random output stream. While sponges separate the input (absorption) and output (squeezing) phases, the duplex mode generates the i-th output block directly after the i-th input block has been absorbed. In both modes, an n-bit permutation absorbs the data in r-bit chunks, where the outer part of the state $r < n$ is called the rate. The hidden inner part of the state of $c = n - r$ bits is called the capacity, where r and c are a trade-off between performance and security.

Keyed Sponge Variants were introduced by Bertoni et al. [18] and can be categorized into inner-keyed, outer-keyed, and full-keyed variants (cf. [48]); recently, Dobraunig and Mennink added the suffix-keyed sponge [35]. The inner-keyed sponge [28] initializes the inner part with the key, $(0 \parallel K)$, whereas the outer-keyed sponge [18] (so-dubbed by [8]) concatenates key and message $K \parallel M$ for the output. The full-keyed sponge [16] employs the full state in the absorption phase; the suffix-keyed sponge uses a keyed function only at the end.

Permutation-based modes are analyzed mostly in the ideal-permutation model. For authenticated encryption, an adversary $\mathbf{A}$ shall distinguish between two worlds consisting of two oracles: each world has (1) a construction oracle that $\mathbf{A}$ can ask encryption and verification queries to and (2) a primitive oracle that provides access to the internal permutation. The former oracle represents on-line queries, whereas the latter represents off-line queries. $\mathbf{A}$ can ask q_e encryption queries, q_v verification queries, and q_p construction queries; σ usually denotes the number of blocks over all construction queries.

Sponge Modes for Authenticated Encryption started with the Duplex construction and the AE scheme SpongeWrap [15] and MonkeyDuplex [16], and led to a considerable corpus of analysis, e.g., [8, 32, 37, 49, 52]. Early, Bertoni et al. [14] showed that the sponge is indifferentiable from a random oracle [47] for up to $O(2^{c/2})$ calls to the permutation. Their follow-up work [18] improved the bounds for the unkeyed sponge to $O(\frac{q_p \sigma}{2^c} + \frac{q_c}{2^k})$ if $\sigma \ll 2^{c/2}$. For SpongeWrap, Bertoni et al. [15] had shown a privacy bound of $O(\frac{q}{2^k} + \frac{\sigma^2}{2^c})$ and an authenticity bound of $O(\frac{q}{2^k} + \frac{\sigma^2}{2^c} + \frac{q}{2^r})$.

Jovanovic et al. [41] improved the asymptotic authenticity bound, although under the limitation of at most $\sigma \ll 2^{c/2}$ decryption queries. Summarizing many previous results, Mennink [48] showed that keyed sponges achieve PRF security of around $O(\frac{q_c^2 + q_c q_p}{2^c}) + \mathbf{Adv}_{\Pi}^{\mathrm{KP}}$. He coined the final term the key-prediction security, e.g., in $O(\frac{q_p}{2^k})$ for full-keyed sponges and $k < n$. The recent duplex-based AE scheme Beetle [25] added a transform to the output so that the plaintext input that is added to the inner part and to the ciphertext output block differ. As a result, Beetle offered a bound of $O(\frac{r q_p + r\sigma}{2^c} + \frac{q_v + q_p}{2^r} + \frac{\sigma^2 + q_p^2}{2^n})$. Table 1 summarizes some of the most noteworthy results in the past. Improvements to those general bounds appear hard, which motivates the search for novel constructions.

Table 1: Existing PRF bounds for keyed sponges. FKS/IKS/OKS = full-/inner-/outer-keyed sponge, IP = ideal permutation, indiff. = indifferentiability.

Scheme					
FKS	IKS	OKS	Model	Bound	Work
–	–	•	Indiff.	$O(\frac{\sigma^2+\sigma}{2^c})$	[14]
–	•	•	IP	$O(\frac{\sigma^2}{2^c})$	[28]
–	–	•	IP	$O(\frac{\sigma^2+\sigma q_p}{2^c})$	[8]
–	•	–	IP	$O(\frac{\sigma^2+\sigma q_p}{2^c} + \frac{q_p}{2^n})$	[8]
NORX-like			IP	$O(\min(\frac{\sigma^2}{2^n}, \frac{\sigma}{2^c}, \frac{q}{2^k}))$	[41]
–	–	•	IP	$O(\frac{q_c^2+\sigma q_c+q_c q_p}{2^c})$	[37]
•	–	–	IP	$O(\frac{\sigma^2}{2^n} + \frac{q_c^2 m}{2^c} + \frac{\sigma_c q_p}{2^k})$	[49]
–	•	•	IP	$O(\frac{q_c^2+q_c q_p}{2^c})$	[52]
•	–	–	IP	$O(\frac{q_c q_p+q_c^2}{2^c} + \frac{q_p}{2^k})$	[32]
•	–	–	IP	$O(\frac{q_c^2+q_c q_p}{2^c} + \frac{q_p}{2^k})$	[48]
–	–	•	IP	$O(\frac{q_c^2+q_c q_p}{2^c} + \frac{c^{k/r} q_p}{2^k})$	[48]

Correct Authenticated Decryption requires the entire plaintext to be buffered until the tag has been verified. On certain architectures, this requirement can exceed the available storage and induce unacceptable latency.

Andreeva et al. [7] introduced notions for privacy and integrity under the release of unverified plaintext material. Security guarantees such as INT-RUP represent valuable additional levels of robustness.

1.2 Research Gap

While Beetle improved the bound for nonce-based authenticated encryption, this bound does not hold in the INT-RUP setting: as we will outline, there exists an attack with an advantage of $O(\frac{q_p q_v}{2^c})$. The naturally arising question is whether one can achieve higher INT-RUP security. This is motivated by a practical demand: while the primary advantage of sponges is simplicity, they are also useful for lightweight applications in resource-constrained environments. In such contexts, buffering multi-block decryption outputs is likely infeasible, which renders INT-RUP security advantageous. For example, the ongoing NIST lightweight competition [53] requests 112-bit integrity security for AE schemes and support of at least $2^{50} - 1$ encrypted bytes of data. Inserting them into Mennink's bound [48] of $\frac{q_c^2 + q_c q_p}{2^c}$, the NIST requirements would imply permutation sizes of at least 172 bits plus a plausible rate. A higher INT-RUP security could lead to smaller permutations, reducing area, and energy consumption simultaneously.

1.3 Contribution

This work contains three contributions: First, it answers the question above in the affirmative by showing a sponge with dynamic s-bit masks that achieves NAE security of $O(\frac{q_p}{2^k} + \frac{q_d}{2^\tau} + \frac{q_p \sigma}{2^{c+s}} + \frac{\sigma^2}{2^n})$. Replacing the terms by the NIST requirements of $k \geq 128$, $\tau \geq 64$, $q_p \leq 2^{112}$ and $\sigma \geq 2^{50}$, a trade-off could use $c + s \geq 162$, or better $c + s \geq 192$ bits to be secure for up to $\sigma \in O(2^{64})$ blocks and $q_p \leq 2^{128}$ primitive queries. Thus, the rate could be reduced considerably compared to the bound from [48]. We prove that the INT-RUP advantage is at most $O(q_d^2/2^c)$, i.e., depends solely on the number of on-line queries, contrasting the bound of $O(q_d q_p/2^c)$ for the generic duplex and previous constructions. The difference may seem minor; though, eliminating the dependency of off-line (i.e., primitive) queries is a valuable gain. We propose a novel permutation-based AE scheme Oribatida that applies dynamic masks to the outputs and provides the NIST security bounds with permutation sizes of only 192 bits. As our second contribution, we show that the bound provided by our proposal is tight with an attack that matches the proved bound. Moreover, we show that it applies also to other duplex-based designs in general if masks would be added.

Remark 1. Finally, we acknowledge an observation by Rohit and Sarkar on the NIST lightweight mailing list [59]. We note that our proposal here represents a slightly updated variant Oribatida compared to the NIST submission [21] that addresses their observation by masking the authentication tag. We call it also Oribatida v1.3, but use Oribatida hereafter. We will discuss the effect of the slight update later.

1.4 Outline

After a brief recall of the necessary preliminaries, Section 3 motivates our proposal by showing INT-RUP attacks on the duplex mode and other existing schemes. Section 4 describes Oribatida in general. We close the parenthesis of INT-RUP attacks on Oribatida and other duplex-based modes when in Section 5. We analyze the security on Oribatida for the standard nonce-based AE setting in Section 6 and in the INT-RUP setting in Section 7. Next, Section 8 compares it with those second-round NIST lightweight candidates that claim INT-RUP security. Section 9 discusses the slight update from [21] and the associated improvement. Subsequently, Section 10 specifies an instance with a Simon-based permutation, whose security is discussed in Section 11 from previous works. Section 12 reports on the result of a hardware implementation of Oribatida before Section 13 concludes this work.

2 Preliminaries

2.1 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 bit of X, and define the bit order by $X = (X[n-1] \,\|\, \dots \,\|\, X[1] \,\|\, X[0])$. For $t \le n$, we use $\mathsf{msb}_t(X) = (X[n-1] \dots X[n-t])$ to return the t rightmost (or most significant when interpreting X as integer) and $\mathsf{lsb}_t(X) = (X[t-1] \dots X[1]X[0])$ to return the t leftmost (or least significant when interpreting X as integer) bits of X. We write $\emptyset$ for the empty set and ε for the empty string.

We denote by $X[x..y]$ the range of $X[x], \dots, X[y]$ 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 $x \ge y$, 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 $(X_1, X_2, \dots, X_x) \xleftarrow{n} X$ the splitting of X into n-bit strings $X_1, \dots, X_{x-1}$, and $|X_x| \le n$, in form of $X_1 \,\|\, \dots \,\|\, X_x = X$. For $Y \in \mathbb{F}_2^x$, we write $(X_1, X_2, \dots, X_m) \xleftarrow{x_1, x_2, \dots, x_m} Y$ for the splitting of Y into $X_1 = Y[x-1..x-x_1]$, $X_2 = Y[x-x_1-1..x-x_1-x_2], \dots, X_m = Y[x_m-1..0]$, where $x = x_1 + x_2 + \dots + x_m$ holds. For a given set $\mathcal{X}$ and non-negative integer x, we write $\mathcal{X}^{\le x}$ for the union set $\cup_{i=0}^{x} \mathcal{X}^i$. For a natural $x < 2^n$, we write $\langle x \rangle_n$ for its conversion to an n-bit binary string with the most significant bit left, e.g., $\langle 135 \rangle_8 = (10000111)$. We omit n if clear from the context.

2.2 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 AE 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 $\mathcal{D} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{C} \times \mathcal{T} \to \mathcal{M} \cup \{\perp\}$ 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 AE schemes are correct, i.e., for all $(K, N, A, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M}$, it holds that $\mathcal{D}_K^{N,A}(\mathcal{E}_K^{N,A}(M)) = M$.

The ideal AE scheme provides two oracles $\$: \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{C} \times \mathcal{T}$ and $\perp : \mathcal{N} \times \mathcal{A} \times \mathcal{C} \times \mathcal{T} \to \mathcal{M} \cup \{\perp\}$ that offer access to encryption and verification. We overload the $\perp$ notation to mean the oracle and the symbol of invalid decryption. Given (N, A, M), the ideal encryption oracle outputs ciphertext-tag tuples (C, T) that are random bits of the expected length. The ideal decryption oracle ensures correctness, i.e., 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 to an earlier encryption query.

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 permutation 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. Note that in the analysis, the permutation is chosen uniformly at random from the set of all permutations over $\mathcal{B}$. Though, the model is an adaption of the ideal-cipher model [60], and not of the random-oracle model [12]. We write $\Delta_{\mathbf{A}}(\mathcal{O}^1; \mathcal{O}^2)$ for the advantage of **A** to distinguish between $\mathcal{O}^1$ and $\mathcal{O}^2$.

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 **A** be a nonce respecting adversary. Then,

$$\mathbf{Adv}^{\mathrm{NAE}}_{\Pi[\pi]}(\mathbf{A}) \overset{\mathrm{def}}{=} \underset{\mathbf{A}}{\Delta} \left(\mathcal{E}[\pi]_K, \mathcal{D}[\pi]_K, \pi^{\pm}; \$, \perp, \pi^{\pm}\right) .$$

In the RUP model, the understanding of nonce-based AE differs slightly from the previous definition. Here, we use the notion that the adversary can always see the full resulting plaintext from a decryption query. 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 three 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.

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 **A** be a nonce-respecting adversary. Then,

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

2.3 H-coefficient Technique

We will need a proof method in the later parts, where we opt for Patarin's well-suited H-coefficient technique in the variant by Chen and Steinberger [30, 54]. The results of the interaction of an adversary **A** with its oracles are collected in a transcript τ. The oracles can sample randomness before the interaction (often a key or an ideal primitive that is sampled beforehand), and are then deterministic throughout the experiment [30]. The challenge of **A** is to distinguish the real world $\mathcal{O}_{\mathrm{real}}$ from the ideal world $\mathcal{O}_{\mathrm{ideal}}$. We denote by Θ_{real} and Θ_{ideal} random variables that represent the distribution of transcripts in the real and the ideal world, respectively. In general, a transcript τ is called *attainable* if the probability to obtain τ in the ideal world – i.e. over Θ_{ideal} – is non-zero. The Fundamental Lemma of the H-coefficients technique, the proof to which is given in [30, 54], states that we can partition the set of all attainable transcripts into two disjoint sets GOODT and BADT.

Lemma 1 (Fundamental Lemma of H-coefficient Technique [54]). Assume that there exist $\epsilon_1, \epsilon_2 \geq 0$ such that for any transcript $\tau \in$ GOODT, it holds that $\Pr[\Theta_{\mathrm{real}} = \tau] / \Pr[\Theta_{\mathrm{ideal}} = \tau] \geq 1 - \epsilon_1$. Moreover, assume that $\Pr[\Theta_{\mathrm{ideal}} \in \mathrm{BADT}] \leq \epsilon_2$. Then under the aforementioned assumptions, for all adversaries **A**, it holds that $\Delta_{\mathbf{A}}(\mathcal{O}_{\mathrm{real}}; \mathcal{O}_{\mathrm{ideal}}) \leq \epsilon_1 + \epsilon_2$.

Note that in all our analyses, we consider information-theoretic distinguishers **A**, where we assume idealized primitives. Thus, their resources are bounded only in terms of their maximal numbers of queries and blocks that they can ask to their available oracles. One can derive the computation-theoretic counterparts easily by adding a parameter of the distinguishers' maximal computational efforts.

3 Int-RUP Attacks on Existing AE Schemes

This section shows attacks under Int-RUP adversaries on the duplex mode, as well as on recent more secure AE schemes Beetle or SPoC. For each construction, we briefly recall the necessary parts of their definition. As summarized in Table 2, the proposed attacks possess an advantage of $O(\frac{q_p q_d}{2^c})$ on the previous constructions. Thus, the improved bounds of Beetle or SPoC do not carry over to the Int-RUP setting. For all attacks, we consider a random permutation $\pi \twoheadleftarrow \mathsf{Perm}(\mathcal{B})$ and a randomly chosen secret key $K \twoheadleftarrow \mathcal{K}$. We denote by $\mathbf{A}$ a nonce-respecting Int-RUP adversary against the individual schemes.

Table 2: Comparison of the security bounds for Int-RUP attacks on previous permutation-based AE schemes and our construction.

	Bound	
Scheme	Unmasked	Masked
Generic Duplex [15]	$O(q_d q_p/2^c)$	$O(q_d^2/2^c)$
Beetle [25]	$O(q_d q_p/2^c)$	$O(q_d^2/2^c)$
SPoC [5]	$O(q_d q_p/2^c)$	$O(q_d^2/2^r)$
Oribatida [This work]	–	$O(q_d^2/2^c)$

The main idea of all the attacks in this section is as follows: $\mathbf{A}$ asks q_d decryption queries s. t. any predetermined r bits (e.g., the first r bits) of the input to one of the permutations of the construction are fixed and known (say X). The remaining $n - r = c$ bits may vary. Next, $\mathbf{A}$ asks q_p primitive queries $Q^1, Q^2, \ldots, Q^{q_p}$ with the first r bits fixed to X, but with pairwise distinct c bits, and receives $R^1, R^2, \ldots, R^{q_p}$. When $q_d \cdot q_p \approx O(2^c)$, $\mathbf{A}$ can expect a state collision between an on-line input to the permutation and an (off-line) permutation query. This collision can be detected from the first r bits of the outputs of the corresponding construction queries, which will be equal for the colliding inputs. Once $\mathbf{A}$ knows the full state at the input to the permutation of the construction, it can revert the permutation calls in the construction and finally recover the key. We adapt this strategy for the duplex mode and Beetle before we consider the differences in SPoC and hybrids.

3.1 Int-RUP Attack on The Duplex Mode

Let us consider the Sponge-Wrap mode [15].

1. The adversary $\mathbf{A}$ asks q_d decryption queries $(N, A^1, C), (N, A^2, C), \ldots, (N, A^{q_d}, C)$, and receives $M^1, M^2, \ldots, M^{q_d}$. The associated data A^i consist of a single block, the ciphertexts $C = (C_1, C_2)$ are fixed to the same two blocks for each query.
2. Now $\mathbf{A}$ can follow the generic idea to complete the attack.

The attack complexity is $q_d q_p \approx O(2^c)$.

3.2 Int-RUP Attack on Beetle

Beetle [25] is a recent permutation-based light-weight AE scheme. From now onward, we consider the updated variant [26] that fixed the proof and details (such as no double call to the permutation). An overview of the encryption process is provided in Figure 1. The map $\rho : \{0,1\}^r \times \{0,1\}^r \to \{0,1\}^r \times \{0,1\}^r$ computes $\rho(I_1, I_2) \overset{\text{def}}{=} (\mathsf{shuffle}(I_1) \oplus I_2, I_1 \oplus I_2)$ for all inputs $I_1, I_2 \in \{0,1\}^r$, where $\mathsf{shuffle}(x) =^{\text{def}} (\mathsf{lsb}_{r/2}(x) \,\|\, \mathsf{lsb}_{r/2}(x) \oplus \mathsf{msb}_{r/2}(x))$. The results of ρ are ordered as $(X_{a+i}, C_i) \leftarrow \rho(Y_{a+i}, M_i)$ and $(Y_{a+i}, M_i) \leftarrow \rho^{-1}(X_{a+i}, C_i)$, respectively.

1. The adversary **A** asks q_d encryption queries $(N^1, A^1, M), (N^2, A^2, M), \ldots, (N^{q_d}, A^{q_d}, M)$ to the encryption oracle, and receives $C^1, C^2, \ldots, C^{q_d}$. Size of M and A^i is one block for each i.
2. Then, **A** asks q_d decryption queries $(N^1, A^1, C^{1'}), (N^2, A^2, C^{2'}), \ldots, (N^{q_d}, A^{q_d}, C^{q'_d})$ to the decryption oracle where $C^{i'} \leftarrow Y_2^i \oplus \mathsf{shuffle}(Y_2^i)$. This ensures that the first r bits of the input to the third permutation always equal zero.
3. Now **A** can follow the generic idea to complete the attack.

The attack complexity is again $q_d q_p \approx O(2^c)$.

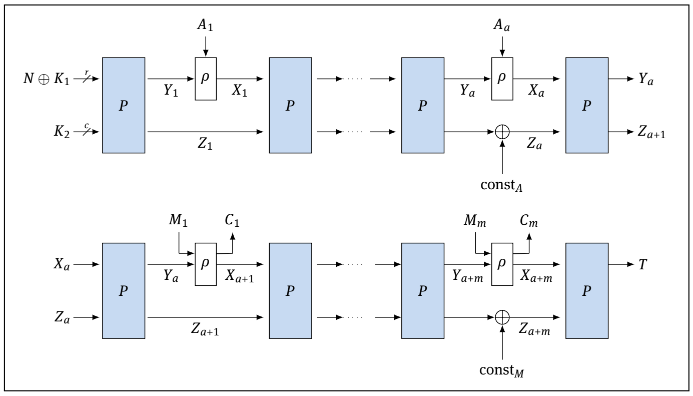
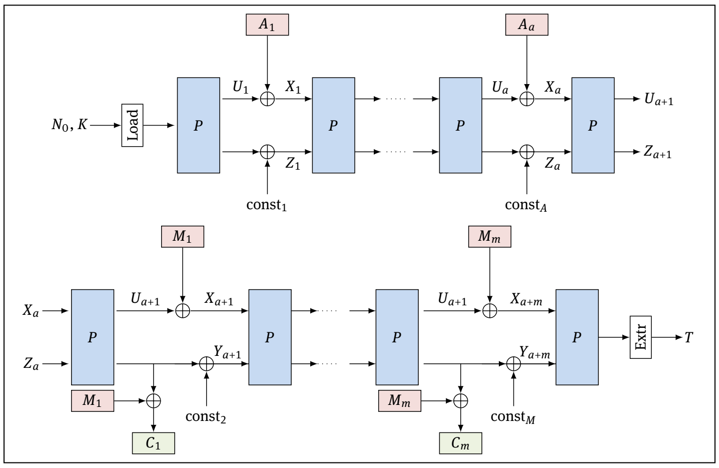


Figure 1: The Beetle authenticated encryption scheme.

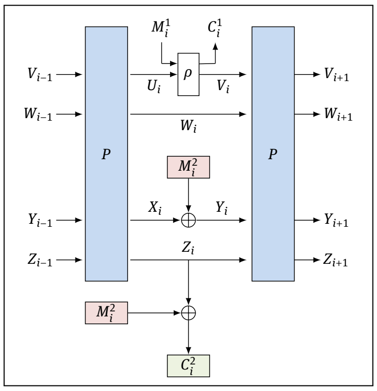
3.3 INT-RUP Attack on SPoC

SPoC (see Figure 2a) is a permutation-based NIST Lightweight candidate [5]. that uses the capacity to derive ciphertext outputs from, while it still absorbs the message in the rate. In SPoC, the adversary cannot fix any part of the state in contrast to SpongeWrap and Beetle– though, there is a similar attack:

1. **A** asks q_d queries $(N, A, C^1), (N, A, C^2), \ldots, (N, A, C^{q_d})$ to the decryption oracle and receives $M^1, M^2, \ldots, M^{q_d}$. The associated data A and ciphertext C^i consist of a single block for every i. This ensures that the first r bits of the input to the third permutation always equal $M^1 \oplus C^1$.



(a) The SPoC authenticated encryption scheme.



(b) A hybrid of Beetle and SPoC.

Figure 2

2. Now **A** can follow the generic idea to complete the attack.

So, the attack needs $q_d q_p \in O(2^c)$ to work, as before.

3.4 Int-RUP Attack on A Hybrid of Beetle and SPoC

We can generalize our attacks to hybrid modes of Beetle and SPoC as well. Such a hybrid would use both modes Beetle and SPoC in parallel to process the queries. We illustrate it in Figure 2b. Each message block (say M) is parsed into two sections (say M^1 and M^2), where $|M^1| = r_1$ and $|M^2| = r_2$. M^1 is processed with Beetle to a ciphertext block C^1; M^2 with SPoC to a ciphertext block C^2; The final ciphertext block becomes $C \leftarrow C^1 \| C^2$, and the associated-data blocks and the ciphertext blocks for decryption are treated in a similar manner. Note that the hybrid mode is parameterized by r_1, r_2 and c with the condition $c \geq r_2$. The size of rate and capacity of the Beetle part are r_1 and $c - r_2$; the size of both rate and capacity of the SPoC part is r_2. As a result, the size of rate and capacity of the hybrid mode is $r = r_1 + r_2$ and c. When $r_2 = 0$, the hybrid mode translates to the Beetle mode. Similarly, when $r_1 = 0$, the hybrid mode is equivalent to the SPoC mode.

An Int-RUP attack on such modes could be defined as follows:

1. **A** asks q_d decryption queries $(N, A^1, C), (N, A^2, C), \ldots, (N, A^{q_d}, C)$ to the decryption oracle and receives $M^1, M^2, \ldots, M^{q_d}$. The ciphertext C and associated data A^i consist of a single block for every i.
2. There exists at least one value of the last r_2 bits of the input to the third permutation which remains same for at least $q = \frac{q_d}{2^{r_2}}$ queries. Suppose q such queries are $(N, A^{1'}, C), (N, A^{2'}, C), \ldots, (N, A^{q'}, C)$.
3. **A** can detect the previous step as it knows the value of the last r_2 bits of the input to the third permutation because that will be equal to the last r_2 bits of $C \oplus M^i$.
4. **A** retains those q queries and discards the rest.
5. For each of the above queries, **A** updates the value of the first r_1 bits of the ciphertext to $Y_2 \oplus \mathsf{shuffle}(Y_2)$ and varies the remaining r_2 bits. This ensures that the first r_1 bits of the input to the third permutation always equal zero.
6. In the way mentioned above, **A** can ask q_d more decryption queries to the decryption oracle. This time, a total of r bits (first r_1 bits and last r_2 bits) of the input to the third permutation are fixed and known to **A**.
7. Then, **A** can follow the generic idea to complete the attack.

The attack needs again $q_d q_p \in O(2^c)$ as before.

3.5 Discussion

As a takeaway from this section, the unmasked sponge-based AE schemes allow Int-RUP attacks whose advantage can depend linearly on the number of off-line primitive queries. We think that any AE construction which uses linear feedback is vulnerable to such an attack ($q_d q_p \in O(2^c)$) unless it uses more state. The next section defines Oribatida that masks its ciphertexts for higher Int-RUP resistance. After its definition, we will get back to attacks on it and on strengthened versions of the schemes sketched here to show that they can be similarly extended by a ciphertext masking.

4 Specification of Oribatida

At its core, Oribatida is a variant of the monkey-wrap design [16], but adds a ciphertext masking. This section considers a slightly updated version of Oribatida. Section 9 discusses the update from Oribatida (v1.2) from [21] to Oribatida v1.3 in this work.

In the following, let $P \in \mathsf{Perm}(\mathcal{B})$ be a permutation. We denote by (X_i, Y_i) the inputs and by (U_i, V_i) the outputs of the primitive(s). As in the classical sponge, Oribatida considers the state $S_i = (U_i \| V_i)$ as a rate U_i

Algorithm 1 Specification of Oribatida v1.3. The domain-encoding functions GetDomainForN, GetDomainForA, and GetDomainForE are instantiation-specific. They are defined in Algorithm 2.

```
101: function $\mathcal{E}_K^{N,A}(M)$
102:     $\ell_A \leftarrow |A|$
103:     $\ell_E \leftarrow |M|$
104:     $d_N \leftarrow \textsc{GetDomainForN}(\ell_A, \ell_E)$
105:     $d_A \leftarrow \textsc{GetDomainForA}(\ell_A, \ell_E)$
106:     $d_E \leftarrow \textsc{GetDomainForE}(\ell_E)$
107:     if $|A| = 0$ then $A \leftarrow 1 \| 0^{r-1}$
108:     $A \leftarrow \mathsf{pad}_r(A)$
109:     $M \leftarrow \mathsf{pad}_r(M)$
110:     $(S_1, V_1) \leftarrow \textsc{Init}(K, N, d_N, \ell_A)$
111:     $S_{a+1} \leftarrow \textsc{ProcessAD}(S_1, A, d_A)$
112:     $(C, T) \leftarrow \textsc{Encrypt}(S_{a+1}, M, V_1, d_E, \ell_E)$
113:     return $(C, T)$

121: function $\textsc{Encrypt}(S_{a+1}, M, V_1, d_E, \ell_E)$
122:     $x \leftarrow \ell_E \bmod r$
123:     $(M_1, \cdots, M_m) \xleftarrow{r} M$
124:     $V \leftarrow V_1$
125:     for $i = 1..m$ do
126:         $(U_{a+i}, V_{a+i}) \xleftarrow{r,c} S_{a+i}$
127:         $X_{a+i} \leftarrow M_i \oplus U_{a+i}$
128:         $C_i \leftarrow X_{a+i} \oplus_s \mathsf{lsb}_s(V)$
129:         $Y_{a+i} \leftarrow V_{a+i}$
130:         if $i = m$ then
131:             $Y_{a+i} \leftarrow Y_{a+i} \oplus_d d_E$
132:             $C_m \leftarrow \mathsf{msb}_x(C_m)$
133:         $V \leftarrow V_{a+i}$
134:         $S_{a+i+1} \leftarrow P(X_{a+i} \| Y_{a+i})$
135:     $C \leftarrow (C_1 \| C_2 \| \cdots \| C_m)$
136:     $T \leftarrow \mathsf{msb}_\tau(S_{a+m+1}) \oplus_s \mathsf{lsb}_s(V)$
137:     return $(C, T)$

141: function $\textsc{Init}(K, N, d_N, \ell_A)$
142:     $V_0 \leftarrow \mathsf{lsb}_s(N \| K)$
143:     $S_1 \leftarrow P((N \| K) \oplus_d d_N)$
144:     $V_1 \leftarrow \mathsf{lsb}_s(S_1)$
145:     return $(S_1, V_1)$

151: function $\textsc{ProcessAD}(S_1, A, d_A)$
152:     $(A_1, \cdots, A_a) \xleftarrow{r} A$
153:     for $i = 1..a-1$ do
154:         $S_{i+1} \leftarrow P(S_i \oplus (A_i \| 0^c))$
155:     $S_{a+1} \leftarrow P(S_a \oplus (A_a \| 0^c) \oplus_d d_A)$
156:     return $S_{a+1}$

201: function $\mathcal{D}_K^{N,A}(C, T)$
202:     $\ell_A \leftarrow |A|$
203:     $\ell_E \leftarrow |C|$
204:     $d_N \leftarrow \textsc{GetDomainForN}(\ell_A, \ell_E)$
205:     $d_A \leftarrow \textsc{GetDomainForA}(\ell_A, \ell_E)$
206:     $d_E \leftarrow \textsc{GetDomainForE}(\ell_E)$
207:     if $|A| = 0$ then $A \leftarrow 1 \| 0^{r-1}$
208:     $A \leftarrow \mathsf{pad}_r(A)$
209:     $C \leftarrow \mathsf{pad}_r(C)$
210:     $(S_1, V_1) \leftarrow \textsc{Init}(K, N, d_N, \ell_A)$
211:     $S_{a+1} \leftarrow \textsc{ProcessAD}(S_1, A, d_A)$
212:     $(M, T') \leftarrow \textsc{Decrypt}(S_{a+1}, C, V_1, d_E, \ell_E)$
213:     if $T = T'$ then return $M$
214:     else return $\perp$

221: function $\textsc{Decrypt}(S_{a+1}, C, V_1, d_E, \ell_E)$
222:     $x \leftarrow \ell_E \bmod r$
223:     $V \leftarrow V_1$
224:     if $\ell_E = 0$ then
225:         $T' \leftarrow \mathsf{msb}_\tau(S_{a+1}) \oplus_s \mathsf{lsb}_s(V)$
226:         return $(\varepsilon, T')$
227:     $(C_1, \cdots, C_m) \xleftarrow{r} C$
228:     for $i = 1..m$ do
229:         $(U_{a+i}, V_{a+i}) \xleftarrow{r,c} S_{a+i}$
230:         $X_{a+i} \leftarrow C_i \oplus_s \mathsf{lsb}_s(V)$
231:         $Y_{a+i} \leftarrow V_{a+i}$
232:         $M_i \leftarrow U_{a+i} \oplus X_{a+i}$
233:         if $i = m$ then
234:             $Y_{a+i} \leftarrow Y_{a+i} \oplus_d d_E$
235:             $M_m \leftarrow \mathsf{msb}_x(M_m)$
236:         $V \leftarrow V_{a+i}$
237:         $S_{a+i+1} \leftarrow P(X_{a+i} \| Y_{a+i})$
238:     $M \leftarrow (M_1 \| M_2 \| \cdots \| M_m)$
239:     $T' \leftarrow \mathsf{msb}_\tau(S_{a+m+1}) \oplus_s \mathsf{lsb}_s(V)$
240:     return $(M, T')$

241: function $\textsc{pad}_x(X)$
242:     if $|X| \bmod x = 0$ then return $X$
243:     return $X \| 1 \| 0^{x-(|X| \bmod x)-1}$

251: function $\textsc{lsb}_x(X)$
252:     if $|X| \le x$ then return $X$
253:     return $X[(|X| - x - 1)..0]$

261: function $\textsc{msb}_x(X)$
262:     if $|X| \le x$ then return $X$
263:     return $X[(|X| - 1)..(|X| - x)]$
```

of r bits, where inputs are XORed to, and a capacity 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. We assume that the key size is at most the capacity, $k \le c$, and the tag size is at most $\tau \le r$ bits.

4.1 Initialization

Each variant of Oribatida uses a fixed-size nonce N, whose length ν is such that $k + \nu = n$ bits. N is concatenated with the key K to initialize the state: $N \| K$: $(X_0, Y_0) \leftarrow (N \| K) \oplus_d \langle d_N \rangle_d$. The domain d_N is XORed to the d least significant bits of the initial state. The first value S_1 results from $S_1 \leftarrow P(U_0 \| V_0)$; V_1 is stored for masking the first block of ciphertext later.

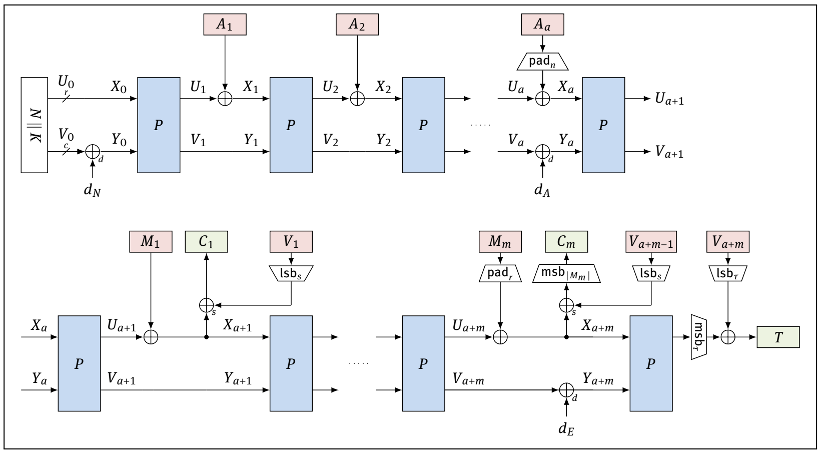


Figure 3: Authenticated encryption of a-block associated data A and m-block message M with Oribatida.

4.2 Processing Associated Data

After the initialization, the associated data A is split into r-bit blocks and is absorbed in the rate. If its length is not a multiple of r bits, A is padded with a 10^*-padding if $|A| \bmod r \not\equiv 0$ such that its length becomes the next highest multiple of r bits. If the associated data is empty, it is padded to one full block 10^{r-1}. In this case, we denote the length of the padded associated data A in blocks also as $a = 1$. The padded A is split into r-bit blocks $(A_1, \cdots, A_a)$. Given $(U_i, V_i) \overset{r,c}{\longleftarrow} S_i$, A_i is XORed to the rate 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 of the permutation output, V_i, is simply forwarded to that of the subsequent input to the permutation P: $Y_i \leftarrow V_i$. The state is updated with P afterwards, for all $1 < i < a$ (that is, except the final a-th block of A): $S_{i+1} \leftarrow P(X_i \| Y_i)$. When the final block A_a is processed, a domain d_A is XORed to the least significant byte of the capacity.

4.3 Encryption

After A has been processed, the message M is encrypted. Similarly as for the associated data, if its length 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. An empty message $M = \varepsilon$ will not be padded.

After M is split into r-bit blocks $(M_1, \cdots, M_m)$ (after padding if necessary), the blocks M_i are processed one after the other. Given the state value $(U_{a+i}, V_{a+i}) \overset{r,c}{\longleftarrow} S_{a+i}$, the current block M_i is XORed to the rate U_{a+i}: $X_{a+i} \leftarrow M_i \oplus U_{a+i}$. The capacity 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 mask value from the capacity. The first ciphertext block is computed from $C_1 \leftarrow X_{a+i} \oplus_s \mathsf{lsb}_s(V_1)$. If C_1 is the final ciphertext block, it is computed as $C_1 \leftarrow \mathsf{msb}_{\ell_E}(X_{a+i} \oplus_s \mathsf{lsb}_s(V_1))$, where ℓ_E denotes the length of M before padding. Non-final ciphertext blocks C_i, $1 < i < m$ are computed from $C_i \leftarrow X_{a+i} \oplus_s \mathsf{lsb}_s(V_{a+i-1})$, for $1 < i < m$. If $m > 1$, the final ciphertext block results from $C_m \leftarrow \mathtt{msb}_{\ell_E \bmod r}(X_{a+m} \oplus_s \mathtt{lsb}_s(V_{a+m-1}))$. For the final message block, a domain d_E is XORed to the least significant byte of the capacity: $Y_{a+m} \leftarrow V_{a+m} \oplus_d \langle d_E \rangle_d$. Similar as for A, d_E uses three pairwise distinct domains depending on whether M was empty, non-empty and required no padding, or non-empty and has been padded. P is called another time to derive $S_{a+m+1} \leftarrow P(X_{a+m} \| Y_{a+m})$. Its rate is XORed with the most significant τ bits of the key V_{a+m}, and – truncated to τ bits if necessary – is released as the authentication tag: $T \leftarrow \mathtt{msb}_\tau(S_{a+m+1}) \oplus_s \mathtt{lsb}_s(V_{a+m})$. Note that, for $s = \tau$ as for our instantiations, the tag is masked as the ciphertext output blocks, which unifies this process.

4.4 Decryption

The decryption takes a tuple (K, N, A, C, T). 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 $X_{a+i} \leftarrow C_i \oplus_s \mathtt{lsb}_s(V)$ and $M_i \leftarrow (U_{a+i} \oplus X_{a+i})$, where $V = V_1$ for $i = 1$, and $V = V_{a+i-1}$ otherwise. The capacity is simply forwarded to the next call of the permutation: $Y_{a+i} \leftarrow V_{a+i}$. The subsequent state is then $(U_{a+i+1} \| V_{a+i+1}) \leftarrow S_{a+i+1} \leftarrow P(X_{a+i} \| Y_{a+i})$.

The final plaintext block is computed from the padded ciphertext block C_m as $X_{a+m} \leftarrow C_m \oplus_s \mathtt{lsb}_s(V)$ and $M_m \leftarrow \mathtt{lsb}_x(U_{a+m} \oplus X_{a+m})$, where $x \leftarrow \ell_E \bmod r$. For the final block, the domain d_E is XORed to the least significant byte of the capacity: $Y_{a+m} \leftarrow V_{a+m} \oplus_d \langle d_E \rangle_d$. The would-be tag T' is derived by computing $(T' \| Z) \leftarrow P(X_{a+m} \| Y_{a+m}) \oplus_s \mathtt{lsb}_s(V_{a+m})$, and using only its most significant τ bits: $T' \leftarrow \mathtt{msb}_\tau(T' \| Z)$ 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.

4.5 Domain Separation

For domain separation, Oribatida defines constants d_N, d_A and d_E. The domains are XORed with the least significant byte of the state at three stages. Domains are encoded as d-bit strings, where $d = 4$ bits suffice in practice. The value depends on the presence of A and M and whether their final blocks are absent, partial, or integral to prevent trivial collisions of inputs to P among blocks of A and M. The constants are determined by four bits (t_3, t_2, t_1, t_0) that reflect inputs in the hardware API, similar to, e.g., [25]:

- **EOI:** t_3 is the **end-of-input** control bit. This bit is set to 1 if the current data block is the final block of the input. Note that if the associated data is empty, then the created 10^{r-1}-block is never treated as the final block of the input.
- **EOT:** 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, then neither the nonce nor the created 10^{r-1}-block of the associated data is considered as the final block of its type.
- **Partial:** t_1 is the **partial-control** bit. This bit is set to 1 if the size of the current block is less than the block size. Note that if the associated data is empty and the message is non-empty, then the created 10^{r-1}-block is treated as a partial block.
- **Type:** t_0 is the **type-control** bit, identifying the type of the current block. For the nonce and the final message block, $t_0 = 1$. If the associated data is empty and the message is non-empty, then $t_0 = 1$ for the created 10^{r-1}-block of the associated data. For all other cases, $t_0 = 0$.

While processing a data block, the domains are set as the integer representation of $t_3 \| t_2 \| t_1 \| t_0$. For example, processing the nonce (which is always a complete r-bit block) with empty associated data and non-empty message yields $d_N = (t_3 t_2 t_1 t_0) = (0101)_2 = 5$. Details are provided in Algorithm 2; ℓ_A denotes the length of A and ℓ_E that of M in bits before padding. An overview is given in Table 3.

Algorithm 2 Instantiated Domains.

```
11: function GetDomainForN($\ell_A, \ell_E$)
12:     if $\ell_A = 0 \wedge \ell_E = 0$ then return $\langle 9 \rangle_d$
13:     return $\langle 5 \rangle_d$

21: function GetDomainForA($\ell_A, \ell_E$)
22:     if $\ell_A = 0 \wedge \ell_E = 0$ then return $\langle 0 \rangle_d$
23:     if $\ell_A = 0 \wedge \ell_E > 0$ then return $\langle 7 \rangle_d$
24:     if $\ell_E > 0 \wedge \ell_A \bmod r \equiv 0$ then return $\langle 4 \rangle_d$
25:     if $\ell_E > 0 \wedge \ell_A \bmod r \not\equiv 0$ then return $\langle 6 \rangle_d$
26:     if $\ell_E = 0 \wedge \ell_A \bmod r \equiv 0$ then return $\langle 12 \rangle_d$
27:     if $\ell_E = 0 \wedge \ell_A \bmod r \not\equiv 0$ then return $\langle 14 \rangle_d$

31: function GetDomainForE($\ell_E$)
32:     if $\ell_E \bmod r \equiv 0$ then return $\langle 13 \rangle_d$
33:     if $\ell_E \bmod r \not\equiv 0$ then return $\langle 15 \rangle_d$
```

Table 3: Instantiated domains in Oribatida. • = yes, – = no.

$\lvert A\rvert$		$\lvert M\rvert$		Domains		
> 0	$\bmod r \equiv 0$	> 0	$\bmod r \equiv 0$	d_N	d_A	d_E
–	–	–	–	$\langle 9\rangle_d$	$\langle 0\rangle_d$	–
–	–	•	–	$\langle 5\rangle_d$	$\langle 7\rangle_d$	$\langle 15\rangle_d$
–	–	•	•	$\langle 5\rangle_d$	$\langle 7\rangle_d$	$\langle 13\rangle_d$
•	–	–	–	$\langle 5\rangle_d$	$\langle 14\rangle_d$	–
•	–	•	–	$\langle 5\rangle_d$	$\langle 6\rangle_d$	$\langle 15\rangle_d$
•	–	•	•	$\langle 5\rangle_d$	$\langle 6\rangle_d$	$\langle 13\rangle_d$
•	•	–	–	$\langle 5\rangle_d$	$\langle 12\rangle_d$	–
•	•	•	–	$\langle 5\rangle_d$	$\langle 4\rangle_d$	$\langle 15\rangle_d$
•	•	•	•	$\langle 5\rangle_d$	$\langle 4\rangle_d$	$\langle 13\rangle_d$

5 Int-RUP Attacks on Schemes with Masked Ciphertexts

The approach of Oribatida is to employ (a portion of) the capacity of the previous permutation output to mask the ciphertext outputs. This strategy is generic enough to also apply it to other modes, such as Beetle or SPoC. We can informally define a masked variant of Beetle and SPoC. The masked beetle uses Z_{i-1} and XORs $\mathsf{lsb}_s(Z_{i-1})$ to the s rightmost bits of the ciphertext block C_i, for $i > 1$ and $s \le c$. If there is no associated data present, we define $Z_0 = K_2$. A masked variant of SPoC would employ the rate (since it is the hidden part). Thus, for the masked SPoC, we define $s \le r$ and define that $\mathsf{lsb}_s(U_{i-1})$ is XORed to C_i for $i > 0$. If no associated data is present, we define U_0 for the rate of the initial input to the permutation P.

For Oribatida, the masked Beetle or the masked SPoC, the attacks in Section 3 do not apply directly. Though, there exist attacks on each of them with complexity $O(q_d^2/2^c)$.

5.1 The Generic Int-RUP Attack on Oribatida (Masked Duplex)

Here, we consider an attack on Oribatida that shows that our Int-RUP bound of $O(q_d^2/2^c)$ will be tight, so the attack will be successful for $q_d^2 \approx O(2^c)$:

1. **A** asks q_d decryption queries (N^i, A^i, C^i, T^i), where N^i and C^i is static for all queries. We assume that the associated data A^i are pairwise distinct and consist of a single block for all queries. **A** obtains M^i from the encryption oracle, for $1 \le i \le q_d$. The rate $X_2^i \leftarrow C_1^i \oplus_s \mathsf{lsb}_s(V_1^i)$ is constant for all queries.
2. For $q_d \approx O(2^{c/2})$, **A** can expect a collision in the capacity of the input: $V_2^i \leftarrow V_2^j$ for some distinct $i \neq j$, $i, j \in [1..q_d]$. Then, this collision leads to a full-state collision that can be detected when $M_1^i = M_1^j$.
3. **A** asks encryption queries (N, A^i, M^i) and obtains (C^i, T) for some tag T.
4. (N, A^j, C^i, T) is a valid forgery and yields M^j.

5.2 Int-RUP Attack on The Masked Beetle

1. The adversary **A** asks q_d encryption queries $(N^1, A^1, M), (N^2, A^2, M), \ldots, (N^{q_d}, A^{q_d}, M)$ to the encryption oracle, and receives $C^1, C^2, \ldots, C^{q_d}$. The associated data A^i consist of a single block for each i; the message M contains $\lceil \frac{c}{r} \rceil$ blocks.
2. **A** asks $q_d - 1$ decryption queries, one for each encryption query except the first encryption query, to the decryption oracle. The decryption query of the i-th encryption query is $(N^i, A^i, C^{i'})$, where $C^{i'}$ =

$Y_2^1 \oplus Y_2^i \oplus \mathsf{shuffle}(Y_2^1) \oplus \mathsf{shuffle}(Y_2^i)$. **A** can calculate the r.h.s. as $\mathsf{shuffle}(Y_2^1) \oplus \mathsf{shuffle}(Y_2^i) = C^1 \oplus C^i$, and $Y_2^1 \oplus Y_2^i$ directly from $\mathsf{shuffle}(Y_2^1) \oplus \mathsf{shuffle}(Y_2^i)$ with the definition of shuffle. So, the first r bits of the input to the third permutation call always equal those of the first encryption query.
3. Afterwards, **A** repeats Step 2 to 6 from Section 5.1 to complete the attack.

The attack is successful for $q_d^2 \approx O(2^c)$. However, the attack strategy differs for SPoC.

5.3 Int-RUP Attack on The Masked SPoC

Here, **A** has to perform the attack in two stages.

1. First, **A** asks q_d decryption queries $(N, A^1, C), (N, A^2, C), \ldots, (N, A^{q_d}, C)$ to the decryption oracle, and receives $M^1, M^2, \ldots, M^{q_d}$. The associated data A^i consists of a single block for each i; the ciphertext C consists of two blocks.
2. When $q_d \approx \mathcal{O}(2^r)$, **A** expects a collision in the first r bits of the input to the third permutation call. **A** can detect this collision by looking at the first message block because it will be equal only for the two colliding queries.
3. Suppose the associated data of the two colliding queries are A^i and A^j.
4. **A** makes q_1 queries $(N, A^i, C^1), (N, A^i, C^2), \ldots, (N, A^i, C^{q_1})$, and q_2 queries $(N, A^j, C^1), (N, A^j, C^2), \ldots, (N, A^j, C^{q_2})$ to the decryption oracle.
5. When $q_1 \cdot q_2 \approx \mathcal{O}(2^r)$, **A** expects a full state collision at the input to the third permutation, between one query with associated data A^i and another query with associated data A^j.
6. Suppose the two ciphertexts corresponding to the two colliding queries are C^p and C^q, and the corresponding messages are M^p and M^q.
7. **A** identifies those pairs of queries $(N, A^i, C^x), (N, A^j, C^y)$, $1 \le x \le q_1$ and $1 \le y \le q_2$, for which the sum of the second message blocks equals that of the first message blocks. For each such pair, **A** updates C^x and C^y by appending $\lceil \frac{c}{r} \rceil - 1$ ciphertext blocks to each s.t., $C_2^x = C_2^y, \ldots, C^x_{\lceil \frac{c}{r} \rceil} = C^y_{\lceil \frac{c}{r} \rceil}$, and makes decryption queries with the updated ciphertexts. For $k > 2$, M_k^p will equal M_k^q.
8. Next, **A** asks (N, A^i, M^p) to the encryption oracle; suppose, the tag is T.
9. Then, **A** successfully forges with the query (N, A^j, C^q, T).

Again, the probability for forgeries becomes non-negligible when $q_d^2 \approx O(2^c)$.

6 nAE Security Analysis

This section analyzes the nAE security of Oribatida. In the following, let $K \twoheadleftarrow \mathcal{K}$ and $\pi \twoheadleftarrow \mathsf{Perm}(\mathcal{B})$. We use $\Pi[\pi, \pi]_K = \Pi[\pi]_K$ as short form of Oribatida, instantiated with π for P *and* for P', and keyed by K. Let **A** be a nonce-respecting nAE adversary w.r.t. $\Pi[\pi]_K$. We denote by $q_p, q_f, q_b, q_c, q_e, q_d, \sigma_c, \sigma_e, \sigma_d$ the number of primitive queries, forward primitive queries, backward primitive queries, construction queries, encryption queries, decryption queries, blocks summed over all construction queries, blocks summed over only all encryption queries, and blocks summed over all decryption queries, respectively. It holds that $q_p = q_f + q_b$, $q_c = q_e + q_d$, and $\sigma_c = \sigma_e + \sigma_d$. For simplicity, we define a function ρ as

$$\rho(i, j) \overset{\text{def}}{=} \begin{cases} 1 & \text{if } j = 1 \\ a^i + j - 1 & \text{otherwise}. \end{cases}$$

So, $V^i_{\rho(i,j)}$ denotes the used block for masking the ciphertext block C^i_j.

Recall the notion of a longest common prefix from [11]. Let $Q = (N, A, M, C, T)$ be a query of **A** with the response. Let $\mathcal{Q}$ denote a set of queries without Q, i.e., $Q \notin \mathcal{Q}$. We define the length of the longest common prefix of M and another message M' as $\mathsf{LCP}(M, M') \overset{\text{def}}{=} \max_i \left\{1 \le j \le i : M_j = M'_j\right\}$. Given $\mathcal{Q}$ and M, we overload the notation by considering the longest common prefix of M with the queries in $Q' = (N', A', M', C', T') \in \mathcal{Q}$: $\mathsf{LCP}(M, \mathcal{Q}) \overset{\text{def}}{=} \max_{M' \in \mathcal{Q}} \{\mathsf{LCP}(M, M')\}$. We also define

$$\mathsf{LCP}^{N,A}(M, \mathcal{Q}) \overset{\text{def}}{=} \max_{\substack{(N',A',M',C',T') \in \mathcal{Q} \\ N'=N \wedge A'=A}} \{\mathsf{LCP}(M, M')\} .$$

Collisions of chaining values are trivial if in the longest common prefix and non-trivial otherwise.

Theorem 1 (NAE Security of Oribatida)**.** Let **A** be a nonce-respecting adversary w.r.t. $\Pi[\pi]_K$. Then, $\mathbf{Adv}^{\mathrm{NAE}}_{\Pi[\pi]_K}(\mathbf{A})$ is upper bounded by

$$\frac{\binom{\sigma}{r} + 2\binom{q_p}{r}}{2^{r(r-1)}} + \frac{\sigma^2}{2^n} + \frac{3q_p}{2^k} + \frac{r(q_d + \sigma_d) + 2\sigma_e q_p + q_p q_c + q_d(\sigma_e + q_p) + 2rq_p}{2^{c+s}} + \frac{q_d}{2^\tau} .$$

Proof. We follow the strategy of the NAE proof of Beetle [25]. The queries by **A** and their corresponding answers are collected in a transcript $\tau = (\tau_e, \tau_d, \tau_p)$. In that transcript, the encryption construction queries are stored as tuples $\tau_e = \{(N^i, A^i, M^i, C^i, T^i)\}$, for $1 \le i \le q_e$, the decryption construction queries are stored as tuples $\tau_d = \{(N^i, A^i, M^i, C^i, T^i)\}$, for $1 \le i \le q_d$, and primitive queries are stored as tuples $\tau_p = \{(Q^i, R^i)\}$, where $\pi(Q^i) = R^i$, for $1 \le i \le q_p$.

Sampling.

We define the ideal oracle to consist of an on-line and an off-line phase. In the on-line phase, the ideal oracle samples the responses (C^i, T^i) uniformly at random from the bit strings of expected lengths for encryption queries. For decryption queries, it always outputs $\perp$. For forward primitive queries Q^i, it forwards the result of $\pi(Q^i)$ to **A**; for backward primitive queries R^i, it forwards the result of $\pi^{-1}(R^i)$.

In the off-line phase, the ideal oracle samples the internal chaining values $V^i_j \twoheadleftarrow \{0,1\}^c$ and $U^i_j \twoheadleftarrow \{0,1\}^r$ uniformly at random for all construction queries in encryption direction. It derives the analogous internal chaining values V^i_j, for $1 \le j \le k$ for all construction queries in decryption direction (N^i, A^i, C^i, T^i) that share $N^i, A^i = N^{i'}, A^{i'}$, and for which $C^i_1 = C^{i'}_1, \ldots, C^i_k = C^{i'}_k$ holds for some i'-th construction query, where $i \ne i'$. Moreover, for construction queries whose plaintext or ciphertext length is not a multiple of r bits, the oracle samples exactly the missing bits C^i_m uniformly independently at random that are not fixed from previous queries, at most at most $r - |C^i_{m^i}|$ bits at a time. The so-sampled values for the final blocks $C^i_{m^i}$ are stored also in the transcript. Moreover, the random key K is revealed to **A** after the off-line phase.

Bad Events.

We define the following bad events. If any of them occurs, the adversary aborts, and we define that it wins in this case.

- bad_1: Multi-collision on the rate X in encryption construction queries. For some $w \ge r$, $\exists$ indices $(i_1, j_1), (i_2, j_2), \ldots, (i_w, j_w)$ with $i_1, i_2, \ldots, i_w \in [1..q_e]$, and $j_1 \in [1..a^{i_1}]$, $j_2 \in [1..a^{i_2}]$, etc., s. t. $X^{i_1}_{j_1} = X^{i_2}_{j_2} = \ldots = X^{i_w}_{j_w}$.
- bad_2: Collision of permutation inputs in encryption construction queries: $\exists$ indices $(i, j) \ne (i', j')$ with $i, i' \in [1..q_e]$, $j \in [1..m^i]$, and $j' \in [1..m^{i'}]$ s. t. $\left(X^i_j \parallel Y^i_j\right) = \left(X^{i'}_{j'} \parallel Y^{i'}_{j'}\right)$.
- bad_3: Collision of permutation outputs in encryption construction queries: $\exists$ indices $(i, j) \ne (i', j')$ with $i, i' \in [1..q_e]$, $j \in [1..m^i]$, and $j' \in [1..m^{i'}]$ s. t. $\left(U^i_j \parallel V^i_j\right) = \left(U^{i'}_{j'} \parallel V^{i'}_{j'}\right)$.
- bad_4: Collision of permutation inputs between a construction and a primitive query: $\exists$ indices (i, j, i') with $i \in [1..q_e]$, $j \in [1..m^i]$, and $i' \in [1..q_p]$ s. t. $\left(X^i_j \parallel Y^i_j\right) = Q^{i'}$.
- bad_5: Collision of permutation outputs between a construction and a primitive query: $\exists$ indices (i, j, i') with $i \in [1..q_e]$, $j \in [1..m^i]$, and $i' \in [1..q_p]$ s. t. $\left(U^i_j \parallel V^i_j\right) = R^{i'}$.
- bad_6: Initial-state collision with a primitive query: $\exists$ indices (i, i') with $i \in [1..q_e]$ and $i' \in [1..q_p]$ s. t. $\left(X^i_0 \parallel Y^i_0\right) = Q^{i'}$.

- bad_7: Multi-collision in the rate of w outputs of forward primitive queries: for some $w \geq r$, $\exists\, i_1, i_2, \ldots, i_w \in [1..q_p]$ s. t. $\mathsf{msb}_r(R^{i_1}) = \mathsf{msb}_r(R^{i_2}) = \cdots = \mathsf{msb}_r(R^{i_w})$.
- bad_8: Multi-collision in the rate of w outputs of backward primitive queries: for some $w \geq r$, $\exists\, i_1, i_2, \ldots, i_w \in [1..q_p]$ s. t. $\mathsf{msb}_r(Q^{i_1}) = \mathsf{msb}_r(Q^{i_2}) = \cdots = \mathsf{msb}_r(Q^{i_w})$.

We define that the adversary is provided with all internal chaining values V_j^i and U_j^i after its interaction, but before it outputs its decision bit, which only strengthens the adversary. We define the set of bad transcripts $\textsc{BadT}$, to contain exactly those attainable transcripts τ for which at least one of the bad events occurred. It holds that $\Pr[\Theta_{\text{ideal}} \in \textsc{BadT}] \leq \sum_{i=1}^{8} \Pr[\mathsf{bad}_i]$. The probability of bad transcripts in the ideal world is treated in the proof of Lemma 2. The ratio of good transcripts is bounded in Lemma 3. Our bound in Theorem 1 follows from them and the fundamental Lemma of the H-coefficient Technique [54], using $w = r$ in the bounds.

Lemma 2. Let $w \geq r$ be a positive integer. It holds that

$$\Pr[\Theta_{\text{ideal}} \in \textsc{BadT}] \leq \frac{\binom{\sigma}{w} + 2\binom{q_p}{w}}{2^{r(w-1)}} + \frac{\sigma^2}{2^n} + \frac{3q_p}{2^k} + \frac{2\sigma_e q_p + q_p q_c + 2w \cdot q_p}{2^{c+s}} .$$

Proof. In the following, we upper bound the probabilities of the individual bad events.

bad_1: Multi-collision on X in encryption construction queries.
In the ideal world, the ciphertext blocks are sampled independently and uniformly at random from the strings of expected length. The internal values X_j^i can be computed by $\mathbf{A}$ once it is given the transcript including the internal chaining values V_j^i. It must hold that $X_j^i \leftarrow C_{j-a^i}^i \oplus_s \mathsf{lsb}_s(V_{\rho(i,j-a^i)}^i)$. The random sampling of C implies that the probability of the values X_j^i to take any specific r-bit value is $1/2^r$. Note that in the case of a padded ciphertext block, each padded bit of $C_{m^i}^i$ is also sampled once randomly and given in the transcript. Hence, the probability for $X_{a^i+m^i}^i$ to take any r-bit value is also 2^{-r} in the ideal world. For fixed indices $(i_1, j_1), (i_2, j_2), \ldots, (i_w, j_w)$, it holds that

$$\Pr\left[X_{j_1}^{i_1} = X_{j_2}^{i_2} = \ldots = X_{j_w}^{i_w}\right] \leq 2^{-r(w-1)} .$$

Over all queries and blocks of τ_e, it follows that

$$\Pr[\mathsf{bad}_1] \leq \frac{\binom{\sigma}{w}}{2^{r(w-1)}} .$$

bad_2: Collision of two permutation inputs in encryption construction queries.
Here, we consider

$$\Pr\left[\left(X_j^i \,\|\, Y_j^i\right) = \left(X_{j'}^{i'} \,\|\, Y_{j'}^{i'}\right)\right] .$$

All ciphertext blocks and the internal chaining values $V_{\rho(i,j-a^i)}^i$, $j > a^i$ are sampled independently and uniformly at random. Moreover, padded bits of ciphertexts are sampled also independently and uniformly at random. Though, we have to consider two cases;

- $j = 0 \wedge j' = 0$: since X_0^i and $X_0^{i'}$ contain nonces, and since we assume $\mathbf{A}$ to be nonce-respecting, the probability for a collision is zero in this case.
- $j > 0$: In this case, $Y_j^i = V_j^i \oplus \text{const}$, where $\text{const} \in \{d_N, d_A, d_E\}$ is a public constant. Moreover, X_j^i is derived from C_j^i; so, both X_j^i and Y_j^i are chosen independently and uniformly at random, and the probability for a collision in this case is at most 2^{-n}.

Therefore, for fixed indices $(i, j) \neq (i', j')$, the probability is

$$\Pr\left[\left(X_j^i \,\|\, Y_j^i\right) = \left(X_{j'}^{i'} \,\|\, Y_{j'}^{i'}\right)\right] \leq 2^{-n} .$$

Over all combinations of indices, it follows that

$$\Pr[\mathsf{bad}_2] \leq \frac{\binom{\sigma}{2}}{2^n}.$$

bad_3: Collision of two permutation outputs in encryption construction queries.
This case is analogous to bad_2. The permutation outputs V_j^i are sampled randomly. In all cases, it holds that

$$\Pr\left[\left(U_j^i \parallel V_j^i\right) = \left(U_{j'}^{i'} \parallel V_{j'}^{i'}\right)\right] \leq 2^{-n}.$$

Over all combinations of indices, it follows that

$$\Pr[\mathsf{bad}_3] \leq \frac{\binom{\sigma}{2}}{2^n}.$$

bad_4: Collision of permutation inputs between a construction and a primitive query.
Again, we consider $X_j^i \leftarrow C_{j-a^i}^i \oplus_s \mathsf{lsb}_s(V_{\rho(i,j-a^i)}^i)$ and $Y_j^i \leftarrow V_j^i \oplus \mathrm{const}$, where const is a public constant. The values $C_{j-a^i}^i$ and $V_{\rho(i,j-a^i)}^i$ are sampled randomly, the values $(X_j^i \parallel Y_j^i)$ take any value with probability at most 2^{-n}.

1. Assume, the primitive query was asked before the construction query. If the construction query was in encryption direction, the collision probability for fixed queries is at most 2^{-n}, for $q_p \cdot q_c$ combinations.
2. If the primitive query was asked after an encryption query, then, the latter one produced a tag. If the primitive query starts at any other block, $\mathbf{A}$ can see $r - s$ bits. Hence, the probability is at most $2^{-(c+s)}$ for $q_p \cdot \sigma_e$ combinations. If the primitive query starts from the tag, the adversary sees $c + s$ unmasked bits. Assuming $\overline{\mathsf{bad}_1}$, there are at most w equal tags over all encryption queries. So, the probability for a collision is $2^{-(n-\tau)}$, for at most $w \cdot q_p$ combinations.

Over all combinations of indices, it follows that

$$\Pr\left[\mathsf{bad}_4 | \overline{\mathsf{bad}_1}\right] \leq \max\left(\frac{\sigma_e \cdot q_p}{2^n}, \frac{\sigma_e \cdot q_p}{2^{c+s}}\right) + \frac{w \cdot q_p}{2^{c+s}} \leq \frac{\sigma_e \cdot q_p}{2^{c+s}} + \frac{w \cdot q_p}{2^{c+s}}.$$

bad_5: Collision of permutation outputs between an encryption construction query and a primitive query.
Again, $U_{j+a^i}^i$ can be derived from $M_j^i \oplus C_j^i \oplus_s \mathsf{lsb}_s(V_{\rho(i,j)}^i)$ and the values C_j^i and $V_{\rho(i,j)}^i$ are sampled randomly. So, the values $U_{j+a^i}^i \parallel V_{j+a^i}^i$ take any value with probability at most 2^{-n}. If the primitive query starts at any other block, $\mathbf{A}$ can see $r - s$ bits. Hence, the probability is at most $2^{-(c+s)}$ for $q_p \cdot \sigma_e$ combinations. Following a similar argument as for bounding bad_4 and excluding bad_1, we obtain over all combinations of indices that

$$\Pr\left[\mathsf{bad}_5 | \overline{\mathsf{bad}_1}\right] \leq \frac{\sigma \cdot q_p}{2^{c+s}} + \frac{w \cdot q_p}{2^{c+s}}.$$

bad_6: Initial-state collision with a primitive query.
Here, we know that the key is chosen uniformly at random. We distinguish between collisions depending on whether the primitive query was a forward query or a backward query.

1. If the primitive query was a forward query, it must hit the correct value of $K \oplus_d d_N$. So, the probability is at most $q_p/2^k$ to collide with encryption construction queries. Considering also decryption queries, a nonce can repeat but change d_N. Since there exist at most three distinct values for d_N, the probability is at most $3q_p/2^k$ to collide.
2. If the primitive query was in backward direction, its response must hit any initial state of a construction query. If the construction query was asked before the primitive, $\mathbf{A}$ sees at best $r - s$ bits of C_1. Then, the probability is at most $q_c \cdot q_p/2^{c+s}$.

3. If the primitive query was asked before the construction query, **A** can use the nonce part of the primitive query's result as a nonce. Though, a collision needs the key part to be correct, which holds with probability at most $3q_p/2^k$.

Over all possible options, we obtain

$$\Pr[\mathsf{bad}_6] \le \frac{3q_p}{2^k} + \frac{q_c \cdot q_p}{2^{c+s}} .$$

bad_7: Multi-collision in the rate of w outputs of forward primitive queries.
Since π is chosen randomly from the set of all permutations, the outputs are chosen randomly from a set of size $2^n - (i-1)$ for the i-th primitive query. So, the probability for w distinct queries to collide in their rate is at most $1/2^{r(w-1)}$ as for bad_7 in the NAE proof. Over all queries, the probability is upper bounded by

$$\Pr[\mathsf{bad}_7] \le \frac{\binom{q_p}{w}}{2^{r(w-1)}} .$$

bad_8: Multi-collision in the rate of w outputs of backward primitive queries.
Following a similar argumentation as for bad_7, we obtain

$$\Pr[\mathsf{bad}_8] \le \frac{\binom{q_p}{w}}{2^{r(w-1)}} .$$

Our bound in Lemma 2 follows from summing up all probabilities.

Lemma 3. Let $\tau \in \text{GoodT}$. Then

$$\frac{\Pr[\Theta_{\text{real}} = \tau]}{\Pr[\Theta_{\text{ideal}} = \tau]} \le 1 - \left(\frac{q_d}{2^\tau} + \frac{q_d(q_p + \sigma_e)}{2^{c+s}} + \frac{(\sigma_d + q_d) \cdot w}{2^{c+s}} \right) .$$

Proof. It remains to lower bound the ratio of real and ideal probability of obtaining a good transcript τ. Let $\tau = (\tau_e, \tau_d, \tau_p)$ be an attainable transcript, where $\tau_d = \bot_{\text{all}}$ contains only $\bot$ for all responses. Since all ciphertext-block outputs and all internal chaining values in encryption queries are sampled independently and uniformly at random, their probability is $1/2$ per bit. We define σ_{distinct} for the number of distinct calls to the permutation over all encryption and decryption queries. In the ideal world, it holds that

$$\begin{aligned}\Pr[\Theta_{\text{ideal}} = \tau] &= \Pr[K] \cdot \Pr[\tau_e \wedge \tau_p \wedge \tau_d] \\ &= \Pr[K] \cdot \Pr[\tau_e] \cdot \Pr[\tau_p] \cdot \Pr[\tau_d] = \frac{1}{2^k} \cdot \frac{1}{(2^n)^{\sigma_{\text{distinct}}}} \cdot \frac{1}{(2^n)_{q_p}} \cdot 1\end{aligned}$$

since the outputs from encryption queries are sampled uniformly at random; so, the encryption and decryption transcripts τ_e and τ_d are independent from τ_p.

In the real world, the probabilities for choosing K as key and π as permutation are equal to those of the ideal world. We can separate the probability into

$$\Pr[\Theta_{\text{real}} = \tau] = \Pr[K] \cdot \Pr[\tau_e \wedge \tau_p \wedge \tau_d] = \frac{1}{2^k} \cdot \Pr[\tau_e, \tau_d | \tau_p] \cdot \Pr[\tau_p]$$

since the encryption and the decryption transcript depend on the choice of the permutation π. Let $\top_i$ denote that the i-th decryption query was a valid forgery. We can upper bound

$$\begin{aligned}\Pr[\tau_e, \tau_d | \tau_p] \cdot \Pr[\tau_p] &\le \Pr[\tau_e | \tau_p] \cdot \Pr[\tau_p] - \left(\sum_{i=1}^{q_d} \Pr[\tau_e \wedge \top_i | \tau_p] \cdot \Pr[\tau_p] \right) \\ &= \Pr[\tau_e | \tau_p] \cdot \Pr[\tau_p] - \Pr[\tau_e | \tau_p] \cdot \Pr[\tau_p] \cdot \left(\sum_{i=1}^{q_d} \Pr[\top_i | \tau_e | \tau_p] \cdot \Pr[\tau_p] \right)\end{aligned}$$

$$= (\Pr[\tau_e|\tau_p] \cdot \Pr[\tau_p]) \cdot (1 - \epsilon) \ , \tag{1}$$

where we define

$$\epsilon \overset{\text{def}}{=} \sum_{i=1}^{q_d} \epsilon_i \qquad \text{and} \qquad \epsilon_i \overset{\text{def}}{=} \Pr[\top_i|\tau_e|\tau_p] \cdot \Pr[\tau_p] \ .$$

The probability of primitive queries is given by the fraction of all permutations π that would produce τ_p, which is

$$\Pr[\tau_p] = \frac{1}{(2^n)_{q_p}} \ ,$$

as in the ideal world. The ciphertext blocks C^i_j from encryption queries as well as the chaining values V^i_j are results from the permutation π and hence, depend on the permutation. Since τ is a good transcript, there are no undesired collisions, e.g., between primitive and construction queries. Hence, all internally computed values $(U^i_j \parallel V^i_j)$ – note that U^i_j can be derived from $C^i_{j-a^i} \oplus_s \mathsf{lsb}_s(V^i_{\rho(i,j-a^i)}) \oplus M^i_{j-a^i}$ by the adversary – are results of fresh values or predefined in decryption queries from the result of previous encryption queries. Then, the probabilities for the outputs of π in construction queries are given by $1/\left(2^n\right)_{\sigma_{\text{distinct}}}$. It is not difficult to see that for positive σ_{distinct}, the ratio of the interpolation probabilities from Equation (1) can be bounded by

$$\frac{(\Pr[\tau_e|\tau_p|\Theta_{\text{real}}] \cdot \Pr[\tau_p|\Theta_{\text{real}}])}{(\Pr[\tau_e|\Theta_{\text{ideal}}])} = \frac{\frac{1}{(2^n)_{\sigma_{\text{distinct}}}}}{\frac{1}{(2^n)^{\sigma_{\text{distinct}}}}} = \frac{(2^n)^{\sigma_{\text{distinct}}}}{(2^n)_{\sigma_{\text{distinct}}}} \geq 1 \ .$$

It remains to upper bound ϵ. For this purpose, we upper bound the values ϵ_i for transcripts that contain forgeries. Since τ is a good transcript, we assume that bad events do not hold. Hence, either $\top_i$ does not hold, which yields $\epsilon_i = 0$; in the opposite case, we have to consider a few mutually exclusive cases in the following. We assume that there exists a decryption query (N^i, A^i, C^i, T^i) s. t. T^i is valid. In all cases, the tag can simply be guessed correctly if the block $(X^i_{a^i+m^i} \parallel Y^i_{a^i+m^i})$ is fresh. Then, the probability for the tag to be correct is $2^{-\tau}$. So, we can concentrate on the cases where it is non-fresh in the following. Prior, we define $(X_{i_1}, X_{i_2}, \ldots, X_{i_{w+1}})$ as a w-chain if there exist $(Y_{i_1}, Y_{i_2}, \ldots, Y_{i_{w+1}})$ s. t. the following chain has been obtained from primitive queries:

$$\begin{aligned}
\pi\left(X_{i_1} \parallel Y_{i_1}\right) &= \left(U_{i_2} \parallel V_{i_2}\right) = \left(U_{i_2} \parallel Y_{i_2}\right) \ , \\
\pi\left(X_{i_2} \parallel Y_{i_2}\right) &= \left(U_{i_3} \parallel V_{i_3}\right) = \left(U_{i_3} \parallel Y_{i_3}\right) \ , \\
&\vdots \\
\pi\left(X_{i_w} \parallel Y_{i_w}\right) &= \left(U_{i_{w+1}} \parallel V_{i_{w+1}}\right) = \left(U_{i_{w+1}} \parallel Y_{i_{w+1}}\right) \ .
\end{aligned}$$

The cases are:

- **Case (A):** N^i is fresh; so, there is no earlier construction query $i' \neq i$ s. t. $N^i = N^{i'}$.
- **Case (B):** N^i is old, but (N^i, A^i) is fresh, i.e., there exists no earlier construction query $i' \neq i$ with $(N^i, A^i) = (N^{i'}, A^{i'})$.
- **Case (C):** (N^i, A^i) is old, but (N^i, A^i, C^i) is fresh, i.e., there exists no earlier construction query $i' \neq i$ with $(N^i, A^i, C^i) = (N^{i'}, A^{i'}, C^{i'})$, and no w-chain of primitive queries is hit.
- **Case (D):** (N^i, A^i, C^i) is old; (N^i, A^i, C^i) a prefix of another construction query.
- **Case (E):** (N^i, A^i) is old and there exists a w-chain of primitive queries that is hit.

Clearly, the cases cover all possible options. We assume that no previous bad events occur, in particular, no w-multi-collisions or collisions with the primitive queries occurred.

Case (A).

We excluded bad_6 in this case. The probability that $(N^i \parallel K) \oplus_d d_N$ hits any block $(X^{i'}_j \parallel Y^{i'}_j)$ from another construction query so that the final block is old is at most

$$\Pr\left[\left(\left(N^i \parallel K\right) \oplus_d d_N\right) = \left(X^{i'}_j \parallel Y^{i'}_j\right)\right] \leq \frac{\sigma_e}{2^{c+s}} \ .$$

Case (B).
Let $p \le a^i + m^i$ denote the length of the longest common prefix of the i-th query with all other queries. In Case (B), the probability that any block $(X^i_j \parallel Y^i_j)$ with $j \ge p+1$ matches the permutation input of any other encryption-query block or primitive query can be upper bounded by

$$\Pr\left[\left(X^i_j \parallel Y^i_j\right) = \left(X^{i'}_{j'} \parallel Y^{i'}_{j'}\right)\right] \le \frac{\sigma_e}{2^{c+s}} + \frac{q_p}{2^{c+s}}\,.$$

Case (C).
A similar argument as for Case (B) can be applied in Case (C). The probability that there exists $i' \neq i$, s. t. for some block indices, it holds that (j, j'): $(X^i_j \parallel Y^i_j) = (X^{i'}_{j'} \parallel Y^{i'}_{j'})$ is at most $1/2^{c+s}$. So, it holds that

$$\Pr\left[\left(X^i_j \parallel Y^i_j\right) = \left(X^{i'}_{j'} \parallel Y^{i'}_{j'}\right)\right] \le \frac{\sigma_e}{2^{c+s}} + \frac{q_p}{2^{c+s}}\,.$$

Case (D).
This case needs that $(X^i_{a^i+m^i+1} \parallel Y^i_{a^i+m^i+1})$ matches the permutation input of any other encryption-query block or primitive query. The probability can be upper bounded by

$$\Pr\left[\left(X^i_j \parallel Y^i_j\right) = \left(X^{i'}_{j'} \parallel Y^{i'}_{j'}\right)\right] \le \frac{\sigma_e}{2^{c+s}} + \frac{q_p}{2^{c+s}}\,.$$

Case (E).
Assume that $(X^i_{p+1} \parallel Y^i_{p+1})$ hits a w-chain of primitive queries. Under the assumption that no other bad events occurred, the probability is at most

$$\Pr\left[\left(X^i_{p+1} \parallel Y^i_{p+1}\right) \middle| \bigwedge_{i=1}^{8} \overline{\mathsf{bad}_i}\right] \le \frac{(m^i+1)\cdot w}{2^{c+s}}\,.$$

Over all decryption queries, we obtain

$$\epsilon \le \sum_{i=1}^{q_d}\left(\frac{1}{2^\tau} + \frac{\sigma_e}{2^{c+s}} + \frac{q_p}{2^{c+s}} + \frac{(m^i+1)\cdot w}{2^{c+s}}\right) \le \frac{q_d}{2^\tau} + \frac{q_d(q_p+\sigma_e)}{2^{c+s}} + \frac{(\sigma_d+q_d)\cdot w}{2^{c+s}}\,.$$

Our claim in Lemma 3 follows.

7 Int-RUP Analysis

We use the same notations as in Section 6 but add some. Let q_d and σ_d be the number of decryption queries and blocks over decryption queries, respectively, and q_v and σ_v the analogs for verification queries. We replace $\pi \twoheadleftarrow \mathsf{Perm}(\mathcal{B})$, assume $K \twoheadleftarrow \mathcal{K}$, and denote $\Pi[\pi]_K$ for Oribatida with π and K.

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

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

Proof. The Int-RUP analysis of Oribatida follows a similar strategy as our nAE analysis. However, this time, the adversary has access to three oracles for encryption, decryption and verification. Moreover, the encryption and decryption oracles are the same in both the real *and* the ideal world. Both worlds differ only in the verification oracle. To alleviate the task, we replace the oracles for encryption and decryption $\widetilde{\mathcal{E}}[\pi]_K$, $\widetilde{\mathcal{D}}[\pi]_K$ with a pair of **consistent** pseudo-random oracles $\$_{\widetilde{\mathcal{E}}}[\pi]$ and $\$_{\widetilde{\mathcal{D}}}[\pi]$ (we define our intent of consistency for encryption and decryption in a moment). The advantage between both settings can be upper bounded by $\Delta_{\mathbf{A}_1}\left(\widetilde{\mathcal{E}}[\pi]_K, \widetilde{\mathcal{D}}[\pi]_K, \widetilde{\mathcal{V}}[\pi]_K, \pi^{\pm}; \$_{\widetilde{\mathcal{E}}}, \$_{\widetilde{\mathcal{D}}}, \bot, \pi^{\pm}\right)$. Note that the oracles $\$_{\widetilde{\mathcal{E}}}$ and $\$_{\widetilde{\mathcal{D}}}$ differ from the *independent* random oracles in the stronger RUPAE notion. In the RUPAE notion, they sample **independently** from each other without considering common prefixes between queries, which would be impossible to achieve for an on-line AE scheme. Again, we consider the H-coefficient approach. So, we define several bad events and bad as well good transcripts. If any of the bad events occurs, the adversary aborts and is defined to win. Next, we consider the probability of forgeries under those idealized oracles. So, we can exclude the previous bad events and study the probability of forgeries. Finally, we study the ratio of interpolation probabilities for good transcripts.

Sampling Consistently in the On-line Phase.
This on-line phase contains much from the off-line phase of the NAE analysis. We define the ideal encryption oracle as in the NAE proof: it samples the responses (C^i, T^i) uniformly at random from all bit strings of expected lengths for encryption queries. The ideal decryption oracle, however, must sample plaintext outputs consistently. For this purpose, the ideal encryption oracle has to sample also the internal chaining values $V_j^i \twoheadleftarrow \{0,1\}^c$ and $U_j^i \twoheadleftarrow \{0,1\}^r$ uniformly at random for all construction queries already in the on-line phase. It stores the values of C_j^i, V_j^i, and U_j^i also internally, but does not release U_j^i and V_j^i in this phase.

On each input (N^i, A^i, C^i, T^i), the ideal decryption oracle looks up the length of the longest common prefix of the query $p \leftarrow \mathsf{LCP}^{N^i, A^i}(C^i, \mathcal{Q})$ with all previous queries $\mathcal{Q}$. For all blocks in the common prefix $1 \le j \le p$, it uses the same outputs M_j^i that have been fixed from previous queries. Since the oracle has sampled V_{p+1}^i for the $(p+1)$-th block, it can deduce all bits not fixed from previous query outputs. Assume, $i \neq i'$, $(N^i, A^i) = (N^{i'}, A^{i'})$, and $p = \mathsf{LCP}^{N^i, A^i}(C^i, C^{i'})$ where $p < m^i, m^{i'}$. Then, $C_{p+1}^i = C_{p+1}^i \oplus \Delta$ is the block directly after the common prefix. Sampling V_j^i and deriving U_j^i ensures consistent sampling, i.e., $M_{p+1}^i = M_{p+1}^{i'} \oplus \Delta$ for all such queries with non-empty common prefix.

Starting from the $(p+2)$-th block, the ideal decryption oracle samples the responses $M_j^i \twoheadleftarrow \{0,1\}^r$, $V_j^i \twoheadleftarrow \{0,1\}^c$, and $U_j^i \twoheadleftarrow \{0,1\}^r$ uniformly and independently at random from the bit strings of expected lengths, for $p+2 \le j \le a^i + m^i$. Note that queries whose ciphertext lengths are not multiples of r bits are answered consistently since the oracle samples V_j^i, and all bits fixed from previous queries are used. For verification queries, the ideal verification oracle always outputs $\bot$. For forward primitive queries Q^i, the ideal oracle forwards $\pi(Q^i)$; for backward primitive queries R^i, it returns $\pi^{-1}(R^i)$.

Off-line phase.
Here, the ideal oracle releases the internal chaining values (U_j^i, V_j^i), after the considered adversary made all queries, but before outputting the decision bit. The ideal oracle also reveals a random key $K \twoheadleftarrow \mathcal{K}$ then.

Bad Events.
Whenever we consider a non-trivial collision between blocks or chaining values at block indices j, j' of two messages, we assume that at least one of them exceeds the longest common prefix.

- bad_1: Non-trivial collision of permutation inputs in construction queries: $\exists\,(i,j) \neq (i',j')$ with $i, i' \in [1..q_C]$, $j \in [1..m^i]$, and $j' \in [1..m^{i'}]$ s.t. $(X_j^i \parallel Y_j^i) = (X_{j'}^{i'} \parallel Y_{j'}^{i'})$.
- bad_2: Non-trivial collision of permutation outputs in construction queries: $\exists\,(i,j) \neq (i',j')$ with $i, i' \in [1..q_C]$, $j \in [1..m^i]$, and $j' \in [1..m^{i'}]$ s.t. $(U_j^i \parallel V_j^i) = (U_{j'}^{i'} \parallel V_{j'}^{i'})$.
- bad_3: Multi-collision between w tags. For some $w \ge r$, there exist $i_1, i_2, \ldots, i_w$ with $i_1, i_2, \ldots, i_w \in [1..q_e]$, s.t. $T^{i_1} = T^{i_2} = \ldots = T^{i_w}$.

- bad_4: Non-trivial collision of permutation inputs between construction and primitive query: $\exists\,(i, j, i')$ with $i \in [1..q_c]$, $j \in [1..m^i]$, and $i' \in [1..q_p]$ s.t. $(X^i_j \parallel Y^i_j) = Q^{i'}$.
- bad_5: Non-trivial collision of permutation outputs between construction and primitive query: $\exists\, i \in [1..q_c]$, $j \in [1..a^i + m^i]$ and $i' \in [1..q_p]$ s.t. $(U^i_j \parallel V^i_j) = R^{i'}$.
- bad_6: Initial-state collision with a primitive query: $\exists\, i \in [1..q_c]$ and $i' \in [1..q_p]$ s.t. $(X^i_0 \parallel Y^i_0) = Q^{i'}$.
- bad_7: Multi-collision in the rate of w outputs of forward primitive queries: for some $w \geq r$, $\exists\, i_1, i_2, \ldots, i_w \in [1..q_p]$ s.t. $\mathsf{msb}_r(R^{i_1}) = \cdots = \mathsf{msb}_r(R^{i_w})$.
- bad_8: Multi-collision in the rate of w outputs of backward primitive queries: for some $w \geq r$, $\exists\, i_1, i_2, \ldots, i_w \in [1..q_p]$ s.t. $\mathsf{msb}_r(Q^{i_1}) = \cdots = \mathsf{msb}_r(Q^{i_w})$.
- bad_9: Forgery in decryption queries if all blocks are old: There exists some $i \in [1..q_d]$ s.t. for all blocks $0 \leq j \leq a^i + m^i$, there exist indices i', j' with $i' \in [1..q_c]$, $j' \in [1..m^{i'}]$ or $i' \in [1..q_p]$ s.t. $(X^i_j \parallel Y^i_j) = (X^{i'}_{j'} \parallel Y^{i'}_{j'})$ or $(X^i_j \parallel Y^i_j) = Q^{i'}$ and the tag is valid: $\mathsf{msb}_\tau\left(\pi\left(X^i_{a^i+m^i} \parallel Y^i_{a^i+m^i}\right)\right) = T^i$.

We define BADT to contain exactly the attainable transcripts τ for which at least one bad events occurred. All other attainable transcripts are in GOODT. Then $\Pr[\Theta_{\text{ideal}} \in \text{BADT}] \leq \sum_{i=1}^{9} Pr[bad_i]$. The probability of bad transcripts in the ideal world is treated in the proof of Lemma 4. The ratio of obtaining a good transcript is bounded in Lemma 5. Our bound in Theorem 1 follows from those and the fundamental Lemma of the H-coefficient Technique [54]. We apply $w = r$ in the bound of Lemma 4.

Lemma 4. Let $w \geq r$ be a positive integer. It holds that

$$\Pr[\Theta_{\text{ideal}} \in \text{BADT}] \leq \frac{\sigma_e^2}{2^n} + \frac{3\sigma_e\sigma_d + 3\sigma q_p + q_c q_p + q_p + w(\sigma_d + q_d) + 3wq_p}{2^{c+s}} + \frac{q_d^2 + \binom{q_d+q_v}{2}}{2^c} + \frac{\binom{q_e}{w}}{2^{\tau(w-1)}} + \frac{3q_p}{2^k} + \frac{2\binom{q_p}{w}}{2^r(w-1)} + \frac{q_v}{2^\tau}.$$

Proof. In the following, we upper bound the probabilities of the individual bad events. For most of them, we differentiate between encryption and decryption queries.

bad_1: Collision of two permutation inputs in construction queries.

1. Among encryption queries only: Here, it holds that
$$\Pr\left[\left(X^i_j \parallel Y^i_j\right) = \left(X^{i'}_{j'} \parallel Y^{i'}_{j'}\right)\right] \leq 2^{-n}.$$
Since there exist $\binom{\sigma_e}{2}$ block combinations, we obtain $\binom{\sigma_e}{2}/2^n$.
2. Dec-then-Enc: If we consider an encryption query block to collide with a block from a previous decryption query, the probability is at most $2^{-(c+s)}$ since **A** can see $r - s$ bits that it can use as the nonce. We have $q_e\sigma_d$ combinations of such blocks. For the remaining $\sigma_e\sigma_d$ blocks, the probability is 2^{-n}.
3. Among decryption queries only: w.l.o.g., we consider the first such collision. If **A** modifies the nonce in the later following query, the bound is the same as for encryption-only queries. So, we assume in the remainder of that the later query is a decryption query. Let $j - 1$ be the first modified block and assume it is in the message-processing part. If the block indices differ $j \neq j'$, the probability is $2^{-(c+s)}$. Otherwise, assume $j = j'$ and $A^i = A^{i'}$. Then, the permutation output $(U^{i'}_j \parallel V^{i'}_j)$ is sampled randomly in the ideal world. If **A** leaves $C^i_{j-a^i} = C^{i'}_{j-a^i}$, it automatically holds that
$$X^i_j = C^i_{j-a^i} \oplus \mathsf{lsb}_s\left(V^i_{\rho(i,j-a^i)}\right) = X^{i'}_j = C^i_{j-a^i} \oplus \mathsf{lsb}_s\left(V^i_{\rho(i,j-a^i)}\right).$$
So, $X^i_j = X^{i'}_j$. With probability 2^{-c}, it also holds for the capacity $V^i_{j+1} = V^{i'}_{j+1}$ and thus $Y^i_{j+1} = Y^{i'}_{j+1}$. Note that this approach holds only for the first differing block, for which which yields a term of $\binom{q_d}{2}/2^c$. If the collision does not hold, the masks beginning for the $(j+2)$-th block will differ and the probability decreases to $2^{-(c+s)}$, which produces a term of $\binom{\sigma_d}{2}/2^{c+s}$.

4. Enc-then-Dec: It remains to consider collisions between an encryption query, followed by a decryption query. If the block indices $j \neq j'$ differ, the probability is again $2^{-(c+s)}$, for at most $\sigma_e \cdot \sigma_d$ combinations. Otherwise, if $j = j'$, **A** can apply the strategy above for a collision. Then, the probability is 2^{-c}; though, the q_d queries can collide at most with one encryption query each since we consider the first collision, producing a term of $q_d/2^c$.

Over all cases, we obtain

$$\Pr[\mathsf{bad}_1] \le \frac{\binom{\sigma_e}{2}}{2^n} + \max\left(\frac{q_e\sigma_d}{2^{c+s}} + \frac{\sigma_e\sigma_d}{2^{c+s}} + \frac{q_d}{2^c}\right) + \frac{\binom{\sigma_d}{2}}{2^{c+s}} + \frac{\binom{q_d}{2}}{2^c} \le \frac{\binom{\sigma_e}{2}}{2^n} + \frac{\sigma_e\sigma_d}{2^{c+s}} + \frac{\binom{q_d}{2}}{2^c} .$$

bad$_2$: Collision of two permutation outputs in encryption construction queries.
This case is analogous to bad_1. Over all combinations of indices, it follows that

$$\Pr[\mathsf{bad}_2] \le \frac{\binom{\sigma_e}{2}}{2^n} + \frac{\sigma_e\sigma_d}{2^{c+s}} + \frac{\binom{q_d}{2}}{2^c} .$$

bad$_3$: Multi-collision on w tags from encryption queries.
Since the tags are sampled uniformly and independently at random in the ideal world, it holds that

$$\Pr\left[T^{i_1}_{j_1} = T^{i_2}_{j_2} = \ldots = T^{i_w}_{j_w}\right] \le \frac{\binom{q_e}{w}}{2^{\tau(w-1)}} .$$

bad$_4$: Collision of permutation inputs between a construction and a primitive query.
Again, we consider $X^i_j \leftarrow C^i_{j-a^i} \oplus_s \mathsf{lsb}_s(V^i_{\rho(i,j-a^i)})$ and $Y^i_j \leftarrow V^i_j \oplus \text{const}$, where const is a public constant. The values $C^i_{j-a^i}$ and $V^i_{\rho(i,j-a^i)}$ are sampled randomly, the values $(X^i_j \,\|\, Y^i_j)$ take any value with probability at most 2^{-n}.

1. Assume, the primitive query was asked before the construction query. If the construction query was in encryption direction, the collision probability for fixed queries is at most 2^{-n}, for $q_p \cdot q_c$ combinations.
2. Otherwise, if the construction query was a decryption query, **A** can see $r - s$ bits. Hence, the probability is at most $2^{-(c+s)}$, for $q_p \cdot q_c$ combinations.
3. The same argument can be applied in the case when the primitive query was asked after a decryption query. Then, the adversary can see $r - s$ unmasked bits of the rate from C^i_j. Again, the probability is at most $2^{-(c+s)}$ and we have $q_p \cdot q_c$ combinations.
4. If the primitive query was asked after an encryption query, then, the latter produced a tag. If the primitive query targets any other block, the argument is the same as in Case c). If the primitive query starts from the tag, the adversary sees $\tau - s$ unmasked bits. Assuming $\overline{\mathsf{bad}_3}$, there are at most w equal tags over all encryption queries. So, the probability for a collision is $2^{-(c+s)}$, for $w \cdot q_p$ combinations.

Over all combinations of indices, it follows that

$$\Pr\left[\mathsf{bad}_4 | \overline{\mathsf{bad}_3}\right] \le \max\left(\frac{\sigma_e \cdot q_p}{2^n}, \frac{\sigma_e \cdot q_p}{2^{c+s}}\right) + \frac{\sigma_d \cdot q_p}{2^{c+s}} + \frac{w \cdot q_p}{2^{c+s}} \le \frac{\sigma \cdot q_p}{2^{c+s}} + \frac{w \cdot q_p}{2^{c+s}} .$$

bad$_5$: Collision of permutation outputs between a construction and a primitive query.
Again, $U^i_{j+a^i}$ can be derived from $M^i_j \oplus C^i_j \oplus_s \mathsf{lsb}_s(V^i_{\rho(i,j)})$; the values C^i_j and $V^i_{\rho(i,j)}$ are sampled randomly. This case is similar as bad_4. Over all index combinations

$$\Pr\left[\mathsf{bad}_5 | \overline{\mathsf{bad}_3}\right] \le \max\left(\frac{\sigma_e \cdot q_p}{2^n}, \frac{\sigma_e \cdot q_p}{2^{c+s}}\right) + \frac{\sigma_d \cdot q_p}{2^{c+s}} + \frac{w \cdot q_p}{2^{c+s}} \le \frac{\sigma \cdot q_p}{2^{c+s}} + \frac{w \cdot q_p}{2^{c+s}} .$$

$\mathtt{bad}_6$: Initial-state collision with a primitive query.
Here, we distinguish between the cases whether the construction query was asked before or after the primitive query and whether the primitive query was in forward or backward direction.

1. Assume, the primitive query was asked after the construction query. If the primitive query was a forward query, it must hit the correct value of $K \oplus_d d_N$. This probability is at most $q_p/2^k$ to collide when considering encryption construction queries. Considering also decryption queries, a nonce can repeat often; though, the initial state can take three different values for the same nonce, namely if the decryption query changes the length of associated data and message, affecting d_N. Since there exist at most three distinct values for d_N, the probability to collide is at most $3q_p/2^k$.
2. If the primitive query was in backward direction, its response must hit any initial state of a construction query. If the construction query was asked before the primitive, **A** sees at best $r - s$ bits of C_1. Then, the probability is at most $q_c \cdot q_p/2^{c+s}$.
3. If the primitive query was asked before the construction query, **A** can use the nonce part of the primitive query's result as the nonce. However, a collision must hit the key part, which holds with probability at most $3q_p/2^k$.
4. If the primitive query was in backward direction, **A** sees at best $r - s$ bits of C_1. Then, there is at most one starting state, assuming $\overline{\mathtt{bad}_1}$, which yields $q_p/2^{c+s}$.

Over all possible options, we obtain

$$\Pr\left[\mathtt{bad}_6 | \overline{\mathtt{bad}_1}\right] \leq \frac{3q_p}{2^k} + \max\left(\frac{q_c \cdot q_p}{2^{c+s}}, \frac{q_p}{2^{c+s}}\right) \leq \frac{3q_p}{2^k} + \frac{q_c \cdot q_p}{2^{c+s}} + \frac{q_p}{2^{c+s}} .$$

$\mathtt{bad}_7$: Multi-collision in the rate of w outputs of forward primitive queries.
Since π is chosen randomly from the set of all permutations, the outputs are sampled uniformly at random from a set of size at least $2^n - (i-1)$ for the i-th query. So, the probability for w distinct queries to collide in their rate is upper bounded by

$$\frac{2^c - 1}{2^n - 1} \cdot \frac{2^c - 2}{2^n - 2} \cdot \dots \cdot \frac{2^c - (w-1)}{2^n - (w-1)} = \prod_{i=1}^{w-1} \frac{2^c - i}{2^n - i} \leq \left(\frac{2^c}{2^n}\right)^{w-1} = 2^{-r(w-1)} .$$

Over all primitive query indices, it holds that

$$\Pr[\mathtt{bad}_7] \leq \frac{\binom{q_p}{w}}{2^{r(w-1)}} .$$

$\mathtt{bad}_8$: Multi-collision in the rate of w outputs of backward primitive queries.
Using a similar argumentation as for $\mathtt{bad}_7$, we obtain

$$\Pr[\mathtt{bad}_8] \leq \frac{\binom{q_p}{w}}{2^{r(w-1)}} .$$

$\mathtt{bad}_9$: Forgeries if all blocks are old.
It remains to bound the probability of a successful forgery of a verification query (N^i, A^i, C^i, T^i) s. t. T^i is valid and where each block is old.

We consider the same five mutually exclusive cases as in the nAE proof. In all cases, the tag can simply be guessed correctly if the block $(X^{a^i+m^i} \| Y^{a^i+m^i})$ is fresh. Then, the probability for the tag to be correct is upper bounded by $2^{-\tau}$. We adopt the cases and the notions from the nAE proof and assume that no previous bad events occur, in particular no w-multi-collisions described earlier or collisions with primitive queries.

- **Case (A):** N^i is fresh; so, there is no earlier construction query $i' \neq i$ s. t. $N^i = N^{i'}$.
- **Case (B):** N^i is old, but (N^i, A^i) is fresh, i.e., there exists no earlier construction query $i' \neq i$ with $(N^i, A^i) = (N^{i'}, A^{i'})$.

- **Case (C):** (N^i, A^i) is old, but (N^i, A^i, C^i) is fresh, i.e., there exists no earlier construction query $i' \neq i$ with $(N^i, A^i, C^i) = (N^{i'}, A^{i'}, C^{i'})$, and no w-chain of primitive queries is hit.
- **Case (D):** (N^i, A^i, C^i) is old; (N^i, A^i, C^i) is a prefix of another construction query.
- **Case (E):** (N^i, A^i) is old and there exists a w-chain of primitive queries that is hit.

Clearly, the cases cover all possible options. We assume that no previous bad events occur, in particular no w-multi-collisions described earlier or collisions with primitive queries.

Case (A).
We excluded bad_4, i.e., collisions of permutation inputs between construction and primitive queries in this case. The probability that $(N^i \| K) \oplus_d d_N$ hits any block $(X_j^{i'} \| Y_j^{i'})$ from another construction query so that the final block is old is at most

$$\Pr\left[\left(\left(N^i \| K\right) \oplus_d d_N\right) = \left(X_j^{i'} \| Y_j^{i'}\right)\right] \le \frac{\sigma + q_p}{2^{c+s}} .$$

Cases (B)–(D).
Let $p \le a^i + m^i$ denote the length of the longest common prefix of the i-th query with all other queries. The probability that any block $(X_j^i \| Y_j^i)$ with $j \ge p + 1$ matches the permutation input of any other query block or primitive query can be upper bounded analogously as bad_1 and bad_4:

$$\Pr\left[\left(X_j^i \| Y_j^i\right) = \left(X_{j'}^{i'} \| Y_{j'}^{i'}\right)\right] \le \frac{\sigma_e + q_d}{2^{c+s}} + \frac{\sigma_e \cdot \sigma_d}{2^{c+s}} + \frac{\binom{q_d}{2}}{2^c} + \frac{\sigma \cdot q_p}{2^{c+s}} + \frac{w \cdot q_p}{2^{c+s}} .$$

Over all verification queries, we obtain

$$\Pr\left[\mathsf{bad}_9 \,\middle|\, \bigwedge_{i=1}^{8} \overline{\mathsf{bad}_i}\right] \le \frac{q_v}{2^\tau} + \frac{\sigma_e \cdot \sigma_d}{2^{c+s}} + \frac{\binom{q_d+q_v}{2}}{2^c} + \frac{\sigma \cdot q_p}{2^{c+s}} + \frac{w \cdot q_p}{2^{c+s}} + \frac{w \cdot (\sigma_d + q_d)}{2^{c+s}} .$$

Case (E).
Assume that $(X_{p+1}^i \| Y_{p+1}^i)$ hits a w-chain of primitive queries. Under the assumption that no other bad events occurred, the probability is at most

$$\Pr\left[\left(X_{p+1}^i \| Y_{p+1}^i\right)\right] \le \frac{(m^i + 1) \cdot w}{2^{c+s}} .$$

Over all verification queries, we obtain

$$\Pr\left[\mathsf{bad}_9 \,\middle|\, \bigwedge_{i=1}^{8} \overline{\mathsf{bad}_i}\right] \le \frac{q_v}{2^\tau} + \frac{\sigma_e \cdot \sigma_d}{2^{c+s}} + \frac{\binom{q_d+q_v}{2}}{2^c} + \frac{\sigma \cdot q_p}{2^{c+s}} + \frac{w \cdot q_p}{2^{n-\tau}} + \frac{w \cdot (\sigma_d + q_d)}{2^{c+s}} .$$

Our bound in Lemma 4 follows from summing up all probabilities.

Lemma 5. Let $\tau \in \textsc{GoodT}$. Then

$$\frac{\Pr[\Theta_{\text{real}} = \tau]}{\Pr[\Theta_{\text{ideal}} = \tau]} \le 1 - \left(\frac{q_d}{2^\tau} + \frac{\sigma_d(\sigma_e + q_p)}{2^{c+s}}\right) .$$

Proof. It remains to bound the ratio of the probabilities for obtaining a good transcript τ in the real and the ideal world, respectively. The bound is similar to that of Lemma 3. The difference to the nAE proof is that the ideal decryption oracle also generates pseudorandom output blocks M_j^i beyond the longest common prefix. The nAE transcript also contained the sampled internal values, as does the transcript τ here. Since we assume that no bad events have occurred, we revisit the following cases for forgeries:

- **Case (A):** The final input to π, $(X^i_{a^i+m^i} \| Y^i_{a^i+m^i})$ is fresh, i.e., has not occurred before. Then, the probability that the authentication tag τ^i is valid is at most $1/2^\tau$.
- **Case (B):** The final input to π, $(X^i_{a^i+m^i} \| Y^i_{a^i+m^i})$ is old, but there exists some block index $j \in [1..a^i + m^i]$ s. t. $(X^i_j \| Y^i_j)$ is fresh. Since the input is old, the probability that the result of any of the next blocks is old is at most $\frac{(\sigma+q_p)}{2^{c+s}}$.
- **Case (C):** There exists no $j \in [1..a^i + m^i]$ s. t. $(X^i_j \| Y^i_j)$ is fresh. The probability that all of those blocks are old is at most $\frac{m^i(\sigma_e+q_p)}{2^{c+s}}$.

It follows that

$$\epsilon_i \le \frac{1}{2^\tau} + \frac{\sigma_e + q_p}{2^{c+s}} + \frac{m^i(\sigma_e + q_p)}{2^{c+s}} .$$

Over all indices $i \in [1..q_d]$, it follows that

$$\epsilon \le \sum_{i=1}^{q_d} \epsilon_i \le \frac{q_d}{2^\tau} + \frac{\sigma_d(\sigma_e + q_p)}{2^{c+s}} .$$

Our claim in Lemma 5 follows. □

8 Comparison with Lightweight Int-RUP-secure Schemes

Among the submissions to the NIST lightweight competition [53], ESTATE [27], LAEM [61], LOTUS-AEAD and LOCUS-AEAD [24] claimed security in the Int-RUP model. Among these modes, ESTATE, LOTUS-AEAD, and LOCUS-AEAD were elected into the second round. This section compares our proposal to those; Table 4 gives a summary.

Table 4: 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) [21]	64	128	96	192	0	288	96	89	48	•	•
Oribatida-256 (v1.2) [21]	128	128	128	256	0	320	128	121	64	•	•
Oribatida v1.3-192 **[This work]**	64	128	96	192	0	288	96	121	48	•	•
Oribatida v1.3-256 **[This work]**	128	128	128	256	0	320	128	121	64	•	•
ESTATE [27]	128	128	128	128	4	260	64	64	64	–	•
LOCUS-AEAD [24]	128	128	64	64	4	324	32	64	64	•	–
LOTUS-AEAD [24]	128	128	64	64	4	384	32	64	64	•	•

8.1 Brief Description

ESTATE follows SIV [57]: the associated data and message are authenticated using a variant of CBC-MAC with a tweakable block cipher before the tag is used as an 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 [23] to process the associated data with the tweakable block cipher. For encryption, LOTUS-AEAD uses a variant of OTR [50], a two-round, two-branch Feistel structure to process the message in double blocks. LOCUS-AEAD employs an encryption similar to OCB [56] and EME/EME* [38]. Both 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.

8.2 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 must 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.

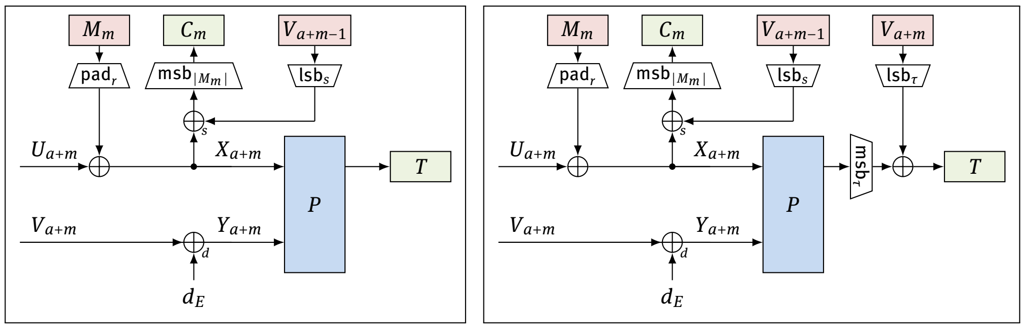
8.3 Security

All three competitors are based on tweakable block ciphers, with Int-RUP claims limited by the birthday bound of the internal primitive. ESTATE inherits Int-RUP security until the birthday bound from SIV, which has been considered in [7, Sect. 6.2]. While LOCUS-AEAD and LOTUS-AEAD share similarities to OCB and OTR, they use intermediate checksums as in EME designs in the tag-generation process. Informally, modifying any message block will result in new pseudorandom internal values and therefore a pseudorandom input to the tag computation.

9 Discussion of the Updated Variant Oribatida v1.3

This section discusses the update from Oribatida (v1.2) from [21] to Oribatida v1.3 in this work, that addresses the observation by Rohit and Sarkar [59] in a straight-forward manner. Here, we briefly discuss only the differences:

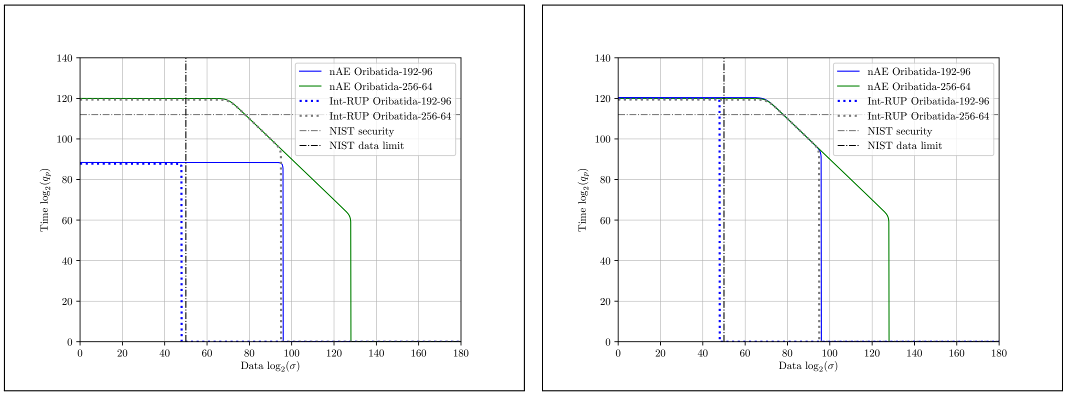
1. Oribatida v1.2 released the tag without masking. As a consequence, the adversary has seen the full rate and had to guess only the $n - \tau$-bit hidden part to be able to invert the encryption process. To succumb this attack, Oribatida v1.3 masks the tag such that the adversary sees $\tau - s$ bits if $s \leq \tau$, which restores the complexity from $q/2^{n-\tau}$ to $q/2^{c+s}$. Figure 4 illustrates both tag-generation processes for comparison. The masking of the authentication tag is performed exactly as for ciphertext blocks, which streamlines this process.
2. Oribatida v1.2 employed two permutations P and P', where the latter was intended to be a more efficient variant of the former. In practice, P' was instantiated with a round-reduced version of P, which was only used for processing intermediate blocks of associated data. This was fine since an upper bound on the probability of differentials was sufficient for security and not pseudo-randomness. Oribatida v1.3 unified the process and uses P at every location.
3. Oribatida v1.2 used a different starting value V_0 for masking the ciphertexts when the associated data was empty and V_1 otherwise. The reason was simply efficiency since empty associated data did not yield a value V_1. In contrast, Oribatida v1.3 always pads the associated data such that there always exists an intermediate value V_1 that is not used as capacity in the message-processing step. This decision implies a slightly lower throughput for empty associated data but adds unification.
4. As a result of Aspect (3), Oribatida v1.3 uses slightly different and more domains to properly address also the additional case when the associated data was empty.



(a) Oribatida v1.2 [21].

(b) Oribatida v1.3.

Figure 4: Tag generation of Oribatida.



(a) Oribatida v1.2 [21].

(b) Oribatida v1.3.

Figure 5: Security of Oribatida using $q_c = 2^{50}$.

Among all changes, only Aspect (1) is crucial. All further aspects helped unify the design. The security effect of the additional tag masking is illustrated in Figure 5 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.

10 Instantiation of Oribatida

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. [31]. We define SimP to use a round-reduced variant of the Simon [10] 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.

10.1 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 6. 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 $\Psi_r, \pi_1, \pi_2, \ldots, \pi_r \in \mathsf{BlockCipher}(\mathbb{F}_2^n \times \mathbb{F}_2^n, \mathbb{F}_2^n)$ are independent block ciphers which use 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.

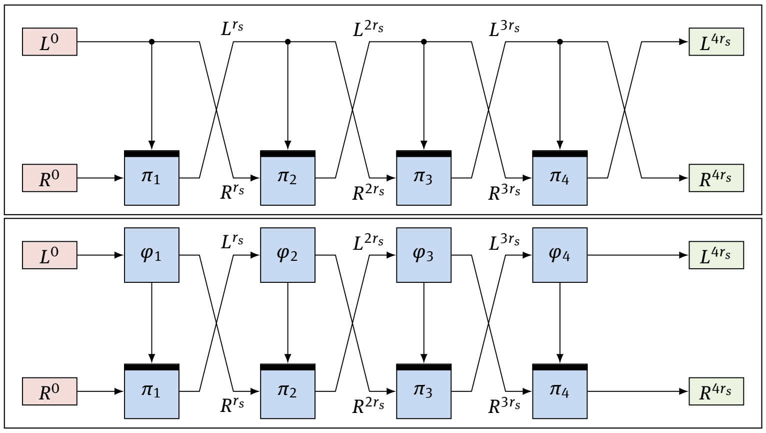


Figure 6: Top: The construction Ψ_4 [31]. The blocks π_i denote block ciphers over $\mathbb{F}_2^n$ with key space $\mathbb{F}_2^n$. **Bottom:** High-level view of the construction Φ_4 as a variant of Ψ_4. The blocks φ_i represent the key schedules that produce the subkeys and which are externalized 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}.

Theorem 3 ([31])**.** 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.

10.2 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 6. 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.

10.3 Simon

The Simon family of block ciphers [10] 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., [3, 29, 44, 55, 64] 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 [40]. 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.

10.4 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 7. 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 7. 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.

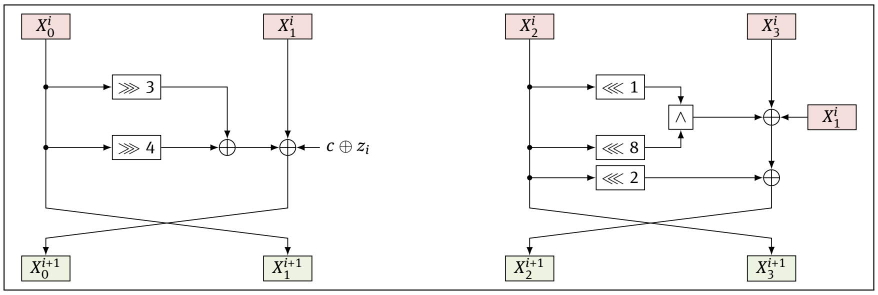


Figure 7: One iteration of the round function of SimP, which is equivalent to the key-update function (left) and the state-update function (right) of Simon-$2w/2w$, where w is the word size.

10.4.1 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

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

10.4.2 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.

10.4.3 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 $1 \le i \le r_s$.

10.4.4 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

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

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

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

SimP-n-θ takes a plaintext $(X_0^0, X_1^0, X_2^0, X_3^0)$ as input 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.

10.4.5 Round Constants

The round constants are those of Simon-96-96 and Simon-128-128 [10], respectively. It holds that c = `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 $z = z_0 z_1 \dots z_{61}$ 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.

10.4.6 Number of Steps θ

We consider only the choice of $\theta = 4$ everywhere in our proposed construction.

10.4.7 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. For simplicity, we also denote SimP-n-4 as SimP-n. The algorithm for SimP-n-θ is given in Algorithm 3.

Table 5: Parameters of SimP.

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

Algorithm 3 Specification of the encryption and decryption algorithms of SimP-n-θ.

```
101: function SIMP-$n$-$\theta(M)$
102:     $(X_0^0, X_1^0, X_2^0, X_3^0) \xleftarrow{w} M$
103:     for $i \leftarrow 1..\theta$ do
104:         for $j \leftarrow 1..r_s$ do
105:             $\ell \leftarrow (i-1) \cdot r_s + j$
106:             $X_0^\ell \leftarrow X_1^{\ell-1} \oplus g(X_0^{\ell-1}) \oplus c \oplus z_{\ell-1}$
107:             $X_1^\ell \leftarrow X_0^{\ell-1}$
108:             $X_2^\ell \leftarrow X_3^{\ell-1} \oplus f(X_2^{\ell-1}) \oplus X_1^{\ell-1}$
109:             $X_3^\ell \leftarrow X_2^{\ell-1}$
110:         if $i \neq \theta$ then
111:             $(X_0^\ell, X_1^\ell, X_2^\ell, X_3^\ell) \leftarrow$ swap$(X_0^\ell, X_1^\ell, X_2^\ell, X_3^\ell)$
112:     $C \leftarrow (X_0^{\theta r_s} \| X_1^{\theta r_s} \| X_2^{\theta r_s} \| X_3^{\theta r_s})$
113:     return $C$
```

```
121: function $f(X)$
122:     return $((X \lll 1) \wedge (X \lll 8))$
123:             $\oplus (X \lll 2)$

131: function $g(X)$
132:     return $(X \ggg 3) \oplus (X \ggg 4)$

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

10.4.8 The Byte Order in Oribatida

For the sake of clarity, Figure 8 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 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 is used first as input to the key-update function; the capacity is used as input to the state-update function.

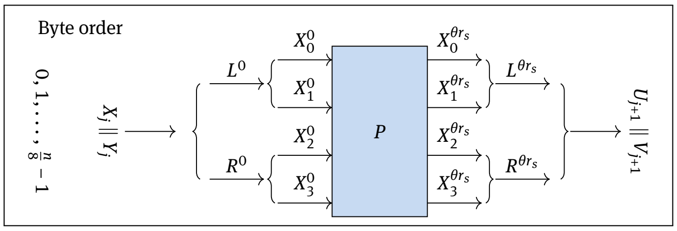


Figure 8: Byte and word orientation of inputs into and outputs from SimP as used in Oribatida.

Remark 2. 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 [39]. 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.

11 Security of SimP

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

11.1 Requirements

Oribatida with a random permutation aims at NAE security of $O(r\sigma_d/2^{c+s})$ and INT-RUP security of $O(q_d^2/2^c)$ 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 us to distinguish it from a random permutation with non-negligible advantage and $<< 2^n$ 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 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.

11.2 Existing Cryptanalysis on Simon

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

11.2.1 Differential Cryptanalysis

Cryptanalysis that appeared early after the proposal of Simon followed mainly heuristics for differential cryptanalysis: Abed et al. [3] followed a heuristic branch-and-bound approach that yielded differentials for up to 30 rounds of Simon-96. Biryukov et al. [22] studied more efficient heuristics, but considered the small variants with state sizes up to 64 bits. Dinur et al. [33] 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. [42] redirected the research focus to the search for optimal characteristics. More recently, Liu et al. [44] employed a variant of Matsui's algorithm [46] 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 an 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 a cumulative probability of $2^{-123.74}$.

11.2.2 Linear Cryptanalysis

Linear cryptanalysis is similarly effective for Simon-like ciphers as its differential counterpart. Alizadeh et al. [1, 4] 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 [29] 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 [45]. 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.

11.2.3 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. [65] 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. [64], and later by Todo and Morii [62]. The latter could show the absence of integrals for 25-round Simon-96, which was confirmed by Kondo et al. [43].

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. [42], 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.

11.2.4 Related-key Distinguishers

Kondo et al. [43] 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 key word 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

Table 6: 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	–	[55]
Integral	22	2^{95}	2^{95} CP	95	[64]
Differential	30	–	–	92.2	[3]
Differential	31	–	–	95.34	[44]
Linear	31	–	–	93.8	[45]
Simon-96-96 Key-recovery Attacks					
Multiple Linear	33	$2^{94.42}$	$2^{94.42}$ KP	94.42	[2]
Linear Hull	37	$2^{88.0}$	$2^{95.2}$ KP	95.2	[29]
Simon-128-128 Distinguishers					
Algebraic	16	–	20 CP	–	[55]
Integral	26	2^{126}	2^{126} CP	126	[64]
Linear	37	–	–	128	[2]
Differential	41	–	–	123.74	[44]
Linear Hull	41	–	–	123.15	[45]
Simon-128-128 Key-recovery Attack					
Linear Hull	49	$2^{127.6}$	$2^{127.6}$ CP	126.6	[29]

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 6.

11.2.5 Algebraic Cryptanalysis

Algebraic attacks are unlikely to be a threat to Simon-like constructions for sufficiently many rounds. Raddum [55] pointed out that the large number of rounds is necessary. He demonstrated that the equation systems for up to 14 rounds of Simon-96-96 and up to 16 rounds of Simon-128 can be solved efficiently in a few minutes on an off-the-shelf laptop. Extensions to considerably more rounds are still unknown.

11.2.6 Meet-in-the-Middle Attacks

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-$2w$-$2w$.

11.2.7 Correlated Sequences

An interesting recent direction may be correlated sequences introduced by Rohit and Gong in [58]. Their technique requires only very few texts and claims to break 27 rounds of Simon-32 and SIMECK-32; thus, it might outperform 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.

11.3 Implications to SimP

Since the key schedule of Simon is fully linear, the two state words that are transformed by the key schedule allow the 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.

11.3.1 Related-key Differential Cryptanalysis

SimP needs cryptanalysis of related-key differential and linear characteristics. Existing methods such as the exhaustive search in [44] or SAT solvers [42], 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. [63]. 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 [42] as well as with the branch-and-bound approach from [44] 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 characteristics found 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 7 compares the probabilities of optimal single- and related-key differential characteristics.

Table 7: 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 / 21	4 / 22	5 / 23	6 / 24	7 / 25	8 / 26	9 / 27	10 / 28	11 / 29	12 / 30	13 / 31	14 / 32	15 / 33	16 / 34	17 / 35	18 / 36	19 / 37	20
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										

11.3.2 Differential Distinguishers

Differential distinguishers are not expected to cover more than two steps. For two steps, we can sketch the high-level idea for a potential distinguisher. Let $\alpha_0, \alpha_{r_s}, \beta_0, \beta_{r_s} \in (\mathbb{F}_2^w)^2$. Let $(\alpha_0, 0) \to (\beta_0, \alpha_{r_s})$ be a differential through one step of SimP. $\alpha_0 \to \alpha_{r_s}$ is a probability-one differential through r_s rounds of the key-update function φ_1, and $0 \to \beta_0$ the related-key differential through r_s rounds of the state-update function π_1, keyed with K and $K \oplus \alpha_0$, respectively. Assume, $p =^{\text{def}} \Pr[(\alpha_0, 0) \to (\beta_0, \alpha_{r_s})] > 2^{-2w}$.

In the second step, the difference β_0 is transformed to β_{r_s} linearly, i.e., $\Pr[\beta_0 \to \beta_{r_s}] = 1$. We can assume that $\beta_0 \neq 0$ and (β_0, α_{r_s}) will be transformed to an output difference $(\beta_{r_s}, 0)$ after the second step with probability $q \approx 2^{-2w}$ by approximating π_2 by a random permutation and using the Markov assumption and the random-key hypothesis. In this case, it holds that

$$\Pr\left[(\alpha_0, 0) \xrightarrow{p} (\beta_0, \alpha_{r_s}) \xrightarrow{2^{-2w}} (\beta_{r_s}, 0)\right] = p \cdot 2^{-2w} > 2^{-4w} .$$

Since π_1 is a round-reduced variant of Simon with 26 or 34 rounds, it is possible to have such trails. This setting is illustrated in Figure 9. However, an adversary would have to find a manyfold of related-key characteristics.

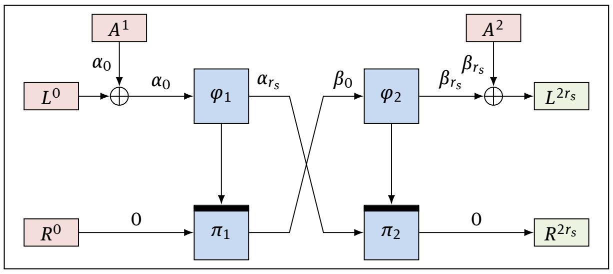


Figure 9: Setting of a differential attack with the step-reduced instance of SimP

11.3.3 Integral and impossible-differential Distinguishers

Integral and impossible-differential distinguishers are possible for up to two steps of SimP in general. We study an integral distinguisher in the following: Consider a structure of texts (L_i^0, R_i^0) that use pairwise distinct values R_i^0 and leave L_i^0 constant. After the first step, the value $R_i^{r_s}$ is also constant for all texts and all values $L_i^{r_s}$ are pairwise distinct. This property is preserved through the linear key schedule to $L_i^{2r_s}$. However, the values of R^{2r_s} are, in general, unknown. Note that there is no word swap after the second step. The third step destroys that knowledge.

Impossible-differential distinguishers can use a similar strategy, by setting $\Delta L^0 = 0$ and testing if $\Delta L^{2r_s} = 0$, which is impossible. Again, note that there is no word swap after the second step.

11.3.4 Cube-like Distinguishers

Cube (and integral) distinguishers exploit that the degree of some output-bit component functions is lower than the state size. As discussed above, the degree of each bit is at least w after more than 22 rounds for Simon-96 and more than 26 rounds for Simon-128. SimP transforms each input bit through at least two steps of Simon, that consist of 26 rounds for Simon-96 and 34 rounds for Simon-128. While two out of four steps do not increase the degree in the part that is used as the key-update function, each bit is transformed through full-round Simon-96 or Simon-192. Thus, cube-like and integral distinguishers are not expected to threaten the security of SimP.

11.3.5 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 differentials and linear characteristics nor by techniques that try to benefit from a larger 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 provides an additional security margin.

12 FPGA Implementations

This section reports on FPGA implementations of SimP and Oribatida.

12.1 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 by the need for 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 allows exploring other countermeasures such as unrolling rounds to achieve higher-order side-channel resistance.

SimP can be implemented in different levels of serialization, from fully serial implementations that update only 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.

12.2 Oribatida

Hardware implementations of our proposed instance of Oribatida are relatively straight-forward. 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 to the 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 are required.

The use of SimP as its main building block allows us 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 8 lists our implementations results obtained from Xilinx Vivado 2018 optimizing for area. All results represent measurements after the place-and-route process.

In Table 8, 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 leave still room for further improvements in the close future.

Table 8: Implementation results for SimP-256 and Oribatida-256-64 encryption/decryption and only encryption on *Virtex 7 FPGA*. LUTs = lookup tables; Enc. = encryption; Dec. = decryption.

	LUTs	FF	#Slices	Frequency (MHz)	Cycles	Throughput (Mb/s)
SimP						
SimP-256	495	340	148	580.51	137	542.37
SimP-192	383	259	122	581.98	105	532.10
Oribatida-256-64						
Enc. and Dec.	940	599	298	554.16	138	514.00
Enc. only	805	595	253	560.71	138	520.08

13 Conclusion

This work presented Oribatida, a nonce-based permutation-based AE scheme that masks the ciphertexts from preceding permutation outputs. As a result, the adversary cannot deduce the masked part of the internal state. Therefore, Oribatida can achieve $O(q_d^2/2^c)$ security against forgeries under the release of unverified plaintexts. Oribatida improves the best known INT-RUP security bound while the permutation can be kept as small as $64 + 128$ bits for 121-bit NAE and 48-bit INT-RUP security.

We showed that even recent previous proposals with high AE security guarantees, such as Beetle or SPoC, succumb to attacks with complexity $O(\frac{q_d q_p}{2^c})$. In contrast, the security bound of Oribatida does not depend primarily on the number of primitive queries. We showed that our bound is tight with a matching INT-RUP attack, generalized our masking approach by applying it also to Beetle or the NIST submission SPoC, and demonstrated its application with similar attacks on their masked variants.

Acknowledgement: We thank Raghvendra Rohit and Sumanta Sarkar for their observation on the NIST lightweight mailing list [59], as well as the anonymous reviewers of the Journal of Mathematical Cryptology for the fruitful comments. Eik List was supported by DFG Grant LU 608/9-1.

Absence of Conflict of Interest.
We the authors hereby declare that we have no conflict of interest in connection with evaluated manuscripts.

References

[1] M. A. Abdelraheem, J. Alizadeh, H. AlKhzaimi, M. R. Aref, N. Bagheri, P. Gauravaram, and M. M. Lauridsen. Improved Linear Cryptanalysis of Round Reduced SIMON. *IACR Cryptology ePrint Archive*, 2014:681, 2014.

[2] M. A. Abdelraheem, J. Alizadeh, H. A. AlKhzaimi, M. R. Aref, N. Bagheri, and P. Gauravaram. Improved Linear Cryptanalysis of Reduced-Round SIMON-32 and SIMON-48. In A. Biryukov and V. Goyal, editors, *INDOCRYPT*, volume 9462 of *LNCS*, pages 153–179. Springer, 2015.

[3] F. Abed, E. List, S. Lucks, and J. Wenzel. Differential Cryptanalysis of Round-Reduced Simon and Speck. In C. Cid and C. Rechberger, editors, *FSE*, volume 8540 of *LNCS*, pages 525–545. Springer, 2014.

[4] J. Alizadeh, N. Bagheri, P. Gauravaram, A. Kumar, and S. K. Sanadhya. Linear Cryptanalysis of Round Reduced SIMON. *IACR Cryptology ePrint Archive*, 2013:663, 2013.

[5] R. AlTawy, G. Gong, M. He, A. Jha, K. Mandal, M. Nandi, and R. Rohit. SpoC: An Authenticated Cipher. Technical report, Feb 24 2019. First-round submission to the NIST Lightweight Cryptography Competition. https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/spoc-spec.pdf.

[6] E. Andreeva, B. Bilgin, A. Bogdanov, A. Luykx, B. Mennink, N. Mouha, and K. Yasuda. APE: Authenticated Permutation-Based Encryption for Lightweight Cryptography. In C. Cid and C. Rechberger, editors, *FSE*, volume 8540 of *LNCS*, pages 168–186. Springer, 2014.

[7] E. Andreeva, A. Bogdanov, A. Luykx, B. Mennink, N. Mouha, and K. Yasuda. How to Securely Release Unverified Plaintext in Authenticated Encryption. In P. Sarkar and T. Iwata, editors, *ASIACRYPT I*, volume 8873 of *LNCS*, pages 105–125. Springer, 2014.

[8] E. Andreeva, J. Daemen, B. Mennink, and G. V. Assche. Security of Keyed Sponge Constructions Using a Modular Proof Approach. In G. Leander, editor, *FSE*, volume 9054 of *LNCS*, pages 364–384. Springer, 2015.

[9] J. Aumasson, P. Jovanovic, and S. Neves. NORX: Parallel and Scalable AEAD. In M. Kutylowski and J. Vaidya, editors, *ESORICS II*, volume 8713 of *LNCS*, pages 19–36. Springer, 2014.

[10] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers. The SIMON and SPECK Families of Lightweight Block Ciphers. *IACR Cryptology ePrint Archive*, 2013:404, 2013.

[11] M. Bellare, A. Boldyreva, L. R. Knudsen, and C. Namprempre. Online Ciphers and the Hash-CBC Construction. In J. Kilian, editor, *CRYPTO*, volume 2139 of *LNCS*, pages 292–309. Springer, 2001.

[12] M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby, editors, *ACM CCS*, pages 62–73. ACM, 1993.

[13] G. Bertoni, J. Daemen, S. Hoffert, M. Peeters, G. V. Assche, and R. V. Keer. Farfalle: parallel permutation-based cryptography. *IACR Trans. Symmetric Cryptol.*, 2017(4):1–38, 2017.

[14] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. On the Indifferentiability of the Sponge Construction. In N. P. Smart, editor, *EUROCRYPT*, volume 4965 of *LNCS*, pages 181–197. Springer, 2008.
[15] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In A. Miri and S. Vaudenay, editors, *SAC*, volume 7118 of *LNCS*, pages 320–337. Springer, 2011.
[16] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. Permutation-based encryption, authentication and authenticated encryption. *Directions in Authenticated Ciphers*, 2012.
[17] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Sponge functions. In *ECRYPT hash workshop*, volume 2007, 2007.
[18] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. On the security of the keyed sponge construction. In *SHA-3 competition (round 3)*, volume 2011, 2011.
[19] G. Bertoni, J. Daemen, M. Peeters, G. van Assche, and R. van Keer. Ketje v2. 2016. Submission to the CAESAR competition http://competitions.cr.yp.to/caesar-submissions.html.
[20] G. Bertoni, J. Daemen, M. Peeters, G. van Assche, and R. van Keer. Keyak v2. 2016. Submission to the CAESAR competition http://competitions.cr.yp.to/caesar-submissions.html.
[21] A. Bhattacharjee, E. List, C. M. López, and M. 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.
[22] A. Biryukov, A. Roy, and V. Velichkov. Differential Analysis of Block Ciphers SIMON and SPECK. In C. Cid and C. Rechberger, editors, *FSE*, volume 8540 of *LNCS*, pages 546–570. Springer, 2014.
[23] J. Black and P. Rogaway. A Block-Cipher Mode of Operation for Parallelizable Message Authentication. In L. R. Knudsen, editor, *EUROCRYPT*, volume 2332 of *LNCS*, pages 384–397. Springer, 2002.
[24] A. Chakraborti, N. Datta, A. Jha, C. M. Lopez, M. Nandi, and Y. Sasaki. LOTUS-AEAD and LOCUS-AEAD. Technical report, Feb 26 2019. First-round submission to the NIST Lightweight Cryptography Competition. https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/lotus-aead-and-locus-aead-spec.pdf.
[25] A. Chakraborti, N. Datta, M. Nandi, and K. Yasuda. Beetle Family of Lightweight and Secure Authenticated Encryption Ciphers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):218–241, 2018. Updated version at https://eprint.iacr.org/2018/805.
[26] A. Chakraborti, N. Datta, M. Nandi, and K. Yasuda. Beetle Family of Lightweight and Secure Authenticated Encryption Ciphers. http://eprint.iacr.org/2018/805, 2018.
[27] A. Chakraborti, A. Jha, C. M. Lopez, M. Nandi, and Y. Sasaki. ESTATE. Technical report, Mar 29 2019. First-round submission to the NIST Lightweight Cryptography Competition. https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/spoc-spec.pdf.
[28] D. Chang, M. Dworkin, S. Hong, J. Kelsey, and M. Nandi. A Keyed Sponge Construction with Pseudorandomness in the Standard Model. In *The Third SHA-3 Candidate Conference*, volume 3, page 7, March 2012.
[29] H. Chen and X. Wang. Improved Linear Hull Attack on Round-Reduced Simon with Dynamic Key-Guessing Techniques. In T. Peyrin, editor, *FSE*, volume 9783 of *LNCS*, pages 428–449. Springer, 2016.
[30] S. Chen and J. P. Steinberger. Tight Security Bounds for Key-Alternating Ciphers. In P. Q. Nguyen and E. Oswald, editors, *EUROCRYPT*, volume 8441 of *LNCS*, pages 327–350. Springer, 2014. Full version at https://eprint.iacr.org/2013/222.
[31] J. Coron, Y. Dodis, A. Mandal, and Y. Seurin. A Domain Extender for the Ideal Cipher. In D. Micciancio, editor, *TCC*, volume 5978 of *LNCS*, pages 273–289. Springer, 2010. Full version at https://eprint.iacr.org/2009/356.
[32] J. Daemen, B. Mennink, and G. V. Assche. Full-State Keyed Duplex with Built-In Multi-user Support. In T. Takagi and T. Peyrin, editors, *ASIACRYPT II*, volume 10625 of *LNCS*, pages 606–637. Springer, 2017.
[33] I. Dinur, O. Dunkelman, M. Gutman, and A. Shamir. Improved Top-Down Techniques in Differential Cryptanalysis. In K. E. Lauter and F. Rodríguez-Henríquez, editors, *LATINCRYPT*, volume 9230 of *LNCS*, pages 139–156. Springer, 2015.
[34] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer. Ascon v1.2 Submission to the CAESAR Competition. September 15 2016. Submission to the CAESAR competition. http://competitions.cr.yp.to/caesar-submissions.html.
[35] C. Dobraunig and B. Mennink. Security of the Suffix Keyed Sponge. *IACR Trans. Symmetric Cryptol.*, 2019(4):223–248, 2019.
[36] M. Dworkin. FIPS 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Technical report, 2015.
[37] P. Gaži, K. Pietrzak, and S. Tessaro. The Exact PRF Security of Truncation: Tight Bounds for Keyed Sponges and Truncated CBC. In R. Gennaro and M. Robshaw, editors, *CRYPTO I*, volume 9215 of *LNCS*, pages 368–387. Springer, 2015.
[38] S. Halevi. EME*: Extending EME to Handle Arbitrary-Length Messages with Associated Data. In A. Canteaut and K. Viswanathan, editors, *INDOCRYPT*, volume 3348 of *LNCS*, pages 315–327. Springer, 2004.
[39] V. T. Hoang, T. Krovetz, and P. Rogaway. Robust Authenticated-Encryption AEZ and the Problem That It Solves. In E. Oswald and M. Fischlin, editors, *EUROCRYPT (1)*, volume 9056 of *LNCS*, pages 15–44. Springer, 2015. Full version at https://eprint.iacr.org/2014/793.
[40] ISO/IEC. Information technology – Automatic identification and data capture techniques – Part 21: Crypto Suite SIMON Security Services for Air Interface Communications. https://www.iso.org/standard/70388.html, Oct 2018.
[41] P. Jovanovic, A. Luykx, and B. Mennink. Beyond $2^{c/2}$ Security in Sponge-Based Authenticated Encryption Modes. In P. Sarkar and T. Iwata, editors, *ASIACRYPT I*, volume 8873 of *LNCS*, pages 85–104. Springer, 2014.
[42] S. Kölbl, G. Leander, and T. Tiessen. Observations on the SIMON Block Cipher Family. In R. Gennaro and M. Robshaw, editors, *CRYPTO*, volume 9215 of *LNCS*, pages 161–185. Springer, 2015.

[43] K. Kondo, Y. Sasaki, Y. Todo, and T. 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.

[44] Z. Liu, Y. Li, and M. Wang. Optimal Differential Trails in SIMON-like Ciphers. *IACR Trans. Symmetric Cryptol.*, 2017(1):358–379, 2017.

[45] Z. Liu, Y. Li, and M. Wang. The Security of SIMON-like Ciphers Against Linear Cryptanalysis. *IACR Cryptology ePrint Archive*, 2017:576, 2017.

[46] M. Matsui. On Correlation Between the Order of S-boxes and the Strength of DES. In A. D. Santis, editor, *EUROCRYPT*, volume 950 of *LNCS*, pages 366–375. Springer, 1994.

[47] U. M. Maurer, R. Renner, and C. Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In M. Naor, editor, *TCC*, volume 2951 of *LNCS*, pages 21–39. Springer, 2004.

[48] B. Mennink. Key Prediction Security of Keyed Sponges. *IACR Trans. Symmetric Cryptol.*, 2018(4):128–149, 2018.

[49] B. Mennink, R. Reyhanitabar, and D. Vizár. Security of full-state keyed sponge and duplex: Applications to authenticated encryption. In T. Iwata and J. H. Cheon, editors, *ASIACRYPT II*, volume 9453 of *LNCS*, pages 465–489. Springer, 2015.

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

[51] N. Mouha, B. Mennink, A. V. Herrewege, D. Watanabe, B. Preneel, and I. Verbauwhede. Chaskey: An Efficient MAC Algorithm for 32-bit Microcontrollers. In A. Joux and A. M. Youssef, editors, *SAC*, volume 8781 of *LNCS*, pages 306–323. Springer, 2014.

[52] Y. Naito and K. Yasuda. New Bounds for Keyed Sponges with Extendable Output: Independence Between Capacity and Message Length. In T. Peyrin, editor, *FSE*, volume 9783 of *LNCS*, pages 3–22. Springer, 2016.

[53] 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.

[54] J. Patarin. The "Coefficients H" Technique. In R. M. Avanzi, L. Keliher, and F. Sica, editors, *SAC*, volume 5381 of *LNCS*, pages 328–345. Springer, 2008.

[55] H. Raddum. Algebraic Analysis of the Simon Block Cipher Family. In K. E. Lauter and F. Rodríguez-Henríquez, editors, *LATINCRYPT*, volume 9230 of *LNCS*, pages 157–169. Springer, 2015.

[56] P. Rogaway, M. Bellare, J. Black, and T. Krovetz. OCB: a block-cipher mode of operation for efficient authenticated encryption. In M. K. Reiter and P. Samarati, editors, *ACM-CCS*, pages 196–205. ACM, 2001.

[57] P. Rogaway and T. Shrimpton. A Provable-Security Treatment of the Key-Wrap Problem. In S. Vaudenay, editor, *EUROCRYPT*, volume 4004 of *LNCS*, pages 373–390. Springer, 2006.

[58] R. Rohit and G. Gong. Correlated Sequence Attack on Reduced-Round Simon-32/64 and Simeck-32/64. *IACR Cryptology ePrint Archive*, 2018:699, 2018.

[59] R. Rohit and S. Sarkar. [lwc-forum] ROUND 2 OFFICIAL COMMENT: Oribatida. NIST lwc forum mailing list, 17 September 17:09 2019.

[60] C. E. Shannon. Communication theory of secrecy systems. *The Bell system technical journal*, 28(4):656–715, 1949.

[61] H. Sui, W. Wu, L. Zhang, and D. Zhang. LAEM (Lightweight Authentication Encryption Mode). Technical report, Mar 25 2019. First-round submission to the NIST Lightweight Cryptography Competition. https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/LAEM-spec.pdf.

[62] Y. Todo and M. Morii. Bit-Based Division Property and Application to Simon Family. In T. Peyrin, editor, *FSE*, volume 9783 of *LNCS*, pages 357–377. Springer, 2016.

[63] X. Wang, B. Wu, L. Hou, and D. Lin. Automatic Search for Related-Key Differential Trails in SIMON-like Block Ciphers Based on MILP. In L. Chen, M. Manulis, and S. Schneider, editors, *ISC*, volume 11060 of *LNCS*, pages 116–131. Springer, 2018.

[64] Z. Xiang, W. Zhang, Z. Bao, and D. Lin. Applying MILP Method to Searching Integral Distinguishers Based on Division Property for 6 Lightweight Block Ciphers. In J. H. Cheon and T. Takagi, editors, *ASIACRYPT I*, volume 10031 of *LNCS*, pages 648–678, 2016.

[65] H. Zhang, W. Wu, and Y. Wang. Integral Attack Against Bit-Oriented Block Ciphers. In S. Kwon and A. Yun, editors, *ICISC*, volume 9558 of *LNCS*, pages 102–118. Springer, 2015.