RUP Security of the SAEF Authenticated Encryption mode

Elena Andreeva[1], Amit Singh Bhati[2], and Damian Vizár[3]

[1] Alpen-Adria University, Klagenfurt, Austria
[2] imec-COSIC, KU Leuven, Belgium
[3] CSEM, Switzerland.
elena.andreeva@aau.at, amitsingh.bhati@esat.kuleuven.be, damian.vizar@csem.ch

Abstract. ForkAE is a family of authenticated encryption (AE) schemes using a forkcipher as a building block. ForkAE was published in Asiacrypt'19 and is a second round candidate in the NIST lightweight cryptography process. ForkAE comes in several modes of operation: SAEF, PAEF, and rPAEF. SAEF is optimized for authenticated encryption of short messages and processes the message blocks in a sequential and *online* manner. SAEF requires a smaller internal state than its parallel sibling PAEF and is better fitted for devices with smaller footprint. At SAC 2020 it was shown that SAEF is also an *online nonce misuse-resistant* AE (OAE) and hence offers enhanced security against adversaries that make blockwise adaptive encryption queries. It has remained an open question if SAEF resists attacks against blockwise adaptive decryption adversaries, or more generally when the decrypted plaintext is released before the verification (RUP).

RUP security is a particularly relevant security target for lightweight (LW) implementations of AE schemes on memory-constrained devices or devices with stringent real-time requirements. Surprisingly, very few NIST lightweight AEAD candidates come with any provable guarantees against RUP.

In this work, we show that the SAEF mode of operation of the ForkAE family comes with integrity guarantees in the RUP setting. The RUP integrity (INT-RUP) property was defined by Andreeva et al. in Asiacrypt'14. Our INT-RUP proof is conducted using the coefficient H technique and it shows that, without any modifications, SAEF is INT-RUP secure up to the birthday bound, i.e., up to $2^{n/2}$ processed data blocks, where n is the block size of the forkcipher. The implication of our work is that SAEF is indeed RUP secure in the sense that the release of unverified plaintexts will not impact its ciphertext integrity.

Keywords: Authenticated encryption, forkcipher, lightweight cryptography, short messages, online, provable security, release of unverified plaintext, RUP.

1 Introduction

An authenticated encryption (AE) scheme provides message confidentiality and authenticity. The majority of AE schemes nowadays process the plaintext data along with two additional inputs: an associated data AD and a nonce N. The associated data is a piece of information, such as a packet header, that is sent in the clear and requires authentication whereas the nonce is a unique value that is used to offload the need for either maintaining a state or a random value. The formalisation of nonce-based authenticated encryption was introduced in 2002 by Rogaway [19].

The design and analysis of AE(AD) protocols have recently attracted a great deal of scientific attention, mostly driven by the past CAESAR competition (2014–2018) [7] and the ongoing NIST lightweight cryptography standardization process (2018–). The winners of CAESAR competition are defined in three categories depending on their use cases: 1. Ascon [11] and ACORN [21] (for resource constrained environments, lightweight); 2. AEGIS-128 [23] and OCB [16] (for high-performance); 3. Deoxys II [15], COLM [1], and MORUS [22] (for stronger security guarantees, defense in depth). The CAESAR winners provide various advantages over the current standards GCM (NIST SP 800-38D) and CCM (IEEE 802.11i, IPsec ESP and IKEv2) and serve as adoption recommendations by the cryptographic community for new applications and standards.

One of target properties for the *defense in depth* CAESAR category was defined as (limited damage under) integrity and confidentiality attacks in the release of unverified plaintext (RUP) setting. More precisely, integrity preservation despite RUP was considered as *critical* for defense in depth security whereas some limited RUP confidentiality damage was still permissible. The AEAD syntax and formal RUP security definitions were introduced by Andreeva et al. [3] in Asiacrypt 2014. First, they defined the security notion of plaintext awareness PA in two variants PA1 and PA2, and then proposed to combine PA1 along with IND-CPA to achieve confidentiality of an AE(AD) scheme. To achieve integrity of ciphertexts under RUP, they used INT-CTXT in the RUP setting, also known as the INT-RUP notion.

Intuitively, the notion of INT-RUP says that an adversary shall not be able to produce a valid forgery after consistent observations of decrypted chosen ciphertexts which have not passed successful verification (else the forgery would have already succeeded). Note that compared to the classical integrity notion where the decryption and verification combined oracle returns failure every time, in the RUP integrity setting the adversary has access to a separate decryption oracle which returns the decryptions of any chosen ciphertext and the verification oracle returns failure unless the forgery succeeds. To accommodate the decryption functionality, one needss to explicitly split the decryption and verification functionalities in the syntax of an AEAD. This split is natural and can be applied to all existing AEAD schemes.

Release of unverified plaintext presents a significant threat for many practical applications which allow for release unverified plaintext (RUP), that is the decrypted ciphertext is released before the verification has been completed. For example, when AE scheme is implemented on resource-constrained devices, like a smart card or RFID, it is almost impossible to store the entire temporary plaintext due to the limited buffer of the device. Another reason could be the real-time requirement for online applications which can not be satisfied if the decrypted ciphertext can only be release after verification.

Our Contributions. Th focus of our RUP-integrity security investigation is the SAEF mode from the ForkAE family. SAEF is a sequential and online nonce-based AEAD which is optimized for the processing of short messages and therefore suitable for lightweight applications where the predominant message size is just a few blocks. SAEF has been shown to achieve confidentiality and authenticity up to the birthday bound in [4]. Moreover, recently in [2] SAEF was proven to be secure when the nonces are repeated up to leakage of identical plaintext prefixes under the notion of OAE [13]. A consequence of this result is that SAEF resist attacks by blockwise adaptive adversaries and hence is suitable for lightweight applications with low latency and/or low memory requirements.

The latter results stand to prove the *defense in depth* resilience of SAEF against both nonce respecting and nonce repeating adversaries. In this work we further investigate the *defense in depth* provable security guaranteed offered by the SAEF mode against an extra strong adversary (compared to the classical nAE setting), namely one with access to a separate decryption oracle. We prove that SAEF indeed provides integrity in the RUP setting or satisfies the INT-RUP security notion.

Our result shows that the SAEF mode is provably INT-RUP secure *without* the need of applying any design modifications i.e. the integrity of SAEF in the presence of a stronger adversary decrypting unverified ciphertexts remains intact.

To complete our proof we use the coefficient H technique [17] as a main analysis tool. More concretely we prove that SAEF is INT-RUP secure up to $2^{n/2}$ blocks of processed data in total, where n is the block size of the underlying forkcipher. Our result further validates and enhances the suitability of SAEF in ForkAE for the *in depth security* protection.

Defense in depth comparison of SAEF in ForkAE with other NIST LW 2nd Round Candidates. Among the 32 candidates in the second round of the NIST lightweight competition there are only 6 AE modes (including SAEF) that come with claims above the conventional nAE security. We provide a comparison of these modes with respect to security properties beyond nAE aka *defense in depth* and summarize them into the Table 1.

The first property is OAE [12] which ensures that the AE mode can be implemented in an online manner with reasonable security guarantees i.e. the mode is secure against blockwise (hence also nonce-misusing) adversaries. The second property is Nonce-Misuse resilience [5] (NMR) which ensures that the mode provides reasonable security for a query even when the nonce is repeated in "other" queries i.e. security against a specific set of nonce-misusing adversaries. The third property is MRAE [20] which is a stronger version of NMR. This property ensures that the AE mode provides reasonable security guarantees against all types of nonce-misusing adversaries.

Note that MRAE is a stronger security definition when compared with OAE (which is stronger than NMR) but an MRAE-secure scheme can not be implemented securely in an online fashion which makes OAE security the optimal choice in such applications with online requirements.

The fourth property is INT-RUP [3] which ensures that the AE mode is secure w.r.t. integrity even when the release of unverified plaintexts is allowed i.e. integrity against adversaries decrypting unverified plaintexts. The fifth property is Plaintext-Awareness (PA) [3] which when combined with IND-CPA ensures that the AE mode is secure w.r.t. confidentiality even when the release of unverified plaintexts is allowed i.e. confidentiality against adversaries decrypting unverified plaintexts. Note that PA is a very strong notion of security and that no OAE-secure scheme can achieve it without weakening its existing definition.

The last remaining property from the table is single pass over data. This property ensures that the mode only require one pass over the data throughout its execution. Again, note that OAE implies single pass but the opposite implication is not always true.

Table 1: Comparison of SAEF with NIST LW submissions with beyond nAE security.

	ESTATE [9]	ROMULUS-M [14]	Spook [6]	Oribatida [8]	LOCUS,LOTUS [10]	SAEF [4]
OAE [12]	✗	✗	✗	✗	✗	✓
MRAE [20]	✓	✓	✗	✗	✗	✗
NMR [5]	✓	✓	✓	✗	✗	✓
INT-RUP [3]	✓	✓	✗	✓	✓	✓
PA [3]	✓	✓	✗	✗	✗	✗
Single pass over data	✗	✗	✓	✓	✓	✓

We further provide a comparison between SAEF and the five modes from NIST candidates with INT-RUP security in Table 2 regarding some of their parametric choices and rate. Our comparison shows that SAEF comes with a comparable state and better rate of 1 as compared to the rest of the illustrated candidates. Here rate means the number of message bits processed per bit of the underlying primitive[4] block. To exemplify, if the rate is 1/2 for a given mode Π then it processes $n/2$ message bits per n-bit block of the underlying primitive.

Table 2: Comparison of SAEF with the other INT-RUP secure NIST LW submissions regarding for their comparable instances. Each entry of this table except the Rate column is in bits. Here n and t represent the block size and the tweak size of the corresponding underlying primitive, respectively.

Parameter	Nonce size	Key size	Tag size	n	t	State	Rate	INT-RUP
ESTATE [9]	128	128	128	128	4	260	1/2	64
ROMULUS-M [14]	128	128	128	128	192	448	1/2	≥ 64
Oribatida [8]	128	128	128	256	0	320	1/2	64
LOCUS [10]	128	128	64	64	4	324	1/2	64
LOTUS [10]	128	128	64	64	4	384	1/2	64
SAEF-128-192 [4]	60	128	128	128	64	448	1	64
SAEF-128-256 [4]	124	128	128	128	128	512	1	64

2 Preliminaries

Strings. All strings are considered as binary strings. The set of all strings of all possible lengths is denoted by $\{0,1\}^*$ and the set of all strings of length n (a positive integer) is denoted by $\{0,1\}^n$. We let $\{0,1\}^{\leq n} = \bigcup_{i=0}^{n}\{0,1\}^n$. We denote by $\mathrm{Perm}(n)$ the set of all permutations of $\{0,1\}^n$ and by $\mathrm{Func}(m,n)$ the set of all functions with domain $\{0,1\}^m$ and range $\{0,1\}^n$.

For a string X of ℓ bits, we denote by $X[i]$ the i^{th} bit of X for $i = 0,\dots,\ell-1$ (counting from left to right) and define $X[i\dots j] = X[i]\|X[i+1]\|\dots\|X[j]$ for $0 \leq i < j < \ell$. We let $\mathsf{left}_\ell(X) = X[0\dots(\ell{-}1)]$ denote the ℓ leftmost bits of X and $\mathsf{right}_r(X) = X[(|X|{-}r)\dots(|X|{-}1)]$ the r rightmost bits of X, such that $X = \mathsf{left}_\chi(X)\|\mathsf{right}_{|X|-\chi}(X)$ for all $0 \leq \chi \leq |X|$. Given an integer (possibly implicit) $n > 0$ and an $X \in \{0,1\}^*$, we use $X\|10^*$ to denote $X\|10^{n-(|X| \bmod n)-1}$ for simplicity. Further, for the same integer n, we define $\mathsf{pad10}(X) = X\|10^*$ that returns X if $|X| \equiv 0 \pmod{n}$ and $X\|10^*$ otherwise.

String partitioning. We fix an arbitrary integer n for this work and call it the block size. For any string X, We let $|X|_n = \lceil |X|/n \rceil$ and use $X_1,\dots,X_x,X_* \xleftarrow{n} X$ to define the partitioning of X into n-bit blocks, such that $X = X_1\|\dots\|X_x\|X_*$ with $|X_i| = n$ for $i = 1,\dots,x$ and $0 < |X_*| \leq n$. Hence, $x = |X|_n - 1$.

Blocks. We use B_n to denote the set of all n-bit strings (or blocks) i.e. $\{0,1\}^n$. We define $\mathsf{B}_n^* = \{\varepsilon\} \cup \bigcup_{i=1}^{\infty} \mathsf{B}_n^i$ where ε denotes the empty string with length 0. We say a string X is "n-aligned" iff $X \in \mathsf{B}_n^*$. We let X_i denote the i^{th} n-bit block of an n-aligned string X. For two distinct and n-aligned strings $X, Y \in \mathsf{B}_n^*$ with $|X|_n \leq |Y|_n$ w.l.o.g, we let $\mathsf{llcp}_n(X,Y) = \max\{1 \leq i \leq |X|_n | X_j = Y_j \text{ for } 1 \leq j \leq i\}$ denote the length of the longest common prefix (in n-bit blocks) of X and Y.

Miscellaneous. We let $X \leftarrow\!\!\$\ \mathcal{X}$ denote the sampling of an element X from a finite set $\mathcal{X}$ under the uniform distribution. We let $(p)_q$ denote the falling factorial $p\cdot(p-1)\cdot(p-2)\cdot\ldots\cdot(p-q+1)$

[4] We resort to indicating rate in terms of opaque primitive calls, as the primitives used to instantiate these schemes in the LWC project do not allow for a meaningful, more fine-grained, yet simple comparison.

where $(p)_0 = 1$. We define a predicate $\mathsf{P}(x)$ as $\mathsf{P}(x) = 1$ if it is true and $\mathsf{P}(x) = 0$ if it is false. We use lexicographic comparison for integer tuples (to exemplify, $(i', j') < (i, j)$ iff $i' < i$ or $i' = i$ and $j' < j$). The symbol $\bot$ denotes an undefined value or an error.

2.1 Syntax of AEAD under RUP setting

A nonce-based AEAD scheme under the RUP setting is a tuple $\mathit{\Pi} = (\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{V})$. The key space $\mathcal{K}$ is a finite set. The deterministic encryption algorithm $\mathcal{E} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{C}$ maps a secret key K, a nonce N, an associated data A and a message M to a ciphertext $C = \mathcal{E}(K, N, A, M)$. The nonce, AD and message domains are all subsets of $\{0, 1\}^*$. The deterministic decryption algorithm $\mathcal{D} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{C} \to \mathcal{M}$ takes a tuple (K, N, A, C) and returns a message $M \in \mathcal{M}$. The deterministic verification algorithm $\mathcal{V} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{C} \to \{\top, \bot\}$ takes a tuple (K, N, A, C) and either returns the distinguished symbol $\top$ to indicate a successful authentication or $\bot$ to indicate an authentication error.

We require that for every $M \in \mathcal{M}$, we have $\{0, 1\}^{|M|} \subseteq \mathcal{M}$ (i.e. for any integer m, either all or no strings of length m belong to $\mathcal{M}$) and that for all $K, N, A, M \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M}$ we have $|\mathcal{E}(K, N, A, M)| = |M| + \theta$ for some non-negative integer θ called the stretch of $\mathit{\Pi}$. For correctness of $\mathit{\Pi}$, we require that for all $K, N, A, M \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M}$ we have $M = \mathcal{D}(K, N, A, \mathcal{E}(K, N, A, M))$ and $\top = \mathcal{V}(K, N, A, \mathcal{E}(K, N, A, M))$. We let $\mathcal{E}_K(N, A, M) = \mathcal{E}(K, N, A, M)$,$\mathcal{D}_K(N, A, C) = \mathcal{D}(K, N, A, C)$ and $\mathcal{V}_K(N, A, C) = \mathcal{V}(K, N, A, C)$.

2.2 Security definition under RUP setting

INT-RUP Authenticity. Traditional requirements for the integrity of an AE scheme can be achieved by the INT-CTXT notion, where the adversary is allowed to make encryption and decryption queries, but the decryption oracle always returns $\bot$. However, under the RUP setting, where the adversary is allowed to observe the unverified plaintext, the integrity requirements as in INT-CTXT need to be modified. The following definition from the work of Andreeva et al. [3] presents the targeted notion of integrity under the RUP setting.

Definition 1 (INT-RUP Advantage). *Let $\mathcal{A}$ be a computationally bounded adversary with access to an encryption, a decryption and a verification oracle namely $\mathcal{E}, \mathcal{D}$, and $\mathcal{V}$ for $\mathit{\Pi}$ then the INT-RUP advantage of $\mathcal{A}$ against $\mathit{\Pi}$ for some $K \leftarrow^{\$} \mathcal{K}$ is defined as*

$$\mathbf{Adv}_{\mathit{\Pi}}^{\mathsf{INT-RUP}}(\mathcal{A}) = \Pr[\mathcal{A}^{\mathcal{E}_K, \mathcal{D}_K, \mathcal{V}_K} \ forges].$$

Here we say $\mathcal{A}$ forges if $\mathcal{A}$ comes up with a challenge ciphertext as the forgery which is not an output from the queries of encryption oracle $\mathcal{E}_K$ but when queried to verification oracle $\mathcal{V}_K$ it results into $\top$ i.e. the forgery is success.

2.3 Forkcipher

We follow the formalism of forkcipher provided by Andreeva et al. [18]. Informally, a forkcipher F is a tweakable symmetric primitive which maps a secret key K, a tweak T and an n-bit input block M to two n-bit ciphertext blocks C_0 and C_1, such that C_0 and C_1 are two independent permutations of M.

Syntax. A forkcipher is formally defined by a pair of deterministic algorithms; named the encryption algorithm: $\mathsf{F} : \{0,1\}^k \times \mathcal{T} \times \{0,1\}^n \times \{0,1,\mathsf{b}\} \rightarrow \{0,1\}^n \cup \{0,1\}^n \times \{0,1\}^n$ and the inversion algorithm: $\mathsf{F}^{-1}\{0,1\}^k \times \mathcal{T} \times \{0,1\}^n \times \{0,1\} \times \{\mathsf{i},\mathsf{o},\mathsf{b}\} \rightarrow \{0,1\}^n \cup \{0,1\}^n \times \{0,1\}^n$.

The encryption algorithm takes a key K, a tweak $\mathsf{T} \in \mathcal{T}$, a plaintext block M and an output selector s, and outputs the left n-bit ciphertext block C_0 if $s = 0$, the right n-bit ciphertext block C_1 if $s = 1$, and both the blocks C_0, C_1 if $s = \mathsf{b}$. We use $\mathsf{F}(K, \mathsf{T}, M, s) = \mathsf{F}_K(\mathsf{T}, M, s) = \mathsf{F}_K^{\mathsf{T}}(M, s) = \mathsf{F}_K^{\mathsf{T},s}(M)$ interchangeably.

Similarly, the inversion algorithm takes a key K, a tweak T, a ciphertext block C (either left or right half of the output block), an indicator b to indicate whether the fed block should be treated as the left or the right ciphertext block and an output selector s, and outputs the plaintext block M if $s = \mathsf{i}$, the other ciphertext block C' if $s = \mathsf{o}$, and both blocks M, C' if $s = \mathsf{b}$. We use $\mathsf{F}^{-1}(K, \mathsf{T}, M, b, s) = \mathsf{F}^{-1}{}_K(\mathsf{T}, M, b, s) = \mathsf{F}^{-1}{}_K^{\mathsf{T}}(M, b, s) = \mathsf{F}^{-1}{}_K^{\mathsf{T},b,s}(M)$ interchangeably.

We say a tweakable forkcipher F is correct if for each pair of key and tweak, the forkcipher applies two independent permutations on the input to produce the corresponding two output blocks. Formally, for every tuple K, T, M, β with $K \in \{0,1\}^k, \mathsf{T} \in \mathcal{T}, M \in \{0,1\}^n$ and $\beta \in \{0,1\}$ the forkcipher must satisfy the following conditions: (i) $\mathsf{F}^{-1}(K, \mathsf{T}, \mathsf{F}(K, \mathsf{T}, M, \beta), \beta, \mathsf{i}) = M$, and (ii) $\mathsf{F}^{-1}(K, \mathsf{T}, \mathsf{F}(K, \mathsf{T}, M, \beta), \beta, \mathsf{o}) = \mathsf{F}(K, \mathsf{T}, M, \beta \oplus 1)$, and (iii) $(\mathsf{F}(K, \mathsf{T}, M, 0),\ \mathsf{F}(K, \mathsf{T}, M, 1)) = \mathsf{F}(K, \mathsf{T}, M, \mathsf{b})$, and (iv) $\big(\mathsf{F}^{-1}(K, \mathsf{T}, C, \beta, \mathsf{i}), \mathsf{F}^{-1}(K, \mathsf{T}, C, \beta, \mathsf{o})\big) = \mathsf{F}^{-1}(K, \mathsf{T}, C, \beta, \mathsf{b})$.

For the rest of the paper, we assume that $\mathcal{T} = \{0,1\}^t$ for some positive t. We call k, n and t the keysize, blocksize and tweaksize of F, respectively.

Forkcipher Security. The security of a forkcipher F is defined as the indistinguishability between the real **prtfp-real**$_{\mathsf{F}}$ and ideal **prtfp-ideal**$_{\mathsf{F}}$ worlds when adversary accesses either worlds in a chosen ciphertext fashion. In the real world, the forkcipher oracle implements the true F algorithm faithfully, whereas in the latter world, the oracle replaces F by two tweakable random permutations $\pi_{\mathsf{T},0}, \pi_{\mathsf{T},1} \leftarrow\!\!\$\ \mathrm{Perm}(n)$ for $\mathsf{T} \in \mathcal{T}$. We then define the advantage of $\mathcal{A}$ as:

$$\mathbf{Adv}_{\mathsf{F}}^{\mathrm{prtfp}}(\mathcal{A}) = \Pr[\mathcal{A}^{\textbf{prtfp-real}_{\mathsf{F}}} \Rightarrow 1] - \Pr[\mathcal{A}^{\textbf{prtfp-ideal}_{\mathsf{F}}} \Rightarrow 1].$$

2.4 Coefficient H Technique

The coefficient H is a simple but powerful proof technique by Patarin [17]. It is often used to prove indistinguishability of a provided construction from an idealized object for an information-theoretic adversary. Coefficient-H based proofs use the concept of "transcripts". A transcript is defined as a complete record of the interaction of an adversary $\mathcal{A}$ with its oracles in the indistinguishability experiment. For example, if (M_i, C_i) represents the input and output of the i-th query of $\mathcal{A}$ to its oracle and the total number of queries made by $\mathcal{A}$ is q then the corresponding transcript (denoted by τ) is defined as $\tau = \langle (M_1, C_1), \ldots, (M_q, C_q) \rangle$. The goal of an adversary $\mathcal{A}$ is to distinguish interactions in the real world $\mathcal{O}_{\mathsf{real}}$ from the ones in ideal world $\mathcal{O}_{\mathsf{ideal}}$.

We denote the distribution of transcripts in the real and the ideal world by Θ_{real} and Θ_{ideal}, respectively. We call a transcript τ *attainable* if the probability of achieving τ in the ideal world is non-zero. Further, w.l.o.g. we also assume that $\mathcal{A}$ does not make any duplicate or prohibited queries. We can now state the fundamental Lemma of coefficient H technique.

Lemma 1 (Fundamental Lemma of the coefficient H Technique [17]). *Consider that the set of attainable transcripts is partitioned into two disjoint sets $\mathcal{T}_{\mathsf{good}}$ and $\mathcal{T}_{\mathsf{bad}}$. Also, as-*

sume there exist $\epsilon_1, \epsilon_2 \geq 0$ *such that for any transcript* $\tau \in \mathcal{T}_{\mathit{good}}$*, we have* $\frac{\Pr[\Theta_{\mathsf{real}}=\tau]}{\Pr[\Theta_{\mathsf{ideal}}=\tau]} \geq 1-\epsilon_1$*, and* $\Pr[\Theta_{\mathsf{ideal}} \in \mathcal{T}_{\mathit{bad}}] \leq \epsilon_2$*. Then, for all adversaries* $\mathcal{A}$*, it holds that*

$$|\Pr[\mathcal{A}^{\mathcal{O}_{\mathsf{real}}}] - \Pr[\mathcal{A}^{\mathcal{O}_{\mathsf{ideal}}}]| \leq \epsilon_1 + \epsilon_2.$$

3 SAEF and its RUP Security

SAEF (short for Sequential AE from a Forkcipher) is a nonce-based AEAD scheme that uses a tweakable forkcipher F (as defined in Section 2.3) as an underlying primitive with $\mathcal{T} = \{0,1\}^t$ for a positive $t \leq n$. SAEF$[\mathsf{F}] = (\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{V})$ has a key space $\mathcal{K} = \{0,1\}^k$, nonce space $\mathcal{N} = \{0,1\}^{t-4}$, and the AD and message spaces are both $\{0,1\}^*$. The ciphertext expansion of SAEF is n bits. The encryption, decryption and verification algorithms are given in Figure 2 and the encryption algorithm is illustrated in Figure 1. Note that the earlier SAEF representation [4] did not have an explicit decryption and verification functionality separation. The present syntax splits explicitly these functionalities and introduces no change to the actual input and output behavior of the SAEF algorithm.

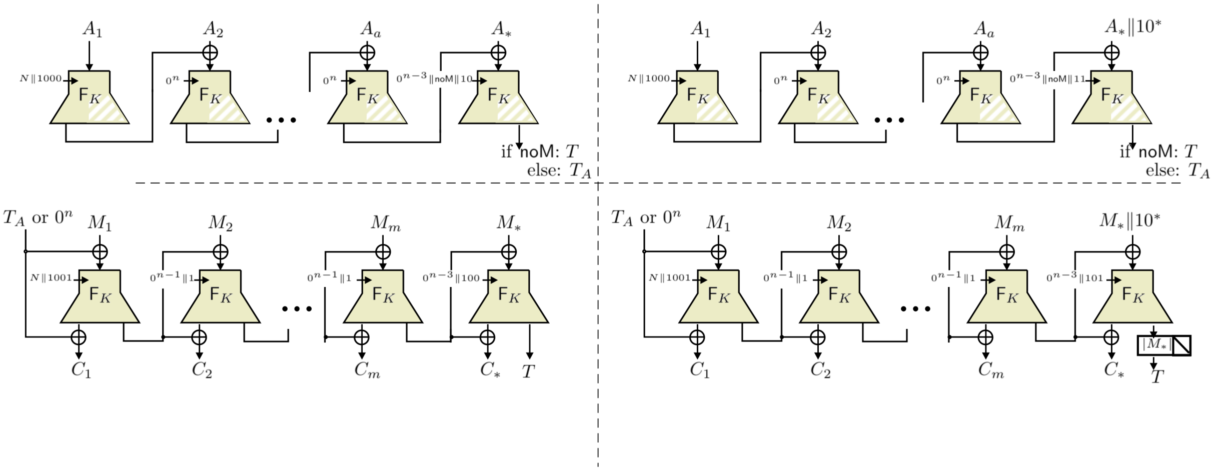

Fig. 1: The encryption algorithm of SAEF[F] mode. The bit $\mathsf{noM} = 1$ iff $|M| = 0$. The picture illustrates the processing of AD when length of AD is a multiple of n (**top left**) and when the length of AD is not a multiple of n (**top right**), and the processing of the message when length of the message is a multiple of n (**bottom left**) and when the length of message is not a multiple of n (**bottom right**). The white hatching denotes that an output block is not computed.

SAEF processes an encryption query in blocks of n bits (in order), with first AD and then the message. It uses single forkcipher call for each block. These forkcipher calls are tweaked by composing: (1) either the nonce followed by a 1-bit (for the first F call of the query) or the string 0^{t-3}, (2) three-bit flag f.

This flag f is used to ensure proper domain separation for various "types" of blocks in the encryption algorithm. The values of f from the set $\{000, 010, 011, 110, 111, 001, 100, 101\}$ are respectively used, when processing non-final AD block, the last n-bit long AD block, the last AD block of $< n$ bits, the last AD block of n bits to produce tag, the last AD block of $< n$ bits to produce tag, non-final message block, the last n-bit message block and the last message block of $< n$ bits.

The right output block of every F call is used as a chaining value to mask either the input (in the case of AD processing) of the following F call or both the input and output (in the case of message processing) of the following F call. The first F call of every query is not masked but contains the nonce in the tweak. The tag for a query is defined as the (possibly truncated) last "right" output block of F. In case of truncation, message padding is used for partial integrity check of the ciphertext. For a decryption (respectively verification) query, the processing of input blocks is similar to the encryption, but now with the chaining values in the message processing part are computed with the "inverse" F algorithm. These chaining values are used (similar to the encryption algorithm) to compute the corresponding plaintext blocks (respectively to verify the final tag).

```
function $\mathcal{E}(K, N, A, M)$
    $A_1, \dots, A_a, A_* \xleftarrow{n} A$
    $M_1, \dots, M_m, M_* \xleftarrow{n} M$
    $\mathsf{noM} \leftarrow 0$
    if $|M| = 0$ then $\mathsf{noM} \leftarrow 1$
    $\Delta \leftarrow 0^n$; $\mathsf{T} \leftarrow N \| 0^{t-4} \| 1$
    for $i \leftarrow 1$ to $a$ do
        $\mathsf{T} \leftarrow \mathsf{T} \| 000$
        $\Delta \leftarrow \mathsf{F}_K^{\mathsf{T},0}(A_i \oplus \Delta)$
        $\mathsf{T} \leftarrow 0^{t-3}$
    end for
    if $|A_*| = n$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| \mathsf{noM} \| 10$
        $\Delta \leftarrow \mathsf{F}_K^{\mathsf{T},0}(A_* \oplus \Delta)$
        $\mathsf{T} \leftarrow 0^{t-3}$
    else if $|A_*| > 0$ or $|M| = 0$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| \mathsf{noM} \| 11$
        $\Delta \leftarrow \mathsf{F}_K^{\mathsf{T},0}((A_* \| 10^*) \oplus \Delta)$
        $\mathsf{T} \leftarrow 0^{t-3}$
    end if                    ▷ Do nothing if $A = \varepsilon, M \neq \varepsilon$
    for $i \leftarrow 1$ to $m$ do
        $\mathsf{T} \leftarrow \mathsf{T} \| 001$
        $C_i, \Delta \leftarrow \mathsf{F}_K^{\mathsf{T},\mathsf{b}}(M_i \oplus \Delta) \oplus (\Delta, 0^n)$
        $\mathsf{T} \leftarrow 0^{t-3}$
    end for
    if $|M_*| = n$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| 100$
    else if $|M_*| > 0$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| 101$
    else
        return $\Delta$
    end if
    $C_*, T \leftarrow \mathsf{F}_K^{\mathsf{T},\mathsf{b}}(\mathsf{pad10}(M_*) \oplus \Delta) \oplus (\Delta \| 0^n)$
    return $C_1 \| \dots \| C_m \| C_* \| \mathsf{left}_{|M_*|}(T)$
end function
```

```
function $\mathcal{D}(K, N, A, C)$
    $A_1, \dots, A_a, A_* \xleftarrow{n} A$
    $C_1, \dots, C_m, C_*, T \xleftarrow{n} C$
    $\mathsf{noM} \leftarrow 0$
    if $|C| = n$ then $\mathsf{noM} \leftarrow 1$
    $\Delta \leftarrow 0^n$; $\mathsf{T} \leftarrow N \| 0^{t-4} \| 1$
    for $i \leftarrow 1$ to $a$ do
        $\mathsf{T} \leftarrow \mathsf{T} \| 000$
        $\Delta \leftarrow \mathsf{F}_K^{\mathsf{T},0}(A_i \oplus \Delta)$
        $\mathsf{T} \leftarrow 0^{t-3}$
    end for
    if $|A_*| = n$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| \mathsf{noM} \| 10$
        $\Delta \leftarrow \mathsf{F}_K^{\mathsf{T},0}(A_* \oplus \Delta)$
        $\mathsf{T} \leftarrow 0^{t-3}$
    else if $|A_*| > 0$ or $|T| = 0$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| \mathsf{noM} \| 11$
        $\Delta \leftarrow \mathsf{F}_K^{\mathsf{T},0}((A_* \| 10^*) \oplus \Delta)$
        $\mathsf{T} \leftarrow 0^{t-3}$
    end if                    ▷ Do nothing if $A = \varepsilon, M \neq \varepsilon$
    for $i \leftarrow 1$ to $m$ do
        $\mathsf{T} \leftarrow \mathsf{T} \| 001$
        $M_i, \Delta \leftarrow \mathsf{F^{-1}}_K^{\mathsf{T},0,\mathsf{b}}(C_i \oplus \Delta) \oplus (\Delta, 0^n)$
        $\mathsf{T} \leftarrow 0^{t-3}$
    end for
    if $|T| = n$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| 100$
    else if $|T| > 0$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| 101$
    else
        return $\varepsilon$
    end if
    $M_*, T' \leftarrow \mathsf{F^{-1}}_K^{\mathsf{T},0,\mathsf{b}}(C_* \oplus \Delta) \oplus (\Delta, 0^n)$
    return $M_1 \| \dots \| M_m \| M_*$
end function
```

```
function $\mathcal{V}(K, N, A, C)$
    $A_1, \dots, A_a, A_* \xleftarrow{n} A$
    $C_1, \dots, C_m, C_*, T \xleftarrow{n} C$
    $\mathsf{noM} \leftarrow 0$
    if $|C| = n$ then $\mathsf{noM} \leftarrow 1$
    $\Delta \leftarrow 0^n$; $\mathsf{T} \leftarrow N \| 0^{t-4} \| 1$
    for $i \leftarrow 1$ to $a$ do
        $\mathsf{T} \leftarrow \mathsf{T} \| 000$
        $\Delta \leftarrow \mathsf{F}_K^{\mathsf{T},0}(A_i \oplus \Delta)$
        $\mathsf{T} \leftarrow 0^{t-3}$
    end for
    if $|A_*| = n$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| \mathsf{noM} \| 10$
        $\Delta \leftarrow \mathsf{F}_K^{\mathsf{T},0}(A_* \oplus \Delta)$
        $\mathsf{T} \leftarrow 0^{t-3}$
    else if $|A_*| > 0$ or $|T| = 0$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| \mathsf{noM} \| 11$
        $\Delta \leftarrow \mathsf{F}_K^{\mathsf{T},0}((A_* \| 10^*) \oplus \Delta)$
        $\mathsf{T} \leftarrow 0^{t-3}$
    end if                    ▷ Do nothing if $A = \varepsilon, M \neq \varepsilon$
    for $i \leftarrow 1$ to $m$ do
        $\mathsf{T} \leftarrow \mathsf{T} \| 001$
        $M_i, \Delta \leftarrow \mathsf{F^{-1}}_K^{\mathsf{T},0,\mathsf{b}}(C_i \oplus \Delta) \oplus (\Delta, 0^n)$
        $\mathsf{T} \leftarrow 0^{t-3}$
    end for
    if $|T| = n$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| 100$
    else if $|T| > 0$ then
        $\mathsf{T} \leftarrow \mathsf{T} \| 101$
    else
        if $C_* \neq \Delta$ then return $\perp$
    end if
    $M_*, T' \leftarrow \mathsf{F^{-1}}_K^{\mathsf{T},0,\mathsf{b}}(C_* \oplus \Delta) \oplus (\Delta, 0^n)$
    $T' \leftarrow \mathsf{left}_{|T|}(T')$; $P \leftarrow \mathsf{right}_{n-|T|}(M_*)$
    if $T' \neq T$ then
        return $\perp$
    else if $P \neq \mathsf{left}_{n-|T|}(10^{n-1})$ then
        return $\perp$
    else
        return $\top$
    end if
end function
```

Fig. 2: The SAEF[F] AEAD scheme.

3.1 Security of SAEF

In the work [2], Andreeva et al. proved that SAEF achieves OAE confidentiality and integrity up to the birthday bound under Nonce-Misuse. However, there have been no investigations into the security of SAEF under release of unverified plaintext (i.e., if the decrypted plaintext is released before the tag verification). We state the formal claim about the integrity of SAEF under RUP in Theorem 1.

Theorem 1. *Let* F *be a tweakable forkcipher with* $\mathcal{T} = \{0,1\}^t$*. Then for any nonce-misuse adversary* $\mathcal{A}$ *who makes at most* q_e *encryption, at most* q_d *decryption and at most* q_v *verification queries with* $q_e + q_d \leq 2^{n-1}$ *such that the total number of forkcipher calls induced by all the queries is at most* σ*, we have*

$$\mathbf{Adv}^{\mathsf{INT-RUP}}_{\mathrm{SAEF}[\mathsf{F}]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{B}) + \frac{\sigma^2 + 4 \cdot q_d q_v}{2^n}$$

for some adversary $\mathcal{B}$*, making at most* 2σ *queries, and running in time given by the running time of* $\mathcal{A}$ *plus* $\gamma \cdot \sigma$ *for some "small" constant* γ*.*

The proof of Theorem 1 follows in Section 3.2.

We use a similar proof approach and case analysis for SAEF RUP integrity as provided by Andreeva et al. in [2] for SAEF integrity under Nonce-Misuse. Our proof and analysis derives the claimed security bound by utilizing the key properties of SAEF that are results of its sequential structure. One of such property is the preservation of common length prefix over queries (encryption or decryption). This can better be understood by an example. Consider two encryption queries (w.l.o.g.) with same nonce N, same associated data of two blocks $A_1 \| A_2$ but different messages (of two blocks) $M_1 \| M_2$ and $M_0 \| M_2$ where $M_1 \neq M_0$. One can follow the encryption algorithm (as shown in Figure 2) for these two queries and can notice that the values of the F tweak strings and of $\varDelta$ masks used to process the input data upto block A_2 are same for both the queries. However, these equalities of tweak strings and the $\varDelta$ masks for A_2 when combined with the inequality $M_1 \neq M_0$ imply that the next input blocks in these encryption queries will necessarily differ i.e. $\varDelta + M_1 \neq \varDelta + M_0$. This will randomize the $\varDelta$ masks that are used to process the next input blocks. In short, the internal variables of SAEF's encryption algorithm preserve a common prefix length over queries and get randomized after that. We use this property of SAEF to prove its RUP integrity.

3.2 Proof of Theorem 1

We switch to an alternative definition of RUP integrity through indistinguishability, which is equivalent with the notion introduced in Section 3. We define two games, $\mathsf{INT\text{-}RUP}\text{-}\mathbf{real}_{\Pi}$ and $\mathsf{INT\text{-}RUP}\text{-}\mathbf{ideal}_{\Pi}$. In both games $\mathcal{A}$ is given access to an encryption, a decryption and a verification oracle. In the game $\mathsf{INT\text{-}RUP}\text{-}\mathbf{real}_{\Pi}$, all three oracles faithfully implement the corresponding algorithms of SAEF using the same randomly sampled secret key, except that the verification oracle returns $\top$ in case of a successful forgery, and $\bot$ otherwise. In the game $\mathsf{INT\text{-}RUP}\text{-}\mathbf{ideal}_{\Pi}$, the encryption and decryption oracles are same as in $\mathsf{INT\text{-}RUP}\text{-}\mathbf{real}_{\Pi}$ but the verification oracle always returns $\bot$. We claim that $\mathbf{Adv}^{\mathsf{INT\text{-}RUP}}_{\Pi}(\mathcal{A}) = \Pr[\mathcal{A}^{\mathsf{INT\text{-}RUP}\text{-}\mathbf{real}_{\Pi}}] - \Pr[\mathcal{A}^{\mathsf{INT\text{-}RUP}\text{-}\mathbf{ideal}_{\Pi}}]$.

One can easily prove the equality of these two definitions, by establishing inequalities in both directions. An adversary $\mathcal{A}$ playing the game $\mathsf{INT\text{-}RUP}_{\Pi}$ can be used to derive a distinguishing adversary $\mathcal{B}$ which forwards $\mathcal{A}$'s queries and outputs 1 if $\mathcal{A}$ forges, achieving the same advantage as $\mathcal{A}$. For the other direction, we can derive an adversary $\mathcal{A}$ for game $\mathsf{INT\text{-}RUP}_{\Pi}$ from an indistinguishability adversary $\mathcal{B}$, which forwards $\mathcal{B}$'s queries and automatically wins if $\mathcal{B}$ produces a valid forgery. Clearly, $\mathcal{A}$ achieves the same advantage as $\mathcal{B}$ because if no forgery occurs, the game $\textsf{INT-RUP-}\mathbf{real}_{\Pi}$ is indistinguishable from $\textsf{INT-RUP-}\mathbf{ideal}_{\Pi}$.

Replacing F. We first replace F with a pair of independent random tweakable permutations $\pi_0 = (\pi_{\mathsf{T},0} \leftarrow\$ \mathrm{Perm}(n))_{\mathsf{T}\in\{0,1\}^t}$ and $\pi_1 = (\pi_{\mathsf{T},1} \leftarrow\$ \mathrm{Perm}(n))_{\mathsf{T}\in\{0,1\}^t}$ and let $\mathrm{SAEF}[(\pi_0, \pi_1)]$ denote the SAEF mode that uses π_0, π_1 instead of F, which yields $\mathbf{Adv}^{\textsf{INT-RUP}}_{\mathrm{SAEF}[\mathsf{F}]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{B}) + \mathbf{Adv}^{\textsf{INT-RUP}}_{\mathrm{SAEF}[(\pi_0,\pi_1)]}(\mathcal{A})$.

Now, the adversary is left with the goal of distinguishing between the games $\textsf{INT-RUP-}\mathbf{real}_{\mathrm{SAEF}[(\pi_0,\pi_1)]}$ and $\textsf{INT-RUP-}\mathbf{ideal}_{\mathrm{SAEF}[(\pi_0,\pi_1)]}$. For simplicity, we denote these games by "real-int world" and "ideal-int world", respectively. Hence, we want to bound $\mathbf{Adv}^{\textsf{INT-RUP}}_{\mathrm{SAEF}[(\pi_0,\pi_1)]}(\mathcal{A}) = \Pr[\mathcal{A}^{\textsf{INT-RUP-}\mathbf{real}_{\mathrm{SAEF}[(\pi_0,\pi_1)]}}] - \Pr[\mathcal{A}^{\textsf{INT-RUP-}\mathbf{ideal}_{\mathrm{SAEF}[(\pi_0,\pi_1)]}}]$.

Transcripts. Following the coefficients H technique [17], we describe the interactions of $\mathcal{A}$ with its oracles in a *transcript*:

$$\tau = \langle (N^i, A^i, M^i, C^i)_{i=1}^{q_e}, (\bar{N}^i, \bar{A}^i, \bar{M}^i, \bar{C}^i)_{i=1}^{q_d}, (\tilde{N}^i, \tilde{A}^i, \tilde{C}^i, b^i)_{i=1}^{q_v} \rangle$$

For the i^{th} query to the encryption oracle with input (N^i, A^i, M^i) and output C^i, SAEF internally processes A^i, M^i and C^i in blocks $A^i_1, \ldots, A^i_{a^i}, A^i_*$, and $M^i_1, \ldots, M^i_{m^i}, M^i_*$, and $C^i_1, \ldots, C^i_{m^i}, C^i_*, T^i$, respectively (defined as per the encryption algorithm of SAEF, Figure 2). Here a^i and m^i represent, the length of A^i and M^i in n-bit, respectively. SAEF also processes and uses the internal chaining values as the whitening masks for encryption which we denote here by a sequence of $\varDelta$s. The whitening masks used to process $A^i_1, \ldots, A^i_{a^i}, A^i_*$ are denoted by $\varDelta^i_1, \ldots, \varDelta^i_{a^i+1}$, respectively, and the whitening masks used to process the blocks $M^i_1, \ldots, M^i_{m^i}, M^i_*$ are denoted by $\varDelta^i_{a^i+2}, \ldots, \varDelta^i_{a^i+m^i+2}$, respectively.

Similarly, for the i^{th} query to the decryption oracle with input $(\bar{N}^i, \bar{A}^i, \bar{C}^i)$ and output $\bar{M}^i$, SAEF internally processes $\bar{A}^i, \bar{C}^i$ and $\bar{M}^i$ in blocks $\bar{A}^i_1, \ldots, \bar{A}^i_{\bar{a}^i}, \bar{A}^i_*$, and $\bar{C}^i_1, \ldots, \bar{C}^i_{\bar{m}^i}, \bar{C}^i_*, \bar{T}^i$, and $\bar{M}^i_1, \ldots, \bar{M}^i_{\bar{m}^i}, \bar{M}^i_*$, respectively (defined as per the decryption algorithm of SAEF, Figure 2). Here $\bar{a}^i$ and $\bar{m}^i$ represent, the length of $\bar{A}^i$ and $\bar{C}^i$ (excluding the tag from the count) in n-bit, respectively. SAEF also processes and uses the internal chaining values as the whitening masks for decryption which we denote here by a sequence of $\bar{\varDelta}$s. The whitening masks used to process $\bar{A}^i_1, \ldots, \bar{A}^i_{\bar{a}^i}, \bar{A}^i_*$ are denoted by $\bar{\varDelta}^i_1, \ldots, \bar{\varDelta}^i_{\bar{a}^i+1}$, respectively, and the whitening masks used to process the blocks $\bar{C}^i_1, \ldots, \bar{C}^i_{\bar{m}^i}, \bar{C}^i_*$ are denoted by $\bar{\varDelta}^i_{\bar{a}^i+2}, \ldots, \bar{\varDelta}^i_{\bar{a}^i+\bar{m}^i+2}$, respectively.

Similarly, for the i^{th} query to the verification oracle with input $(\tilde{N}^i, \tilde{A}^i, \tilde{C}^i)$ and output $b^i \in \{\top, \bot\}$, SAEF internally processes $\tilde{A}^i$ and $\tilde{C}^i$ in blocks, denoted as $\tilde{A}^i_1, \ldots, \tilde{A}^i_{\tilde{a}^i}, \tilde{A}^i_*$ and $\tilde{C}^i_1, \ldots, \tilde{C}^i_{\tilde{m}^i}, \tilde{C}^i_*, \tilde{T}^i$, where $\tilde{a}^i$ and $\tilde{m}^i$ are respectively equal to the length of $\tilde{A}^i$ and the length of $\tilde{C}^i$ in n-bit blocks (excluding the tag from the count). Additionally, SAEF internally computes the plaintext blocks $\tilde{M}^i_1, \ldots, \tilde{M}^i_{\tilde{m}^i}, \tilde{M}^i_*$ as well as $\tilde{\varDelta}^i_1, \ldots, \tilde{\varDelta}^i_{\tilde{a}^i+1}$, the whitening masks used to process $\tilde{A}^i_1, \ldots, \tilde{A}^i_{\tilde{a}^i}, \tilde{A}^i_*$ respectively, and $\tilde{\varDelta}^i_{\tilde{a}^i+2}, \ldots, \tilde{\varDelta}^i_{\tilde{a}^i+\tilde{m}^i+2}$,the whitening masks used to process the blocks $\tilde{C}^i_1, \ldots, \tilde{C}^i_{\tilde{m}^i}, \tilde{C}^i_*$ respectively.

Additional information. To make the proof analysis simple, we additionally provide the adversary with all the whitening masks (encryption masks $\varDelta^i_j$, decryption masks $\bar{\varDelta}^i_j$ and verification masks $\tilde{\varDelta}^i_j$), and internally computed plaintexts $\tilde{M}^i_j$ when it has made all its queries and only the final response is pending.

In the real-int world, all these variables are internally computed by oracles that faithfully evaluate SAEF. In the ideal-int world also, the encryption and decryption oracles evaluate SAEF, hence the $\varDelta_j^i$ and $\bar{\varDelta}_j^i$ masks are defined. However, the verification oracle of the ideal-int world does not make any computations, and hence $\tilde{\varDelta}_j^i$ and $\tilde{M}_j^i$ are not defined. We therefore have to define the sampling of these variables which will be done at the end of the experiment (and thus have no effect on the adversarial queries).

We fix $\tilde{\varDelta}_1^i = 0^n$ for $1 \leq i \leq q_v$ and sample each of the remaining masks $\tilde{\varDelta}_j^i$ uniformly and independently at random, except when such a mask is trivially defined due to a "common prefix" (defined shortly) with a previous query (encryption, decryption or verification). Once these masks are sampled, we use the SAEF decryption algorithm with π_0 and these masks to compute $\tilde{M}_j^i$. Clearly, this give away of additional information can only help the adversary by increasing its advantage and hence can be considered here for upper bounding the targeted (above mentioned) adversarial advantage.

Block-tuple representation. We switch to an equivalent representation (called *block-tuple* representation) by defining the i^{th} encryption query as $(\mathsf{T}_j^i, \varDelta_j^i, X_j^i, Y_j^i)_{j=1}^{\ell^i}, T^i$, such that $\ell^i = a^i + m^i + 2$. The j^{th} quadruple (out of these ℓ^i) represents the processing done in j^{th} forkcipher call in the query, with the string T_j^i used as forkcipher tweak, the corresponding whitening mask $\varDelta_j^i$, the (possibly padded) associated data/plaintext block X_j^i and the empty/ciphertex block Y_j^i. With formal details:

- For the very first block, we always have $\mathsf{T}_1^i = N\|1\|F$ for a flag $F \in \{0,1\}^3$ and $\varDelta_1^i = 0^n$. For blocks with $j > 1$ we have $\mathsf{T}_j^i = 0^{t-3}\|F$ for an $F \in \{0,1\}^3$.
- If $|A| > 0$, for $1 \leq j \leq a^i$ we have $X_j^i = A_j^i$, $Y_j^i = \varepsilon$ and $F = 000$. For $j = a^i + 1$ we have $X_j^i = \mathsf{pad10}(A_*^i)$, $Y_j^i = \varepsilon$ and $F \in \{0,1\}^3$ as defined in Figure 2.
- If $|M| > 0$, for $a^i + 2 \leq j < \ell^i$ we have $X_j^i = M_j^i$, $Y_j^i = C_j^i$ and $F = 001$. For $j = \ell^i$ we have $X_j^i = \mathsf{pad10}(M_*^i)$, $Y_j^i = C_*^i$ and $F \in \{0,1\}^3$ as defined in Figure 2.
- If $A = M = \varepsilon$, we have $j = \ell^i = 1$, $X_j^i = \mathsf{pad10}(\varepsilon), Y_j^i = \varepsilon$ and $F = 111$.

With similar definition, we define the block-tuple representation for decryption queries as $(\bar{\mathsf{T}}_j^i, \bar{\varDelta}_j^i, \bar{X}_j^i, \bar{Y}_j^i)_{j=1}^{\bar{\ell}^i}, \bar{T}^i$ with $\bar{\ell}^i = \bar{m}^i + \bar{a}^i + 2$, and for verification queries as $(\tilde{\mathsf{T}}_j^i, \tilde{\varDelta}_j^i, \tilde{X}_j^i, \tilde{Y}_j^i)_{j=1}^{\tilde{\ell}^i}, \tilde{T}^i, b^i$ with $\tilde{\ell}^i = \tilde{m}^i + \tilde{a}^i + 2$. We now simplify the notation by re-indexing the decryption queries from $q_e + 1$ to $q_e + q_d$ and the verification queries from $q_e + q_d + 1$ to $q_e + q_d + q_v$. Further, with this new indexing, we drop the bars and tildes from the variables. The decryption queries and verification queries are thus denoted as $(\mathsf{T}_j^i, \varDelta_j^i, X_j^i, Y_j^i)_{j=1}^{\ell^i}, T^i$ for $q_e + 1 \leq i \leq q_e + q_d$ and $(\mathsf{T}_j^i, \varDelta_j^i, X_j^i, Y_j^i)_{j=1}^{\ell^i}, T^i, b^i$ for $q_e + q_d + 1 \leq i \leq q_e + q_d + q_v$, respectively.

Blockwise common prefix of queries The block-tuple notation allows us to define the following natural definition of longest common blockwise prefix between any two *queries.* We define the longest common prefix between the i^{th} and i'^{th} query with $\ell^i \leq \ell^{i'}$ w.l.o.g. as

$$\mathsf{llcp}_n(i, i') = \max\{1 \leq u \leq \ell^i | (\mathsf{T}_j^i, \varDelta_j^i, X_j^i, Y_j^i) = (\mathsf{T}_j^{i'}, \varDelta_j^{i'}, X_j^{i'}, Y_j^{i'}) \text{ for } 1 \leq j \leq u\}.$$

Note that this definition covers common blockwise prefix between all types of query pairs (for example, between two encryption queries or between an encryption and a decryption query or between a decryption and a verification query etc). Informally, $\mathsf{llcp}_n(i, i')$ represents the number of internal chaining values $\varDelta$s that are trivially equal between the i^{th} and i'^{th} query. To exemplify, if the nonces N^i and $N^{i'}$ are different then we have $\mathsf{llcp}_n(i, i') = 0$. If we have two queries with $N^i = N^{i'}$ but the i'^{th} query has AD $A^{i'} = A^i \| M_1^i$ i.e. equal to the AD of the i^{th} query appended with its first message block, we will still have $\mathsf{llcp}_n(i, i') = a^i + 1$, due to the inclusion of tweak strings in the block tuples. We now define the length of the longest common blockwise prefix of a query *with all previous queries* as $\mathsf{llcp}_n(i) = \max_{1<i'<i} \mathsf{llcp}_n(i, i')$. One should note here that for a verification query, all the encryption and decryption queries are always taken into account (as per the convention of query indexing).

Sampling of $\varDelta$ masks. Since the block-tuple notation and common prefix are defined, we can now use them to formally define the sampling of $\varDelta^i_j$ masks for the verification queries (i.e., for $q_e + q_d < i \leq q_e + q_d + q_v$) of the ideal-int world.

For the i^{th} verification query with $1 \leq j \leq \mathsf{llcp}_n(i) + 1$, we let $\varDelta^i_j = \varDelta^{i'}_j$ for the smallest $i' < i$ such that i'^{th} query has $\mathsf{llcp}_n(i) = \mathsf{llcp}_n(i, i')$. For the remaining block-tuples with $\mathsf{llcp}_n(i) + 1 < j \leq \ell^i$, $\varDelta^i_j$ is sampled uniformly at random.

Extended transcripts. Using the block-tuple notation, we can now re-define the extended transcripts as

$$\tau = \left\langle \left(\left(\mathsf{T}^i_j, \varDelta^i_j, X^i_j, Y^i_j \right)_{j=1}^{\ell^i}, T^i \right)_{i=1}^{q_e+q_d}, \left(\left(\mathsf{T}^i_j, \varDelta^i_j, X^i_j, Y^i_j \right)_{j=1}^{\ell^i}, T^i, b^i \right)_{i=q_e+q_d+1}^{q_e+q_d+q_v} \right\rangle .$$

Note that the terms q_e, q_d, q_v, a and m here are themselves random variables and hence can vary for different *attainable* transcripts. However, due to the assumption that the adversary can only make at most σ many block queries, we always have $\sum_{i=1}^{q_e+q_d+q_v}(a^i + m^i + 2) = \sigma$. Also, note that it is impossible for two distinct transcripts $\tau = \langle (N^i, A^i, M^i, C^i)_{i=1}^{q_e}, (\bar{N}^i, \bar{A}^i, \bar{C}^i, \bar{M}^i)_{i=1}^{q_d}, (\tilde{N}^i, \tilde{A}^i, \tilde{C}^i, b^i)_{i=1}^{q_v} \rangle$ and $\tau' = \langle (N'^i, A'^i, M'^i, C'^i)_{i=1}^{q'_e}$, $(\bar{N'}^i, \bar{A'}^i, \bar{C'}^i, \bar{M'}^i)_{i=1}^{q'_d}$, $(\tilde{N'}^i, \tilde{A'}^i, \tilde{C'}^i, b'^i)_{i=1}^{q'_v} \rangle$ to have the same block-tuple representation (for the proof of this claim, we refer the reader to [2], Proposition 1).

Coefficient-H. Let us represent the distribution of the transcript in the real-int world and the ideal-int world by Θ_{rein} and Θ_{idin}, respectively.

The proof relies on the fundamental lemma of the coefficient H technique as defined in Lemma 1 above. We say an attainable transcript τ is bad if one of the following conditions occurs:

BadT_1 a.k.a. "Input Collision": There exists $(i', j') < (i, j)$ (the block indexed by (i', j') precedes (i, j)) such that $1 \leq i \leq q_e + q_d + q_v$, $\mathsf{llcp}_n(i) < j \leq \ell^i$ is not in the longest common prefix of the i^{th} query, , and the (i, j) block call has tweak-input collision with the (i', j') block call, i.e., $\mathsf{T}^i_j = \mathsf{T}^{i'}_{j'}$ and $X^i_j \oplus \varDelta^i_j = X^{i'}_{j'} \oplus \varDelta^{i'}_{j'}$.

BadT_2 a.k.a. "Mask Collision": There exists $(i', j') < (i, j)$ such that $1 \leq i \leq q_e + q_d + q_v$, $\mathsf{llcp}_n(i) < j < \ell^i$ (not lies in the longest common prefix), and both the block calls have the same tweaks $\mathsf{T}^i_j = \mathsf{T}^{i'}_{j'}$ and different inputs $X^i_j \oplus \varDelta^i_j \neq X^{i'}_{j'} \oplus \varDelta^{i'}_{j'}$ but the following masks $\varDelta^i_{j+1} = \varDelta^{i'}_{j'+1}$ collide. (although such a collision cannot occur in the real-int world where the masks are generated using permutation, it can still occur in the ideal-int world).

BadT_3 a.k.a. "Forgery": There exists $q_e + q_d + 1 \leq i \leq q_e + q_d + q_v$ such that for $j = \ell^i$ we have any of the following:

Case 1. The last bit of T^i_j is 0 and $\pi_{\mathsf{T}^i_j,1}(X^i_j \oplus \varDelta^i_j) = T^i$.

Case 2. The last bit of T^i_j is 1, $\mathsf{right}_{n-|T^i|}(X^i_j) = 10^{n-|T^i|-1}$
and $\mathsf{left}_{|T^i|}(\pi_{\mathsf{T}^i_j,1}(X^i_j \oplus \varDelta^i_j)) = T^i$.

Case 3. The last bit of T^i_j is 1, there exists $q_e + 1 \leq i_d \leq q_e + q_d$ with $\mathsf{T}^{i_d}_{\ell^{i_d}} = 1$ and $|T^i| = |T^{i_d}|$ such that $\mathsf{right}_{n-|T^{i_d}|}(X^{i_d}_{\ell^{i_d}}) = 10^{n-|T^{i_d}|-1}$

and $\mathsf{left}_{|T^i|}(\pi_{\mathsf{T}^i_j,1}(X^i_j \oplus \mathit{\Delta}^i_j)) = T^i$.

We denote by $\mathcal{T}_{\text{bad}}$, the set of "bad" transcripts that is defined as the subset of attainable transcripts for which the transcript predicate $\mathsf{BadT}(\tau) = (\mathsf{BadT}_1(\tau) \vee \mathsf{BadT}_2(\tau) \vee \mathsf{BadT}_3(\tau)) = 1$. We denote by $\mathcal{T}_{\text{good}}$, the set of attainable transcripts which are not in the set $\mathcal{T}_{\text{bad}}$ (and are therefore called good transcripts).

Lemma 2. *For $\mathcal{T}_{bad}$ above and $q_e + q_d \le 2^{n-1}$, we have*

$$\Pr[\Theta_{idin} \in \mathcal{T}_{bad}] \le \frac{\sigma^2}{2^n} + \frac{4 \cdot q_d q_v}{2^n} \ .$$

Proof. BadT_1**.** For any transcript in $\mathcal{T}_{\text{bad}}$ with BadT_1 set to 1, we know that there exists at least one pair of block indices $(i', j') < (i, j)$ such that $\mathsf{llcp}_n(i) < j \le \ell^i$ and $\mathit{\Delta}^i_j \oplus \mathit{\Delta}^{i'}_{j'} = X^i_j \oplus X^{i'}_{j'}$.

Note that for all $i' < i$ and $j = j' = \mathsf{llcp}_n(i) + 1$, we have $\mathit{\Delta}^i_j = \mathit{\Delta}^{i'}_{j'}$ but $X^i_j \ne X^{i'}_{j'}$ and thus for all such cases the probability that the above equality occurs is 0. Contrastingly, for all $i' \le i$ and $j' \ne j$ or $j \ne \mathsf{llcp}_n(i) + 1$, the two masks has marginal probability $1/2^n$ of being same in Θ_{idin}. Since there are total σ possible values of (i, j) in a transcript, each having no more than σ possible values of (i', j'), we get $\Pr[\mathsf{BadT}_1(\Theta_{idin}) = 1] \le \frac{\sigma^2}{2} \cdot \max\left\{0, \frac{1}{2^n}\right\} = \frac{\sigma^2}{2^{n+1}}$.

BadT_2**.** Similarly, for any transcript in $\mathcal{T}_{\text{bad}}$ with BadT_2 set to 1, we know that there exists at least one pair $(i', j') < (i, j)$ such that $\mathsf{llcp}_n(i) < j < \ell^i$ and $\mathit{\Delta}^i_{j+1} \oplus \mathit{\Delta}^{i'}_{j'+1} = 0$.

Note that from the definition of the predicate BadT_2 we have $j + 1 \ne \mathsf{llcp}_n(i) + 1$. This means that the marginal probability of $\mathit{\Delta}^i_{j+1}$ being equal to $\mathit{\Delta}^{i'}_{j'+1}$ is $1/2^n$. Since there are total σ possible values of (i, j) in a transcript, each with no more than σ possible values of (i', j'), we get $\Pr[\mathsf{BadT}_2(\Theta_{idin}) = 1] \le \frac{\sigma^2}{2^{n+1}}$.

BadT_3**.** Now, for any transcript in $\mathcal{T}_{\text{bad}}$ with BadT_3 set to 1 and BadT_1 set to 0, we know that one of the following can happen for Θ_{idin}:

1. For some $i' \le q_e$, $j = \ell^i$ and $j' = \ell^{i'}$, we have $j = j' = \mathsf{llcp}_n(i)$. Clearly, in such a case $\mathit{\Delta}^i_j = \mathit{\Delta}^{i'}_{j'}$, $X^i_j = X^{i'}_{j'}$ but $T^i \ne T^{i'}$. Since $T^{i'}$ is the correct tag for the given ciphertext, $T^i \ne T^{i'}$ cannot trigger BadT_3, and yields 0 probability.
2. For some $i' \le q_e + q_d$, $j = \ell^i$ and $j' = \ell^{i'}$, we have $j = j' = \mathsf{llcp}_n(i) + 1$. We have $\mathit{\Delta}^i_j = \mathit{\Delta}^{i'}_{j'}$ but $X^i_j \ne X^{i'}_{j'}$ and thus the probability of any of the three conditions of BadT_3 occurring for a given query is at most $4q_d/2^n$ assuming $q_e + q_d \le 2^{n-1}$. For the first condition, this holds as every tag there is produced with a tweak used at most once per encryption query, corresponding to a probability $1/(2^n - q_e) \le 2/2^n$. For the second condition, we can upper bound the product of the probabilities of having the correct padding in the block X^i_j (at most $2^{|T^i|}/(2^n - q_e - q_d)$), and of having the correct truncated tag (at most $2^{n-|T^i|}/(2^n - q_e)$) by $4/2^n$. For the third condition, with any choice of $q_e + 1 \le i_d \le q_e + q_d$ such that $|T^i| = |T^{i_d}|$, we can upper bound the product of the probabilities of having the correct padding in the block $X^{i_d}_{\ell^{i_d}}$ (at most $2^{|T^i|}/(2^n - q_e - q_d)$), and of having the correct truncated tag for the verification query (at most $2^{n-|T^i|}/(2^n - q_e)$) by $4/2^n$. Now, since there are total q_d many possible choices for i_d, the total probability of the third condition is upper bounded by $4q_d/2^n$.

3. For all $i' \leq q_e + q_d$ when we have $j > \mathsf{llcp}_n(i, i') + 1$. We know that the Δ^i_j is not inherited from an encryption or decryption query and is therefore sampled uniformly in Θ_{idin}. The first condition of BadT_3 thus occurs with a probability $1/2^n$. For the second condition, the correct padding is found with probability $1/2^{n-|T^i|}$ (using the randomness of Δ^i_j), and the correct tag is found with probability at most $2^{n-|T^i|}/(2^n - q_e)$, thanks to freshness of $X^i_j \oplus \Delta^i_j$, relying on $\mathsf{BadT}_1(\Theta_{idin}) = 0$ w.l.o.g., yielding a probability of at most $2/2^n$. For the third condition, with similar reasoning the correct padding is found with probability $q_d/2^{n-|T^i|}$ (using the randomness of Δ^i_j), and the correct tag is found with probability at most $2^{n-|T^i|}/(2^n - q_e)$, providing a probability of at most $2q_d/2^n$.

Since there are total q_v possible verification queries, we get $\Pr[\mathsf{BadT}_3(\Theta_{idin}) = 1 | \mathsf{BadT}_1(\Theta_{idin}) = 0] \leq q_v \cdot \max\left\{0, \frac{4q_d}{2^n}, \frac{2q_d}{2^n}\right\} = \frac{4 \cdot q_d q_v}{2^n}$ and we obtain by the union bound that $\Pr[\Theta_{idin} \in \mathcal{T}_{\text{bad}}] \leq \frac{\sigma^2}{2^n} + \frac{4 \cdot q_d q_v}{2^n}$.

Lemma 3. *Let $\tau \in \mathcal{T}_{good}$ i.e. τ is a good transcript. Then* $\frac{\Pr[\Theta_{rein}=\tau]}{\Pr[\Theta_{idin}=\tau]} \geq 1$.

Proof. Note that a good transcript has the following two properties 1. **(i)** For each $(i', j') < (i, j)$ if (i, j) is not in the longest common prefix of the two queries i.e. $\mathsf{llcp}_n(i, i') < j < \ell^i$ and both π_0 calls have same tweaks (i.e. $\mathsf{T}^i_j = \mathsf{T}^{i'}_{j'}$) then both calls will always have different inputs and different outputs. 2. **(ii)** For each query to the verification oracle i.e. $1 \leq i \leq q_v$, the transcript contains $b^i = \perp$ in the verification result i.e. the conditions for a successful verification are not met.

The probability to obtain a good transcript τ in the real-int and the ideal-int worlds can now be computed. Let τ_{ed} and τ_v denote the two parts of a transcript τ consisting respectively encryption-decryption and verification queries, so that $\tau = \langle \tau_{ed}, \tau_v \rangle$. With a slight abuse of notation, we have $\Pr[\Theta_{rein} = \tau] = \Pr[\Theta_{rein,e} = \tau_{ed}] \cdot \Pr[\Theta_{rein,v} = \tau_v | \Theta_{rein,ed} = \tau_{ed}]$ and $\Pr[\Theta_{idin} = \tau] = \Pr[\Theta_{idin,ed} = \tau_{ed}] \cdot \Pr[\Theta_{idin,v} = \tau_v | \Theta_{idin,ed} = \tau_{ed}]$ and consequently

$$\frac{\Pr[\Theta_{rein,ed} = \tau_{ed}] \cdot \Pr[\Theta_{rein,v} = \tau_v | \Theta_{rein,ed} = \tau_{ed}]}{\Pr[\Theta_{idin,ed} = \tau_{ed}] \cdot \Pr[\Theta_{idin,v} = \tau_v | \Theta_{idin,ed} = \tau_{ed}]} = \frac{\Pr[\Theta_{rein,v} = \tau_v | \Theta_{rein,ed} = \tau_{ed}]}{\Pr[\Theta_{idin,v} = \tau_v | \Theta_{idin,ed} = \tau_{ed}]} .$$

This is true because the encryption and decryption oracles in the real-int world and in the ideal-int world are identical, and so $\Pr[\Theta_{rein,ed} = \tau_{ed}] = \Pr[\Theta_{idin,ed} = \tau_{ed}]$. Further abusing notation, we let $\tau_{v,\Delta}$ denote the marginal event of all Δ masks in the verification queries (as variables) being equal to the values in the transcript. We have $\Pr[\Theta_{rein,v} = \tau_v | \Theta_{rein,ed} = \tau_{ed}, \Theta_{rein,v,\Delta} = \tau_{v,\Delta}] = \Pr[\Theta_{idin,v} = \tau_v | \Theta_{idin,ed} = \tau_{ed}, \Theta_{idin,v,\Delta} = \tau_{v,\Delta}]$ because both sides of this equality correspond to mappings defined with random permutations with the input-output pairs fixed from the encryption-decryption parts in both worlds. Further, using this equality, we get

$$\frac{\Pr[\Theta_{rein,v} = \tau_v | \Theta_{rein,ed} = \tau_{ed}]}{\Pr[\Theta_{idin,v} = \tau_v | \Theta_{idin,ed} = \tau_{ed}]} = \frac{\Pr[\Theta_{rein,v,\Delta} = \tau_{v,\Delta} | \Theta_{rein,ed} = \tau_{ed}]}{\Pr[\Theta_{idin,v,\Delta} = \tau_{v,\Delta} | \Theta_{idin,ed} = \tau_{ed}]} .$$

Let us now consider that there are δ many Δs in τ that are fixed/predefined due to all internal common prefixes. Clearly, one can write that $\delta = \sum_{i=1}^{q_e+q_d+q_v} (\mathsf{llcp}_n(i) + 1)$ (the extra 1 represents the Δ^i_1 as it is always fixed to 0). In the ideal-int world, since the Δs corresponding to the remaining $(\sigma - \delta)$ unique block calls are sampled uniformly and independently and all verification oracle results are $\perp$, one has $\Pr[\Theta_{idin,v,\Delta} = \tau_{v,\Delta} | \Theta_{idin,ed} = \tau_{ed}] = \frac{1}{(2^n)^{\sigma-\delta}}$. In the real-int world, these $(\sigma - \delta)$ Δs are no longer uniformly distributed but are instead defined using the random tweakable permutation (π_0, π_1) with at least $g_1 = \sum_{i=1}^{q_e+q_d+q_v} (a^i - 1)$ block calls with the tweak 0^n and at least $g_2 = \sum_{i=1}^{q_e+q_d+q_v} (m^i - 1)$ block calls with the tweak $0^{n-1} \| 1$. Thus, one has

$\Pr[\Theta_{rein,v,\Delta} = \tau_{v,\Delta} | \Theta_{rein,ed} = \tau_{ed}] \geq \frac{1}{(2^n)_{g_1}(2^n)_{g_2}(2^n)^{\sigma-\delta-g_1-g_2}}$.
One should note here that the above expression is not an equality and only provides an upper bound on the targeted probability because there exist more permutation calls that can have tweak collisions (for example, the first block calls of any set of queries will have same tweaks if they all have same nonce). Now, from the above expressions, we get

$$\frac{\Pr[\Theta_{rein} = \tau]}{\Pr[\Theta_{idin} = \tau]} \geq \frac{(2^n)^{\sigma-\delta}}{(2^n)_{g_1}(2^n)_{g_2}(2^n)^{\sigma-\delta-g_1-g_2}} = \frac{(2^n)^{g_1}(2^n)^{g_2}}{(2^n)_{g_1}(2^n)_{g_2}} \geq 1 \ .$$

Combining the results of Lemma 2 and 3 (taking $\epsilon_1 = 0$) into Lemma 1, we obtain the upper bound $\mathbf{Adv}^{\mathsf{INT\text{-}RUP}}_{\mathrm{SAEF}[(\pi_0,\pi_1)]}(\mathcal{A}) \leq \frac{\sigma^2}{2^n} + \frac{4 \cdot q_d q_v}{2^n}$ and hence the result of the Theorem 1.

4 Conclusion

We prove that SAEF is RUP-secure w.r.t. integrity under the existing notion of INT-RUP as long as the total amount of data processed with a single key is $\ll 2^{n/2}$ blocks, with n being the blocksize of the underling forkcipher. This concludes that SAEF continues to provide the same integrity even when the unverified plaintext is released. Clearly, our newly proven security property on the original SAEF construction is of high relevance to many resource-constrained applications of lightweight cryptography where the constrained devices are forced or allow for leaking portions of unverified plaintext in decryption.

5 Acknowledgments

This work was supported by CyberSecurity Research Flanders with reference number VR20192203. This work was supported in part by the Research Council KU Leuven C1 on Security and Privacy for Cyber-Physical Systems and the Internet of Things with contract number C16/15/058 and by the Flemish Government through FWO Project G.0835.16 A security Architecture for IoT.

References

1. Andreeva, E., Bogdanov, A., Datta, N., Luykx, A., Mennink, B., Nandi, M., Tischhauser, E., Yasuda, K.: COLM v1 (2014), "https://competitions.cr.yp.to/round3/colmv1.pdf"
2. Andreeva, E., Bhati, A.S., Vizár, D.: Nonce-Misuse Security of the SAEF Authenticated Encryption mode. In: Selected Areas in Cryptography (2020)
3. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., Yasuda, K.: How to Securely Release Unverified Plaintext in Authenticated Encryption. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 105–125. Springer (2014)
4. Andreeva, E., Lallemand, V., Purnal, A., Reyhanitabar, R., Roy, A., Vizár, D.: Forkcipher: a New Primitive for Authenticated Encryption of Very Short Messages. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 153–182. Springer (2019)
5. Ashur, T., Dunkelman, O., Luykx, A.: Boosting authenticated encryption robustness with minimal modifications. In: Annual International Cryptology Conference. pp. 3–33. Springer (2017)
6. Bellizia, D., Berti, F., Bronchain, O., Cassiers, G., Duval, S., Guo, C., Leander, G., Leurent, G., Levi, I., Momin, C., Pereira, O., Peters, T., Standaert, F.X., Udvarhelyi, B., Wiemer, F.: Spook: Sponge-Based Leakage-Resistant Authenticated Encryption with a Masked Tweakable Block Cipher. IACR Transactions on Symmetric Cryptology **2020**(S1), 295–349 (Jun 2020). https://doi.org/10.13154/tosc.v2020.iS1.295-349, https://tosc.iacr.org/index.php/ToSC/article/view/8623
7. Bernstein, D.J.: Cryptographic competitions: CAESAR. http://competitions.cr.yp.to

8. Bhattacharjee, A., List, E., López, C.M., Nandi, M.: The Oribatida Family of Lightweight Authenticated Encryption Schemes
9. Chakraborti, A., Datta, N., Jha, A., Lopez, C.M., Nandi, M., Sasaki, Y.: ESTATE Authenticated Encryption Mode: Hardware Benchmarking and Security Analysis
10. Chakraborti, A., Datta, N., Jha, A., Mancillas-López, C., Nandi, M., Sasaki, Y.: INT-RUP Secure Lightweight Parallel AE Modes. IACR Transactions on Symmetric Cryptology pp. 81–118 (2019)
11. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: ASCON v1.2 (2014), "https://competitions.cr.yp.to/round3/asconv12.pdf"
12. Fleischmann, E., Forler, C., Lucks, S.: McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes. In: Canteaut, A. (ed.) Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers. Lecture Notes in Computer Science, vol. 7549, pp. 196–215. Springer (2012). https://doi.org/10.1007/978-3-642-34047-5_12, https://doi.org/10.1007/978-3-642-34047-5_12
13. Hoang, V.T., Reyhanitabar, R., Rogaway, P., Vizar, D.: Online Authenticated-Encryption and its Nonce-Reuse Misuse-Resistance. vol. 9215, pp. 493–517. Gennaro, R, Springer Verlag (2015)
14. Iwata, T., Khairallah, M., Minematsu, K., Peyrin, T.: Duel of the Titans: The Romulus and Remus Families of Lightweight AEAD Algorithms. IACR Transactions on Symmetric Cryptology **2020**(1), 43–120 (May 2020). https://doi.org/10.13154/tosc.v2020.i1.43-120, https://tosc.iacr.org/index.php/ToSC/article/view/8560
15. Jean, J., Nikolić, I., Peyrin, T., Seurin, Y.: Deoxys v1.41 v1 (2016), "https://competitions.cr.yp.to/round3/deoxysv141.pdf"
16. Krovetz, T., Rogaway, P.: OCB v1.1 (2014), "https://competitions.cr.yp.to/round3/ocbv11.pdf"
17. Patarin, J.: The "Coefficients H" Technique, p. 328–345. Springer-Verlag, Berlin, Heidelberg (2009), https://doi.org/10.1007/978-3-642-04159-4_21
18. Purnal, A., Andreeva, E., Roy, A., Vizár, D.: What the Fork: Implementation Aspects of a Forkcipher. In: NIST Lightweight Cryptography Workshop 2019 (2019)
19. Rogaway, P.: Authenticated-Encryption with Associated-Data. In: Proceedings of the 9th ACM conference on Computer and communications security. pp. 98–107 (2002)
20. Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 373–390. Springer (2006)
21. Wu, H.: ACORN v3 (2014), "https://competitions.cr.yp.to/round3/acornv3.pdf"
22. Wu, H., Huang, T.: MORUS v2 (2014), "https://competitions.cr.yp.to/round3/morusv2.pdf"
23. Wu, H., Preneel, B.: AEGIS v1.1 (2014), "https://competitions.cr.yp.to/round3/aegisv11.pdf"