Let’s Go Eevee! A Friendly and Suitable Family of AEAD Modes for IoT-to-Cloud Secure Computation

Amit Singh Bhati
COSIC, KU Leuven
Leuven, Belgium
amitsingh.bhati@esat.kuleuven.be

Erik Pohle
COSIC, KU Leuven
Leuven, Belgium
erik.pohle@esat.kuleuven.be

Aysajan Abidin
COSIC, KU Leuven
Leuven, Belgium
aysajan.abidin@esat.kuleuven.be

Elena Andreeva
Technical University of Vienna
Vienna, Austria
elena.andreeva@tuwien.ac.at

Bart Preneel
COSIC, KU Leuven
Leuven, Belgium
bart.preneel@esat.kuleuven.be

ABSTRACT

IoT devices collect privacy-sensitive data, e.g., in smart grids or in medical devices, and send this data to cloud servers for further processing. In order to ensure confidentiality as well as authenticity of the sensor data in the untrusted cloud environment, we consider a transciphering scenario between embedded IoT devices and multiple cloud servers that perform secure multi-party computation (MPC). Concretely, the IoT devices encrypt their data with a lightweight symmetric cipher and send the ciphertext to the cloud servers. To obtain the secret shares of the cleartext message for further processing, the cloud servers engage in an MPC protocol to decrypt the ciphertext in a distributed manner. This way, the plaintext is never exposed to the individual servers.

As an important building block in this scenario, we propose a new, provably secure family of lightweight modes for authenticated encryption with associated data (AEAD), called Eevee. The Eevee family has fully parallel decryption, making it suitable for MPC protocols for which the round complexity depends on the complexity of the function they compute. Further, our modes use the lightweight forkcipher primitive that offers fixed-length output expansion and a compact yet parallelizable internal structure.

All Eevee members improve substantially over the few available state-of-the-art (SotA) MPC-friendly modes and other standard solutions. We benchmark the Eevee family on a microcontroller and in MPC. Our proposed mode Jolteon (when instantiated with ForkSkinny) provides 1.85x to 3.64x speedup in IoT-encryption time and 3x to 4.5x speedup in both MPC-decryption time and data for very short queries of 8 bytes and, 1.55x to 3.04x and 1.23x to 2.43x speedup, respectively, in MPC-decryption time and data for queries up to 500 bytes when compared against SotA MPC-friendly modes instantiated with SKINNY. We also provide two advanced modes, Umbreon and Espeon, that show a favorable performance-security trade-off with stronger security guarantees such as nonce-misuse security. Additionally, all Eevee members have full n-bit security (where n is the block size of the underlying primitive), use a single primitive and require smaller state and HW area when compared with the SotA modes under their original security settings.

1 INTRODUCTION

In today’s era of big data, the tremendous growth of Internet-connected devices such as resource-constrained lightweight sensor nodes enables fine-grained data collection. While the collected data is utilized in good faith, e.g., to improve models, aid energy saving or offer personalized analysis, privacy risks remain due to the centralized nature of the collection and utilization. For example in a smart grid environment, smart electricity meters serve as embedded IoT devices that measure power consumption and transmit readings to energy providers [71]. Amalgamating the data from numerous users enables efficient power grid management, dynamic distribution, and predictive modeling of energy demands. However, it is crucial to consider that frequent individual electricity readings may inadvertently reveal highly sensitive information on the user’s health, habits and beliefs.

Previous privacy-preserving solutions either rely on data aggregation [71] or on blockchain infrastructure [32, 54]. In our solution, the sensor data is encrypted and authenticated at the source by the resource-constrained devices using a suitable authenticated encryption with associated data (AEAD) mode, and the encrypted data is sent to multiple computing parties. The computing parties then perform a distributed decryption via secure multi-party computation (MPC) to obtain secret shares of the plaintext data on which they can perform further privacy-preserving computation in MPC. Once the computation is done, the result is provided in a suitable manner to the legitimate parties. With this approach, we don’t lose accuracy due to aggregation nor do we require expensive distributed ledger technologies. Moreover, the encrypted sensor data can be centrally collected and stored since it is both encrypted and authenticated. The distributed decryption and subsequent data usage can tolerate up to all-but-one malicious computing parties, thus ensuring that private data remains private.

1.1 Approach

We start to approach this *transciphering* scenario by exploring the spectrum of security-efficiency trade-offs using traditional AEAD solutions. Considering the globally accepted AEAD standard AES-GCM [33] for the IoT-to-Cloud transciphering problem, we observe very efficient internal components AES and GHASH and the support of hardware acceleration on standard platforms leading to high performance. However, when compared against lightweight tweakable block ciphers (TBCs) such as SKINNY [14], AES suffers from (1) larger hardware (HW) area which increases energy consumption and hinders battery life of the device (see [52, Fig. 4 and 5]) and (2) larger register sizes for threshold implementations (TI) to achieve protection from various side channel attacks (see [62, Table 1] and [63, Table 1][1]). Further, the drawbacks of GCM as a mode include its sub-optimized structure for small size IoT messages due to the additional costly pre- and post-processing cipher calls, and lack of nonce-misuse resistance which brings devastating attacks when nonces are repeated [20, 72] and at most 64-bit security. As a result, AES-GCM is not a suitable lightweight solution for small embedded/IoT devices[2].

Another option is to consider MPC-friendly block ciphers. This choice would prioritize fast decryption and would put more strain on the IoT device. We implemented two MPC-friendly AEAD modes CTR-then-PMAC (short: CTR-PMAC) and CTR-then-Hash-then-MAC (short: CTR-HtMAC) proposed by Rotaru et al. [68] on a microcontroller. The modes are instantiated with MiMC-128, an algebraic block cipher operating on a 128-bit prime field. Our benchmark in Fig. 7 in Appendix A.1 shows that the MPC-friendly AEAD modes encrypt messages between one and two orders of magnitude slower than all other studied modes. As a consequence, the increased workload for encryption impacts, among others, power consumption, heat management and battery life of a device significantly. Thus, current MPC-friendly AEAD modes that are proposed for transciphering can hardly be called lightweight and do not seem well suited for many IoT applications.[3] In this work, we assume that the implementation cost of an MPC-friendly solution is prohibitively large for practical deployment in IoT applications. We focus our approach on traditional lightweight primitives.

The term lightweight is both a subjective and relative term the meaning of which depends on the context, for example the specific IoT application. For this work, we specify some common properties that are targeted (or considered important) by major IoT applications to reduce production costs and to comply with platform resource constraints. Our lightweight solution aims at (1) minimal RAM and HW area requirements (as low as possible) and thus low power consumption to reduce the heat and cooling costs, (2) built-in defence against accidental or adversarially controlled nonce repetitions, (3) robust 128-bit security levels with small key sizes, (4) use of a small number of primitives with common components, (5) support for online/block-wise processing of streamed data removing the need to buffer large chunks of the message and (6) provably secure standard model guarantees.

Some of the submitted AEAD designs to the NIST Lightweight Cryptography competition [64] fit only partially to our lightweight criteria. Note that security robustness and nonce-misuse resilience is met only by Romulus-M [45] and partially by the Elephantv2 candidates [19]. Yet, Romulus-M requires a double pass over the data which penalizes performance both in IoT and MPC. Elephantv2 on the other hand, does not provide security guarantees in the standard model but rather in the ideal cipher model.

In addition, if deployed in our transciphering scenario, existing lightweight solutions have to be adapted to support efficient and secure computation on the decryption side. Ideally, the target design has to reduce the MPC cost (computational and communicational) by a significant margin without trading any security properties or significantly increasing the costs on the IoT device. Unlike the IoT end, where area and battery life are hard constraints, MPC servers have high computational processing capabilities and constraints on the communication costs. Consequently, to reduce the overall cost to the user which includes the MPC-as-a-service cost, we look for an IoT-friendly solution with a computation/communication cost for MPC decryption that is as low as possible.

As a result, a scheme that is mainly optimized for lightweight applications has to integrate additional nontrivial design changes for performance and security, such as decryption parallelizability. All these considerations render existing lightweight solutions inappropriate for our target scenario.

In Table 1, we provide a summary of the existing solutions and conclude that they still lack some crucial properties for being IoT-to-MPC-friendly.

1.2 Desirable Properties

Summarizing our IoT-to-Cloud computation scenario, the following properties of the AEAD mode are important.

Parallel Decryption. In many MPC protocols, e.g. [17, 25, 30, 31, 37, 50, 51], multiplications (or equivalent non-linear operations) require interaction between the parties but independent multiplications can be performed in parallel. Here, the communication cost is directly proportional to the number of non-parallelizable secret-secret multiplications. In an AEAD mode which is based on some underlying primitives, this property turns into minimizing the number of non-parallelizable primitive calls. To exemplify the impact, we note that the performance of serial decryption modes degrades more than their parallel counterpart in slow networks since only one block can be computed at a time.

Nonce-Misuse Resistance. Nonce misuse presents a greater risk for embedded devices where nonces could repeat due to various memory or space constraints or an increased attack surface. Ensuring the correct use of nonces is very challenging in distributed systems where nodes and connections can fail at any time. Recent attacks [20, 53, 72] that exploit implementation flaws causing an application to reuse the same nonce illustrate the severity of nonce misuse in practice.

Block-wise Processing for Encryption. Due to buffer-size constraints most IoT devices cannot process large messages as a whole and follow the online or block-wise processing strategy. This allows an attacker to see a processed block output before the next block input and hence the attacker can choose each upcoming block adaptively which can lead to severe attacks in practice [12, 16, 48]. We target the online AE (OAE) [35] notion of security which resists block-wise adaptive attackers [34] and is a more realistic and stronger notion when compared with the basic nonce-based AE (nAE) [67] and the nonce misuse resilience (NMR) [10] notions of security. An OAE secure scheme is also nAE- and NMR-secure, however, it additionally provides a well-defined level of security (both confidentiality and integrity) for queries with reused nonces, i.e., security against nonce-misuse.

[1]TBCs are shown here as better and efficient alternatives to block ciphers or sponges for TI.
[2]See, e.g., the NIST LWC Standardization Process criteria [64] and the well-summarized SoK [41, 65].
[3]Another indicator may be the lack of MiMC implementations for IoT/microcontrollers. To the best of our knowledge, MiMC has only been implemented in high-level languages for the use in zero-knowledge proof systems.

Beyond-Birthday Security. Modes with security beyond the birthday bound (i.e., $> n/2$ bits) or even close to full n bits (where n is the block size) can avoid the use of large block primitives to achieve the desired level of security. Smaller primitives require less area and consume less power which both are important factors in IoT devices.

Length-Independent Security. Security bounds that are independent of the length of queried messages are called length-independent (LI) [58]. Such security does not degrade with an increase in message lengths and is useful when the average message length in queries is longer, e.g., when sensor data is streamed. The long message will have equal confidentiality and integrity as a single message block. This enables processing more data securely with the same key and hence avoids the heavy cost of frequent rekeying.

Single Primitive. AEAD constructions that use the same primitive both for encryption and authentication are beneficial for embedded devices since only one primitive has to be securely (e.g., resistant to side-channels) and efficiently implemented. Moreover, HW area can be saved if the primitive performs two functions.

Reusable Encryption. We finalize our IoT-to-MPC computation model by considering another useful property of the IoT encryption. We call the encryption share-independent/reusable when ciphertext does not depend on the number or nature of the MPC parties and thus the ciphertext can be reused for different MPC computations by simply resharing the key. This saves the cost of storing old data for re-encryptions on the IoT device or performing multiple encryptions for different groups of servers.

We note that storage and re-encryption of previously collected data is not feasible for real-time streaming or sensing IoT applications. This includes but is not limited to applications such as smart grids, healthcare IoTs, SCM IoTs, and agriculture IoTs. These classes of applications are novel and unexplored for IoT-to-MPC setting. Consequently, the devices supporting such applications are widespread but not MPC-aware at present, i.e., they were not installed with MPC processing in mind. To adapt them to the IoT-to-MPC setting, one can surely benefit from a cost-effective solution that does not require upgrading the device hardware.

Using a symmetric AEAD to build a share-independent/reusable scheme has multiple benefits over simply secret-sharing the collected data with the MPC engines. When using secret-sharing to input data, the IoT devices have to know MPC protocol specific details about the involved computing parties already during data collection. Coupled with the IoT devices' lack of sufficient memory capacity to store large amounts of sensor data, this hinders the flexibility since data collected for a specific analysis setting cannot be reused in a differing MPC setup, nor can the data collection start before the MPC parties are selected and ready.

In addition, the input shares have to be sent confidentially to each MPC party, increasing the encryption and communication work on the IoT device linearly in the number of parties. Battery-powered devices or those with tiny uplink bandwidth suffer performance drops due to this approach.

Our targeted AEAD approach shifts the workload to the more powerful MPC side instead. Here, only the symmetric key is (secret-) shared with the MPC parties once before the data analysis, reducing the linear cost (for the IoT device) to a few hundred bits. The MPC parties then use the key shares to compute a distributed decryption in MPC over the symmetrically AEAD encrypted data, where the ciphertext is a public input to the protocol, before computing the data analysis. To further increase flexibility, the symmetric key can be shared *at the latest possible instant, shortly before the MPC computation*, and *can be shared again* with different MPC parties to perform a different analysis on the same collected data.

Recently, an initial public draft of a call for multi-party threshold schemes by NIST [22] calls the same property "interchangeable" and defines it as a desirable property for interaction with thresholdized schemes.

1.3 Contributions

We design AEAD modes based on forkciphers (FC) [9] that are a type of tweakable ciphers with a versatile input and output structure. ForkSkinny [8, 66] is a forkcipher instance that is based on the recent ISO standard SKINNY [14]. The latter fact, together with the forkcipher's promising functional and security features in efficient modes of operation [4–6, 9] makes it a suitable candidate towards building AEAD modes that (1) are both *lightweight* for IoT devices and *efficient* in MPC for distributed decryption, and (2) guard against relevant security threats on both ends. A forkcipher (compared to a tweakable block cipher) as underlying primitive in the mode allows further parallelization on the primitive level in MPC such as parallel branch computation after the forking point. This makes Eevee members parallelizable both at mode and primitive levels.

We propose the Eevee family of three lightweight AEAD schemes based on a forkcipher. The three modes: Umbreon, Jolteon and Espeon are designed to be fully parallelizable in decryption, i.e., parallelizing all cipher calls, which is the best strategy so far [68] for decryption over MPC, and are at the same time IoT-friendly, as they are based on the lightweight forkcipher ForkSkinny. In addition, to use Eevee with any block cipher, we provide a generic forkcipher instantiation based on, e.g., AES.

Eevee modes provide security up to the full block length of the primitive and resistance against nonce-misuse and block-wise adversaries. Unlike existing MPC-friendly designs, the Eevee family contains dedicated yet simple AEAD modes based on a single primitive. The efficiency and suitability advantages of Eevee for both encryption and decryption over MPC make it a well-suited for secure data sharing solution from IoT-to-Cloud. Our specific contributions are as follows.

- Novel IoT- and MPC-friendly modes with better security properties.
  1. Umbreon (prioritizing security) provides full n bits of OAE security that logarithmically degrades with nonce-misuse.
  2. Jolteon (prioritizing performance) has smaller state requirements and provides better performance when compared with Umbreon at the cost of losing security under nonce-misuse.
  3. Espeon (intermediate trade-off) provides similar performance as Jolteon but with tweak-size-dependent security under nonce-misuse.

Table 1: Comparison of Eevee modes with other possible solutions for the target IoT-to-MPC setting. n, t and k represent the block, tweak, and key sizes of the underlying primitive in bits, respectively. Here, μ, $|M|$, s_H and A_H represent the maximum number of nonce repetitions, message length (in bits), the extra state size (in bits) and HW area used by the hash function in the AEAD scheme, respectively. We use the abbreviations Nonce-Respecting (NR), Nonce-Misuse (NM), Gate Equivalent (GE), Non-parallelizable Primitive Calls in MPC Decryption (NPCMD), Length-Independent security (LI), Single Primitive based design (SP) and Block-wise Processing Support (BPS). Security entries are in bits. Green color gradients in a column illustrate the result strength.
* **To the best of our knowledge, a HW implementation of Forkskinny-128-384 has not been studied yet. We give the area for Forkskinny-128-288 which also uses a $3n$ tweakey schedule albeit optimized since some bits are zero.**

Mode (Instantiations)	Minimal State (part of RAM)	Security Margin	Minimal HW Area	NPCMD	NR Security (nAE)	NM Security (OAE / MRAE)	LI(NR)	SP	BPS
AES-GCM [33] (AES,GHASH)	$5n+k$	<64	7215 GE [11] $+A_H$	3	$< n/2$	✗	✗	✗	✓
AES-GCM-SIV [39] (AES,POLYVAL)	$5n+k+\|M\|$	<64	7215 GE [11] $+A_H$	3	$< n/2$	$(n-2\log_2\mu)/3$ [46]	✗	✗	✗
CTR-PMAC [68] (SKINNY)	$3n+t+k$	64	3312 GE [14]	2	$n/2$	✗	✗	✓	✓
CTR-HtMAC [68] (SKINNY, BLAKE2s)	$2n+t+k$ $+s_H$	64	3312 GE [14] $+A_H$	1	$n/2$	✗	✗	✗	✓
Jolteon [This work] (ForkSkinny)	$2n+t+k$	64 128	2718 GE (64), 3917 GE (128) [7]	1	n	✗	✓	✓	✓
Espeon [This work] (ForkSkinny)	$2n+t+k$	128	$> 4567^*$ GE [7]	1	$t/2$	✓ $\min(t/2, n-\log_2\mu)$	✗	✓	✓
Umbreon [This work] (ForkSkinny)	$3n+t+k$	64 128	2718 GE (64), 3917 GE (128) [7]	1	n	✓ $n-\log_2\mu$	✓	✓	✓

- Compact provable security analysis of Eevee modes due to
  (1) a joint confidentiality analysis of Eevee as a family with non-trivial ways of defining intermediate adversaries, using their advantages to express individual mode bounds and therefore maximizing the common parts of the analyses.
  (2) the possibility of relying on (or reusing) the analysis of confidentiality to give a short proof for integrity.
- Full performance evaluation of the Eevee family for both encryption in IoT devices and distributed decryption in MPC. We compare the Eevee family with the state of the art MPC-friendly modes by Rotaru et al. [68] and the standard AEAD solutions AES-GCM [33] and AES-GCM-SIV [39, 40]. In order to keep underlying primitives similar, we instantiate Eevee with ForkSkinny and the MPC-friendly baseline with SKINNY and instantiate Eevee with AES (see Sect. 6.1 for details) to compare to the standardized AEAD solutions.

Our results show that all Eevee members improve substantially over the few available SotA MPC-friendly modes and standard solutions. Jolteon (with ForkSkinny) provides 1.85x to 3.64x speed-up in IoT-encryption time and 3x to 4.5x speed-up in both MPC-decryption time and data for very short messages of 8 bytes and, 1.55x to 3.04x and 1.23x to 2.43x speed-up, respectively, in MPC-decryption time and data for messages up to 500B when compared to SotA MPC-friendly modes instantiated with SKINNY. We also provide two advanced modes, Umbreon and Espeon, that show a favourable performance-security trade-off with stronger security guarantees. In addition, all Eevee members use a single primitive and have smaller state requirements when compared to the other modes under their original security settings.

1.4 Related Work

Historically, AES has long been the de facto benchmark for efficiency of the MPC protocols, e.g., [26, 29, 42]. As MPC is getting more traction in practical applications, the performance limitations of AES are becoming more apparent. Therefore, there has been an interest in MPC-friendly cryptographic primitives such as block ciphers [1, 2] and modes-of-operations [68]. The block cipher and hash function primitives aim for low multiplication depth [2] or native support for operations in GF(p) for a large prime p, as in [1]. Rotaru et al. [68] proposed MPC-friendly AEAD modes CTR-HtMAC and CTR-PMAC that are based on MPC-friendly block primitives MiMC/Leg (i.e., based on the Legendre symbol).

1.5 Outline

The paper is organized as follows. We present the preliminary notations and MPC-related background in Sect. 2. This is followed by the design definitions of Eevee AEAD family and its OAE security in Sect. 3. Next, in Sect. 4, we detail the design rationale of each Eevee mode and discuss their targeted instantiations in Sect. 5. Following a benchmark of Eevee for software performance on microcontrollers and MPC in Sect. 6, we present a security analysis of the proposed Eevee family in Sect. 7. Finally, we conclude the paper in Sect. 8.

2 NOTATION AND PRELIMINARIES

Strings and Operations. All strings are binary strings, $\{0,1\}^n$ is the set of strings of length $n > 0$, $\{0,1\}^*$ denotes strings of arbitrary length. We denote by $\mathrm{Perm}(n)$ and $\mathrm{Func}(m,n)$ the sets of all permutations of $\{0,1\}^n$ and functions with domain $\{0,1\}^m$ and range $\{0,1\}^n$, respectively. For a string X of ℓ bits, let $X[i]$ denote the i^{th} bit of X for $i = 0, \ldots, \ell-1$ (starting from the left) and $X[i \ldots j] = X[i] \| X[i+1] \| \ldots \| X[j]$ for $0 \le i < j < \ell$. We let $\mathrm{left}_\ell(X) = X[0 \ldots (\ell-1)]$ denote the ℓ leftmost bits of X.

For the rest of the section, we fix an arbitrary integer n and call it the block size. Given an $X \in \{0,1\}^*$, we let $X\|10^*$ denote $X\|10^{n-(|X| \bmod n)-1}$ for simplicity. Then let $\mathrm{pad10}(X)$ return X if $|X| \equiv 0 \pmod{n}$ and $X\|10^*$ otherwise. Given a string X, we let $X_1, \ldots, X_x, X_* \xleftarrow{n} X$ denote partitioning X into n-bit blocks with the last block X_* possibly incomplete. For two distinct strings $X_1, \ldots, X_x \xleftarrow{n} X$, $Y_1, \ldots, Y_y \xleftarrow{n} Y$, $\mathrm{llcp}_n(X,Y)$ denotes the length of the longest common prefix of X and Y in n-bit blocks, i.e., the largest $i \le x, y$ with $X_j = Y_j$, $1 \le j \le i$. For the same string, we denote by $\langle i \rangle_d$, some d-bit encoding of a number i with leading zeros, if needed. For two strings $X, Y \in \{0,1\}^*$ with $|X| \le |Y|$, we let $X \oplus Y$ denote the bitwise XOR of $X\|0^{|Y|-|X|}$ and Y. For the same strings, we define $X \oplus_a Y = (X \oplus Y)[0 \ldots a-1]$.

We denote by $X \leftarrow\!\!\$\ \mathcal{X}$ sampling an element X from a finite set $\mathcal{X}$ following the uniform distribution. The symbol $\perp$ denotes an error signal or an undefined value. We use lexicographic comparison of tuples of integers, i.e., $(i', j') < (i, j)$ iff. $i' < i$ or $i' = i$ and $j' < j$.

Syntax of AEAD. We follow the AEAD syntax by Rogaway [67]. A nonce-based AEAD scheme is a triplet $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. The key space $\mathcal{K}$ is a finite set. The deterministic encryption algorithm $\mathcal{E}$ maps a secret key K, a nonce N, associated data A and a message M to a ciphertext $C = \mathcal{E}(K, N, A, M)$. The nonce, associated data and message domains are all subsets of $\{0,1\}^*$. The deterministic decryption algorithm $\mathcal{D}$ takes a tuple (K, N, A, C) and either returns a message M, or a distinguished symbol $\perp$ to indicate an authentication error. For correctness of Π, we require that for all K, N, A, M we have $M = \mathcal{D}(K, N, A, \mathcal{E}(K, N, A, M))$.

OAE Security. Our target AE notion is online AE (OAE) by Fleischmann et al. [35]. We use a variant defined by Hoang et al. [43], who extend it to deal with messages that are not n-bit (block) aligned. We opt for the two-requirement flavor of the notion, separating confidentiality as **oprpf** and authenticity as **auth**. We refer the reader to Appendix D or [5, 35] for the full security definition.

Forkcipher. We follow the formalism by Andreeva et al. [9]. A forkcipher F is a tweakable symmetric-key cipher that maps a secret key K, a tweak T and an input block M of n bits to *two* ciphertext blocks C_0 and C_1, each of size n bit, such that C_0 and C_1 are each an (independent) permutation of M. Formally, a forkcipher is defined as a pair of deterministic algorithms, the encryption algorithm $\mathsf{F} : \{0,1\}^k \times \mathcal{T} \times \{0,1\}^n \times \{0,1,\mathsf{b}\} \to \{0,1\}^n \cup (\{0,1\}^n \times \{0,1\}^n)$ and the inversion algorithm $\mathsf{F}^{-1} : \{0,1\}^k \times \mathcal{T} \times \{0,1\}^n \times \{0,1\} \times \{\mathsf{i}, \mathsf{o}, \mathsf{b}\} \to \{0,1\}^n \cup (\{0,1\}^n \times \{0,1\}^n)$ as illustrated in Fig. 1 where k, n and $\mathcal{T}$ denote the key size, block size and tweak space of F, respectively. Here in the forward call $s \in \{0,1,\mathsf{b}\}$ identifies the desired output branch(es) and in the inverse call $b \in \{0,1\}$ identifies the input's branch and $s \in \{\mathsf{i}, \mathsf{o}, \mathsf{b}\}$ identifies the desired outputs i.e. the inverse, the other branch ciphertext or both. For more details on the syntax of a forkcipher we refer the reader to [9].

The security of a forkcipher F is defined through indistinguishability of the games $\textbf{prtfp-real}_{\mathsf{F}}$ (which implements F faithfully) and $\textbf{prtfp-ideal}_{\mathsf{F}}$ (which replaces F by a pair of tweakable random permutations $\pi_{\mathsf{T},0}, \pi_{\mathsf{T},1} \leftarrow\!\!\$\ \mathrm{Perm}(n)$ for $\mathsf{T} \in \mathcal{T}$ in a natural way), in a chosen ciphertext attack. We define the **prtfp**-advantage of an adversary $\mathcal{B}$ against F as $\mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{B}) = \Pr[\mathcal{A}^{\textbf{prtfp-real}_{\mathsf{F}}} \Rightarrow 1] - \Pr[\mathcal{A}^{\textbf{prtfp-ideal}_{\mathsf{F}}} \Rightarrow 1]$.

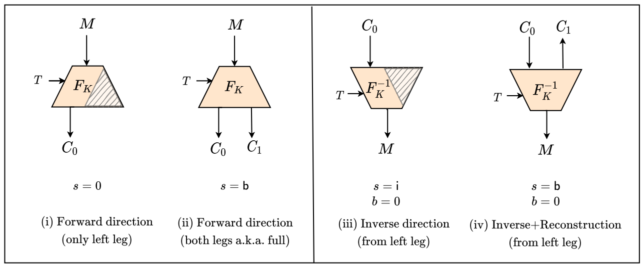

M
T
F_K
C_0
$s = 0$
(i) Forward direction (only left leg)
M
T
F_K
C_0
C_1
$s = \mathsf{b}$
(ii) Forward direction (both legs a.k.a. full)
C_0
T
F_K^{-1}
M
$s = \mathsf{i}$
$b = 0$
(iii) Inverse direction (from left leg)
C_0
C_1
T
F_K^{-1}
M
$s = \mathsf{b}$
$b = 0$
(iv) Inverse+Reconstruction (from left leg)

Figure 1: The forkcipher algorithms.

Statistical Distance. The statistical distance between two random variables X and Y is $\mathrm{SD}(X,Y) = \frac{1}{2} \sum_{x \in \mathcal{X}} |\Pr[X = x] - \Pr[Y = x]|$.

MPC Model of Computation. In many MPC protocols, e.g., [17, 25, 30, 31, 37, 50, 51], the number of communication rounds is proportional to the circuit depth, i.e., the number of multiplications in series in the function to compute.

We illustrate this by the actively secure, SPDZ family of protocols for dishonest majority [18, 30, 31, 50]. These protocols work in the pre-processing model, where a computationally heavy pre-processing (offline) phase precedes a fast online phase. Since the pre-processing only depends on the characteristics of the function but not on the input, the phase can be run in advance, e.g., during low system load.

For computation, additive secret-sharing in a finite field $\mathbb{F}$ is used. We write $[\![x]\!]$ to denote the authenticated share of x. During the online phase, addition/multiplication by constants $a \cdot [\![x]\!] + b$ and addition $[\![x]\!] + [\![y]\!]$ are local operations, i.e., they do not incur communication cost. Multiplications $[\![x]\!] \cdot [\![y]\!]$ require multiplication triples [13] ($[\![a]\!], [\![b]\!], [\![a \cdot b]\!]$) obtained in the offline phase. The parties blind $[\![x]\!] - [\![a]\!]$, $[\![y]\!] - [\![b]\!]$ and then broadcast and reconstruct $[\![x - a]\!]$ as γ and $[\![y - b]\!]$ as ϵ. The multiplication can now be obtained via a local operation $[\![xy]\!] \leftarrow [\![ab]\!] + \gamma \cdot [\![b]\!] + \epsilon \cdot [\![a]\!] + \gamma \cdot \epsilon$. Due to the broadcast, every multiplication in the function requires one interaction between the MPC parties. However, multiplications with independent arguments can be *parallelized* in the same interaction. We call the number of multiplications in series, i.e., those that cannot be parallelized, the depth L of a circuit.

3 THE EEVEE FAMILY AND ITS OAE SECURITY

The Eevee family consists of three AEAD modes, Umbreon, Jolteon and Espeon, that share a common way of processing the associated data (AD). Each of them provides distinct security-performance trade-offs. The underlying cipher of the Eevee family is the tweakable forkcipher F (as defined in Sect. 2).

In encryption, Jolteon and Espeon both use a single branch of the forkcipher up to the last processed message block to boost performance. Additionally, in Jolteon, the forkcipher evaluations can be parallelized during encryption but at the cost of reduced security. On the other hand, Umbreon and Espeon both work sequentially and use either both forkcipher branches throughout (in Umbreon) or longer tweaks (in Espeon) which brings extra security benefits. Decryption is fully parallelizable for all modes, while on the primitive level Umbreon allows the reconstruction and inverse evaluations to be parallelized.

We illustrate the Eevee family AEAD encryption and decryption algorithms in Fig. 2 and 3, and present their complete pseudocode in Fig. 9 (App. B). We state the formal claim about the OAE security of Umbreon, Jolteon and Espeon in Theorem 3.1 and defer its proof to Sect. 7.

Theorem 3.1. *Let* F *be a tweakable forkcipher with tweak space $\mathcal{T} = \{0,1\}^t$. Then for any nonce-respecting adversary $\mathcal{A}_{nr}$ and nonce-misuse adversary $\mathcal{A}_{nm}$ who make at most q_e encryption queries with a nonce repeating at most $\mu \le 2^{n-1}$ times and at most q_v verification queries such that the total number of forkcipher calls induced by all the message parts of encryption and verification queries is at most $\sigma_{m,e}$ and $\sigma_{m,v} \le 2^{n-2}$, respectively, we have*

$$\begin{aligned}
&\mathbf{Adv}^{\mathbf{auth}}_{\mathsf{Umbreon[F]}}(\mathcal{A}_{nm}) - \frac{q_v}{2^{n-4}} \le \mathbf{Adv}^{\mathbf{oprpf}}_{\mathsf{Umbreon[F]}}(\mathcal{A}_{nm}) \\
&\quad \le \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{B}) + \frac{3(\sigma_{m,e}+q_e)(\mu-1)}{2^{n-1}}, \\
&\mathbf{Adv}^{\mathbf{auth}}_{\mathsf{Jolteon[F]}}(\mathcal{A}_{nr}) - \frac{q_v}{2^{n-2}} \le \mathbf{Adv}^{\mathbf{oprpf}}_{\mathsf{Jolteon[F]}}(\mathcal{A}_{nr}) \\
&\quad \le \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{B}), \\
&\mathbf{Adv}^{\mathbf{auth}}_{\mathsf{Espeon[F]}}(\mathcal{A}_{nm}) - \frac{q_v}{2^{n-4}} \le \mathbf{Adv}^{\mathbf{oprpf}}_{\mathsf{Espeon[F]}}(\mathcal{A}_{nm}) \\
&\quad \le \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{B}) + \frac{2q_e(\mu-1)}{2^n} + \frac{\sigma^2_{m,e}}{2^{t-3}}
\end{aligned}$$

for some **prtfp** *adversary $\mathcal{B}$, making at most σ queries to* F *and running in time given by the running time of $\mathcal{A}$ plus $\gamma \cdot \sigma$ where σ denotes the total number of* F *calls induced by all encryption and verification queries and γ is the runtime of an* F *call in the model of computation. Here for* Espeon, *$t \in \{n, 2n\}$.*

Eevee puts forward the first modes optimized for IoT-to-MPC computation. We now compare Eevee modes with the baseline ForkAE [9] modes (PAEF, SAEF and RPAEF). Like SAEF, Umbreon achieves OAE security but additionally offers fully parallel decryption by applying only one mask (i.e. XORing internal states only to the input blocks). This change requires a novel way to get authenticity. We also present a parallel tag-verification to process the whole ciphertext-tag pair which is not provided by existing forkcipher modes. We achieve n-bit security whereas SAEF has $n/2$-bit. The designs of Jolteon and Espeon are inspired by Umbreon. Espeon does not use the second forkcipher leg, and to preserve dependence among calls for error propagation, we include two ciphertext blocks into the tweak and achieve a security of $t/2$-bit equaling n-bit for instantiations with larger tweak of size $2n$ bits. In Jolteon the tags need to depend on every bit of the message, hence a distinct checksum of message and ciphertext is input to the last forkcipher call. We use PAEF's AD processing with a modified tweak encoding for efficiency. Since the consequences of these design choices are not obvious they require a dedicated security analysis which we address in our unified provable treatment of the Eevee family.

4 DESIGN RATIONALE

This section explains the Eevee design decisions that make it suitable for IoT-to-Cloud computation.

Parallel Decryption for MPC. In decryption, the calls to the forkcipher primitive are completely independent, making the circuit depth independent of the ciphertext size. This property is essential for MPC protocols with a round complexity depending on the circuit depth, such as GWM [37], SPDZ-style [17, 30, 31, 50, 51], and CCD [25]. Moreover, the use of a forkcipher allows further parallelization of the branch computation after the forking point which makes Eevee members parallelizable both at mode and primitive levels.

OAE Security. OAE security is achieved in Umbreon by using a full forkcipher call where one branch is used to generate the ciphertexts and the other one is used as both the feed-forward for input blocks to get high error propagation in the following output blocks and to get security against nonce-misusing and/or blockwise adversaries, and to maintain a state for the final authentication tag. Espeon, on the other hand, uses a slightly different approach of feed-forwarding (by using the past two ciphertext blocks) into the tweak instead of the input. This avoids the internal state accumulation of the costly second forkcipher branch and replaces this accumulation by the message checksum but at the cost of larger tweaks. Jolteon, the last member of the family trades the large tweak cost with nonce-misuse security by completely eliminating feed-forwarding and ensuring the tag authentication using combined checksums of message and ciphertext.

Length-Independent Security. All Eevee modes except Espeon (due to the tweak occupied by ciphertexts) use a counter, which results in independence among the cipher calls of a query and hence length-independent security at no additional cost.

Beyond-Birthday Security. Extending the length independence argument, we note that achieving independence among the primitive calls (especially the last ones) not only inside a query but also over all the queries allows us to prevent typical attacks, like birthday in n, on the design (see Sect. 7 for proof). All Eevee modes except Espeon use the nonce along with the counter in the tweak to achieve independence in the nonce-respecting setting and a gradual increase in dependence under nonce-misuse. Espeon, on the other hand, relies on the random collision probability of the tweaks and hence provides birthday security in tweak size t which can be considered beyond-birthday in n for $t > n$. Our instantiation ForkSkinny supports these larger tweak sizes.

Single Primitive. All Eevee modes are based on a single dedicated tweakable forkcipher which, combined with beyond birthday security, allows us to avoid large block primitives, large code size and area for multiple primitives in hardware implementations.

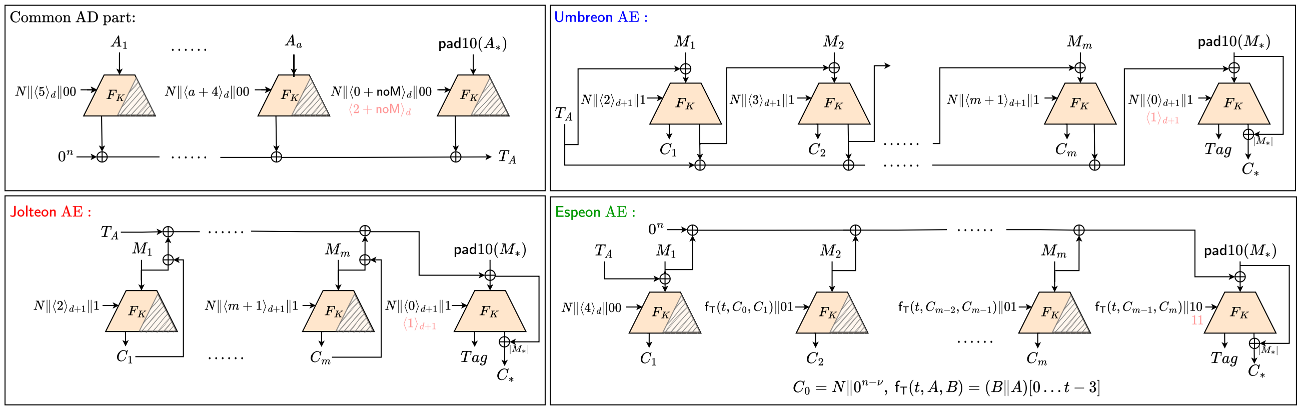

Common AD part:
Umbreon AE :
Jolteon AE :
Espeon AE :
$C_0 = N\|0^{n-\nu}$, $\mathsf{f}_\mathsf{T}(t, A, B) = (B\|A)[0 \ldots t-3]$

Figure 2: The Eevee family of AEAD modes (Encryption diagrams). The top left figure and the rest illustrate the corresponding processing of an associated data A and of a message M of size $na + x_a$ and $nm + x_m$ bits, respectively, for some positive integers $a, m, x_a \leq n$ and $x_m \leq n$. Further, for the last processed AD and message blocks, the tweak counter is set to $\langle 2 + \mathsf{noM} \rangle_d$ and $\langle 1 \rangle_{d+1}$ (11 for Espeon) if the fed block has size n bits and to $\langle 0 + \mathsf{noM} \rangle_d$ and $\langle 0 \rangle_{d+1}$ (10 for Espeon), otherwise, where d is a fixed positive integer defined for the counter size. Here $\nu = |N| \leq n - 2$ and noM represents an indicator function that is set to 1 if $|M| = 0$ and to 0, otherwise.

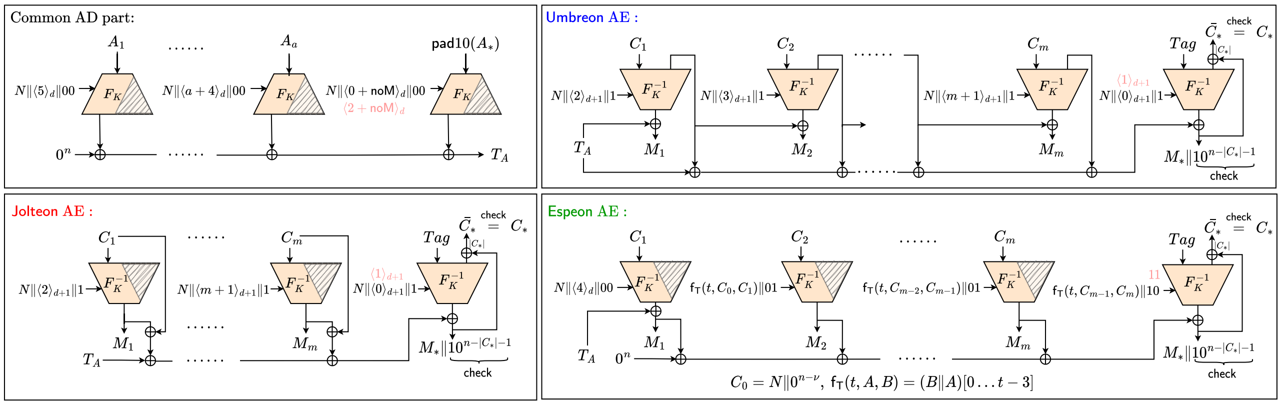

Common AD part:
Umbreon AE :
Jolteon AE :
Espeon AE :
check
$C_0 = N\|0^{n-\nu}$, $\mathsf{f}_\mathsf{T}(t, A, B) = (B\|A)[0 \ldots t-3]$

Figure 3: The Eevee family of AEAD modes (Decryption diagrams). The top left figure and the rest illustrate the corresponding processing of an associated data A and of a ciphertext C of size $na + x_a$ and $nm + x_m$ bits, respectively, for some positive integers $a, m, x_a \leq n$ and $x_m \leq n$. Further, for the last processed AD and ciphertext blocks, the tweak counter is set to $\langle 2 + \mathsf{noM} \rangle_d$ and $\langle 1 \rangle_{d+1}$ (11 for Espeon) if the fed block has size n bits and to $\langle 0 + \mathsf{noM} \rangle_d$ and $\langle 0 \rangle_{d+1}$ (10 for Espeon), otherwise, where d is a fixed positive integer defined for the counter size. Here $\nu = |N| \leq n - 2$ and noM represents an indicator function that is set to 1 if $|C| = 0$ and to 0, otherwise.

5 DISCUSSION

This section discusses the security and efficiency of Eevee. First we talk about the mode level improvements and trade-offs among Eevee modes followed by a performance estimation of Eevee modes when instantiated with ForkSkinny.

Eevee Modes and their Trade-offs. Umbreon achieves a high security level, namely full n-bit OAE security that logarithmically degrades with nonce-misuse. Since Umbreon needs to evaluate both branches of the forkcipher to process the message, its evaluation is relatively costly in MPC decryption when compared with SotA mode CTR-HtMAC that uses a single tweakable block cipher. As a trade-off, Jolteon uses only one leg of the forkcipher to process the message blocks. Consequently, we obtain smaller state requirements and better performance at the cost of losing security under nonce-misuse.

In the same direction of trade-offs but with retaining a good level of nonce-misuse security, Espeon provides similar performance as Jolteon with a well-defined level of tweak-dependent security under nonce-misuse. In other words, its overall security level is bounded by $t/2$ bits. Overall, Espeon is our preferred instantiation as it offers a fixed security level under nonce misuse.

We note that all Eevee modes provide either n-bit or $t/2$-bit OAE security in nonce-respecting settings while previous modes in the MPC literature only provide $n/2$-bit. Hence, Eevee members can use smaller primitives for the same security level. A smaller primitive usually requires less cycles and consumes less power. Table 1 provides a brief comparison on security and software performance estimates of the Eevee modes with the SotA MPC-friendly AEAD schemes.

Instantiating with ForkSkinny. We instantiate the Eevee AEAD modes with ForkSkinny. Its dedicated tweak support (due to the tweakey framework [47]) allows us to achieve the claimed security (being beyond-birthday or full n-bit) of the Eevee modes.

To exemplify, CTR-HtMAC when instantiated with SKINNY-128-256 (128-bit block size, 256-bit tweakey size) and BLAKE2s (256-bit internal state size) provides 64-bit nAE security (due to the typical birthday attacks). On the other hand, Umbreon and Jolteon which are based on a single tweakable forkcipher instantiated with ForkSkinny-64-192 (64-bit block size, 64-bit tweak size and 128-bit key size) achieve the same or a higher level of security.

With a concrete instantiation, we can illustrate the design differences between Eevee modes with respect to the MPC decryption. We count the total number M of multiplications that are required to compute the primitive as well as the depth L, i.e., the number of multiplications in series. This point of view justifies the transition from Umbreon to Jolteon since $M_{\mathsf{FC}^{-1}} > M_{\mathsf{hFC}^{-1}}$ holds for a reasonable forkcipher primitive. Concretely for ForkSkinny-64-192 (denoting the full- and one-legged ForkSkinny version by FS and h-FS, respectively), $M_{\mathsf{FS\text{-}64\text{-}192}^{-1}} = 4000$ and $M_{\mathsf{h\text{-}FS\text{-}64\text{-}192}^{-1}} = 2528$, hence we save around 37% of multiplications in this instantiation. Note further that for ForkSkinny, the depth of a full forkcipher and one-legged forkcipher is the same, namely 230 for ForkSkinny-64-192. The change from Umbreon to Espeon can be studied in the same way. For the 128-bit security level, the number of multiplications used in Umbreon is $M_{\mathsf{FS\text{-}128\text{-}256}^{-1}} = 9536$ and in Espeon $M_{\mathsf{h\text{-}FS\text{-}128\text{-}384}^{-1}} = 7104$ (note the larger ForkSkinny instance since we require a longer tweak to reach 128-bit security). While this change saves 26% of multiplications, the larger instance ForkSkinny-128-384 entails that Espeon's circuit is 10% deeper, from $L_{\mathsf{FS\text{-}128\text{-}256}^{-1}} = 280$ to $L_{\mathsf{FS\text{-}128\text{-}384}^{-1}} = 310$ rounds of communication. Table 2 lists the cost for each mode in terms of pre-processing data and circuit depth. Since AEAD decryption performs a tag check of the n-bit tag, $M_{\mathsf{Tag},n}$ pre-processed triples and $L_{\mathsf{Tag},n}$ rounds are added. This is detailed in Sect. 6.2 and Eq. (1).

6 PERFORMANCE

We want to showcase the performance and security benefits of the Eevee family in two distinct comparisons. In Sect. 6.1, we compare Eevee with the standardized mode AES-GCM [33] and its nonce misuse-resistant variant AES-GCM-SIV [39]. To keep this comparison fair, we instantiate the forkcipher in Eevee with AES by defining a generically composed forkcipher instantiation as a pair of tweakable block ciphers as explained in Sect. 6.1. This way we also illustrate that although a dedicated forkcipher fits the mode best (as it has some built in advantages), one can also fit in any other cipher such as AES following the composition.

Despite the generic instantiation, we show comparable performance and/or superior security properties of Eevee (due to mode level optimizations) in this setting.

Further, in Sect. 6.2, we show concrete performance and security improvements when a dedicated forkcipher primitive is available. We also demonstrate the suitability of forkciphers in the MPC setting using ForkSkinny to instantiate Eevee and using SKINNY in various MPC-friendly modes from the literature [68]. Then, in Sect. 6.3, we briefly sketch a possible instantiation with MPC-friendly primitives but don't report performance results. A MPC-friendly instantiation is not the focus of this work.

We implement the modes in question on a 32-bit ARM Cortex M4 microcontroller to perform the encryption, and benchmark the decryption in a two/three-party MPC cluster. The software implementation on the microcontroller is constant-time, i.e., avoiding secret-dependent control-flow and memory access. For AES, we use dedicated assembly code [69]. The decryption benchmark uses the MP-SPDZ framework [49] where each party runs on a separate machine (4 core E3-1220v3 (3.1 GHz) with 16 GB RAM) in the same network.

6.1 Comparison of Eevee and AES-GCM(-SIV)

We instantiate the AES-based forkcipher from two parallel tweakable block ciphers and use the general LRW [57] construction to make AES tweakable as $E_{k_1}(m \oplus H_{k_2}(t)) \oplus H_{k_2}(t)$ where m denotes the message/input, k_1, k_2 are two secret keys (each of size $|m|$), t is the tweak (with $|t|$ as a multiple of $|m|$) and $E_k(\cdot)$ denotes the AES call with key k. The function H is PolyCW [24, 55], i.e., $H_k(t_1||...||t_a) = (t_1 \otimes k) \oplus \cdots \oplus (t_a \otimes k^a)$ where $|t_i| = |m|$ for all $1 \leq i \leq a$. This generic forkcipher provides around 64 bits of security when instantiated with AES-128. The proof of this claim is straightforward from the results of [57, Theorem 3] and [24].

Implementation. In MPC, AES is implemented by embedding the AES implementation from MP-SPDZ [29] into the GHASH field to turn GHASH multiplication into standard field multiplication in the MPC protocol.

Comparison. Figure 4 shows the encryption vs. decryption cost for the studied modes. The encryption cost is listed as the average number of cycles per message byte to encrypt and the decryption cost is the online time of a three-party distributed decryption in a LAN setting using the MASCOT [50] actively secure MPC protocol that tolerates a dishonest majority. The modes AES-GCM and Jolteon-AES are not nonce misuse resistant while AES-GCM-SIV and Espeon-AES are. Jolteon improves over AES-GCM both in encryption and in decryption performance. We see an improvement of ≈10% for encryption and ≈39% for decryption. However, both of these modes don't offer nonce misuse resistance. In the nonce misuse setting, Espeon-AES has the same security level as AES-GCM-SIV. Espeon improves the decryption time by ≈77% while we measure a slower encryption performance of around 15%. It appears that in our prototype implementation, computing H for the longer tweak in Espeon is not sufficiently optimized. The choice of H mainly allows the tweak extension of AES to be almost free in MPC, costing only one additional multiplication and bit-decomposition operation per block. This choice together with Espeon's fully parallel nature allows for the decryption performance boost.

6.2 Comparison of Eevee in Lightweight Settings

In the previous section, we already demonstrated that the Eevee family leads to performance and security improvements even if no dedicated (or lightweight) forkcipher primitive is available. In this section, we expand on a scenario where a lightweight forkcipher primitive is available, and showcase the additional performance gain due to an optimized primitive. We instantiate Umbreon, Jolteon and

Table 2: The amount of pre-processing data and the number of communication rounds for computing the decryption of each mode. M^p_{P} is the number of multiplications needed to evaluate the primitive P with public input and secret key; M^s_{P} denotes the evaluation with secret input and secret key. The primitives are block cipher (BC), tweakable block cipher (TBC), forkcipher (FC) and half/one-legged forkcipher (hFC) with instatiations AES, SKINNY and ForkSkinny (FS), respectively. P^{-1} denotes the inverse of the primitive. L with the same notation denotes the number of communication rounds with L^p and L^s denoting public and secret input, respectively. $|C|$ is the ciphertext length and n is the block size. For tag verification, $M_{\mathsf{Tag},64} = 63$, $M_{\mathsf{Tag},128} = 127$, $L_{\mathsf{Tag},64} = 7$ and $L_{\mathsf{Tag},128} = 8$.

Mode	Pre-processing Data	Minimum Communication Rounds	Instantiations	
			Data	Rounds
CTR-HtMAC	$\left(\left\lceil \frac{\lvert C\rvert}{n} \right\rceil + 1\right) \cdot M^p_{\mathsf{TBC}} + M_{\mathsf{Tag},n}$	$L^p_{\mathsf{TBC}} + L_{\mathsf{Tag},n}$	$M^p_{\mathsf{SKINNY\text{-}128\text{-}256}} = 6016$	$L^p_{\mathsf{SKINNY\text{-}128\text{-}256}} = 235$
CTR-PMAC	$2\left\lceil \frac{\lvert C\rvert}{n} \right\rceil \cdot M^p_{\mathsf{TBC}} + M^s_{\mathsf{TBC}} + M_{\mathsf{Tag},n}$	$L^p_{\mathsf{TBC}} + L^s_{\mathsf{TBC}} + L_{\mathsf{Tag},n}$	$M^s_{\mathsf{SKINNY\text{-}128\text{-}256}} = 6144$	$L^s_{\mathsf{SKINNY\text{-}128\text{-}256}} = 240$
Umbreon	$\left\lceil \frac{\lvert C\rvert}{n} \right\rceil \cdot M^p_{\mathsf{FC}^{-1}} + M_{\mathsf{Tag},n}$	$L^p_{\mathsf{FC}^{-1}} + L_{\mathsf{Tag},n}$	$M^p_{\mathsf{FS\text{-}64\text{-}192}^{-1}} = 4000$ $M^p_{\mathsf{FS\text{-}128\text{-}256}^{-1}} = 9536$	$L^p_{\mathsf{FS\text{-}64\text{-}192}^{-1}} = 230$ $L^p_{\mathsf{FS\text{-}128\text{-}256}^{-1}} = 280$
Jolteon Espeon	$\left(\left\lceil \frac{\lvert C\rvert}{n} \right\rceil - 1\right) \cdot M^p_{\mathsf{hFC}^{-1}} + M^p_{\mathsf{FC}^{-1}} + M_{\mathsf{Tag},n}$	$\max\left\{L^p_{\mathsf{hFC}^{-1}}, L^p_{\mathsf{FC}^{-1}}\right\} + L_{\mathsf{Tag},n}$	$M^p_{\mathsf{h\text{-}FS\text{-}64\text{-}192}^{-1}} = 2528$ $M^p_{\mathsf{h\text{-}FS\text{-}128\text{-}256}^{-1}} = 6080$ $M^p_{\mathsf{FS\text{-}128\text{-}384}^{-1}} = 11072$ $M^p_{\mathsf{h\text{-}FS\text{-}128\text{-}384}^{-1}} = 7104$	$L^p_{\mathsf{h\text{-}FS\text{-}64\text{-}192}^{-1}} = 230$ $L^p_{\mathsf{h\text{-}FS\text{-}128\text{-}256}^{-1}} = 280$ $L^p_{\mathsf{FS\text{-}128\text{-}384}^{-1}} = 310$ $L^p_{\mathsf{h\text{-}FS\text{-}128\text{-}384}^{-1}} = 310$
GCM	$\left(\left\lceil \frac{\lvert C\rvert}{n} \right\rceil + 2\right) \cdot M_{\mathsf{BC}} + M_{\mathsf{Tag},n}$	$L_{\mathsf{BC}} + \max\{L_{\mathsf{BC}}, \left\lceil \frac{\lvert C\rvert}{n} \right\rceil + L_{\mathsf{Tag},n}\}$	$M_{\mathsf{AES}} = 1200$	$L_{\mathsf{AES}} = 52$
GCM-SIV	$\left(\left\lceil \frac{\lvert C\rvert}{n} \right\rceil + 5\right) \cdot M_{\mathsf{BC}} + M_{\mathsf{Tag},n}$	$2L_{\mathsf{BC}} + \left\lceil \frac{\lvert C\rvert}{n} \right\rceil + L_{\mathsf{Tag},n}$		

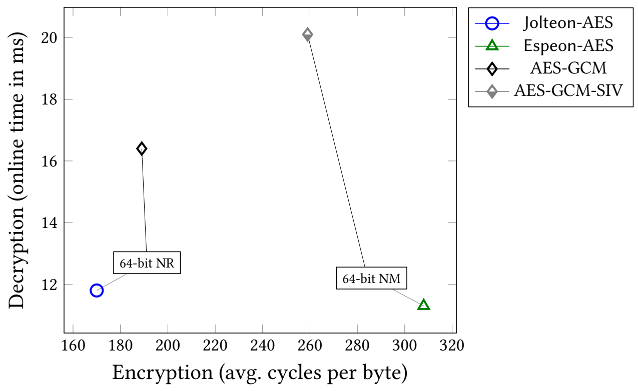


Figure 4: Encryption vs. decryption performance of AES-based modes for message lengths ≥ 100-byte for three MPC parties in a LAN setting. Here NR and NM refers the modes with nonce-respecting and nonce-misuse security, respectively.

Espeon with the forkcipher ForkSkinny. We compare their performance with the MPC-friendly modes in the literature [68] CTR-HtMAC-SKINNY-128-256 – counter mode encryption then hash-then-MAC using SKINNY-128-256 and BLAKE2s, and with CTR-PMAC-SKINNY-128-256 – counter mode encryption then PMAC using SKINNY-128-256. To minimize the impact of the different primitives and allow for a stronger comparison, we choose the SKINNY cipher as the underlying cipher instance for these modes.

Implementation. At the core of the MPC implementation lies the representation of the SKINNY round function since ForkSkinny reuses the same steps with minor modifications. The forward round function consists of SubCells, AddConstants, AddRoundTweakey, ShiftRows and MixColumns. All the steps except for SubCells are linear and therefore local operations. Furthermore, if the tweakey is available in shared bits, the key schedule of SKINNY is also linear. We represent each 4-bit (8-bit) cell of the state as element in $\mathbb{F}_{2^4}$ ($\mathbb{F}_{2^8}$) embedded in $\mathbb{F}_{2^{40}}$. The inverse round function consists of the respective inverse steps in reverse order. The SubCells step applies the SKINNY 4-bit (8-bit) S-box in parallel to each cell of the state. We compute the S-box by first decomposing the cell into 4 (8) bits and then apply NOT, XOR and AND operations to compute the desired function. Naturally in the arithmetic setting with fields of characteristic two, NOT corresponds to addition with a constant 1, XOR to addition and AND to multiplication.

Consequently, the forward SKINNY round function consumes $16 \cdot 4$ ($16 \cdot 8$) multiplication triples and $16 \cdot 4$ ($16 \cdot 8$) random bits in 3 (5) rounds of communication in the online phase of MASCOT. The inverse SKINNY round function consumes the same number of multiplication triples and random bits as the forward direction but takes 5 rounds of communication both for the 64-bit and 128-bit state.

There is a subtle difference between the forward direction for SKINNY, used in CTR-HtMAC and CTR-PMAC, and the inverse

ForkSkinny, used in Umbreon, Jolteon and Espeon. Since the input to both primitives is public with a secret-shared key, the first round is not computed completely in secret. For SKINNY using the forward direction of the SKINNY round function, all S-boxes are computed in the clear since the key addition and mixing layers follow the substitution layer. The state becomes secret only after the key addition and mixing. For ForkSkinny, which uses the inverse SKINNY round function, this is reversed, so the key addition precedes the substitution layer. Consequently, half of the S-boxes are computed in secret because the round key is XORed to half of the cells in the state. The other half of the S-boxes is computed in the clear. Therefore, the number of required multiplication triples is $(r-1)\cdot 16\cdot s$ for SKINNY and $(r-1)\cdot 16\cdot s+8\cdot s$ for inverse ForkSkinny where r is the number of rounds in the primitive, e.g., $r=48$ for SKINNY-128-256, and s is the cost to compute one S-box, i.e., $s=4$ for SKINNY-64-* and $s=8$ for SKINNY-128-*.

Umbreon, Jolteon and Espeon, but also CTR-HtMAC don't require non-linear operations on the mode level except for the tag equality check. To check that two ℓ-bit tags $t_1,\ldots,t_\ell$ and $t'_1,\ldots,t'_\ell$ match, we compute

$$[\![c]\!]=\bigwedge_{i=1}^{\ell}\overline{[\![t_i]\!]\oplus t'_i}=\prod_{i=1}^{\ell}\left([\![t_i]\!]+t'_i+0\text{x}1\right). \quad (1)$$

Note that since for this check only one tag (the computed one) is secret-shared while the other is part of the public input ciphertext, we write $[\![t_i]\!]$ and t'_i. The shared output bit $[\![c]\!]$ is 1 if all bits in t match the bits at the same position in t'. Computing the product consumes $\ell-1$ multiplication triples. To minimize the depth of the computation, we arrange the multiplications in a binary tree with $\lceil\log_2(\ell)\rceil$ levels. We further consume ℓ random bits to decompose the output state of the primitive into shared bits, making the total number of rounds of communication $\lceil\log_2(\ell)\rceil+1$.

Comparison. In Fig. 5 we show the encryption and decryption cost for SKINNY/ForkSkinny-based modes. As in the previous section, the encryption cost is the average number of cycles required to encrypt messages normalized to the message size. The decryption cost is the online time of a two-party distributed decryption in a LAN setting.

Figure 8 in Appendix A illustrates further performance metrics of the MPC decryption such as offline time and online/offline communication cost. First, we compare the Eevee family with CTR-PMAC. Afterwards, we assess the performance with CTR-HtMAC as baseline.

CTR-PMAC. All our proposed instantiations of Umbreon, Jolteon and Espeon outperform CTR-PMAC-SKINNY-128-256. The modes with the same security level, Jolteon-ForkSkinny-64-192 and Umbreon-ForkSkinny-64-192, encrypt messages faster with factor 1.75 to 3.64, depending on the message size. The modes with a doubled security level, namely Jolteon-ForkSkinny-128-256, Espeon-ForkSkinny-128-384 and Umbreon-ForkSkinny-128-256 also have improved encryption performance with a factor 1.29 to 2.03.

Jolteon-ForkSkinny-64-192 decrypts faster and with less communication data by at least factor 2.42 to 4.5 and Umbreon-ForkSkinny-64-192 improves by a factor 1.55 to 4.5. The remaining instantiations, Jolteon-ForkSkinny-128-256, Espeon-ForkSkinny-128-384, and Umbreon-ForkSkinny-128-256, also improve in decryption time over CTR-PMAC, despite the doubled security level. They improve by a factor of 1.89 to 1.97, 1.63 to 1.69, and 1.89 to 1.29. Umbreon and Espeon provide nonce-misuse resistance.

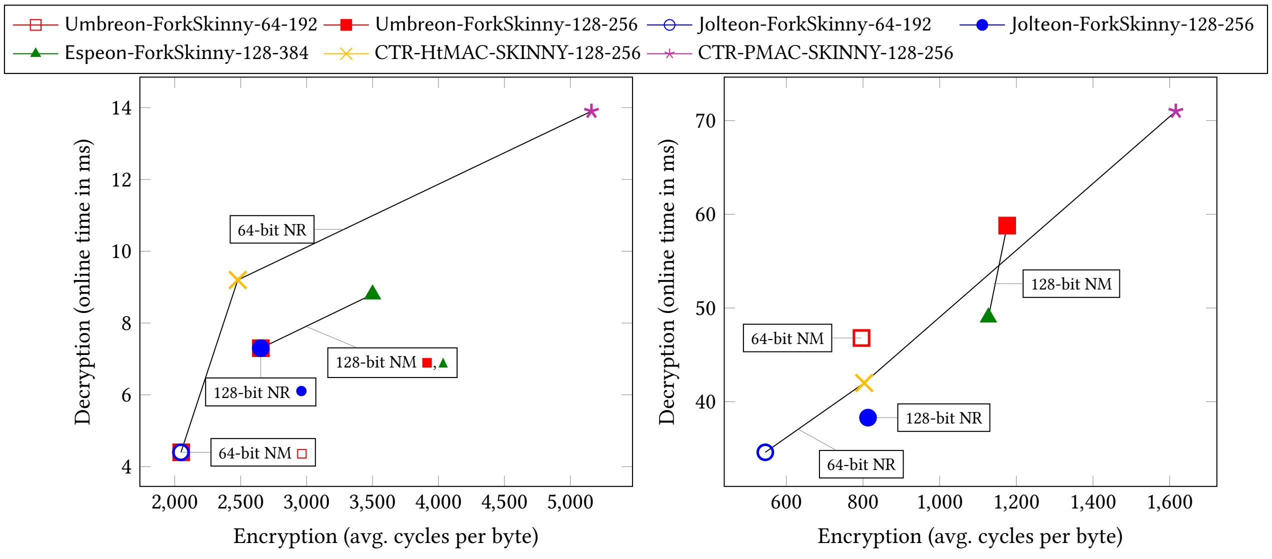
Thus, for an AEAD mode with a single primitive use, all Eevee members surpass CTR-PMAC both in performance and security. First, Eevee modes that are instantiated with a smaller primitive still attain an equivalent level of security as CTR-PMAC with a larger primitive, e.g., both Umbreon-ForkSkinny-64-192 and CTR-PMAC-SKINNY-128-256 achieve 64-bit of nAE security. Second, Eevee modes that are instantiated with a primitive of the same size as CTR-PMAC, benefit from doubled nAE security. Third, Umbreon and Espeon offer graceful security degradation in the nonce-misuse setting while CTR-PMAC is trivially insecure under nonce- misuse attacks (see [4, App. C]).

CTR-HtMAC. We now compare Jolteon, Espeon and Umbreon to CTR-HtMAC-SKINNY-128-256. For the same security level Jolteon-ForkSkinny-64-192 and Umbreon-ForkSkinny-64-192 encrypt faster with factor up to 1.85. For Espeon-ForkSkinny-128-384, Umbreon-ForkSkinny-128-256, and Jolteon-ForkSkinny-128-256, we measure slower encryption for messages ranging from 8 bytes to 1500 bytes of 52% to 33%, 9% to 49%, and 9% to −3% (i.e., for messages of length 500 bytes or larger, Jolteon-ForkSkinny-128-256 gains 3% speed-up), respectively. Note the doubled security level of these instantiations compared to CTR-HtMAC-SKINNY-128-256.

Regarding decryption, Jolteon-ForkSkinny-64-192 shows better performance for all message sizes by factor of 3 to 1.23 in all metrics. Umbreon-ForkSkinny-64-192 improves decryption time by factor 3 for 8 byte messages over CTR-HtMAC-SKINNY-128-256 while we measure between 18% and 27% slower decryption time and more communication data for longer messages. Jolteon-ForkSkinny-128-256 shows a speed up by factor 1.26 for short messages and has very similar performance in all metrics for the other message lengths. Both Espeon-ForkSkinny-128-384 and Umbreon-ForkSkinny-128-256 have a slight performance improvement for short messages but are slower and use more data for longer messages. We measure a difference of 12% − 16% and 41% − 54%, respectively.

For short messages that fit into one or two blocks, the parallel nature of the forkcipher shows to be especially advantageous since all our instantiations improve even over CTR-HtMAC which outsources the tag computation to a local hash, i.e., with negligible cost, as seen in Fig. 5a. Furthermore, our results validate the design of Jolteon to improve the performance. We see a reduced amount of exchanged online and offline data as well as faster online and offline times to the extend that we obtain superior (with same security) or similar (with double security) performance than CTR-HtMAC for our two instantiations. The additional security of Espeon when compared to Jolteon comes from the larger tweak. We note a slight performance improvement when compared to Umbreon but due to the larger required instance ForkSkinny-128-384, Espeon doesn't outperform SKINNY-128-256.

As for the comparison for CTR-PMAC, we stress that Umbreon and Espeon offer nonce-misuse resistance. For hardware implementations, CTR-HtMAC requires two primitives, the block cipher and a hash function, which will likely require more area than a single primitive (cf. Table 1). Jolteon, Espeon and Umbreon all use a single primitive in the mode.



(a) Message length = 8 byte. **(b) Message length ≥ 100 byte.**

Figure 5: Encryption vs. decryption performance of the studied modes for very short messages (a) and longer messages (b). Modes with the same security level lie on the same curve. The annotation denotes the OAE security level for confidentiality (64-bit or 128-bit). NR and NM denote the nonce respecting and the nonce misuse settings, respectively.

6.3 Eevee in the MPC-friendly setting

A comparison with MPC-friendly primitives is outside the scope of this paper since we prioritize a lightweight, IoT-friendly encryption. However, the Eevee modes can also be instantiated with an MPC-friendly block cipher, such as MiMC, Hades, Vision, etc. [1, 3, 38], using the LRW [57] construction (as described in Sect. 6.1). Note that decryption in Eevee requires inverting the primitive which may be more expensive in some MPC-friendly block ciphers. Special-purpose MPC protocols can optimize primitive inversion and create a trade-off with a slightly more expensive offline phase (see, e.g., [3, Appendix C.3]). Our implementation of transciphering based on MiMC-128 from [68] yields the following performance. For CTR-PMAC-MiMC-128, encryption and decryption of a 100 byte message takes 67623 cpb and 3.08 ms, respectively. The same for a short 8 byte message takes 184771 cpb and 0.82 ms. For CTR-HtMAC-MiMC-128, long messages take 36004 cpb to encrypt and 1.58 ms to decrypt while short messages take 109456 cpb and 0.51 ms. Compared to these, Jolteon-ForkSkinny-64-192 (which has the same security level) comes with 545 cpb to encrypt and 34.6 ms to decrypt for long messages and 2147 cpb and 4.4 ms for short messages, respectively.

7 SECURITY ANALYSIS

Proof of Theorem 3.1. For brevity of the proof, we use throughout (w.l.o.g.) the same notation of $\mathcal{A}$ and $\mathcal{B}$ (unless defined) for the adversary against the AEAD mode and its underlying primitive disregarding the corresponding setup (the security notion and the targeted design), respectively. We clarify that $\mathcal{A}$ and $\mathcal{B}$ are not some pre-fixed adversaries and can be modified to the best strategy for the targeted notion and design when rementioned.

Replacing F. We first replace F with a pair of independent random tweakable permutations $\pi_0 = (\pi_{T,0} \xleftarrow{\$} \mathrm{Perm}(n))_{T\in\{0,1\}^t}$ and $\pi_1 = (\pi_{T,1} \xleftarrow{\$} \mathrm{Perm}(n))_{T\in\{0,1\}^t}$ and let $\Pi' = \Pi[(\pi_0, \pi_1)]$ denote the $\Pi \in \{\mathsf{Umbreon}, \mathsf{Jolteon}, \mathsf{Espeon}\}$ mode that uses π_0, π_1 instead of F, which yields

$$\mathbf{Adv}^{\mathrm{oprpf}}_{\Pi[\mathsf{F}]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{prtfp}}_{\mathsf{F}}(\mathcal{B}) + \mathbf{Adv}^{\mathrm{oprpf}}_{\Pi'}(\mathcal{A})$$
$$\mathbf{Adv}^{\mathrm{auth}}_{\Pi[\mathsf{F}]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{prtfp}}_{\mathsf{F}}(\mathcal{B}) + \mathbf{Adv}^{\mathrm{auth}}_{\Pi'}(\mathcal{A})\ .$$

Now, $\mathcal{A}$ is left with the goal of distinguishing between the games **oprpf-real**$_{\Pi'}$ and **oprpf-ideal**$_{\Pi'}$ for the confidentiality of Π and similarly between the games **auth-real**$_{\Pi'}$ and **auth-ideal**$_{\Pi'}$ for the integrity of Π. For simplicity, we denote these games by "real world" and "ideal world" regarding confidentiality and integrity, respectively. Hence, we now want to bound $\mathbf{Adv}^{\mathrm{oprpf}}_{\Pi'}(\mathcal{A})$ and $\mathbf{Adv}^{\mathrm{auth}}_{\Pi'}(\mathcal{A})$.

Integrity. Let us recall from Sect. 2 and [5] that

$$\mathbf{Adv}^{\mathrm{auth}}_{\Pi'}(\mathcal{A}) = \Pr\left[\mathcal{A}^{\mathbf{auth}_{\Pi'}} \text{forges}\right] \leq \mathbf{Adv}^{\mathrm{oprpf}}_{\Pi'}(\mathcal{A})$$
$$+ \Pr\left[\mathcal{A}^{\mathbf{auth}_{\Pi'}} \text{forges} \mid \Pi' \text{ is } \mathbf{oprpf}\text{-secure}\right]\ .$$

Now, since Π being **oprpf**-secure implies by definition that all generated tags for non-duplicate queries (even when the queried message length is zero, i.e., only AD is queried) are indistinguishable from uniform random n-bit strings and are independently generated from the ciphertexts, we can apply the single to multiple verification queries relation [15, Theorem 5.1] and get,

$$\mathbf{Adv}^{\mathrm{auth}}_{\Pi'}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{oprpf}}_{\Pi'}(\mathcal{A}) + q_v \cdot \Pr\Big[\mathcal{A}^{\mathbf{auth}_{\Pi'}} \text{forges} \mid \Pi' \text{ is } \mathbf{oprpf}\text{-secure and } \mathcal{A} \text{ makes only 1 verification query}\Big]\ , \quad (2)$$

where q_v represents the total number of verification/decryption queries made by $\mathcal{A}$ during the whole session. Now the probability of adversary $\mathcal{A}$ forging a valid ciphertext-tag pair can be computed easily for all possible scenarios (describing how an adversary can make a single verification query) as follows:

Case 1. The verification query contains a new tuple of nonce N, AD A and ciphertext $C_1\|\ldots\|C_m$, i.e., a $N\|A\|C_1\|\ldots\|C_m$ value is new when compared with previously made encryption queries. Under this scenario, we know that Π' being **oprpf**-secure implies that at least one of the corresponding checksum blocks, i.e., blocks that are xored to the input of the final primitive call of the corresponding verification query (for example, in Espeon these blocks are $M_1, \ldots, M_m$) is fresh, independent and randomly sampled with probability at most $1/(2^n - (x_{\Pi'} - 1))$ where $x_{\Pi'}$ denotes the maximum number of times a tweak can repeat during a full session of q_e encryption and q_v verification queries to Π'. Hence, the corresponding padded input block of the form $M_*\|10^*$ (which can be computed by inverting the last π_0 call of the verification query with the Tag value and xoring it with the checksum) will be fresh, independent and randomly sampled with probability at most $1/(2^n - (x_{\Pi'} - 1))$.

Case 2. The verification query contains a nonce, AD and ciphertext from an old encryption query, i.e., $N\|A\|C_1\|\ldots\|C_m$ value repeats from one of the already made encryption queries: Since Π' is a nonce-based deterministic AEAD scheme, old input will derive the corresponding old checksum value and hence to have a valid forgery with non-zero success probability, Tag and C_* both are required to differ from the corresponding old query. We note that since Π' is **oprpf**-secure, we can consider all q_e encryption query tags and last ciphertext blocks as independent uniform random strings. Hence, disregarding the fact of having a Tag value which is completely new or is equal to one of the other old encryption query tags, we have that the padded input block of the form $M_*\|10^*$ (which corresponds to the Tag and C_* blocks of the verification query) is equal to any arbitrary n-bit string with random probability of $1/(2^n - (x_{\Pi'} - 1))$.

Note that both of these cases are disjoint and exhaustive, therefore, the probability of $\mathcal{A}$ forging a ciphertext-tag pair for Π' in a single verification query (which can be defined as the product of two probabilities: (1) the probability that the padding 10^* from the last π_0 call input $M_*\|10^*$ matches $10^{n-|M_*|-1}$; (2) the probability that the last π_1 call output when xored with C_* and truncated by the last $n - |M_*| - 1$ bits matches M_*) can be upper bounded by $(2^{n-|M_*|}/(2^n - (x_{\Pi'} - 1))) \cdot (2^{|M_*|}/(2^n - (x_{\Pi'} - 1))) = 2^n/(2^n - (x_{\Pi'} - 1))^2$. Hence, revising Eq. (2), we get for $\Pi \in \{\mathsf{Umbreon}, \mathsf{Jolteon}, \mathsf{Espeon}\}$

$$\mathbf{Adv}^{\mathbf{auth}}_{\Pi'}(\mathcal{A}) \le \mathbf{Adv}^{\mathbf{oprpf}}_{\Pi'}(\mathcal{A}) + q_v 2^n/(2^n - (x_{\Pi'} - 1))^2 . \quad (3)$$

We defer the individual definition of $x_{\Pi'}$ for $\Pi \in \{\mathsf{Umbreon}, \mathsf{Jolteon}, \mathsf{Espeon}\}$ to the final bounds segment which is defined after the following segment of confidentiality.

Confidentiality. We now bound the **oprpf**-security advantage of a confidentiality adversary $\mathcal{A}$ against $\Pi' = \Pi[(\pi_0, \pi_1)]$. Let us denote by q_e the number of total encryption queries made by $\mathcal{A}$ during the whole session. Further, let us denote by $\mathcal{U}^{\Pi',\mathcal{A}}_{i,\text{case}}$ the event when $\mathcal{A}$ successfully distinguishes the received output for its i^{th} encryption query from being a real Π' output or an ideal ORP+RF (as defined by **oprpf** notion in Sect. 2 and [5]) output. Here case defines the type of the i^{th} encryption query among the all possible types/cases as defined in Fig. 6.

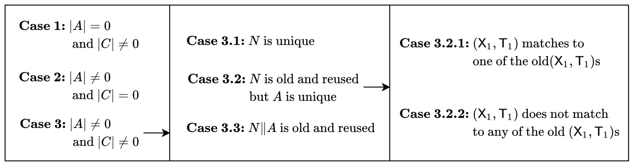

Case 1: $|A| = 0$ and $|C| \neq 0$
Case 2: $|A| \neq 0$ and $|C| = 0$
Case 3: $|A| \neq 0$ and $|C| \neq 0$
Case 3.1: N is unique
Case 3.2: N is old and reused but A is unique
Case 3.3: $N\|A$ is old and reused
Case 3.2.1: $(\mathsf{X}_1, \mathsf{T}_1)$ matches to one of the old$(\mathsf{X}_1, \mathsf{T}_1)$s
Case 3.2.2: $(\mathsf{X}_1, \mathsf{T}_1)$ does not match to any of the old $(\mathsf{X}_1, \mathsf{T}_1)$s

Figure 6: Exhaustive and disjoint cases of a possible encryption query to $\Pi \in \{\mathsf{Umbreon}, \mathsf{Jolteon}, \mathsf{Espeon}\}$. Here $(\mathsf{X}_1, \mathsf{T}_1)$ represents the input-tweak pair of the first primitive call that outputs the first ciphertext for the query.

With the notations defined, we can infer the following expression for all $1 \le i \le q_e$ and $\Pi \in \{\mathsf{Umbreon}, \mathsf{Jolteon}, \mathsf{Espeon}\}$:

$$\mathcal{U}^{\Pi',\mathcal{A}}_{i,1} \supseteq \left(\mathcal{U}^{\Pi',\mathcal{A}}_{i,3.1} \cup \mathcal{U}^{\Pi',\mathcal{A}}_{i,3.2.2} \cup \mathcal{U}^{\Pi',\mathcal{A}}_{i,3.3}\right) . \quad (4)$$

This holds because under Case 1, T_A is set to 0 and therefore, $(\mathsf{X}_1, \mathsf{T}_1)$ is solely defined by $N\|M_1$ which means the values of X_1 and T_1 are fully in the control of $\mathcal{A}$. Hence, for every adversary $\mathcal{A}'$ who corresponds to either $\mathcal{U}^{\Pi',\mathcal{A}'}_{i,3.1}$ or $\mathcal{U}^{\Pi',\mathcal{A}'}_{i,3.2.2}$ or $\mathcal{U}^{\Pi',\mathcal{A}'}_{i,3.3}$, there exists another adversary $\mathcal{A}$ corresponding to $\mathcal{U}^{\Pi',\mathcal{A}}_{i,1}$ who modifies the input-tweak values X_1, T_1 according to $\mathcal{A}'$ and achieves the same distinguishing (oprpf) advantage as $\mathcal{A}'$. Now, since all these cases (as shown in Fig. 6) are disjoint and exhaustive, we can say from Eq. (4) that

$$\mathbf{Adv}^{\mathbf{oprpf}}_{\Pi'}(\mathcal{A}) \le \sum_{i=1}^{q_e} \max\left\{\Pr\left[\mathcal{U}^{\Pi',\mathcal{A}}_{i,1}\right], \Pr\left[\mathcal{U}^{\Pi',\mathcal{A}}_{i,2}\right], \Pr\left[\mathcal{U}^{\Pi',\mathcal{A}}_{i,3.2.1}\right]\right\} . \quad (5)$$

Hence, we are now left with bounding these three $\mathcal{U}$ terms for each Eevee mode. We defer the case analysis and bounding of these three $\mathcal{U}$ terms to Appendix C and recall from there the results of Eq. (6), (7), (8), (9), (10) and (12) as shown below.

For $\Pi \in \{\mathsf{Umbreon}, \mathsf{Jolteon}, \mathsf{Espeon}\}$:

$$\Pr\left[\mathcal{U}^{\Pi',\mathcal{A}}_{i,3.2.1}\right] < \frac{\mu - 1}{2^n - \mu} \qquad (6) \text{ in Appendix C}$$

$$\Pr\left[\mathcal{U}^{\Pi',\mathcal{A}}_{i,2}\right] \le \frac{(\mu - 1)}{2^n} \qquad (7) \text{ in Appendix C}$$

$$\Pr\left[\mathcal{U}^{\mathsf{Jolteon}',\mathcal{A}_{nr}}_{i,1}\right] = 0 \qquad (8) \text{ in Appendix C}$$

For $\Pi \in \{\mathsf{Umbreon}, \mathsf{Espeon}\}$ and $L_i \le 2^{n-1}$:

$$\Pr\left[\mathcal{U}^{\Pi',\mathcal{A}_{nm}}_{i,1}\right] \le \Pr\left[\mathcal{U}^{\Pi',\mathcal{A}^\star}_{i,1}\right] + \frac{\mu - 1}{2^n - \mu} \qquad (9) \text{ in Appendix C}$$

$$\Pr\left[\mathcal{U}^{\mathsf{Umbreon}',\mathcal{A}^\star}_{i,1}\right] \le \frac{3\ell_i(\mu - 1)}{2^n} \qquad (10) \text{ in Appendix C}$$

$$\Pr\left[\mathcal{U}^{\mathsf{Espeon}',\mathcal{A}^\star}_{i,1}\right] \le \frac{(L_i - \ell_i + (\ell_i - 1)/2)\ell_i}{2^{t-4}} , \qquad (12) \text{ in Appendix C}$$

where μ denotes the maximum number of times a nonce can repeat during a session, $\mathcal{A}_{nr}$ (resp. $\mathcal{A}_{nm}$) represents a nonce-respecting (resp. nonce-misusing) adversary, ℓ_i represents the total number of primitive calls that are required to process M^i, i.e., $\ell_i = \lceil |M^i|/n \rceil \ge 0$ and $L_i = \sum_{a=1}^{i} \ell_a$.

Final Bounds. Now, combining Eq. (5), (6), (7) and (8) with $\mu = 1$ (for $\mathcal{A} = \mathcal{A}_{nr}$), we get

$$\mathbf{Adv}^{\mathbf{oprpf}}_{\mathsf{Jolteon}'}(\mathcal{A}_{nr}) \leq \sum_{i=1}^{q_e} \max\left\{0, 0, 0\right\} = 0\,.$$

Further, let $\sigma_{m,e} = \sum_{i=1}^{q_e} \ell_i$ denote the total number of primitive calls induced by all the message parts from $\mathcal{A}_{nm}$'s q_e encryption queries to Π' then combining Eq. (5), (6), (7), (9) and (10), we get

$$\begin{aligned}\mathbf{Adv}^{\mathbf{oprpf}}_{\mathsf{Umbreon}'}(\mathcal{A}_{nm}) &\leq \sum_{i=1}^{q_e} \max\left\{\frac{\mu-1}{2^n-\mu}, \frac{(\mu-1)}{2^n}, \frac{3\ell_i(\mu-1)}{2^n} + \frac{\mu-1}{2^n-\mu}\right\}\\ &\leq \sum_{i=1}^{q_e} \frac{3(\ell_i+1)(\mu-1)}{2^n};\ \text{assuming } \mu \leq 2^{n-1}\\ &= \frac{3(\sigma_{m,e}+q_e)(\mu-1)}{2^n}\,.\end{aligned}$$

Similarly, combining Eq. (5), (6), (7), (9) and (12) for $t \in \{n, 2n\}$ and assuming $\sigma_{m,e} \leq 2^{n-1}$, we get

$$\begin{aligned}&\mathbf{Adv}^{\mathbf{oprpf}}_{\mathsf{Espeon}'}(\mathcal{A}_{nm})\\ &\leq \sum_{i=1}^{q_e}\left\{\frac{\mu-1}{2^n-\mu}, \frac{(\mu-1)}{2^n}, \frac{(L_i-\ell_i+(\ell_i-1)/2)\ell_i}{2^{t-4}} + \frac{\mu-1}{2^n-\mu}\right\}\\ &\leq \frac{2q_e(\mu-1)}{2^n} + \frac{1}{2^{t-4}}\sum_{i=1}^{q_e}\left(L_i-\ell_i+\frac{(\ell_i-1)}{2}\right)\ell_i;\ \text{assuming } \mu \leq 2^{n-1}\\ &= \frac{2q_e(\mu-1)}{2^n} + \frac{\sigma_{m,e}(\sigma_{m,e}-1)}{2^{t-3}}\,.\end{aligned}$$

Note that for $\sigma_{m,e} > 2^{n-1}$ this bound becomes void hence the assumption of $\sigma_{m,e} \leq 2^{n-1}$ can be dropped. Finally in Eq. (3), setting $x_{\Pi'}$ (which denotes the maximum number of times a tweak can repeat during a full session of q_e encryption and q_v verification queries to Π') to the maximum possible values $q_v + \mu$, $q_v + 1$ and $\sigma_{m,v} + \sigma_{m,e}$ for $\Pi' = \mathsf{Umbreon}'$ ($\mathcal{A} = \mathcal{A}_{nm}$), Jolteon′ ($\mathcal{A} = \mathcal{A}_{nr}$) and Espeon′ ($\mathcal{A} = \mathcal{A}_{nm}$), respectively, we get the integrity bounds for Eevee modes and hence the results of Theorem 3.1. Here $\sigma_{m,v}$ denote the total number of primitive calls induced by all the message parts from $\mathcal{A}$'s q_v verification queries to Π'. ◻

8 CONCLUSION

We studied the IoT-to-Cloud computation problem and proposed Eevee, a family of three AEAD modes as a suitable solution. Our rigorous security analysis shows that in the nonce respecting setting the Eevee modes Umbreon, Jolteon and Espeon achieve full (n-bit) security with a single primitive (along with some other desirable features) while the existing possible solutions only achieve birthday-bound security and mostly require two primitives. In the nonce misuse setting, Umbreon and Espeon have graceful security degradation which previous online solutions do not offer.

Our practical evaluations show that Jolteon (with ForkSkinny) provides 1.85x to 3.64x speedup in IoT-encryption time and 3x to 4.5x speed-up in both MPC-decryption time and data for very short queries of 8 bytes and, 1.55x to 3.04x and 1.23x to 2.43x speedup, respectively, in MPC-decryption time and data for queries up to 500 bytes when compared against state-of-the-art MPC-friendly modes instantiated with SKINNY whereas Umbreon and Espeon show a favorable performance-security trade-off and provide stronger security guarantees. As an intermediate trade-off, Espeon can be a viable alternative in low latency networks targeting better security. We also compared our modes to the standard AEAD mode AES-GCM and its misuse-resistant counterpart AES-GCM-SIV and observed that despite the use of a generic forkcipher instantiation, the Eevee modes showed comparable performance and/or superior security properties to these modes which highlights the optimality of Eevee modes to the transciphering setting.

Our work can be viewed as transciphering for MPC. While transciphering has been studied for fully homomorphic encryption (FHE) [21, 44, 61], to the best of our knowledge, only the implementation of block cipher primitives [27, 36, 56, 70] and stream ciphers [23, 28, 59, 60] have been studied. The study of the FHE-related cost for modes of operations to turn those block cipher primitives into an encryption scheme is yet missing. Moreover, the commonly used counter mode construction suffers from birthday-bound security of $n/2$-bit, which was used by Canteaut et al. [23] as a motivation to study stream ciphers in this context. On the other hand, the Eevee family achieves full n-bit security. We leave the further study of the Eevee family and similar modes for FHE to future work.

ACKNOWLEDGMENTS

This work was supported by the Flemish Government through FWO SBO project MOZAIK S003321N and by CyberSecurity Research Flanders with reference number VR20192203. Amit Singh Bhati and Bart Preneel were 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] Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. 2016. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 191–219.

[2] Martin Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. 2015. Ciphers for MPC and FHE. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 430–454.

[3] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. 2020. Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols. *IACR Transactions on Symmetric Cryptology* 2020, Issue 3 (2020), 1–45. https://doi.org/10.13154/tosc.v2020.i3.1-45

[4] Elena Andreeva, Amit Singh Bhati, Bart Preneel, and Damian Vizár. 2021. 1, 2, 3, Fork: Counter Mode Variants based on a Generalized Forkcipher. *IACR Trans. Symmetric Cryptol.* 2021, 3 (2021), 1–35.

[5] Elena Andreeva, Amit Singh Bhati, and Damian Vizár. 2020. Nonce-Misuse Security of the SAEF Authenticated Encryption mode. In *Selected Areas in Cryptography*.

[6] Elena Andreeva, Amit Singh Bhati, and Damian Vizar. 2021. RUP Security of the SAEF Authenticated Encryption mode. Cryptology ePrint Archive, Paper 2021/103. https://eprint.iacr.org/2021/103

[7] Elena Andreeva, Arne Deprez, Jowan Pittevils, Arnab Roy, Amit Singh Bhati, and Damian Vizár. 2020. New Results and Insighs on ForkAE. In *NIST LWC workshop*.

[8] Elena Andreeva, Virginie Lallemand, Antoon Purnal, Reza Reyhanitabar, Arnab Roy, and Damian Vizár. 2019. ForkAE v. *Submission to NIST LwC Standardization Process* (2019).

[9] Elena Andreeva, Virginie Lallemand, Antoon Purnal, Reza Reyhanitabar, Arnab Roy, and Damian Vizár. 2019. Forkcipher: a New Primitive for Authenticated Encryption of Very Short Messages. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 153–182.

[10] Tomer Ashur, Orr Dunkelman, and Atul Luykx. 2017. Boosting authenticated encryption robustness with minimal modifications. In *Annual International Cryptology Conference*. Springer, 3–33.

[11] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. 2017. GIFT: a small present. In *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 321–345.

[12] Gregory V. Bard. 2006. A challenging but feasible blockwise-adaptive chosen-plaintext attack on SSL. *Cryptology ePrint Archive* (2006).

[13] Donald Beaver. 1991. Efficient Multiparty Protocols Using Circuit Randomization. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '91)*. Springer-Verlag, Berlin, Heidelberg, 420–432.

[14] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. 2016. The SKINNY family of block ciphers and its low-latency variant MANTIS. In *Annual International Cryptology Conference*. Springer, Springer-Verlag, 123–153.

[15] Mihir Bellare, Oded Goldreich, and Anton Mityagin. 2004. The Power of Verification Queries in Message Authentication and Authenticated Encryption. *IACR Cryptology ePrint Archive* 2004 (2004), 309.

[16] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. 2002. Authenticated encryption in SSH: provably fixing the SSH binary packet protocol. In *Proceedings of the 9th ACM conference on Computer and communications security*. 1–11.

[17] Aner Ben-Efraim, Michael Nielsen, and Eran Omri. 2019. Turbospeedz: Double Your Online SPDZ! Improving SPDZ Using Function Dependent Preprocessing. In *Applied Cryptography and Network Security*, Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung (Eds.). Springer International Publishing, Cham, 530–549. https://doi.org/10.1007/978-3-030-21568-2_26

[18] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. 2011. Semi-homomorphic Encryption and Multiparty Computation. In *Advances in Cryptology – EUROCRYPT 2011*, Kenneth G. Paterson (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 169–188. https://doi.org/10.1007/978-3-642-20465-4_11

[19] Tim Beyne, Yu Long Chen, Christoph Dobraunig, and Bart Mennink. 2020. Dumbo, Jumbo, and Delirium: Parallel Authenticated Encryption for the Lightweight Circus. *IACR Transactions on Symmetric Cryptology* 2020, S1 (Jun. 2020), 5–30. https://doi.org/10.13154/tosc.v2020.iS1.5-30

[20] Hanno Böck, Aaron Zauner, Sean Devlin, Juraj Somorovsky, and Philipp Jovanovic. 2016. Nonce-Disrespecting Adversaries: Practical Forgery Attacks on GCM in TLS. In *10th USENIX Workshop on Offensive Technologies, WOOT*, Natalie Silvanovich and Patrick Traynor (Eds.). USENIX Association.

[21] Zvika Brakerski and Vinod Vaikuntanathan. 2011. Efficient Fully Homomorphic Encryption from (Standard) LWE. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*. 97–106. https://doi.org/10.1109/FOCS.2011.12

[22] Luís T. A. N. Brandão and René Peralta. 2023. NIST IR 8214C ipd NIST First Call for Multi-Party Threshold Schemes (Initial Public Draft). (2023). https://doi.org/10.6028/NIST.IR.8214C.ipd

[23] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. 2018. Stream Ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression. *J. Cryptology* 31 (2018), 885–916. https://doi.org/10.1007/s00145-017-9273-9

[24] J Lawrence Carter and Mark N Wegman. 1977. Universal classes of hash functions. In *Proceedings of the ninth annual ACM symposium on Theory of computing*. 106–112.

[25] David Chaum, Claude Crépeau, and Ivan Damgård. 1988. Multiparty Unconditionally Secure Protocols. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing* (Chicago, Illinois, USA) *(STOC '88)*. Association for Computing Machinery, New York, NY, USA, 11–19. https://doi.org/10.1145/62212.62214

[26] Koji Chida, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, and Benny Pinkas. 2018. High-throughput secure AES computation. In *Proceedings of the 6th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. 13–24.

[27] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. 2014. Scale-Invariant Fully Homomorphic Encryption over the Integers. In *Public-Key Cryptography – PKC 2014*, Hugo Krawczyk (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 311–328. https://doi.org/10.1007/978-3-642-54631-0_18

[28] Orel Cosseron, Clément Hoffmann, Pierrick Méaux, and François-Xavier Standaert. 2022. Towards Globally Optimized Hybrid Homomorphic Encryption - Featuring the Elisabeth Stream Cipher. Cryptology ePrint Archive, Paper 2022/180. arXiv:https://eprint.iacr.org/2022/180

[29] Ivan Damgård, Marcel Keller, Enrique Larraia, Christian Miles, and Nigel P. Smart. 2012. Implementing AES via an actively/covertly secure dishonest-majority MPC protocol. In *International Conference on Security and Cryptography for Networks*. Springer, 241–263.

[30] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. 2013. Practical covertly secure MPC for dishonest majority–or: breaking the SPDZ limits. In *European Symposium on Research in Computer Security*. Springer, 1–18.

[31] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. 2012. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*. Springer, 643–662.

[32] Jun Du, Chunxiao Jiang, Erol Gelenbe, Lei Xu, Jianhua Li, and Yong Ren. 2018. Distributed Data Privacy Preservation in IoT Applications. *IEEE Wireless Communications* 25, 6 (2018), 68–76. https://doi.org/10.1109/MWC.2017.1800094

[33] Morris J Dworkin. 2007. SP 800-38D. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. *National Institute of Standards & Technology* (2007).

[34] Guillaume Endignoux and Damian Vizár. 2017. Linking online misuse-resistant authenticated encryption and blockwise attack models. *Cryptology ePrint Archive* (2017).

[35] Ewan Fleischmann, Christian Forler, and Stefan Lucks. 2012. McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes. In *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)*, Anne Canteaut (Ed.). Springer, 196–215. https://doi.org/10.1007/978-3-642-34047-5_12

[36] Craig Gentry, Shai Halevi, and Nigel P. Smart. 2012. Homomorphic Evaluation of the AES Circuit. In *Advances in Cryptology – CRYPTO 2012*, Reihaneh Safavi-Naini and Ran Canetti (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 850–867. https://doi.org/10.1007/978-3-642-32009-5_49

[37] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to Play ANY Mental Game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing* (New York, New York, USA) *(STOC '87)*. Association for Computing Machinery, New York, NY, USA, 218–229. https://doi.org/10.1145/28395.28420

[38] Lorenzo Grassi, Reinhard Lüftenegger, Christian Rechberger, Dragos Rotaru, and Markus Schofnegger. 2020. On a Generalization of Substitution-Permutation Networks: The HADES Design Strategy. In *39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12105)*. Springer. https://doi.org/10.1007/978-3-030-45724-2_23

[39] Shay Gueron, Adam Langley, and Yehuda Lindell. 2017. AES-GCM-SIV: specification and analysis. *Cryptology ePrint Archive* (2017).

[40] Shay Gueron and Yehuda Lindell. 2015. GCM-SIV: Full Nonce Misuse-Resistant Authenticated Encryption at Under One Cycle per Byte. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, Colorado, USA) *(CCS '15)*. Association for Computing Machinery, New York, NY, USA, 109–119. https://doi.org/10.1145/2810103.2813613

[41] Nilupulee A. Gunathilake, Ahmed Al-Dubai, and William J. Buchana. 2020. Recent Advances and Trends in Lightweight Cryptography for IoT Security. In *2020 16th International Conference on Network and Service Management (CNSM)*. 1–5. https://doi.org/10.23919/CNSM50824.2020.9269083

[42] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. 2017. Low Cost Constant Round MPC Combining BMR and Oblivious Transfer. In *ASIACRYPT (1)*. Springer, 598–628. https://doi.org/10.1007/978-3-319-70694-8_21

[43] Viet Tung Hoang, Reza Reyhanitabar, Phillip Rogaway, and Damian Vizar. 2015. Online Authenticated-Encryption and its Nonce-Reuse Misuse-Resistance. In *ADVANCES IN CRYPTOLOGY, PT I*, Vol. 9215. Gennaro, R, Springer Verlag, 493–517.

[44] Clément Hoffmann, Pierrick Méaux, and Thomas Ricosset. 2020. Transciphering, Using FiLIP and TFHE for an Efficient Delegation of Computation. In *Progress in Cryptology – INDOCRYPT 2020*, Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran (Eds.). Springer International Publishing, Cham, 39–61. https://doi.org/10.1007/978-3-030-65277-7_3

[45] Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, and Thomas Peyrin. 2020. Duel of the Titans: The Romulus and Remus Families of Lightweight AEAD Algorithms. *IACR Transactions on Symmetric Cryptology* 2020, Issue 1 (2020), 43–120. https://doi.org/10.13154/tosc.v2020.i1.43-120

[46] Tetsu Iwata and Yannick Seurin. 2017. Reconsidering the security bound of AES-GCM-SIV. *Cryptology ePrint Archive* (2017).

[47] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. 2014. Tweaks and keys for block ciphers: the TWEAKEY framework. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 274–288.

[48] Antoine Joux, Gwenaëlle Martinet, and Frédéric Valette. 2002. Blockwise-adaptive attackers revisiting the (in) security of some provably secure encryption modes: CBC, GEM, IACBC. In *Annual International Cryptology Conference*. Springer, 17–30.

[49] Marcel Keller. 2020. MP-SPDZ: A Versatile Framework for Multi-Party Computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. https://doi.org/10.1145/3372297.3417872

[50] Marcel Keller, Emmanuela Orsini, and Peter Scholl. 2016. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 830–842.

[51] Marcel Keller, Valerio Pastro, and Dragos Rotaru. 2018. Overdrive: Making SPDZ great again. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 158–189.

[52] Mustafa Khairallah and Shivam Bhasin. 2022. Hardware implementations of romulus: Exploring nonce misuse resistance and boolean masking. In *NIST Lightweight Cryptography Workshop*.

[53] Mustafa Khairallah, Shivam Bhasin, and Anupam Chattopadhyay. 2019. On Misuse of Nonce-Misuse Resistance : Adapting Differential Fault Attacks on (few) CAESAR Winners. In *2019 IEEE 8th International Workshop on Advances in Sensors and Interfaces (IWASI)*. 189–193. https://doi.org/10.1109/IWASI.2019.8791393

[54] Tiffany Hyun-Jin Kim and Joshua Lampkins. 2019. SSP: Self-Sovereign Privacy for Internet of Things Using Blockchain and MPC. In *2019 IEEE International Conference on Blockchain (Blockchain)*. 411–418. https://doi.org/10.1109/Blockchain.2019.00063

[55] Ted Krovetz and Phillip Rogaway. 2001. Fast universal hashing with small keys and no preprocessing: The PolyR construction. In *Information Security and Cryptology—ICISC 2000: Third International Conference Seoul, Korea, December 8–9, 2000 Proceedings 3*. Springer, 73–89.

[56] Tancrède Lepoint and Michael Naehrig. 2014. A Comparison of the Homomorphic Encryption Schemes FV and YASHE. In *Progress in Cryptology – AFRICACRYPT 2014*, David Pointcheval and Damien Vergnaud (Eds.). Springer International Publishing, Cham, 318–335. https://doi.org/10.1007/978-3-319-06734-6_20

[57] Moses Liskov, Ronald L Rivest, and David Wagner. 2002. Tweakable block ciphers. In *Advances in Cryptology—CRYPTO 2002: 22nd Annual International Cryptology Conference Santa Barbara, California, USA, August 18–22, 2002 Proceedings 22*. Springer, 31–46.

[58] Atul Luykx, Bart Preneel, Elmar Tischhauser, and Kan Yasuda. 2016. A MAC Mode for Lightweight Block Ciphers. In *Fast Software Encryption*, Thomas Peyrin (Ed.). Springer Berlin Heidelberg, 43–59.

[59] Pierrick Méaux, Claude Carlet, Anthony Journault, and François-Xavier Standaert. 2019. Improved Filter Permutators for Efficient FHE: Better Instances and Implementations. In *Progress in Cryptology – INDOCRYPT 2019*, Feng Hao, Sushmita Ruj, and Sourav Sen Gupta (Eds.). Springer International Publishing, Cham, 68–91. https://doi.org/10.1007/978-3-030-35423-7_4

[60] Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. 2016. Towards Stream Ciphers for Efficient FHE with Low-Noise Ciphertexts. In *Advances in Cryptology – EUROCRYPT 2016*, Marc Fischlin and Jean-Sébastien Coron (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 311–343. https://doi.org/10.1007/978-3-662-49890-3_13

[61] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. 2011. Can Homomorphic Encryption Be Practical?. In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop* (Chicago, Illinois, USA) *(CCSW '11)*. Association for Computing Machinery, New York, NY, USA, 113–124. https://doi.org/10.1145/2046660.2046682

[62] Yusuke Naito, Yu Sasaki, and Takeshi Sugawara. 2020. Lightweight authenticated encryption mode suitable for threshold implementation. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part II 30*. Springer, 705–735.

[63] Yusuke Naito, Yu Sasaki, and Takeshi Sugawara. 2022. Secret Can Be Public: Low-Memory AEAD Mode for High-Order Masking. In *Advances in Cryptology–CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part III*. Springer, 315–345.

[64] NIST. 2018. DRAFT Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process. https://csrc.nist.gov/Projects/Lightweight-Cryptography.

[65] Thomas Peyrin. 2018. Lightweight Symmetric-Key Cryptography. *Invited talk, CTCRYPT* (2018). https://thomaspeyrin.github.io/web/assets/docs/invited/CTCRYPT2018_slides.pdf.

[66] Antoon Purnal, Elena Andreeva, Arnab Roy, and Damian Vizár. 2019. What the Fork: Implementation Aspects of a Forkcipher. In *NIST Lightweight Cryptography Workshop 2019*.

[67] Phillip Rogaway. 2002. Authenticated-Encryption with Associated-Data. In *Proceedings of the 9th ACM conference on Computer and communications security*. 98–107.

[68] Dragos Rotaru, Nigel P. Smart, and Martijn Stam. 2017. Modes of Operation Suitable for Computing on Encrypted Data. *IACR Transactions on Symmetric Cryptology* 2017, 3 (Sep. 2017), 294–324. https://doi.org/10.13154/tosc.v2017.i3.294-324

[69] Peter Schwabe and Ko Stoffelen. 2017. All the AES you need on Cortex-M3 and M4. In *Selected Areas in Cryptography–SAC 2016: 23rd International Conference, St. John's, NL, Canada, August 10-12, 2016, Revised Selected Papers*. Springer, 180–194.

[70] Nigel P. Smart and Frederik Vercauteren. 2014. Fully homomorphic SIMD operations. *Designs, codes and cryptography* 71, 1 (2014), 57–81. https://doi.org/10.1007/s10623-012-9720-4

[71] Samet Tonyali, Kemal Akkaya, Nico Saputro, A. Selcuk Uluagac, and Mehrdad Nojoumian. 2018. Privacy-preserving protocols for secure and reliable data aggregation in IoT-enabled Smart Metering systems. *Future Generation Computer Systems* 78 (2018), 547–557. https://doi.org/10.1016/j.future.2017.04.031

[72] Mathy Vanhoef and Frank Piessens. 2017. Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2. In *Proceedings of the 2017 ACM SIGSAC, CCS 2017*, Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 1313–1328.

A ADDITIONAL FIGURES

A.1 Software

Figure 7 details the encryption performance in the transciphering scenario which happens on a microcontroller in software. In addition to the modes discussed in the main body, we implement the MPC-friendly versions of the same modes CTR-HtMAC-MiMC-128 and CTR-pPMAC-MiMC-128 [68] with the underlying cipher MiMC-128 operating on a 128-bit prime field.

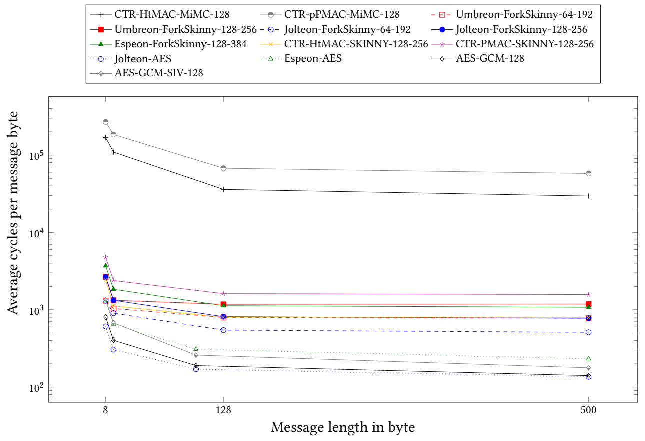


Figure 7: Software performance of Umbreon, Jolteon, Espeon, **CTR-HtMAC, CTR-(p)PMAC, AES-GCM and AES-GCM-SIV. The dashed plot (- -) denotes the same mode (denoted by a straight line —) but with a smaller primitive.**

A.2 LAN Setting

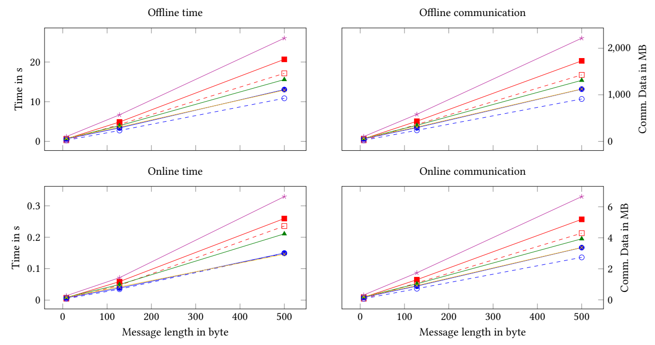
In Fig. 8a we show the execution time and the amount of exchanged data during the offline and online phase for the distributed decryption of messages of length 8, 128 and 500 bytes. In this scenario, two parties are connected in a fast network (< 1 ms latency).

A.3 WAN Setting

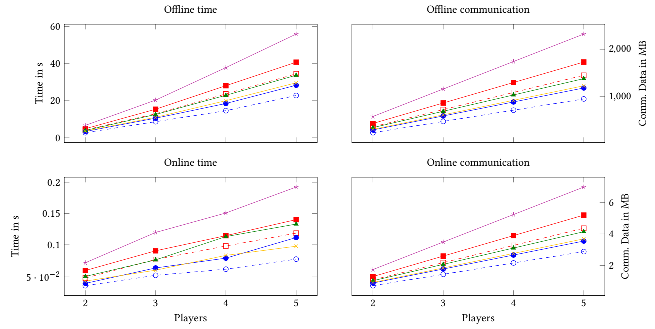
Next, we examine the performance in a network with latency. Figure 8c visualizes offline and online phase execution time for a low latency and higher latency network for a distributed decryption of a 128-byte message. Since the amount of exchanged data does not change, we do not report those values. In this setting, we want to recall the importance of fully parallel decryption in an AEAD mode (see Sect. 2). The performance of AEAD modes with fully parallel decryption, like all Eevee family modes and CTR-HtMAC, degrades at a lower rate than for modes that have multiple primitive calls in series. In our benchmark CTR-PMAC which has two calls in series has a higher performance degradation when compared with the rest. The degradation is aggravated the more primitive calls a mode performs in series.

A.4 More Players

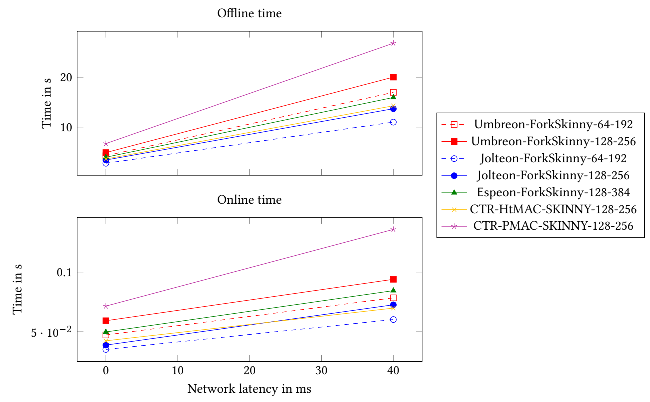
Finally, Fig. 8b shows the decryption performance of a 128-byte message among 2 to 5 players in a low latency network. We observe that the relation between the performance of the studied



(a) Performance for decryption of 8, 128 and 500 byte long messages among two players with at most 1 ms latency and at least 950 Mbit/s connection.



(b) Performance of the decryption of 128 byte long messages with a connection with at most 1 ms latency and at least 950 Mbit/s of bandwidth for a varying number of players.



(c) Performance of the decryption of 128 byte long messages among two players with a connection of at least 950 Mbit/s for varying network latency.

Figure 8: Performance of Umbreon, Jolteon, Espeon, **CTR-HtMAC and CTR-PMAC in MASCOT. The dashed plot (- -) denotes the same mode (denoted by a straight line —) but with a smaller primitive. We give wall-clock time and the amount of data that is exchanged separately for the offline and online phase of the protocol.**

modes does not change much when the number of participating parties is increased. All Eevee family members still outperform CTR-PMAC while both instantiations of Jolteon still show superior (with same security) or comparable (with double security) performance to CTR-HtMAC. Note that due to the increased communication cost when adding new parties in MASCOT, the performance difference for Jolteon-ForkSkinny-64-192 compared to CTR-HtMAC-SKINNY-128-256 improves further and becomes more evident. The fewer multiplications and thus fewer partial openings during the online phase clearly show in those scenarios. For similar reasons, the online time performance of Espeon-ForkSkinny-128-384 degrades with a growing number of parties due to the larger primitive.

B EEVEE AEAD MODES: PSEUDOCODE

```
function Pad($n, X$)
    pad $\leftarrow 1$; res $\leftarrow |X|\%n$
    if res $\neq 0$ then
        pad $\leftarrow 0$
        $X \leftarrow X\|10^{n-1-\mathrm{res}}$
    else
        res $\leftarrow n$
    end if
    return $X$, pad, res
end function

△ function $\mathsf{f}_{\mathsf{T}}(t, A, B)$
    △ return $(B\|A)[0 \dots t-3]$
△ end function

function AD($K, N, A, T_A$)
    $A, \mathrm{pad}_A, \mathrm{res}_A \leftarrow \mathrm{Pad}(n, A)$
    $A_1, \dots, A_a, A'_* \xleftarrow{n} A$
//$A'_* = A_*\|10^*$
    for $i \leftarrow 1$ to $a$ do
        $\mathsf{T} \leftarrow N\|\langle i+4\rangle_d\|00$
//$d = t - |N| - 2$
        $T_A \leftarrow \mathsf{F}_K^{\mathsf{T},0}(A_i) \oplus T_A$
    end for
    $\mathsf{T} \leftarrow N\|\langle 2 \cdot \mathrm{pad}_A + \mathrm{noM}\rangle_d\|00$
    $T_A \leftarrow \mathsf{F}_K^{\mathsf{T},0}(A'_*) \oplus T_A$
    return $T_A$
end function

function Enc($K, N, A, M$)
    if $|M| = |A| = 0$ then
        return $\perp$
    end if
    noM $\leftarrow 1$
    if $|M| \neq 0$ then
        noM $\leftarrow 0$
        $M, \mathrm{pad}_M, \mathrm{res}_M \leftarrow \mathrm{Pad}(n, M)$
        $M_1, \dots, M_m, M'_* \xleftarrow{n} M$
//$M'_* = M_*\|10^*$
    end if
    $T_A, \Delta \leftarrow 0^n$
    if $|A| \neq 0$ then
        $T_A \leftarrow \mathrm{AD}(K, N, A, T_A)$
    end if
    if $|M| = 0$ then
        return $T_A$
    end if
⊖ $C_0 \leftarrow T_A$
△ if $|N| > n-2$ then
        △ return $\perp$
△ $C_0 \leftarrow N\|0^{n-|N|}$
    for $i \leftarrow 1$ to $m$ do
        □ $\mathsf{T} \leftarrow N\|\langle i+1\rangle_{d+1}\|1$
        □ $\Delta \leftarrow \Delta \oplus T_A$
        □ $(C_i, T_A) \leftarrow \mathsf{F}_K^{\mathsf{T},b}(M_i \oplus T_A)$
        ⊖ $\mathsf{T} \leftarrow N\|\langle i+1\rangle_{d+1}\|1$
        ⊖ $\Delta \leftarrow \Delta \oplus M_i \oplus C_{i-1}$
        ⊖ $C_i \leftarrow \mathsf{F}_K^{\mathsf{T},0}(M_i)$
        △ $\Delta \leftarrow \Delta \oplus M_i$
        △ if $i = 1$ then
            △ $\mathsf{T} \leftarrow N\|\langle 4\rangle_d\|00$
            △ $C_i \leftarrow \mathsf{F}_K^{\mathsf{T},0}(T_A \oplus M_i)$
        △ else
            △ $\mathsf{T} \leftarrow \mathsf{f}_{\mathsf{T}}(t, C_{i-2}, C_{i-1})\|01$
            △ $C_i \leftarrow \mathsf{F}_K^{\mathsf{T},0}(M_i)$
        △ end if
    end for
□ $\mathsf{T} \leftarrow N\|\langle \mathrm{pad}_M\rangle_{d+1}\|1$
⊖ $\mathsf{T} \leftarrow N\|\langle \mathrm{pad}_M\rangle_{d+1}\|1$
△ if $m = 0$ then
        △ $\mathsf{T} \leftarrow N\|\langle 4\rangle_d\|1\|\mathrm{pad}_M$
△ else
        △ $\mathsf{T} \leftarrow \mathsf{f}_{\mathsf{T}}(t, C_{m-1}, C_m)\|1\|\mathrm{pad}_M$
□ $\Delta \leftarrow \Delta \oplus T_A$
⊖ $\Delta \leftarrow \Delta \oplus C_m$
    $(Tag, C'_*) \leftarrow \mathsf{F}_K^{\mathsf{T},b}(M'_* \oplus \Delta)$
    $C_* \leftarrow (C'_* \oplus M'_*)[0 \dots \mathrm{res}_M - 1]$
    return $C_1, \dots, C_m, C_*, Tag$
end function

function Dec($K, N, A, C, Tag$)
    if $|C| = |A| = 0$ then
        return $\perp$
    end if
    noM $\leftarrow 1$
    if $|C| \neq 0$ then
        noM $\leftarrow 0$
        $C, \mathrm{pad}_C, \mathrm{res}_C \leftarrow \mathrm{Pad}(n, C)$
        $C_1, \dots, C_m, C'_* \xleftarrow{n} C$
//$C'_* = C_*\|10^*$
    end if
    $T_A, \Delta \leftarrow 0^n$
    if $|A| \neq 0$ then
        $T_A \leftarrow \mathrm{AD}(K, N, A, T_A)$
    end if
    if $|C| = 0$ then
        if $Tag = T_A$ then
            return $\top$
        else
            return $\perp$
        end if
    end if
⊖ $C_0 \leftarrow T_A$
△ if $|N| > n-2$ then
        △ return $\perp$
△ $C_0 \leftarrow N\|0^{n-|N|}$
    for $i \leftarrow 1$ to $m$ do
        □ $\mathsf{T} \leftarrow N\|\langle i+1\rangle_{d+1}\|1$
        □ $\Delta \leftarrow \Delta \oplus T_A$
        □ $(M_i, T_A) \leftarrow \mathsf{F}^{-1}{}_K^{\mathsf{T},0,b}(C_i) \oplus (T_A, 0^n)$
        ⊖ $\mathsf{T} \leftarrow N\|\langle i+1\rangle_{d+1}\|1$
        ⊖ $M_i \leftarrow \mathsf{F}^{-1}{}_K^{\mathsf{T},0,i}(C_i)$
        ⊖ $\Delta \leftarrow \Delta \oplus M_i \oplus C_{i-1}$
        △ if $i = 1$ then
            △ $\mathsf{T} \leftarrow N\|\langle 4\rangle_d\|00$
            △ $M_i \leftarrow \mathsf{F}^{-1}{}_K^{\mathsf{T},0,i}(C_i) \oplus T_A$
        △ else
            △ $\mathsf{T} \leftarrow \mathsf{f}_{\mathsf{T}}(t, C_{i-2}, C_{i-1})\|01$
            △ $M_i \leftarrow \mathsf{F}^{-1}{}_K^{\mathsf{T},0,i}(C_i)$
        △ end if
        △ $\Delta \leftarrow \Delta \oplus M_i$
    end for
□ $\mathsf{T} \leftarrow N\|\langle \mathrm{pad}_C\rangle_{d+1}\|1$
⊖ $\mathsf{T} \leftarrow N\|\langle \mathrm{pad}_C\rangle_{d+1}\|1$
△ if $m = 0$ then
        △ $\mathsf{T} \leftarrow N\|\langle 4\rangle_d\|1\|\mathrm{pad}_C$
△ else
        △ $\mathsf{T} \leftarrow \mathsf{f}_{\mathsf{T}}(t, C_{m-1}, C_m)\|1\|\mathrm{pad}_C$
□ $\Delta \leftarrow \Delta \oplus T_A$
⊖ $\Delta \leftarrow \Delta \oplus C_m$
    $(M'_*, C'_*) \leftarrow \mathsf{F}^{-1}{}_K^{\mathsf{T},0,b}(Tag) \oplus (\Delta, 0^n)$
    if $(M'_* \oplus C'_*)[0 \dots \mathrm{res}_C - 1] = C_*$ and $M'_*[\mathrm{res}_C \dots n-1] = 10^{n-1-\mathrm{res}_C}$ then
        $M_* \leftarrow M'_*[0 \dots \mathrm{res}_C - 1]$
        return $M_1, \dots, M_m, M_*$
    else
        return $\perp$
    end if
        end function
```

Figure 9: The Eevee family of AEAD modes. Red (denoted by □), blue (denoted by ⊖) and green (denoted by △) lines represent pseudocode subparts specific to Umbreon, Jolteon and Espeon, respectively, whereas black lines are common among all the modes.

C BOUNDING $\Pr[\mathcal{U}]$ TERMS

Block Notation. In the following analysis, we use (unless defined) superscript and subscripts to denote the query number and the primitive call number for that variable/parameter. To exemplify, M^i_j represents j^{th} message block input in i^{th} encryption query.

A) Bounding $\Pr\left[\mathcal{U}^{\Pi',\mathcal{A}}_{i,3.2.1}\right]$**.** Note that $\Pr\left[\mathcal{U}^{\Pi',\mathcal{A}}_{i,3.2.1}\right]$ is equal to the product of two probabilities - 1. probability of i^{th} encryption query of $\mathcal{A}$ (to Π') satisfying Case 3.2.1 and 2. probability of oprpf-adversary $\mathcal{A}$ distinguishing the i^{th} encryption query output from being real or ideal, therefore, we have

$$\begin{aligned}
\Pr\left[\mathcal{U}^{\Pi',\mathcal{A}}_{i,3.2.1}\right] &\leq \Pr[i^{th} \text{ encryption query of } \mathcal{A} \text{ to } \Pi' \\
&\quad \text{satisfies Case 3.2.1}] \\
&= \Pr[\text{For given } \Pi', \mathcal{A} \text{ and } 1 \leq i \leq q_e \text{ with } |A^i| \neq 0 \neq \\
&\quad |C^i|, \exists\, 1 \leq i' < i \text{ s.t. } (\mathsf{X}^i_1, \mathsf{T}^i_1) = (\mathsf{X}^{i'}_1, \mathsf{T}^{i'}_1) \text{ and} \\
&\quad \forall\, 1 \leq r < i, A^i \neq A^r]\,.
\end{aligned}$$

Now, let us consider that the adversary $\mathcal{A}$ repeats the nonce $N \in \mathcal{N}$ for μ_N many times over q_e encryption queries to Π'. We note that with Π', T^i_A is generated as xor of random tweakable permutations' outputs with any tweak repeating over first i queries at most $\max_{N\in\mathcal{N}}\{\mu_N\}$ many times. Hence for $\mu = \max_{N\in\mathcal{N}}\{\mu_N\}$, we know that for any $i' < i$, the probability of $(\mathsf{X}^i_1, \mathsf{T}^i_1) = (\mathsf{X}^{i'}_1, \mathsf{T}^{i'}_1)$, i.e., $(T^i_A, N^i) = (T^{i'}_A \oplus M^{i'}_1 \oplus M^i_1, N^{i'})$ is at most $1/(2^n - (\mu - 1))$ and we have

$$\Pr\left[\mathcal{U}^{\Pi',\mathcal{A}}_{i,3.2.1}\right] < \frac{\mu - 1}{2^n - \mu}\,. \tag{6}$$

B) Bounding $\Pr\left[\mathcal{U}^{\Pi',\mathcal{A}}_{i,2}\right]$**.** Similarly defined as $\mathcal{U}^{\Pi',\mathcal{A}}_{i,3.2.1}$ above (as product of two probabilities and thus bounded by the second probability), we get

$$\begin{aligned}
\Pr\left[\mathcal{U}^{\Pi',\mathcal{A}}_{i,2}\right] &\leq \Pr[\text{For given } \Pi', \mathcal{A} \text{ distinguishes its } i^{th} \\
&\qquad \text{query output } T^i_A \text{ (queried with only AD)} \\
&\qquad\qquad \text{from } n\text{-bit uniform random string}]\,.
\end{aligned}$$

Note that $\mathcal{A}$ here is trivially assumed to make non-duplicate queries and thus we can say that

$$\begin{aligned}
\Pr\left[\mathcal{U}^{\Pi',\mathcal{A}}_{i,2}\right] &\leq \frac{1}{2}\sum_{y\in\{0,1\}^n}\Big|\Pr\left[T^i_A = y \mid T^i_A \leftarrow \Pi'.Enc(\right. \\
&\quad \left.N^i, A^i, M^i = \varepsilon)\right] - \Pr\left[T^i_A = y \mid T^i_A \leftarrow\!\$\ \{0,1\}^n\right]\Big| \\
&\leq \frac{1}{2}\sum_{y\in\{0,1\}^n}\Big|\Pr\left[T^i_A = y \mid T^i_A \leftarrow \Pi'.Enc(N^i, A^i \in\right. \\
&\quad \left.\{0,1\}^n, M^i = \varepsilon)\right] - \Pr\left[T^i_A = y \mid T^i_A \leftarrow f_{N^i}(A^i \in\right. \\
&\quad \left.\{0,1\}^n), f_{N^i} \leftarrow\!\$\ \mathrm{Func}(n,n)\right]\Big| \\
&= \frac{1}{2}\sum_{y\in\{0,1\}^n}\Big|\Pr\left[\pi_{N^i}(X^i) = y \mid \pi_{N^i} \leftarrow\!\$\ \mathrm{Perm}(n)\right] \\
&\quad - \Pr\left[f_{N^i}(X^i) = y \mid f_{N^i} \leftarrow\!\$\ \mathrm{Func}(n,n)\right]\Big| \\
&= \mathrm{SD}(\pi_{N^i}(X^i), f_{N^i}(X^i))
\end{aligned}$$

with $\pi_{N^i} \leftarrow\$ \text{Perm}(n)$ and $f_{N^i} \leftarrow\$ \text{Func}(n,n)$ where X^i represents the i^{th} n-bit block in the set of arbitrary and ordered n-bit block queries to π (resp. f) $\mathbb{X}^i = \{X^1, X^2, \ldots, X^i\}$ such that $X^a \neq X^b \ \forall\ a \neq b$. In other words, $\mathbb{X}^i$ contains no duplicate queries. Here the first inequality holds because this desired $\Pr[\mathcal{U}]$ is upper bounded by the statistical distance between the uniform distribution and the distribution of T_A^i. The second inequality holds because $\Pi'.Enc(N^i, A^i, M^i = \varepsilon)$ is defined as a sum of independent random permutations for $\Pi \in \{\text{Umbreon}, \text{Jolteon}, \text{Espeon}\}$ (independence is ensured by tweak domain separation) and due to this independence, the probability of this sum of random permutations' outputs being equal to a given value cannot be higher than the probability of any of the individual random permutation's output being equal to the same. The first equality holds since querying single block AD A^i with nonce N^i to Π' is by definition equivalent to querying a random tweakable permutation π_{N^i} with block $X^i = A^i$.

Let us denote by $\mathbb{Y}^i$ the set $\{(N^1, \pi_{N^1}(X^1)), (N^2, \pi_{N^2}(X^2)), \ldots, (N^i, \pi_{N^i}(X^i))\}$ and let $\overline{\mathbb{Y}^i} \subseteq \mathbb{Y}^i$ be defined as $\{Y \in \{0,1\}^n \mid (N, Y) \in \mathbb{Y}^i \text{ and } N = N^i\}$. In simple words, $\overline{\mathbb{Y}^i}$ represents the set of all previous query outputs for π that share the same nonce N^i in tweak as targeted (i^{th}) query. Clearly, $|\overline{\mathbb{Y}^i}| \leq \mu - 1$. With this notation, we have for some $\pi_{N^i} \leftarrow\$ \text{Perm}(n)$ and $f_{N^i} \leftarrow\$ \text{Func}(n,n)$,

$$\begin{aligned}
\Pr\left[\mathcal{U}_{i,2}^{\Pi',\mathcal{A}}\right] &\leq \text{SD}(\pi_{N^i}(X^i), f_{N^i}(X^i)) \\
&= \frac{1}{2} \sum_{y \in \{0,1\}^n \setminus \overline{\mathbb{Y}^i}} \Big| \Pr\left[\pi_{N^i}(X^i) = y \mid \pi_{N^i} \leftarrow\$ \text{Perm}(n)\right] \\
&\qquad - \Pr\left[f_{N^i}(X^i) = y \mid f_{N^i} \leftarrow\$ \text{Func}(n,n)\right] \Big| \\
&\qquad + \frac{1}{2} \sum_{y \in \overline{\mathbb{Y}^i}} \Big| \Pr\left[\pi_{N^i}(X^i) = y \mid \pi_{N^i} \leftarrow\$ \text{Perm}(n)\right] \\
&\qquad - \Pr\left[f_{N^i}(X^i) = y \mid f_{N^i} \leftarrow\$ \text{Func}(n,n)\right] \Big| \\
&= \frac{1}{2} \sum_{y \in \{0,1\}^n \setminus \overline{\mathbb{Y}^i}} \left| \frac{1}{2^n - |\overline{\mathbb{Y}^i}|} - \frac{1}{2^n} \right| + \frac{1}{2} \sum_{y \in \overline{\mathbb{Y}^i}} \left| 0 - \frac{1}{2^n} \right| \\
&= \frac{|\overline{\mathbb{Y}^i}|}{2^n} \leq \frac{(\mu - 1)}{2^n}\,.
\end{aligned} \tag{7}$$

The second last equality here is true as for any value of i, the random permutation π_{N^i} (when fed with a distinct input) returns an arbitrary string $y \in \{0,1\}^n$ as output with probability 0 if $y \in \overline{\mathbb{Y}^i}$ and with random probability of $1/(2^n - |\overline{\mathbb{Y}^i}|)$, otherwise. On the other hand, the probability of a random function returning an arbitrary string $y \in \{0,1\}^n$ as output is $1/2^n$ as all outputs there are uniformly distributed for distinct queries.

C) Bounding $\Pr\left[\mathcal{U}_{i,1}^{\Pi',\mathcal{A}}\right]$**.** Note that under this event, $\mathcal{A}$ makes its i^{th} query with no AD (hence T_A is set to 0).

C.1) When $\Pi' = \text{Jolteon}'$ and $\mathcal{A} = \mathcal{A}_{nr}$ is nonce-respecting: Since $\mathcal{A}_{nr}$ is nonce-respecting, every encryption query is determined to have unique nonce. Further, since Jolteon uses the nonce concatenated with a block counter as the tweak, we have different tweak for each block call throughout the queries and hence all the output blocks (as they are outputs of the tweakable random permutation π_0) for all queries (including the i^{th}) are independent, random and uniformly distributed and so are indistinguishable from uniform random strings. Or

$$\Pr\left[\mathcal{U}_{i,1}^{\text{Jolteon}',\mathcal{A}_{nr}}\right] = 0\,. \tag{8}$$

C.2) When $\Pi' \in \{\text{Umbreon}', \text{Espeon}'\}$ and $\mathcal{A} = \mathcal{A}_{nm}$ is nonce-misusing: We recall the notation that $\mathcal{A}_{nm}$ can use a nonce $N \in \mathcal{N}$ over q_e queries for $\mu_N \leq \mu$ many times, hence the i^{th} query nonce N_i can be used before the i^{th} query for at most $\mu - 1$ many times.

Observation 1. Since the i^{th} query of $\mathcal{A}_{nm}$ has $|M^i| \neq 0$, any of its first $i-1$ queries, let say i'^{th} with $1 \leq i' < i$, that contains no message part does not help it in distinguishing the i^{th} query output from being real or ideal. This is true thanks to the noM parameter which ensures distinct tweaks in the final π calls processing AD in the queries i and i' and this by definition implies that the output distribution of any i'^{th} query (i.e., $T_A^{i'}$s) is independent to the output distribution of i^{th} query.

Observation 2. Further, we recall that the freedom of choosing non-empty AD for encryption queries to Π' can only help $\mathcal{A}_{nm}$ if the first primitive call that corresponds to the first ciphertext block of i^{th} encryption query contain a non-prefixed input-tweak pair $(\mathsf{X}_1, \mathsf{T}_1)$ that collides with one of the old/previously queried $(\mathsf{X}_1, \mathsf{T}_1)$ pairs.

Observation 3. Let us consider for a moment that $\mathcal{A}_{nm}$ makes its first i queries with no AD. Now, as per the oprpf security definition, we have that to successfully distinguish the i^{th} output of Π' from a corresponding ideal ORP+RF output, the oprpf-adversary $\mathcal{A}_{nm}$ has to distinguish the non-prefixed part of this output from the corresponding non-prefixed part of the ideal ORP+RF output. More cocnretely, if p_i represents the number of prefixed output blocks C_j^is (i.e., $C_1^i, \ldots, C_{p_i}^i$) in the i^{th} query of $\mathcal{A}_{nm}$ to Π' then in order to successfully win the distinguishing game, $\mathcal{A}_{nm}$ must be able to distinguish either the $(p_i+1)^{th}$ block output $C_{p_i+1}^i$ of Π' from an output of a random permutation $\pi_{N^i, M_1^i, \ldots, M_{p_i}^i} \leftarrow\$ \text{Perm}(n)$ or the rest output blocks corresponding to the i^{th} query to Π' from equal length uniform random strings.

Let us now consider a new oprpf-adversary $\mathcal{A}^\star$ against Π' who makes similar queries as $\mathcal{A}_{nm}$ but ensures that all of its first i queries contain non-zero message lengths and no AD. Hence, for an event $\mathcal{V}_i^{\Pi',\mathcal{A}_{nm}}$ defined as $\mathcal{V}_i^{\Pi',\mathcal{A}_{nm}} = \{\text{For given } \Pi', \mathcal{A}_{nm} \text{ and } 1 \leq i \leq q_e,\ \exists\ 1 \leq i' < i \text{ such that } (\mathsf{X}_1^{i'}, \mathsf{T}_1^{i'}) = (\mathsf{X}_1^i, \mathsf{T}_1^i) \text{ and } A^{i'} \neq A^i = \varepsilon\}$ and $\Pi' \in \{\text{Umbreon}', \text{Espeon}'\}$ we can say that

$$\begin{aligned}
\Pr\left[\mathcal{U}_{i,1}^{\Pi',\mathcal{A}_{nm}}\right] &\leq \Pr\left[\mathcal{U}_{i,1}^{\Pi',\mathcal{A}_{nm}} \mid \neg\mathcal{V}_i^{\Pi',\mathcal{A}_{nm}}\right] + \Pr\left[\mathcal{V}_i^{\Pi',\mathcal{A}_{nm}}\right] \\
&\leq \Pr\left[\mathcal{U}_{i,1}^{\Pi',\mathcal{A}^\star}\right] + \Pr\left[\mathcal{V}_i^{\Pi',\mathcal{A}_{nm}}\right] \\
&\leq \Pr\left[\mathcal{U}_{i,1}^{\Pi',\mathcal{A}^\star}\right] + \frac{\mu - 1}{2^n - \mu}
\end{aligned} \tag{9}$$

where the last inequality holds with an analogous explanation as Exp. 6 but this time considering probability distribution of $T_A^{i'}$ instead of T_A^i.

C.2.1) When $\Pi' = \text{Umbreon}'$: At this step, we define and denote a mode Π' by Π'' that has (π_0, π_1); the two families of random permutations in the i^{th} query further replaced for all except the first p_i (π_0, π_1) calls and the $(p_i+1)^{th}$ π_0 call of i^{th} query by families of random functions f_0 and f_1 with the same signature, i.e., $f_b = f_{\mathsf{T},b} \leftarrow\$ \text{Func}(n))_{\mathsf{T}\in\{0,1\}^t}$ where $p_i \geq 0$ represents the number of prefixed output blocks C^i_js (i.e., $C^i_1, \ldots, C^i_{p_i}$) in the i^{th} query of $\mathcal{A}^*$ to Π'. We note that for $\Pi' = \mathsf{Umbreon}'$, $\Pr\left[\mathcal{U}^{\Pi'',\mathcal{A}^\star}_{i,1}\right] = 0$ as Π'' behaves no different than the corresponding ideal ORP+RF oracle for $\mathcal{A}^\star$. Further, we note that in $\mathsf{Umbreon}'$ each query contains different tweaks for its each (π_0, π_1) call (thanks to the block counter) which means all these calls in a query are independent from each other and hence using the standard statistical distance bounding with slight abuse of notations denoting $\pi_{N^i\|\langle j\rangle_{d+1}\|1,b}$ and $f_{N^i\|\langle j\rangle_{d+1}\|1,b}$ by $\pi^{j,b}_{N^i}$ and $f^{j,b}_{N^i}$, respectively, we get for $\Pi' = \mathsf{Umbreon}'$ that

$$\begin{aligned}
\Pr\left[\mathcal{U}^{\Pi',\mathcal{A}^\star}_{i,1}\right] &\leq \Pr\left[\mathcal{U}^{\Pi'',\mathcal{A}^\star}_{i,1}\right] + \sum_{h=p_i+1}^{\ell_i} \Pr\Big[\exists\ 1 \leq i' < i \\
&\qquad \text{s.t. } X^i_h = X^{i'}_h \text{ and } N^i = N^{i'}\Big] + \\
&\qquad \sum_{b\in\{0,1\}} \Big\{ \sum_{h=p_i+1}^{\ell_i-1} \mathsf{SD}\big(\pi^{h+1,b}_{N^i}(X^i_h), f^{h+1,b}_{N^i}(X^i_h)\big) + \\
&\qquad \max_{j\in\{0,1\}} \big\{\mathsf{SD}\big(\pi^{j,b}_{N^i}(X^i_{\ell_i}), f^{j,b}_{N^i}(X^i_{\ell_i})\big)\big\}\Big\} \\
&\leq \frac{\ell_i(\mu-1)}{2^n} + 2\cdot\sum_{h=1}^{\ell_i} \mathsf{SD}(\pi_{N^i}(X^i), f_{N^i}(X^i)) \\
&\qquad \text{for some } \pi_{N^i} \leftarrow\$ \text{ Perm}(n) \text{ and } f_{N^i} \leftarrow\$ \text{ Func}(n,n) \\
&\leq \frac{3\ell_i(\mu-1)}{2^n} \qquad (10)
\end{aligned}$$

where X^i_h represents the value which is fed to the h^{th} primitive call (i.e., $f_{N^i\|\langle j\rangle_{d+1}\|1,b}$ where for $1 \leq h < \ell_i$, $j = h+1$ and for $h = \ell_i$, $j = 0$ or 1 as per the message padding) as input block in order to process the i^{th} queried message M^i of $\mathcal{A}^\star$ with Π''. Here the term ℓ_i represents the total number of primitive calls that are required to process M^i. In other words, $\ell_i = \lceil |M^i|/n \rceil \geq 0$. The second last inequality holds because the statistical distance between the distributions of $\pi_{N^i\|\langle j\rangle_{d+1}\|1,b}(X^i)$, i.e., random permutation outputs and $f_{N^i\|\langle j\rangle_{d+1}\|1,b}(X^i)$, i.e., random function outputs for arbitrary distinct inputs X^is does not depend on the resampling/reindexing of either of these functions. In other words, the distance between these two distribution under the same set of X^is will be equal for all values of indices (j, b). Further, the term representing (X^i_h, N^i) collision with any prior $(X^{i'}_h, N^{i'})$ is upper bounded here by $(\mu-1)/2^n$ as the N^i can repeat at most $\mu - 1$ times in the first $i-1$ queries and among those queries X^i_h (which is either unique or is defined by an XOR with a random function output) has collision probability of at most $1/2^n$ for each value of $X^{i'}_h$. Finally, the last inequality here follows from Exp. 7.

C.2.2) When $\Pi' = \mathsf{Espeon}'$: Let us denote by $p_{i,i'}$, the term $\mathsf{llcp}_n(N^i\|0^{n-\nu}, N^{i'}\|0^{n-\nu}) \cdot \mathsf{llcp}_n(M^i, M^{i'})$ where ν is the nonce size. In simple words, this term represents the prefixed output blocks in the i^{th} query response when only compared with the i'^{th} query-response pair. Clearly then p_i which is previously defined as the number of prefixed output blocks in the i^{th} query response of $\mathcal{A}^\star$ to Π', can now be defined as $p_i = \max_{1\leq i'<i}\{p_{i,i'}\}$.

Let us assume fo a moment that in Espeon' all tweaks T^i_js in the i^{th} query for $1 \leq j \leq \ell_i$ are unique when compared with all tweaks $\mathsf{T}^{i'}_{j'}$ among the first i queries with $(j \neq j') \vee (j \geq p_{i,i'}+2)$. Clearly then we know that the output of the i^{th} query will have a form of prefixed blocks followed by an n-bit random permutation block followed by a uniform random string of the remaining length. Note that this output is statistically no different than the corresponding ideal OPRP+RF output and hence the distinguishing advantage of $\mathcal{A}^\star$ will be zero under this event. More concretely, for $\Pi' = \mathsf{Espeon}'$ and event $E^{i',j'}_{i,j}$ defined as {for given (i,j) and (i',j') we have $((j \neq j') \vee (j \geq p_{i,i'}+2)) \wedge (\mathsf{T}^i_j = \mathsf{T}^{i'}_{j'})$} we can say that

$$\begin{aligned}
\Pr\left[\mathcal{U}^{\Pi',\mathcal{A}^\star}_{i,1}\right] &\leq \Pr\left[\bigvee_{\substack{(i',j')<(i,j)\\1\leq j\leq \ell_i}} E^{i',j'}_{i,j}\right] \\
&\leq \sum_{j=1}^{\ell_i}\sum_{(i'j')<(i,j)} \Pr\left[E^{i',j'}_{i,j} \mid \bigwedge_{(i'',j'')<(i',j')} \neg E^{i'',j''}_{i,j} \bigwedge_{\substack{(i'',j'')<(i,j^*)\\1\leq j^*\leq j-1}} \neg E^{i'',j''}_{i,j^*}\right]. \qquad (11)
\end{aligned}$$

We note the following trivial observations – 1. When $\min\{j, j'\} = 1$: $\Pr[E^{i',j'}_{i,j}] = 0$ for all values of $(i',j') < (i,j)$, due to the domain separation bits. 2. When $j = j' = 2$: $\Pr[E^{i',j'}_{i,j}]$ for any $1 \leq i' < i$ is equal to 0 when $t = 2n$ and is at most $2^2/(2^n - L_i)$ when $t = n$ bits where $L_i = \sum_{a=1}^{i} \ell_a$ denotes the total number of block/primitive calls made by $\mathcal{A}^\star$ in its first i queries to Π'. 3. When $\min\{j, j'\} \geq 2 < \max\{j, j'\}$: In Exp. 11, each probability term for $E^{i',j'}_{i,j}$ is conditioned on negations of all previous $E^{i'',j''}_{i,j}$ s with $(i'',j'') < (i',j')$ and $E^{i'',j''}_{i,j^*}$ s with $(i'',j'') < (i,j^*)$ for all $1 \leq j^* \leq j-1$, which implies for the same that the sampling of corresponding output ciphertext block C^i_{j-1} is independent of $C^i_{j-2}, C^{i'}_{j'-1}$ and $C^{i'}_{j'-2}$ and at least one of the pairs (C^i_{j-2}, C^i_{j-1}) and $(C^{i'}_{j'-2}, C^{i'}_{j'-1})$ is generated as the outputs of random permutation/s. This implies that the probability of $\mathsf{f_T}(t, C^i_{j-2}, C^i_{j-1}) = \mathsf{f_T}(t, C^{i'}_{j'-2}, C^{i'}_{j'-1})$ (and hence the conditioned event) is at most $2^{2n-(t-2)}/(2^n - L_i)^2$ where $t \in \{n, 2n\}$. Since these hold for all indexes i, i', j and j', we can write Exp. 11 for $t \in \{n, 2n\}$ as

$$\begin{aligned}
\Pr\left[\mathcal{U}^{\Pi',\mathcal{A}^\star}_{i,1}\right] &\leq \sum_{j=1}^{\ell_i}\sum_{(i'j')<(i,j)} \frac{2^{2n-(t-2)}}{(2^n - L_i)^2} \\
&\leq \sum_{j=1}^{\ell_i}\sum_{(i'j')<(i,j)} \frac{1}{2^{t-4}};\ \text{assuming } L_i \leq 2^{n-1} \\
&= \sum_{j=1}^{\ell_i} \frac{L_i - \ell_i + (j-1)}{2^{t-4}} = \frac{(L_i - \ell_i + (\ell_i - 1)/2)\ell_i}{2^{t-4}}. \qquad (12)
\end{aligned}$$

D OAE SECURITY DEFINITION

OAE Confidentiality. Let us first denote $\mathsf{B}_n = \{0,1\}^n$ and define $\mathsf{B}^*_n = \{\varepsilon\} \cup \bigcup_{i=1}^{\infty} \mathsf{B}^i_n$ with ε denoting the empty string. An online permutation can now be defined as a length preserving permutation $\pi : \mathsf{B}^*_n \to \mathsf{B}^*_n$ that also preserves the length of blockwise prefix. More concretely, for any non-negative integer m, the function π, applied to mn-bit inputs acts as a permutation. Moreover, for any pair of values M and $M' \in \mathsf{B}^*_n$, the length of their longest common prefix remains same even after applying π to them i.e. $\mathsf{llcp}_n(M, M') = \mathsf{llcp}_n(\pi(M), \pi(M'))$. Let us denote the set of all such permutations by $\mathrm{OPerm}(n)$. Note that $\mathrm{OPerm}(n)$ is a countably infinite set, therefore, uniform sampling of an online permutation is not defined on it. We stick to the well-defined lazy sampling of random online permutations from $\mathrm{OPerm}(n)$ as defined and used in [5] and denote this by $\pi \leftarrow_{\$} \mathrm{OPerm}(n)$.

The OAE confidentiality of an AEAD scheme Π can now be defined using two games, $\textbf{oprpf-real}_\Pi$ and $\textbf{oprpf-ideal}_\Pi$. In both games $\mathcal{A}$ can make arbitrary chosen plaintext queries (even with nonce repetitions) to a black box encryption oracle.

In the $\textbf{oprpf-real}_\Pi$ game, the encryption oracle implements the actual encryption algorithm of Π using a random secret key. On the other hand, in the $\textbf{oprpf-ideal}_\Pi$ game, when presented with an encryption query N, A, M, the oracle responds with $\pi_{N,A}(M_L) \| f_{N,A,M}$, where M_L is derived from M as its longest n-bit (block) aligned prefix, $\pi_{N,A} \leftarrow_{\$} \mathrm{OPerm}(n)$ is a randomly selected online permutation for each $N, A \in \mathcal{N} \times \mathcal{AD}$ and $f_{N,A,M} \leftarrow_{\$} \{0,1\}^{n+(|M| \bmod n)}$ is a random string of length $n + |M| - |M_L|$ bits (with $|M| - |M_L|$ bits corresponding to the last incomplete block of M and n bits reserved for the tag) for each $N, A, M \in \mathcal{N} \times \mathcal{AD} \times \mathcal{M}$. The **oprpf** advantage of an adversary $\mathcal{A}$ against Π can now be precisely defined as: $\mathbf{Adv}_\Pi^{\mathbf{oprpf}}(\mathcal{A}) = \Pr[\mathcal{A}^{\textbf{oprpf-real}_\Pi}] - \Pr[\mathcal{A}^{\textbf{oprpf-ideal}_\Pi}]$.

OAE Authenticity. OAE authenticity of a nonce-based AE scheme Π is defined by the unforgeability of a nonce-misusing adversary $\mathcal{A}$ against it under chosen ciphertext attacks. It is modeled through the security game **auth** where adversary is given a black-box access to the encryption and decryption oracles of Π. This means $\mathcal{A}$ can freely make arbitrary chosen plaintext queries, even allowing for nonce repetitions, to the encryption oracle. Furthermore, $\mathcal{A}$ can also issue arbitrary chosen ciphertext queries to the decryption/verification oracle with the goal of finding a valid forgery i.e. a ciphertext-tag pair that successfully decrypts and is not trivially known from the encryption queries. The advantage of $\mathcal{A}$ in breaking the OAE authenticity of Π is then defined as $\mathbf{Adv}_\Pi^{\mathbf{auth}}(\mathcal{A}) = \Pr[\mathcal{A}^{\mathbf{auth}_\Pi} \text{ forges}]$.