

Structural Cryptanalysis of the Eevee AEAD Family

Shuping Mao¹, Peng Wang²(✉), Jiadong Han², Tingting Guo³, Yan Jia^{4,5}, Ying Chen¹

¹ Beijing Electronic Science & Technology Institute, Beijing, China

² School of Cryptology, University of Chinese Academy of Sciences, Beijing, China

p-wang@ucas.ac.cn

³ Ningbo Key Laboratory of Information Technology Application Innovation and Security, School of Cyber Science and Engineering, Ningbo University of Technology, Ningbo, China

⁴ State Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China

⁵ School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

Abstract. Eevee is a family of three AEAD modes, Umbreon, Jolteon, and Espeon, proposed at CCS 2023 for IoT-to-cloud secure computation. We analyze structural flaws in the algorithms of the supplied full-version specification and the proof steps underlying their online authenticated-encryption claims. An AD-free, two-encryption prefix distinguisher has advantage $1 - 2^{-n}$ in the ideal-forkcipher model, contradicting the specified online ideal object. Independently, the separable associated-data transform yields a four-encryption nonce-misuse distinguisher with advantage $1 - 2^{-n}$, a deterministic AD-only forgery, and nonempty-message forgeries obtained from cross-domain state differences. These attacks contradict the advertised nonce-misuse bounds of Umbreon and Espeon. With nonce-respecting encryption queries, one two-block Jolteon encryption permits a fresh one-block forgery with the same tag, while Espeon’s short-message path permits arbitrary AD substitution. We trace these counterexamples to failures of online-prefix consistency, conditional transcript distributions, domain separation, and checksum freshness in the original proof. The authenticity attacks follow from deterministic identities, and the privacy counterexamples persist with an ideal forkcipher. We derive corresponding requirements for AD processing and message finalization, and briefly summarize supporting checks of the published C routines. The central failures concern the mode design and cannot be resolved solely by correcting implementation encodings.

Keywords: Eevee · Authenticated encryption · Forkcipher · Nonce misuse · Forgery

1 Introduction

Authenticated encryption with associated data (AEAD) protects message confidentiality and binds public associated data to the ciphertext [BN00,Rog02]. Online authenticated encryption (OAE) additionally addresses incremental message processing [FFL12,HRRV15]. Its guarantees must distinguish a nonce-respecting (NR) adversary, whose encryption nonces are pairwise distinct, from a nonce-misusing (NM) adversary, which may repeat them. Verification queries may reuse an encryption nonce in either setting; freshness applies to the complete verification record.

The Eevee family, proposed by Bhati et al. at CCS 2023 [BPA⁺23a], combines IoT-friendly encryption with parallel MPC-friendly decryption. Its three members, Umbreon, Jolteon, and Espeon, use a tweakable forkcipher with two output branches. Our primary object of analysis is the mode design in Fig. 9 and the claims in Theorem 3.1 of the supplied full version, ePrint 2023/1361 [BPA⁺23b]. The full version advertises NM OAE confidentiality and authenticity for Umbreon and Espeon, and NR guarantees for Jolteon. Figures 1–4 give the original diagrams and illustrate complete-final-block processing. Full algorithms are given in Section 2 and Appendix A.

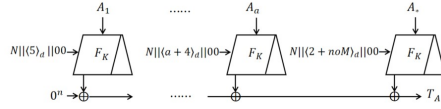


Fig. 1: Common AD part.

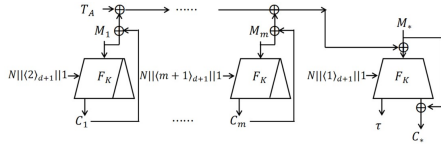


Fig. 3: Jolteon.

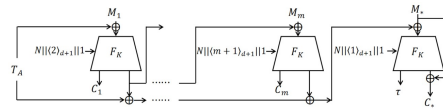


Fig. 2: Umbreon.

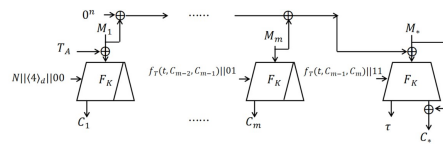


Fig. 4: Espeon.

Our analysis focuses on three aspects of the construction: whether the message layer preserves the prefix consistency required by its online ideal object, whether AD processing hides joint relations across queries and message domains, and whether finalization binds the complete ciphertext record. The counterexamples exploit deterministic identities and control-flow dependencies in the algorithms; they do not rely on weak keys or a weakness in ForkSkinny.

Our contributions. We make the following contributions and Table 1 provides a summary of the attacks.

1. **Nonce-misuse attacks on online processing and AD aggregation.** Attack 1 uses two encryption queries with empty AD to distinguish Umbreon, Jolteon and Espeon from the original OAE ideal object with advantage $1 - 2^{-n}$ in the ideal-forkcipher model. The separable two-block AD state $\mathcal{H}_b(X \parallel Y) = P(X) \oplus Q_b(Y)$ independently gives the four-encryption distinguisher of Attack 2 and the three-encryption AD-only forgery of Attack 3. AD-only responses also disclose exact differences between nonempty-message AD states, yielding Attacks 4 and 5. These attacks refute Umbreon’s and Espeon’s NM bounds; their repeated encryption nonces do not contradict Jolteon’s NR-only claim.
2. **Nonce-respecting forgeries against message finalization.** Attack 6 turns one encryption of two complete Jolteon message blocks into a fresh, accepted one-block ciphertext $C' = C_2 \oplus M_1 \oplus C_1$ with the old tag. The transfer works with empty AD and extends to removal of an arbitrary complete prefix. Attack 7 exploits Espeon’s

short-message branch to substitute arbitrary AD without changing the ciphertext or tag. Both attacks use one encryption query and succeed with probability one.

3. **Proof failures and design requirements.** We identify a missing argument for online-prefix consistency, the treatment of conditional AD-tag distributions in Appendix C, Eq. (7), the independence assertion in its Observation 1, and the checksum-freshness inference in Section 7’s integrity Case 1. We substitute explicit attack resources into Theorem 3.1 and derive repair requirements for joint AD-state security, domain separation, and finalization. Supporting observations from the published C routines are summarized briefly in Section 6.1.

Table 1: Algorithm-level attacks and representative input lengths. U, J, E denote Umbleon, Jolteon, Espeon. Lengths are in bits unless marked as bytes. The query tuple is (q_e, q_v, μ) , with μ counting encryption-nonce multiplicity only. Every forgery succeeds with probability 1; both distinguishing advantages are $1 - 2^{-n}$.

Attack	Modes	Mechanism	Threat / goal	Queries	AD length	Message length
1	U,J,E	Online prefix inconsistency	NM / privacy	(2, 0, 2)	0	$n, 2n$
2	U,J,E	Four-point AD-tag relation	NM / privacy	(4, 0, 4)	$2n$	0
3	U,J,E	AD-only interpolation	NM / auth.	(3, 1, 3)	$2n$	0
4	U,J	Cross-domain AD difference	NM / auth.	(3, 1, 3)	$2n$	$0, n$
5	E	Cross-domain AD difference	NM / auth.	(3, 1, 3)	$2n$	$0, 2n$
6	J	Cross-length truncation	NR / auth.	(1, 1, 1)	0	$2n$
7	E	Short-message AD omission	NR / auth.	(1, 1, 1)	1 byte	n

The two privacy attacks distinguish the claimed ideal OAE distribution through ciphertext-prefix consistency and AD-tag relations, respectively; they do not establish key recovery or general plaintext recovery. The NR attacks target authenticity and do not by themselves demonstrate an NR confidentiality break. The representative algorithm-level attacks in Table 1 use short AD strings and messages, so their small oracle counts do not conceal long-input processing requirements.

Section 2 defines the games and recalls the claims. Sections 3 and 4 present the seven numbered attacks and the structural lemmas that support them. Section 5 traces the contradictions to the original bounds and proof. Section 6 states algorithmic repair requirements and briefly summarizes implementation observations and validation. Appendix A restates the analyzed algorithms, Appendix B collects the attack procedures in the same order, and Appendix C identifies the source material.

2 Preliminaries

A block cipher $E : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ defines a permutation E_K on $\{0, 1\}^n$ for every key K . A tweakable block cipher $\tilde{E} : \{0, 1\}^k \times \{0, 1\}^n \times \{0, 1\}^t \rightarrow \{0, 1\}^n$ defines a permutation $\tilde{E}_K^T$ for every key K and tweak T . We write E_K^{-1} and $\tilde{D}_K^T = (\tilde{E}_K^T)^{-1}$ for their inverses. With the key fixed, inversion takes the ciphertext block and tweak as inputs, not another key argument.

A tweakable forkcipher F with key space $\{0, 1\}^k$, tweak space $\{0, 1\}^t$, and block space $\{0, 1\}^n$ consists of a forward algorithm and an inversion algorithm. For $K \in \{0, 1\}^k$, $T \in \{0, 1\}^t$, and $X \in \{0, 1\}^n$, F returns two n -bit branches: $F_K^{T,b}(X) = (F_K^{T,0}(X), F_K^{T,1}(X))$. For each T and branch $b \in \{0, 1\}$, $F_K^{T,b}$ is a permutation. We use $F_K^{-1,T,0,b}(Y) = (X, F_K^{T,1}(X))$ where $X = (F_K^{T,0})^{-1}(Y)$, and $F_K^{-1,T,0,i}(Y) = X$. Superscripts 0, 1, b denote a selected branch or both.

Let $n > 0$ be a block length. All strings are binary; ε is empty, $|X|$ is bit length, $X \parallel Y$ concatenation, and $X \oplus Y$ equal-length bitwise XOR. When operands have different lengths, the original paper right-extends the shorter operand with zeros before XOR; we use equal-length operands unless explicitly stated. Bit slices $X[u : v]$ use inclusive bit indices. We write $\text{left}_r(X)$ for the first r bits, and $\langle i \rangle_d$ for the d -bit binary encoding of i , provided $i < 2^d$. Put $\{0, 1\}^n$ for the n -bit strings and $\text{Perm}(n)$ for permutations thereon.

2.1 Security notation

Let $\Pi = (\mathcal{K}, \text{Enc}, \text{Dec})$ be a deterministic, nonce-based authenticated-encryption scheme with associated data (AEAD). Its input domains are a nonce set $\mathcal{N}$, an associated-data set $\mathcal{AD}$, and a message set $\mathcal{M}$. Given a key $K \in \mathcal{K}$, nonce $N \in \mathcal{N}$, associated data A , and message M , the encryption algorithm returns $(C, \tau) \leftarrow \text{Enc}_K(N, A, M)$, where $|C| = |M|$ and the authentication tag satisfies $|\tau| = n$. The decryption algorithm accepts (N, A, C, τ) and either recovers the plaintext or rejects. We write $\text{Acc}_K(N, A, C, \tau) := [\text{Dec}_K(N, A, C, \tau) \neq \perp]$. In particular, Eevee excludes $A = M = \varepsilon$, but permits nonempty AD with an empty message. The printed $\top$ output on successful AD-only verification denotes acceptance and may be represented as returning ε .

Nonce-respecting and nonce-misusing adversaries. An adversary is nonce-respecting (NR) if the nonces in its encryption queries are pairwise distinct; it is nonce-misusing (NM) if it may reuse a nonce in encryption queries. We denote these adversary classes by $\mathcal{A}_{\text{nr}}$ and $\mathcal{A}_{\text{nm}}$. For q_e encryption queries with nonce sequence $(N_1, \dots, N_{q_e})$, define the maximal encryption-nonce multiplicity by

$$\mu := \max_{N \in \mathcal{N}} |\{j \in \{1, \dots, q_e\} : N_j = N\}|.$$

Thus $\mu \leq 1$ for an NR adversary (and $\mu = 1$ when $q_e > 0$), whereas $1 \leq \mu \leq q_e$ is possible for NM adversaries. Crucially, NR does not forbid verification queries using a nonce that has appeared in an encryption query. In both classes, let q_v be the number of verification queries. We write $\sigma_{m,e}$ and $\sigma_{m,v}$ for the total numbers of message-block processing calls induced by encryption and verification queries, respectively.

OAE confidentiality (oprpf). The target AE notion of Eevee family is online AE (OAE) [FFL12, HRRV15]. Eevee's OAE privacy notion compares a real encryption oracle with an ideal online permutation plus random final output oracle (Appendix D of [BPA⁺23b]). The real game samples one key $K \xleftarrow{\$} \mathcal{K}$ at initialization and uses it for every query. Let

$$\ell(M) := n \left\lfloor \frac{|M|}{n} \right\rfloor, M_L := \text{left}_{n \lfloor |M|/n \rfloor}(M), r(M) := |M| - \ell(M) \in \{0, \dots, n-1\}.$$

Here M_L is the longest block-aligned prefix and is empty if $|M| < n$. A blockwise online permutation maps a full-block string to an equally long string, preserves prefix consistency, and is a permutation for each fixed full-block input length. We use the standard consistent lazy-sampling convention. For each distinct pair (N, A) , sample an independent ideal online permutation $\pi_{N,A}$. Independently, for each distinct triple (N, A, M) , lazily sample a uniform string $R_{N,A,M} \xleftarrow{\$} \{0, 1\}^{n+r(M)}$. If the same triple is queried again, the same sampled $R_{N,A,M}$ is reused. Parse $R_{N,A,M} = R_C \| R_\tau$, where $|R_C| = r(M)$ and $|R_\tau| = n$. The two oracle worlds are

$$\begin{aligned}\mathcal{O}_H^{\text{real}}(N, A, M) &:= \text{Enc}_K(N, A, M), \\ \mathcal{O}_H^{\text{ideal}}(N, A, M) &:= (\pi_{N,A}(M_L) \| R_C, R_\tau).\end{aligned}$$

The permutation objects and final random values are sampled consistently across an adaptive query transcript. In particular, for fixed (N, A) and full blocks $X, Y \in \{0, 1\}^n$, the first ciphertext block returned on $X \| Y$ equals the ciphertext returned on X , with probability one. Both queries use the same $\pi_{N,A}$, and a complete last block belongs to M_L . The OAE confidentiality advantage for $\nu \in \{\text{nr}, \text{nm}\}$ is

$$\text{Adv}_H^{\text{oprpf}}(\mathcal{A}_\nu) := \Pr[\mathcal{A}_\nu^{\mathcal{O}_H^{\text{real}}} = 1] - \Pr[\mathcal{A}_\nu^{\mathcal{O}_H^{\text{ideal}}} = 1].$$

OAE authenticity (auth). The authenticity experiment gives $\mathcal{A}_\nu$ an encryption oracle and a decryption/verification oracle under one randomly chosen key K . An initially empty set $\mathcal{L}$ stores the records produced by the encryption oracle:

$\mathcal{O}_{\text{Enc}}(N, A, M) : (C, \tau) \leftarrow \text{Enc}_K(N, A, M); \mathcal{L} \leftarrow \mathcal{L} \cup \{(N, A, C, \tau)\}; \text{return } (C, \tau).$
 $\mathcal{O}_{\text{Ver}}(N, A, C, \tau) : \text{if } (N, A, C, \tau) \in \mathcal{L}, \text{ return } 0; b \leftarrow \text{Acc}_K(N, A, C, \tau); \text{return } b.$

A successful forgery is a fresh record, namely $(N, A, C, \tau) \notin \mathcal{L}$, for which $\text{Acc}_K(N, A, C, \tau) = 1$. Equivalently, the experiment wins if at least one of the adversary's at most q_v verification queries is fresh and accepted. $\mathcal{L}_j$ denotes the encryption set immediately before the j th verification query, define

$$\text{Adv}_H^{\text{auth}}(\mathcal{A}_\nu) := \Pr[\exists j \leq q_v : (N_j, A_j, C_j, \tau_j) \notin \mathcal{L}_j \wedge \text{Acc}_K(N_j, A_j, C_j, \tau_j) = 1].$$

2.2 The Eevee Family

The complete Eevee algorithm is presented in Fig. 9 of [BPA⁺23b]; here we rewrite the algorithm and provide the full version in Appendix A (Algorithms 1-4). Our restatement separates the color-coded shared program into member-specific procedures. The common AD subroutine receives noM explicitly, where Fig. 9 treats it as a variable from its caller. Throughout, $d = t - |N| - 2$ and counter encodings are used only when representable. The two-block AD attacks require $d \geq 3$, so that $\langle 5 \rangle_d$ exists. Espeon additionally requires $|N| \leq n - 2$ on the nonempty-message path. For a nonempty string X , X_* is its unpadding last block and X'_* is the corresponding n -bit padded block; they coincide for a complete last block. Thus $1 \leq \text{res}_X = |X_*| \leq n$, and $\text{pad}_X = 1$ iff $\text{res}_X = n$. We write $a = \lceil |A|/n \rceil - 1$ and $m = \lceil |M|/n \rceil - 1$ for the numbers of nonfinal AD and message blocks, respectively, when the corresponding string is nonempty.

These block counts differ from the bit length $\ell(M) = n \lfloor |M|/n \rfloor$ and the remainder $r(M) \in \{0, \dots, n-1\}$. A prime on an indexed message or ciphertext block, such as M'_i or C'_i , instead denotes a block in the modified record; the starred notation X'_* specifically denotes padding. Finally, C_0 is a mode-dependent initialization block. The initial AD state T_A is denoted by S (or S_j for A_j) in the attack proofs, and Δ denotes a running checksum.

Our attacks use the following features of Eevee (Fig. 9 in [BPA⁺23b] and Algorithms 1-4 in Appendix A):

1. The AD state is an XOR of branch-zero evaluations, and an empty message exposes this state directly as the tag.
2. Umbreon and Jolteon inject the AD state into a one-block final input. The final tweak depends on the nonce and completeness flag but not on the number of preceding blocks.
3. Espeon injects the AD state into its first nonfinal message block, and its final input contains a plaintext checksum. With no nonfinal block, the final input omits the AD state.

Security claims of Eevee. Theorem 3.1 of [BPA⁺23b] gives the following implications for Jolteon under NR adversaries:

$$\begin{aligned} \text{Adv}_J^{\text{auth}}(\mathcal{A}_{\text{nr}}) &\leq \text{Adv}_J^{\text{oprpf}}(\mathcal{A}_{\text{nr}}) + \frac{q_v}{2^{n-2}}, \\ \text{Adv}_J^{\text{oprpf}}(\mathcal{A}_{\text{nr}}) &\leq \text{Adv}_F^{\text{prtfp}}(\mathcal{B}). \end{aligned}$$

For Umbreon and Espeon under NM adversaries, the respective confidentiality estimates and authenticity losses are

$$\begin{aligned} \text{Adv}_U^{\text{oprpf}}(\mathcal{A}_{\text{nm}}) &\leq \text{Adv}_F^{\text{prtfp}}(\mathcal{B}) + \frac{3(\sigma_{m,e} + q_e)(\mu - 1)}{2^{n-1}}, \\ \text{Adv}_U^{\text{auth}}(\mathcal{A}_{\text{nm}}) &\leq \text{Adv}_U^{\text{oprpf}}(\mathcal{A}_{\text{nm}}) + \frac{q_v}{2^{n-4}}, \\ \text{Adv}_E^{\text{oprpf}}(\mathcal{A}_{\text{nm}}) &\leq \text{Adv}_F^{\text{prtfp}}(\mathcal{B}) + \frac{2q_e(\mu - 1)}{2^n} + \frac{\sigma_{m,e}^2}{2^{t-3}}, \\ \text{Adv}_E^{\text{auth}}(\mathcal{A}_{\text{nm}}) &\leq \text{Adv}_E^{\text{oprpf}}(\mathcal{A}_{\text{nm}}) + \frac{q_v}{2^{n-4}}. \end{aligned}$$

Here $\sigma_{m,e}$ is the count of forkcipher calls induced by the message portions of encryption queries.

3 Nonce-Misuse Attacks for the Eevee Family

We give five nonce-misuse attacks on the Eevee Family. Attack 1 tests online-prefix consistency without AD. Attacks 2-5 exploit exact relations in the shared AD transform, progressing from distinguishing and AD-only forgery to nonempty-message forgeries. The AD attacks use only deterministic, correctly invertible branch evaluations. Replacing ForkSkinny by an ideal forkcipher does not prevent these relations.

Attack 1. AD-free online-prefix distinguishing

The first counterexample uses no AD and tests the block-prefix consistency required by the original ideal OAE object. Choose legal N and $X, Y \in \{0, 1\}^n$, and make two encryption queries with the same nonce:

$$(C, \tau) \leftarrow \text{Enc}_K(N, \varepsilon, X), (C'_1 \parallel C'_2, \tau') \leftarrow \text{Enc}_K(N, \varepsilon, X \parallel Y).$$

Both inputs are legal because their messages are nonempty. The distinguisher outputs one (“real”) if and only if $C \neq C'_1$; neither tag nor the second ciphertext block is tested.

Attack. For Umbreon, Jolteon and Espeon algorithms instantiated with the ideal fork-cipher used in the original proof, there is a distinguisher with $(q_e, q_v, \mu; \sigma_{m,e}, \sigma_{m,v}) = (2, 0, 2; 3, 0)$ such that

$$\text{Adv}_H^{\text{oprpf}}(\mathcal{D}_{\text{prefix}}) = 1 - 2^{-n}.$$

The nonce and tweak parameters must satisfy the respective printed algorithm’s domain restrictions; no AD-processing calls are made.

Analysis. In the ideal OAE world, $M_L = X$ for the first query and $M_L = X \parallel Y$ for the second. The same online permutation $\pi_{N,\varepsilon}$ processes both strings. Prefix consistency therefore gives

$$\Pr_{\text{ideal}}[C = C'_1] = 1.$$

In real Umbreon (Algorithm 2), the AD state and initial checksum are zero. The one-block query uses $T_f^U = N \parallel \langle 1 \rangle_{d+1} \parallel 1$ and returns $C = F_K^{T_f^U,1}(X) \oplus X$. The first block of the two-block query instead uses $T_1^U = N \parallel \langle 2 \rangle_{d+1} \parallel 1$ and returns $C'_1 = F_K^{T_1^U,0}(X)$.

In real Jolteon (Algorithm 3), the empty AD implies $T_A = C_0 = 0^n$. For the one-block query, $\Delta = C_0 = 0^n$ and $T_f^J = N \parallel \langle 1 \rangle_{d+1} \parallel 1$, returns $C = F_K^{T_f^J,1}(X) \oplus X$. For the two-block query, $T_1^J = N \parallel \langle 2 \rangle_{d+1} \parallel 1$, yielding $C'_1 = F_K^{T_1^J,0}(X)$.

In real Espeon (Algorithm 4), the corresponding outputs are $C = F_K^{T_f^E,1}(X) \oplus X$ and $C'_1 = F_K^{T_1^E,0}(X)$, where $T_f^E = N \parallel \langle 4 \rangle_d \parallel 11$ and $T_1^E = N \parallel \langle 4 \rangle_d \parallel 00$. In each mode, the final and ordinary tweaks are distinct. In the original proof’s replacement model, the branch permutations at these tweaks are independent, so their evaluations at the fixed X are independent uniform n -bit strings. XOR with X preserves uniformity, giving

$$\Pr_{\text{real}}[C = C'_1] = 2^{-n}.$$

Thus the output-one probabilities are $1 - 2^{-n}$ in the real world and zero in the ideal world. Their difference is $1 - 2^{-n}$. The queries process one and two message blocks, respectively, so $\sigma_{m,e} = 3$; since AD is empty and there are no verification queries, also $\sigma = 3$.

3.1 The separable algebra of the shared AD transform

Fix a key K and a nonce N satisfying the applicable nonce restrictions and $d = t - |N| - 2 \geq 3$. As in Algorithm 1, let

$$\mathcal{H}_b(A) := \text{AD}(K, N, A, b), b = \text{noM} \in \{0, 1\}.$$

Here $b = 1$ indicates that the message is empty; $b = 0$ indicates that it is nonempty. For fixed K, N , we suppress their occurrence in the notation $\mathcal{H}_b$. Notice that $\mathcal{H}_b(A)$ is an n -bit internal AD state, not in general a public encryption output. For an empty message, however, the Eevee encryption algorithms return that state directly as the authentication tag (Line 7 in Algorithm 2, Line 7 in Algorithm 3, Line 7 in Algorithm 4):

$$\text{Enc}_K(N, A, \varepsilon) = (\varepsilon, \mathcal{H}_1(A)), A \neq \varepsilon. \quad (1)$$

We only need AD strings made of two complete n -bit blocks. This avoids all padding ambiguities and ensures that the final AD block has $\text{pad}_A = 1$. In Algorithm 1, the first block is a nonfinal AD block with counter $i + 4 = 5$; the second is the final AD block with counter $2\text{pad}_A + b = 2 + b$. Define the following fixed, keyed n -bit permutations:

$$\begin{aligned} T_P &:= N \parallel \langle 5 \rangle_d \parallel 00, P(X) := F_K^{T_P, 0}(X), \\ T_{Q_b} &:= N \parallel \langle 2 + b \rangle_d \parallel 00, Q_b(Y) := F_K^{T_{Q_b}, 0}(Y). \end{aligned}$$

The same P is used for $b = 0$ and $b = 1$; only Q_b changes.

Lemma 1 (Separable two-block AD transform). *For any $X, Y \in \{0, 1\}^n$ and $b \in \{0, 1\}$, the AD subroutine of each Eevee member satisfies*

$$\mathcal{H}_b(X \parallel Y) = P(X) \oplus Q_b(Y).$$

When the plaintext is empty, $b = 1$ and the right-hand side is returned verbatim as the encryption tag.

Proof. From Algorithm 1, the initial accumulator is $T_A = 0^n$. The first full AD block X is handled inside the nonfinal-block loop, with tweak T_P and update $T_A \leftarrow T_A \oplus P(X) = P(X)$. Because the remaining block Y is complete, $\text{pad}_A = 1$; the last-block tweak is T_{Q_b} , so the final update produces $T_A = P(X) \oplus Q_b(Y)$. Equation (1) then proves the claim about the empty-message tag.

Attack 2. Four-point AD-tag distinguishing

Choose four n -bit strings X_0, X_1, Y_0, Y_1 such that $X_0 \neq X_1$ and $Y_0 \neq Y_1$. Define four pairwise distinct AD strings $A_{ij} := X_i \parallel Y_j, i, j \in \{0, 1\}$. Use the same nonce N in each of four empty-message encryption queries, and denote the corresponding tags by $\tau_{ij} := \mathcal{H}_1(A_{ij})$. From Lemma 1:

$$\begin{aligned} \tau_{00} &= P(X_0) \oplus Q_1(Y_0), \tau_{01} = P(X_0) \oplus Q_1(Y_1), \\ \tau_{10} &= P(X_1) \oplus Q_1(Y_0), \tau_{11} = P(X_1) \oplus Q_1(Y_1). \end{aligned}$$

Every one of the four primitive outputs occurs exactly twice in the XOR of these equations. Consequently,

$$\tau_{00} \oplus \tau_{01} \oplus \tau_{10} \oplus \tau_{11} = 0^n. \quad (2)$$

This equality is exact for every key and forkcipher.

Attack. Let Π be any of Umbreon, Jolteon, or Espeon. The four-point test is an OAE distinguisher that makes $q_e = 4$ encryption queries with one nonce ($\mu = 4$), no verification queries, and obtains

$$\text{Adv}_{\Pi}^{\text{oprpf}}(\mathcal{D}) = 1 - 2^{-n}.$$

Analysis. The adversary queries (N, A_{ij}, ε) for all $i, j \in \{0, 1\}$, computes $R = \tau_{00} \oplus \tau_{01} \oplus \tau_{10} \oplus \tau_{11}$, and outputs one (“real”) precisely when $R = 0^n$. By Equation (2), the event holds with probability one in the real world: $\Pr[\mathcal{D}^{\text{real}} = 1] = 1$.

In the ideal OAE confidentiality game, the empty message has no complete message prefix. Its response consists of a random n -bit final tag indexed by (N, A, ε) . Since the four values A_{ij} are pairwise distinct, the four ideal tags $U_{00}, U_{01}, U_{10}, U_{11}$ are independent uniform elements of $\{0, 1\}^n$. After fixing the first three, $U_{00} \oplus U_{01} \oplus U_{10} \oplus U_{11} = 0^n$ holds precisely when $U_{11} = U_{00} \oplus U_{01} \oplus U_{10}$, an event of probability 2^{-n} . Hence $\Pr[\mathcal{D}^{\text{ideal}} = 1] = 2^{-n}$. Hence $\text{Adv}_{\Pi}^{\text{oprpf}}(\mathcal{D}) = 1 - 2^{-n}$.

Attack 3. AD-only forgery by three-tag interpolation

The four-point identity yields a strictly cheaper authentication attack: the fourth encryption response is not needed, because its tag is determined by the first three.

Attack. Let Π be any of Umbreon, Jolteon, or Espeon, and let N satisfy the section’s restrictions. An adversary making $q_e = 3$ empty-message encryption queries using that nonce and $q_v = 1$ verification query produces a fresh accepted record with probability one. Thus $\mu = 3$ and

$$\text{Adv}_{\Pi}^{\text{auth}}(\mathcal{A}_{\text{nm}}) = 1.$$

Analysis. Query A_{00}, A_{01}, A_{10} with empty plaintext, obtaining $\tau_{00}, \tau_{01}, \tau_{10}$, respectively. Compute the new tag $\tau^* := \tau_{00} \oplus \tau_{01} \oplus \tau_{10}$. Rearranging (2) proves that $\tau^* = \tau_{11} = \mathcal{H}_1(A_{11})$ for every possible key. Submit the verification query $(N, A_{11}, C^* = \varepsilon, \tau^*)$. For an empty ciphertext, the decryption branches in Algorithms 2–4 accept exactly when the proposed tag equals $\mathcal{H}_1(A_{11})$. The query therefore succeeds with probability one. The target is fresh. The three encrypted tuples used A_{00}, A_{01}, A_{10} , while the verification tuple uses A_{11} . Their pairwise distinctness follows from $X_0 \neq X_1$ and $Y_0 \neq Y_1$. No query containing (N, A_{11}, C^*) was previously answered by encryption.

Attacks 1 to 3 are nonce-misuse attacks. They directly invalidate the claimed NM confidentiality and authenticity guarantees of Umbreon and Espeon. The same adversaries are valid against Jolteon as a nonce-misusing system, but do not contradict Jolteon’s stated NR security claim. Section 4 deals separately with attacks that satisfy NR constraints.

3.2 Cross-domain differences leaked by AD-only queries

Next, we show that empty-message oracle responses leak exact AD-state differences that can be transplanted into encryptions of nonempty messages. The original domain separator noM changes only the final AD-block tweak, not the tweak used by the preceding AD blocks.

Fix two AD strings $A_0 := X_0 \parallel Y, A_1 := X_1 \parallel Y, X_0 \neq X_1, X_0, X_1, Y \in \{0, 1\}^n$. Using two AD-only encryption queries with the same nonce, the adversary learns $\mathcal{H}_1(A_0)$ and $\mathcal{H}_1(A_1)$, and can form

$$\begin{aligned} \delta &:= \mathcal{H}_1(A_0) \oplus \mathcal{H}_1(A_1) = (P(X_0) \oplus Q_1(Y)) \oplus (P(X_1) \oplus Q_1(Y)) \\ &= P(X_0) \oplus P(X_1). \end{aligned} \quad (3)$$

Since P is a permutation and $X_0 \neq X_1$, necessarily $\delta \neq 0^n$. The same calculation with $b = 0$ proves a stronger structural statement.

Lemma 2 (Cross-domain AD-difference invariance). *For $A_0 := X_0 \parallel Y, A_1 := X_1 \parallel Y$ with $X_0 \neq X_1$ and $X_0, X_1, Y \in \{0, 1\}^n$,*

$$\mathcal{H}_0(A_0) \oplus \mathcal{H}_0(A_1) = \mathcal{H}_1(A_0) \oplus \mathcal{H}_1(A_1) = \delta.$$

In particular, two AD-only encryption queries reveal the exact XOR difference of the otherwise secret nonempty-message AD states.

Proof. By Lemma 1,

$$\begin{aligned} \mathcal{H}_0(A_0) \oplus \mathcal{H}_0(A_1) &= (P(X_0) \oplus Q_0(Y)) \oplus (P(X_1) \oplus Q_0(Y)) \\ &= P(X_0) \oplus P(X_1) = \delta = \mathcal{H}_1(A_0) \oplus \mathcal{H}_1(A_1). \end{aligned}$$

The last equality follows from (3). The final-block outputs $Q_0(Y)$ and $Q_1(Y)$ need not be equal; each cancels within its own domain. Therefore, separating the final AD-block tweak by noM does not hide the shared prefix difference.

Crucially, the adversary does not recover either $S_0 := \mathcal{H}_0(A_0)$ or $S_1 := \mathcal{H}_0(A_1)$ individually. It only learns $S_0 \oplus S_1 = \delta$. Attacks 4 and 5 demonstrate that this differential information is sufficient to generate fully valid nonempty ciphertext.

Attack 4. Nonempty-message forgery for Umbreon and Jolteon

We first use a single complete message block $M \in \{0, 1\}^n$. For each mode, the parsing convention of Eevee gives $m = 0$: there are no nonfinal message blocks, and the only block is the final complete block with $\text{pad}_M = 1$. Write $S_j = \mathcal{H}_0(A_j)$ for $j \in \{0, 1\}$. By Lemma 2, $S_0 \oplus S_1 = \delta$. For Umbreon, Line 14 in Algorithm 2 updates $\Delta \leftarrow \Delta \oplus T_A = S_0$ before the final call. For Jolteon, the initialization is $C_0 = T_A = S_0$ (Line 10 in Algorithm 3), followed by $\Delta \leftarrow \Delta \oplus C_m = C_0 = S_0$ at finalization (Line 16 in Algorithm 3). Thus, although their nonfinal processing differs, their complete one-block finalizations have the same form:

$$T_f := N \parallel \langle 1 \rangle_{d+1} \parallel 1, \quad (4)$$

$$(\tau, Z) = F_K^{T_f, b}(M \oplus S_0), \quad (5)$$

$$C = Z \oplus M. \quad (6)$$

Here τ is the tag branch and Z is the other branch used to form the final ciphertext. The final tweak depends on N and padding status but not on the changed AD block.

Attack. For Umbreon and Jolteon, there exists an adversary using $q_e = 3$ encryption queries under one nonce ($\mu = 3$) and $q_v = 1$ verification query that outputs a fresh, valid complete one-block ciphertext with probability 1. The recovered plaintext differs from the original plaintext.

Analysis. Make the two AD-only queries from Lemma 2 and compute δ . Make the third encryption query on (N, A_0, M) with $M \in \{0, 1\}^n$, obtaining (C, τ) . Define $M' := M \oplus \delta, C' := C \oplus \delta, \tau' := \tau$. Consider processing (N, A_1, M') . Then

$$M' \oplus S_1 = (M \oplus \delta) \oplus S_1 = M \oplus (S_0 \oplus S_1) \oplus S_1 = M \oplus S_0.$$

Moreover, M' has the same complete-block length as M , so the final tweak remains T_f in (4). Consequently, deterministic evaluation of the same forkcipher on the same $(T_f, M \oplus S_0)$ returns the same (τ, Z) . The corresponding ciphertext block is

$$Z \oplus M' = Z \oplus M \oplus \delta = C' \oplus \delta = C'.$$

Thus $\text{Enc}_K(N, A_1, M') = (C', \tau)$, and correctness gives

$$\text{Dec}_K(N, A_1, C', \tau) = M \oplus \delta.$$

The target tuple is fresh because $A_1 \neq A_0$ and the sole nonempty-message encryption query used A_0 ; the earlier query involving A_1 returned an empty ciphertext. In fact, $\delta \neq 0^n$, so also $C' \neq C$ and $M' \neq M$.

Attack 5. Two-block nonempty-message forgery for Espeon

Espeon processes the message differently: its initial ciphertext block mixes T_A into the first nonfinal plaintext block, while its final authentication call receives a checksum of the message blocks and a tweak derived from earlier ciphertext blocks. We use two complete message blocks $M_1, M_2 \in \{0, 1\}^n$, for which $m = 1$ and $\text{pad}_M = 1$.

For the encryption query $(N, A_0, M_1 \parallel M_2)$, let $S_0 = \mathcal{H}_0(A_0)$, from Algorithm 4,

$$C_0 := N \parallel 0^{n-|N|}, T_1 := N \parallel \langle 4 \rangle_d \parallel 00, C_1 := F_K^{T_1, 0}(M_1 \oplus S_0), \Delta := M_1.$$

The final tweak and forkcipher input are

$$T_f := f_T(t, C_0, C_1) \parallel 11, U := M_2 \oplus \Delta = M_2 \oplus M_1.$$

The corresponding forkcipher call and last ciphertext block are

$$(\tau, Z) := F_K^{T_f, b}(U), C_2 := Z \oplus M_2.$$

Attack. For the printed Espeon algorithm, two AD-only encryption queries and one encryption of a two-complete-block message under the same nonce allow a fresh two-block ciphertext forgery using one verification query, with success probability 1. In symbols, $q_e = 3, q_v = 1, \mu = 3$.

Analysis. As before, first obtain $\delta = \mathcal{H}_0(A_0) \oplus \mathcal{H}_0(A_1)$ from two AD-only queries, then encrypt $M_1 \parallel M_2$ under A_0 to obtain $(C_1 \parallel C_2, \tau)$. Define $C'_1 := C_1, C'_2 := C_2 \oplus \delta, \tau' := \tau$. We show that it is a valid ciphertext for the new AD A_1 .

Define $M'_1 := M_1 \oplus \delta$, $M'_2 := M_2 \oplus \delta$. The first nonfinal encryption step under A_1 uses state $S_1 := \mathcal{H}_0(A_1)$ and thus has input

$$M'_1 \oplus S_1 = (M_1 \oplus \delta) \oplus S_1 = M_1 \oplus S_0,$$

where $\delta = S_0 \oplus S_1$. Its tweak remains T_1 , so

$$F_K^{T_1,0}(M'_1 \oplus S_1) = F_K^{T_1,0}(M_1 \oplus S_0) = C'_1 = C_1.$$

Then N and C_1 are unchanged; hence the input pair (C_0, C_1) of f_T is unchanged. Because both last message blocks are complete, the final padding bit is also unchanged. Therefore the new final tweak is exactly

$$T'_f = f_T(t, C_0, C'_1) \parallel 11 = T_f. \quad (7)$$

The new checksum after the first nonfinal message block is $\Delta' = M'_1 = M_1 \oplus \delta$. The final input then becomes

$$U' = M'_2 \oplus \Delta' = (M_2 \oplus \delta) \oplus (M_1 \oplus \delta) = M_2 \oplus M_1 = U. \quad (8)$$

Equations (7) and (8) imply that the same final forkcipher call produces exactly the same (τ, Z) . Its ciphertext consistency relation is preserved because

$$Z \oplus M'_2 = Z \oplus (M_2 \oplus \delta) = C_2 \oplus \delta = C'_2.$$

Thus the complete forged record is an encryption under A_1 of $M'_1 \parallel M'_2$:

$$\text{Dec}_K(N, A_1, C_1 \parallel (C_2 \oplus \delta), \tau) = (M_1 \oplus \delta) \parallel (M_2 \oplus \delta).$$

The tuple is fresh because the only nonempty-message encryption used A_0 , whereas the query under A_1 returned an empty ciphertext. All message blocks are complete, so no nonempty padding suffix is checked. Every step is deterministic, so the forgery succeeds with probability one.

The two-block Espeon construction above uses the final input $M_2 \oplus M_1$ of Fig. 9. The C implementation retains the initial AD state in its checksum, so this exact ciphertext transformation does not transfer unchanged to that implementation. Section 6.1 summarizes this distinction and its validation.

4 Nonce-Respecting Attacks on Jolteon and Espeon

We now turn from the nonce-misusing adversaries of Section 3 to nonce-respecting (NR) attacks against the Eevee family [BPA⁺23b]. They exploit two distinct shortcomings in final-message processing: (i) Jolteon's length-independent final tweak combines with an XOR checksum whose removed-prefix contribution is publicly computable, allowing a transfer across ciphertext lengths; and (ii) the Espeon single-block branch computes the AD accumulator but does not inject it into the final forkcipher call.

In the original NR authenticity experiment, an adversary must not repeat a nonce among encryption queries; it may nonetheless ask its verification oracle to check a fresh

ciphertext under a nonce previously submitted to the encryption oracle. A forgery is fresh when its tuple (N, A, C, τ) is not an encryption-oracle response for that nonce and AD. Both constructions below use exactly one encryption query and one verification query. In particular, $q_e = q_v = \mu = 1$, where μ is the maximum number of times any nonce is used in encryption queries.

Attack 6. Jolteon cross-length forgery

Let n be the block size and let A be any fixed legal AD, including possibly $A = \varepsilon$. Since we consider nonempty messages, define

$$S = \begin{cases} 0^n, & A = \varepsilon, \\ \text{AD}(K, N, A, 0), & A \neq \varepsilon. \end{cases}$$

Thus S is precisely T_A in Algorithm 3; the adversary need not know S . Jolteon initializes $C_0 = S$ and $\Delta = 0^n$. For each nonfinal complete plaintext block M_i , Algorithm 3 updates

$$\Delta \leftarrow \Delta \oplus M_i \oplus C_{i-1}, C_i \leftarrow F_K^{N \parallel \langle i+1 \rangle_{d+1} \parallel 1, 0}(M_i), \quad (9)$$

where $d = t - |N| - 2$. For the final (possibly partial) block, it updates $\Delta \leftarrow \Delta \oplus C_m$ and invokes both branches of the forkcipher with tweak

$$T_f(N, \text{pad}_M) = N \parallel \langle \text{pad}_M \rangle_{d+1} \parallel 1.$$

The final tweak depends on N and on whether the final block is complete; it does not encode the number m of preceding complete blocks. In particular, the final calls for a one-block message and for a two-block message with complete final blocks use the same tweak $T_f(N, 1)$.

Choose arbitrary $M_1, M_2 \in \{0, 1\}^n$ and make the adversary's only encryption query $(C_1 \parallel C_2, \tau) \leftarrow \text{Jolteon.Enc}_K(N, A, M_1 \parallel M_2)$. The padding indicator is $\text{pad}_M = 1$ and Algorithm 3 parsing has one nonfinal block and one final block ($m = 1$, $M'_* = M_2$). For the first block, $C_0 = S$ and Δ becomes

$$\Delta_1 = 0^n \oplus M_1 \oplus C_0 = M_1 \oplus S,$$

while

$$C_1 = F_K^{T_1, 0}(M_1), T_1 = N \parallel \langle 2 \rangle_{d+1} \parallel 1.$$

In the final stage the additional $C_m = C_1$ contributes

$$\Delta_f = \Delta_1 \oplus C_1 = M_1 \oplus S \oplus C_1.$$

Consequently the final forkcipher input is

$$X = M_2 \oplus \Delta_f = M_2 \oplus M_1 \oplus S \oplus C_1,$$

and the two output branches and final ciphertext block satisfy

$$(\tau, Z) = F_K^{T_f, \text{b}}(X), C_2 = Z \oplus M_2, T_f = T_f(N, 1).$$

All values M_1, C_1, C_2, τ are known to the adversary except for the intermediate state S , input X , and branch output Z .

Define the publicly computable difference $D = M_1 \oplus C_1$ and the forged ciphertext

$$C' = C_2 \oplus D = C_2 \oplus M_1 \oplus C_1.$$

The adversary keeps the tag τ unchanged and submits the single verification query (N, A, C', τ) . Observe that the forgery has $|C'| = n$, whereas the only encryption response had ciphertext length $2n$.

Attack. For any valid forkcipher, key K , legal nonce N , legal associated data A , and blocks $M_1, M_2 \in \{0, 1\}^n$, let $(C_1 \parallel C_2, \tau)$ be the result of one encryption of $M_1 \parallel M_2$ under (N, A) . The fresh verification query

$$(N, A, C_2 \oplus M_1 \oplus C_1, \tau)$$

is always accepted by the Jolteon decryption algorithm, which outputs $M_2 \oplus M_1 \oplus C_1$. The adversary satisfies NR and has $q_e = q_v = \mu = 1$ and authenticity success probability exactly 1.

Analysis. We verify the entire decryption path. The forged ciphertext has exactly one complete n -bit block, so $m' = 0$, $C'_* = C' = C_2 \oplus M_1 \oplus C_1$, and $\text{pad}_C = 1$. As a result, no ordinary-block decryption iteration runs. The decryption initialization remains $C_0 = S$, while its checksum is initially 0^n and is finally updated with $C_{m'} = C_0$:

$$\Delta'_f = 0^n \oplus C_0 = S.$$

The forged final block is complete, so its tweak is

$$T'_f = N \parallel \langle 1 \rangle_{d+1} \parallel 1 = T_f.$$

The supplied tag is still τ , and hence the inverse forkcipher call at T'_f recovers exactly the same (X, Z) as the original two-block encryption. Substituting the forged checksum into the decryption formula gives

$$M' = X \oplus \Delta'_f = (M_2 \oplus M_1 \oplus S \oplus C_1) \oplus S = M_2 \oplus M_1 \oplus C_1.$$

For a complete final block, Algorithm 3 checks whether $\text{left}_n(M' \oplus Z) = C'$, with no nonempty padding suffix to validate. In fact,

$$\begin{aligned} M' \oplus Z &= M_2 \oplus M_1 \oplus C_1 \oplus Z = (M_2 \oplus Z) \oplus M_1 \oplus C_1 \\ &= C_2 \oplus M_1 \oplus C_1 = C'. \end{aligned}$$

Thus the check succeeds and $\text{Dec}_K(N, A, C', \tau) = M' \neq \perp$. The target is fresh because its ciphertext has length n , whereas the sole encryption response has length $2n$. A single encryption query satisfies NR.

The attack exploits a length-independent final tweak and an XOR checksum whose removed-prefix contribution is publicly computable. Tweak reuse alone is not the entire cause: the cancellation by $D = M_1 \oplus C_1$ preserves the final input and ciphertext consistency equation without knowledge of S . Figure 5 summarizes this transfer. The same final-input identity extends to an arbitrary complete prefix, as derived below.

$$\begin{array}{c}
M_1 \parallel M_2 \xrightarrow{\text{Enc}_K(N, A, \cdot)} (C_1 \parallel C_2, \tau) \\
D = M_1 \oplus C_1, \quad C' = C_2 \oplus D \\
(N, A, C', \tau) \xrightarrow{\text{Dec}_K} M_2 \oplus D \\
\text{Same final tweak and tag; the public } D \text{ removes the prefix checksum.}
\end{array}$$

Fig. 5: Jolteon’s NR cross-length forgery from one encryption. AD may be empty; no AD state is learned.

Extension: removal of an arbitrary complete prefix. Let $r \geq 1$ and suppose the Jolteon algorithm accepts $(r + 1)n$ -bit plaintexts with all required tweak counters defined. From a single encryption query

$$(C_1 \parallel \cdots \parallel C_{r+1}, \tau) \leftarrow \text{Jolteon.Enc}_K(N, A, M_1 \parallel \cdots \parallel M_{r+1})$$

with every $M_i \in \{0, 1\}^n$, define

$$D_r = \bigoplus_{i=1}^r (M_i \oplus C_i), C' = C_{r+1} \oplus D_r.$$

Then (N, A, C', τ) is a fresh NR forgery, accepted with probability 1, and decryption returns $M_{r+1} \oplus D_r$. **Derivation.** The nonfinal-loop recurrence in (9), with $C_0 = S$, gives

$$\begin{aligned}
\Delta_r &= \bigoplus_{i=1}^r (M_i \oplus C_{i-1}), \\
\Delta_f &= \Delta_r \oplus C_r = \left(\bigoplus_{i=1}^r M_i \right) \oplus S \oplus \left(\bigoplus_{i=1}^r C_i \right) = S \oplus D_r.
\end{aligned}$$

Thus the original final input is $X = M_{r+1} \oplus S \oplus D_r$, and its non-tag branch is Z with $C_{r+1} = Z \oplus M_{r+1}$. The proposed one-block ciphertext has a complete final block, so its final tweak is unchanged, its checksum is S , and inverse evaluation of τ recovers (X, Z) . The resulting message is $M' = X \oplus S = M_{r+1} \oplus D_r$, then

$$Z \oplus M' = (Z \oplus M_{r+1}) \oplus D_r = C_{r+1} \oplus D_r = C'.$$

The original ciphertext length is $(r + 1)n > n$, whereas the forged ciphertext length is n . There is only one encryption query.

Attack 7. Espeon short-message AD substitution

Let N be a legal nonce for Espeon and let A be any legal AD. From Algorithm 4, for nonempty M , the algorithm first computes $T_A = \text{AD}(K, N, A, 0)$ (or leaves $T_A = 0^n$ when $A = \varepsilon$) and initializes $\Delta = 0^n$. Choose $0 < |M| \leq n$. The (possibly padded)

message consists of $m = 0$ nonfinal blocks and a single last block M'_* . The ordinary Espeon message loop, which would otherwise inject T_A for its first iteration, is skipped. Hence $\Delta = 0^n$. The $m = 0$ branch selects the final tweak

$$T_f = N \parallel \langle 4 \rangle_d \parallel 1 \parallel \text{pad}_M, d = t - |N| - 2, \quad (10)$$

where $\text{pad}_M = 1$ for $|M| = n$ and $\text{pad}_M = 0$ otherwise. The Espeon algorithm then evaluates

$$(\tau, Z) = F_K^{T_f, b}(M'_* \oplus \Delta) = F_K^{T_f, b}(M'_*), \quad (11)$$

returning

$$C = \text{left}_{|M|}(Z \oplus M'_*). \quad (12)$$

Crucially, T_A occurs in neither (10) nor (11)–(12). It was computed but never used on this path. Thus both the ciphertext and its authentication tag depend only on K , N , the padded message, and its complete/partial flag, not on the associated data.

Lemma 3 (AD independence in the short-message branch). *For any fixed K, N , any valid nonempty M satisfying $0 < |M| \leq n$, and any legal AD values A_0, A_1 , the printed Espeon encryption algorithm satisfies*

$$\text{Espeon.Enc}_K(N, A_0, M) = \text{Espeon.Enc}_K(N, A_1, M). \quad (13)$$

Proof. The two computations have identical M'_* , pad_M , and T_f ; in both cases $\Delta = 0^n$ and $(\tau, Z) = F_K^{T_f, b}(M'_*)$. Their returned ciphertext–tag pairs are therefore equal, independently of their AD states.

This AD independence yields the following one-encryption NR AD-substitution forgery.

Attack. Assume the legal AD domain contains two distinct strings $A_0 \neq A_1$. For any legal nonce N and any M with $0 < |M| \leq n$, a nonce-respecting adversary can make one encryption query

$$(C, \tau) \leftarrow \text{Espeon.Enc}_K(N, A_0, M)$$

and submit (N, A_1, C, τ) as a fresh forgery. The printed Espeon decryption algorithm always accepts and recovers M . The advantage is 1 with $q_e = q_v = \mu = 1$.

Analysis. By Lemma 3, the ciphertext–tag pair (C, τ) is also an encryption of M under (N, A_1) . It is useful to verify this directly on the corresponding decryption path. Since $|C| = |M| \in (0, n]$, ciphertext parsing also yields $m = 0$ and $\text{pad}_C = \text{pad}_M$. Decryption may recompute a different $T'_A = \text{AD}(K, N, A_1, 0)$, but its ordinary loop is skipped, and its final accumulator remains $\Delta' = 0^n$. It therefore uses

$$T'_f = N \parallel \langle 4 \rangle_d \parallel 1 \parallel \text{pad}_C = T_f$$

with the unchanged tag τ . Inversion of the tag branch recovers (M'_*, Z) from (11), and the recovered message block is

$$M'_{\text{rec}} = M'_* \oplus \Delta' = M'_*.$$

The full-block or partial-block ciphertext consistency condition is satisfied because

$$\text{left}_{|M|}(M'_* \oplus Z) = C.$$

If $|M| < n$, the recovered M'_* has precisely the same 10^* padding as the original encryption, so the additional suffix test succeeds. Decryption returns the original M . The AD differs from the sole encryption query, so the record is fresh, and the one-query adversary satisfies NR.

Attack 6 and Attack 7 are authenticity counterexamples. Neither establishes an NR OAE confidentiality distinguisher. The C implementation changes Espeon's AD-state update, so Attack 7 does not transfer unchanged to its short-message path. Section 6.1 summarizes the checked source-level behavior.

5 Counterexamples to the Bounds and Their Proof Steps

The algorithms and resource definitions in the analyzed full version [BPA+23b] suffice to compare our attacks directly with Theorem 3.1. Its proof first replaces the forkcipher by two independent families of random tweakable permutations. Our identities hold in precisely this replacement model, where the primitive distinguishing term vanishes. The issue is therefore in the mode-level reasoning, independently of the concrete security of ForkSkinny.

5.1 Explicit contradictions within the resource limits

Table 2 substitutes the query resources into the bounds stated in Theorem 3.1. The symbol $\sigma_{m,e}$ counts message-processing primitive calls in encryption queries, not AD-processing calls. In particular, an AD-only encryption contributes zero to $\sigma_{m,e}$ while still contributing to the total primitive-query count σ . Likewise, $\sigma_{m,v}$ counts message-processing calls in the verification query. A two-block message contributes two calls; a one-block message contributes one.

Table 2: Mode-level bounds in the ideal-primitive model versus the constructed adversaries. The resource column gives $(q_e, q_v, \mu; \sigma_{m,e}, \sigma_{m,v})$. Message-call counts exclude AD; the total primitive-call count σ includes both AD and message processing.

Attack / mode	Resources	Claimed upper bound	Actual advantage
Attack 1 / U	(2, 0, 2; 3, 0)	$30/2^n$	$1 - 2^{-n}$
Attack 1 / E	(2, 0, 2; 3, 0)	$4/2^n + 9/2^{t-3}$	$1 - 2^{-n}$
Attack 2 / U	(4, 0, 4; 0, 0)	$72/2^n$	$1 - 2^{-n}$
Attack 2 / E	(4, 0, 4; 0, 0)	$24/2^n$	$1 - 2^{-n}$
Attack 3 / U	(3, 1, 3; 0, 0)	$52/2^n$	1
Attack 3 / E	(3, 1, 3; 0, 0)	$28/2^n$	1
Attack 6 / J	(1, 1, 1; 2, 1)	$4/2^n$	1
Attack 7 / E	(1, 1, 1; 1, 1)	$2^{3-t} + 2^{4-n}$	1

For example, the three-query Umbreon AD-only forgery yields

$$\text{Adv}_U^{\text{auth}} \leq \frac{3(0+3)(3-1)}{2^{n-1}} + \frac{1}{2^{n-4}} = \frac{52}{2^n},$$

whereas Attack 3 gives probability one. The other rows follow by the same substitution. All listed resources satisfy the original limits $\mu \leq 2^{n-1}$ and $\sigma_{m,v} \leq 2^{n-2}$ for the intended block sizes. The largest message-call counts in the table are $\sigma_{m,e} = 3$ and $\sigma_{m,v} = 1$. The attacks use representable counters, with $d \geq 3$ when two-block AD is required. The Espeon comparison uses its stated $t \in \{n, 2n\}$ and legal nonces. These are finite, small-resource counterexamples rather than asymptotic attacks outside the theorem's domain.

Online-prefix consistency is a separate proof obligation. The absence of repeated primitive input-tweak pairs is insufficient to establish OAE security: the ciphertext transcript must also satisfy the structural consistency constraints of the ideal online permutation. Attack 1 uses distinct final and ordinary tweaks. Their fresh random evaluations produce matching first blocks only with probability 2^{-n} , whereas the ideal OAE transcript requires a match with probability one. Thus, fresh primitive evaluations alone do not ensure the prefix consistency required by the ideal online permutation. This structural failure differs from the conditional-distribution error below: the AD attack reuses primitive values whose XOR determines a later response; the prefix attack fails even with fresh random evaluations at distinct tweaks.

5.2 AD-only tags: the joint transcript in Appendix C, Eq. (7)

Appendix C, part B of [BPA⁺23b], attempts to bound the contribution of an AD-only encryption query by comparing its tag with a uniform string, and then by a permutation-versus-function comparison leading to Eq. (7). This reasoning does not account for the conditional distribution of a multi-block XOR accumulator after previous related AD queries.

For the fixed four AD strings in Attack 2, the fourth real tag obeys

$$\Pr[\tau_{11} = \tau_{00} \oplus \tau_{01} \oplus \tau_{10} \mid \tau_{00}, \tau_{01}, \tau_{10}] = 1$$

for every realizable first-three-tag transcript. In the original ideal OAE game, the fourth tag is indexed by a distinct (N, A, M) and remains uniform independently of the first three, so the corresponding conditional probability is 2^{-n} . The conditional statistical distance for this fourth response is therefore $1 - 2^{-n}$.

There is no contradiction between this fact and uniform marginal distributions of individual real tags. For independent ideal permutations at the two distinct AD tweaks, each $P(X) \oplus Q_1(Y)$ is individually uniform. The flaw is replacing the conditional distribution of an XOR of previously reused primitive outputs by the distribution of a fresh single permutation evaluation. Non-duplicate AEAD queries need not contain fresh primitive evaluations; all four contributions to the fourth tag have already occurred in the first three queries.

5.3 Message-domain separation: Appendix C, Observation 1

Appendix C, part C.2, Observation 1 of [BPA⁺23b], asserts that AD-only queries do not help distinguish a subsequent nonempty-message query because *noM* separates the final AD tweaks. Distinct final tweaks do not establish independence of the complete AD states across an adaptive transcript.

Lemma 2 gives a direct counterexample to that independence inference:

$$\mathcal{H}_0(X_0 \parallel Y) \oplus \mathcal{H}_0(X_1 \parallel Y) = \mathcal{H}_1(X_0 \parallel Y) \oplus \mathcal{H}_1(X_1 \parallel Y).$$

The final outputs cancel separately within each domain, exposing the shared prefix difference. Thus the two AD-only oracle responses determine a relation between secret states used by a later nonempty-message query. The individual $\mathcal{H}_0$ states need not be revealed or have nonuniform marginals. The nonempty-message forgeries in Attacks 4 and 5 show how this relation defeats authenticity. They do not rely on treating the complete ciphertexts of related plaintexts as independent random strings.

5.4 Authenticity: checksum freshness in Section 7, Case 1

The integrity argument in Section 7 of [BPA⁺23b], Case 1, infers from a new nonce–AD–nonfinal-ciphertext tuple that at least one corresponding checksum block is fresh, independent, and randomly sampled. Attack 6 invalidates that implication even with one NR encryption query.

The sole two-block encryption has one nonfinal ciphertext block. The forged one-block ciphertext has no nonfinal block, so its nonce–AD–prefix tuple is new. Nevertheless, its final tweak is unchanged. Inverting the old tag consequently returns the old final pair (X, Z) ; removing the publicly known checksum contribution $M_1 \oplus C_1$ leaves a deterministically valid shorter record. The new AEAD record has generated neither a fresh final primitive input nor a fresh branch output. Applying a fresh-output probability estimate to this verification query is therefore unjustified.

Attack 7 is a separate data-flow counterexample: changing AD yields a fresh record without changing any message primitive call. Together these examples show why an encryption-only pseudorandomness estimate cannot supply the claimed authenticity addend without an additional argument about the actual verification computation. The claimed bounds fail; this does not imply that OAE confidentiality generally entails authenticity, nor that the NR authenticity attacks themselves recover a key or distinguish NR encryption transcripts.

6 Discussion, Repair Requirements, and Limitations

The shared XOR-based AD transform has deterministic relations across repeated-nonce queries. The AD-free two-query prefix distinguisher and the independent four-point AD relation contradict the NM confidentiality bounds of Umbreon and Espeon, and interpolation and cross-domain differences give deterministic forgeries. Jolteon’s finalization additionally admits an NR cross-length forgery, while the Espeon short-message

branch omits AD binding. Section 5 identifies the corresponding online-prefix consistency, conditional-distribution, and checksum-freshness failures. None of these counterexamples requires a weak underlying forkcipher.

The online ideal object. Message processing and finalization must be reconciled with the ideal oracle: its longest complete-block prefix includes a complete final block. For fixed (N, A) , related messages must induce the prescribed shared prefix. Distinct tweaks and independently uniform outputs do not alone ensure this consistency. Any alternative ideal object must be stated explicitly and its security proved; changing the model does not restore the original Theorem 3.1.

AD processing. A replacement must hide the joint relations exposed by AD-only queries and prevent reuse of those relations in the nonempty-message domain. Changing only the last AD-block tweak does not eliminate shared-prefix differences. A domain-separated variable-input PRF over an injective encoding of (N, A, noM) is a candidate component, but its interaction with the online message layer and its IoT/MPC cost require fresh analysis. Merely applying a final permutation to an XOR accumulator is not, by itself, a proof that all relevant collision and cross-domain attacks disappear.

Finalization and state binding. Jolteon’s final authentication computation must prevent the demonstrated transfer between different message lengths. Incorporating an appropriate length or final-position identifier into a collision-free final domain would block this particular equality of tweaks, but would not constitute a proof of full authenticity. Espeon’s message paths must bind AD for every permitted length. In particular, computing the AD accumulator is insufficient when a control-flow branch bypasses every use of it. Any revision must analyze the recovered state along the entire decryption path, including partial-last-block checks, and exclude cancellation between its state updates.

6.1 Implementation observations and validation

We also checked the published C source at commit `a8bb0731` [COS]. Jolteon’s cross-length transfer and the shared AD relations reproduce in the tested builds. The C Espeon state update differs from the specification: it binds AD on the short-message path, but for messages longer than one block the same AD state is used in both the first-block mask and the initial checksum. On decryption, these two occurrences cancel after the first block, allowing substitution of AD while retaining the ciphertext and tag. This observation supports the need to analyze state binding across all message lengths; it does not change the scope of the specified-mode attacks.

Separate source-level faults include final-AD conditional-operator precedence, overlapping ordinary and final AD tweaks in Espeon (giving a 145-byte AD collision), and Umbreon/Jolteon counter wrap within accepted AD lengths. These require unambiguous encodings, disjoint tweak domains, and checked counter ranges. They supplement the algorithm analysis and do not account for the design-level counterexamples above.

Our local checks compiled the unmodified supplied routines across six instances and both ForkSkinny backends. The checks reproduced the constructed forgeries and AD relations, rejected the C Espeon short-message substitution controls, and passed

normal-decryption and incorrect-tag rejection controls. Appendix C identifies the analyzed material. The host tests corroborate the reported source behavior for the tested inputs; the algorithmic identities and ideal-model analysis establish the main results.

Scope and remaining work. The algorithm-level conclusions concern the supplied full-version specification, identified in Appendix C. They demonstrate violations of specific OAE distribution and authenticity claims, rather than key recovery. The privacy advantages are ideal-forkcipher statements and are not estimated from concrete prefix observations. The source checks concern the pinned C routines and host builds; they do not establish behavior of ARM firmware, MPC deployments, or other wrappers. Unresolved byte differences between the two 64-bit backends were observed in our local validation and are not used as evidence for the attacks. A revised construction requires a reconciled specification and a new security proof, followed by conformance and regression checks.

References

- BN00. Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *Advances in Cryptology - ASIACRYPT 2000*, volume 1976, pages 531–545, 2000. 1
- BPA⁺23a. Amit Singh Bhati, Erik Pohle, Aysajan Abidin, Elena Andreeva, and Bart Preneel. Let’s go eevee! A friendly and suitable family of AEAD modes for iot-to-cloud secure computation. In *CCS 2023*, pages 2546–2560, 2023. 2
- BPA⁺23b. Amit Singh Bhati, Erik Pohle, Aysajan Abidin, Elena Andreeva, and Bart Preneel. Let’s go Eevee! A friendly and suitable family of AEAD modes for IoT-to-cloud secure computation. Cryptology ePrint Archive, Paper 2023/1361, <https://eprint.iacr.org/2023/1361>, 2023. Full version analyzed; Fig. 9, Theorem 3.1, Section 7 and Appendices C–D. 2, 4, 5, 6, 12, 17, 18, 19, 21, 28
- COS. COSIC, KU Leuven. Eevee: The Eevee family of AEAD modes for IoT-Friendly encryption and MPC-Friendly decryption. Public source repository. The supplied source audit pins commit a8bb0731a75c6fd04e7a3f909513b9d9778e1017. 20
- FFL12. Ewan Fleischmann, Christian Forler, and Stefan Lucks. Mcoe: A family of almost foolproof on-line authenticated encryption schemes. In *Fast Software Encryption, FSE 2012*, volume 7549, pages 196–215, 2012. 1, 4
- HRRV15. Viet Tung Hoang, Reza Reyhanitabar, Phillip Rogaway, and Damian Vizár. On-line authenticated-encryption and its nonce-reuse misuse-resistance. In *Advances in Cryptology - CRYPTO 2015*, volume 9215, pages 493–517, 2015. 1, 4
- Rog02. Phillip Rogaway. Authenticated-encryption with associated-data. In *CCS 2002*, pages 98–107, 2002. 1

A Algorithm of Eevee

The following procedures separate the color-coded algorithm in Fig. 9 of the analyzed full version [BPA⁺23b]. Every member encryption procedure returns (C, τ) , including (ε, T_A) for an empty message; the AD helper itself returns only T_A . The common AD routine receives *noM* explicitly instead of inheriting it from its caller. A successful

AD-only verification is written as returning ε instead of $\top$; both denote acceptance. For a complete last block, the padding-suffix test is vacuous. This makes explicit the intended full-block case of Fig. 9, whose literal 10^{n-1-res_C} expression is otherwise undefined when $\text{res}_C = n$. We preserve the final input and tweak expressions, including the absence of the AD state in Espeon’s short-message path.

Algorithm 1 Common Eevee padding and associated-data processing

```

    Pad( $n, X$ )  $\triangleright X \neq \varepsilon$ 
1:    $\text{res}_X \leftarrow |X| \bmod n; \text{pad}_X \leftarrow 1$ 
2:   if  $\text{res}_X \neq 0$  then
3:      $\text{pad}_X \leftarrow 0; X \leftarrow X \parallel 10^{n-\text{res}_X-1}$ 
4:   else
5:      $\text{res}_X \leftarrow n$ 
6:   return  $(X, \text{pad}_X, \text{res}_X)$ 

    SplitLast( $X$ )  $\triangleright n \mid |X|$ 
7:    $h \leftarrow |X|/n - 1$ 
8:   Parse  $X = X_1 \parallel \dots \parallel X_h \parallel X'_*$ , where  $|X_i| = |X'_*| = n$ 
9:   return  $(X_1, \dots, X_h, X'_*)$ 

     $f_T(t, A, B)$   $\triangleright$  Espeon only
10:  return  $\text{left}_{t-2}(B \parallel A)$ 

    AD( $K, N, A, \text{noM}$ )  $\triangleright A \neq \varepsilon$ 
11:   $(A, \text{pad}_A, \text{res}_A) \leftarrow \text{Pad}(n, A)$ 
12:   $(A_1, \dots, A_a, A'_*) \leftarrow \text{SplitLast}(A)$ 
13:   $T_A \leftarrow 0^n; d \leftarrow t - |N| - 2$ 
14:  for  $i \leftarrow 1$  to  $a$  do
15:     $T \leftarrow N \parallel \langle i+4 \rangle_d \parallel 00$ 
16:     $T_A \leftarrow T_A \oplus F_K^{T,0}(A_i)$ 
17:   $T \leftarrow N \parallel \langle 2\text{pad}_A + \text{noM} \rangle_d \parallel 00$ 
18:   $T_A \leftarrow T_A \oplus F_K^{T,0}(A'_*)$ 
19:  return  $T_A$ 
  
```

Algorithm 2 Umbreon encryption and decryption

```

    Umbreon.Enc( $K, N, A, M$ )
1:   if  $|A| = |M| = 0$  then
2:     return  $\perp$ 
3:    $\text{noM} \leftarrow [|M| = 0]; T_A, \Delta \leftarrow 0^n, 0^n$ 
4:   if  $A \neq \varepsilon$  then
5:      $T_A \leftarrow \text{AD}(K, N, A, \text{noM})$ 
6:   if  $M = \varepsilon$  then
7:     return  $(\varepsilon, T_A)$ 
8:    $(M, \text{pad}_M, \text{res}_M) \leftarrow \text{Pad}(n, M)$ 
9:    $(M_1, \dots, M_m, M'_*) \leftarrow \text{SplitLast}(M)$ 
10:  for  $i \leftarrow 1$  to  $m$  do
11:     $T \leftarrow N \parallel \langle i+1 \rangle_{d+1} \parallel 1; \Delta \leftarrow \Delta \oplus T_A$ 
12:     $(C_i, T_A) \leftarrow F_K^{T,b}(M_i \oplus T_A)$ 
13:   $T \leftarrow N \parallel \langle \text{pad}_M \rangle_{d+1} \parallel 1$ 
14:   $\Delta \leftarrow \Delta \oplus T_A$ 
  
```

```

15:  $(\tau, Z) \leftarrow F_K^{T,b}(M'_* \oplus \Delta)$ 
16:  $C_* \leftarrow \text{left}_{\text{res}_M}(Z \oplus M'_*)$ 
17: return  $(C_1 \parallel \dots \parallel C_m \parallel C_*, \tau)$ 

Umbreon.Dec( $K, N, A, C, \tau$ )
18: if  $|A| = |C| = 0$  then
19:   return  $\perp$ 
20:  $noM \leftarrow [|C| = 0]; T_A, \Delta \leftarrow 0^n, 0^n$ 
21: if  $A \neq \varepsilon$  then
22:    $T_A \leftarrow \text{AD}(K, N, A, noM)$ 
23: if  $C = \varepsilon$  then
24:   if  $\tau = T_A$  then
25:     return  $\varepsilon$ 
26:   else
27:     return  $\perp$ 
28:  $(C, \text{pad}_C, \text{res}_C) \leftarrow \text{Pad}(n, C)$ 
29:  $(C_1, \dots, C_m, C'_*) \leftarrow \text{SplitLast}(C)$ 
30:  $C_* \leftarrow \text{left}_{\text{res}_C}(C'_*)$ 
31: for  $i \leftarrow 1$  to  $m$  do
32:    $T \leftarrow N \parallel \langle i+1 \rangle_{d+1} \parallel 1; \Delta \leftarrow \Delta \oplus T_A$ 
33:    $(M_i, U) \leftarrow F_K^{-1,T,0,b}(C_i)$ 
34:    $M_i \leftarrow M_i \oplus T_A; T_A \leftarrow U$ 
35:    $T \leftarrow N \parallel \langle \text{pad}_C \rangle_{d+1} \parallel 1; \Delta \leftarrow \Delta \oplus T_A$ 
36:    $(X, Z) \leftarrow F_K^{-1,T,0,b}(\tau); M'_* \leftarrow X \oplus \Delta$ 
37:   if  $\text{CheckLast}(M'_*, Z, C_*, \text{res}_C) = 0$  then
38:     return  $\perp$ 
39:   return  $M_1 \parallel \dots \parallel M_m \parallel \text{left}_{\text{res}_C}(M'_*)$ 

```

Algorithm 3 Jolteon encryption and decryption

```

Jolteon.Enc( $K, N, A, M$ )
1: if  $|A| = |M| = 0$  then
2:   return  $\perp$ 
3:  $noM \leftarrow [|M| = 0]; T_A, \Delta \leftarrow 0^n, 0^n$ 
4: if  $A \neq \varepsilon$  then
5:    $T_A \leftarrow \text{AD}(K, N, A, noM)$ 
6: if  $M = \varepsilon$  then
7:   return  $(\varepsilon, T_A)$ 
8:  $(M, \text{pad}_M, \text{res}_M) \leftarrow \text{Pad}(n, M)$ 
9:  $(M_1, \dots, M_m, M'_*) \leftarrow \text{SplitLast}(M)$ 
10:  $C_0 \leftarrow T_A$ 
11: for  $i \leftarrow 1$  to  $m$  do
12:    $T \leftarrow N \parallel \langle i+1 \rangle_{d+1} \parallel 1$ 
13:    $\Delta \leftarrow \Delta \oplus M_i \oplus C_{i-1}$ 
14:    $C_i \leftarrow F_K^{T,0}(M_i)$ 
15:    $T \leftarrow N \parallel \langle \text{pad}_M \rangle_{d+1} \parallel 1$ 
16:    $\Delta \leftarrow \Delta \oplus C_m$ 
17:    $(\tau, Z) \leftarrow F_K^{T,b}(M'_* \oplus \Delta)$ 
18:    $C_* \leftarrow \text{left}_{\text{res}_M}(Z \oplus M'_*)$ 
19:   return  $(C_1 \parallel \dots \parallel C_m \parallel C_*, \tau)$ 

```

```

Jolteon.Dec( $K, N, A, C, \tau$ )
20:  if  $|A| = |C| = 0$  then
21:    return  $\perp$ 
22:     $noM \leftarrow [|C| = 0]$ ;  $T_A, \Delta \leftarrow 0^n, 0^n$ 
23:    if  $A \neq \varepsilon$  then
24:       $T_A \leftarrow \text{AD}(K, N, A, noM)$ 
25:    if  $C = \varepsilon$  then
26:      if  $\tau = T_A$  then
27:        return  $\varepsilon$ 
28:      else
29:        return  $\perp$ 
30:     $(C, \text{pad}_C, \text{res}_C) \leftarrow \text{Pad}(n, C)$ 
31:     $(C_1, \dots, C_m, C'_*) \leftarrow \text{SplitLast}(C)$ 
32:     $C_* \leftarrow \text{left}_{\text{res}_C}(C'_*)$ 
33:     $C_0 \leftarrow T_A$ 
34:    for  $i \leftarrow 1$  to  $m$  do
35:       $T \leftarrow N \parallel \langle i+1 \rangle_{d+1} \parallel 1$ 
36:       $M_i \leftarrow F_K^{-1, T, 0, i}(C_i)$ 
37:       $\Delta \leftarrow \Delta \oplus M_i \oplus C_{i-1}$ 
38:     $T \leftarrow N \parallel \langle \text{pad}_C \rangle_{d+1} \parallel 1$ 
39:     $\Delta \leftarrow \Delta \oplus C_m$ 
40:     $(X, Z) \leftarrow F_K^{-1, T, 0, b}(\tau)$ ;  $M'_* \leftarrow X \oplus \Delta$ 
41:    if  $\text{CheckLast}(M'_*, Z, C_*, \text{res}_C) = 0$  then
42:      return  $\perp$ 
43:    return  $M_1 \parallel \dots \parallel M_m \parallel \text{left}_{\text{res}_C}(M'_*)$ 

```

Algorithm 4 Espeon encryption and decryption

```

Espeon.Enc( $K, N, A, M$ )
1:  if  $|A| = |M| = 0$  then
2:    return  $\perp$ 
3:     $noM \leftarrow [|M| = 0]$ ;  $T_A, \Delta \leftarrow 0^n, 0^n$ 
4:    if  $A \neq \varepsilon$  then
5:       $T_A \leftarrow \text{AD}(K, N, A, noM)$ 
6:    if  $M = \varepsilon$  then
7:      return  $(\varepsilon, T_A)$ 
8:    if  $|N| > n - 2$  then
9:      return  $\perp$ 
10:    $(M, \text{pad}_M, \text{res}_M) \leftarrow \text{Pad}(n, M)$ 
11:    $(M_1, \dots, M_m, M'_*) \leftarrow \text{SplitLast}(M)$ 
12:    $C_0 \leftarrow N \parallel 0^{n-|N|}$ 
13:   for  $i \leftarrow 1$  to  $m$  do
14:      $\Delta \leftarrow \Delta \oplus M_i$ 
15:     if  $i = 1$  then
16:        $T \leftarrow N \parallel \langle 4 \rangle_d \parallel 00$ 
17:        $C_i \leftarrow F_K^{T, 0}(T_A \oplus M_i)$ 
18:     else
19:        $T \leftarrow f_T(t, C_{i-2}, C_{i-1}) \parallel 01$ 
20:        $C_i \leftarrow F_K^{T, 0}(M_i)$ 

```

```

21:   if  $m = 0$  then
22:      $T \leftarrow N \parallel \langle 4 \rangle_d \parallel 1 \parallel \text{pad}_M$ 
23:   else
24:      $T \leftarrow f_T(t, C_{m-1}, C_m) \parallel 1 \parallel \text{pad}_M$ 
25:    $(\tau, Z) \leftarrow F_K^{T,b}(M'_* \oplus \Delta)$ 
26:    $C_* \leftarrow \text{left}_{\text{res}_M}(Z \oplus M'_*)$ 
27:   return  $(C_1 \parallel \dots \parallel C_m \parallel C_*, \tau)$ 

```

Espeon.Dec(K, N, A, C, τ)

```

28:   if  $|A| = |C| = 0$  then
29:     return  $\perp$ 
30:    $noM \leftarrow [|C| = 0]; T_A, \Delta \leftarrow 0^n, 0^n$ 
31:   if  $A \neq \varepsilon$  then
32:      $T_A \leftarrow \text{AD}(K, N, A, noM)$ 
33:   if  $C = \varepsilon$  then
34:     if  $\tau = T_A$  then
35:       return  $\varepsilon$ 
36:     else
37:       return  $\perp$ 
38:   if  $|N| > n - 2$  then
39:     return  $\perp$ 
40:    $(C, \text{pad}_C, \text{res}_C) \leftarrow \text{Pad}(n, C)$ 
41:    $(C_1, \dots, C_m, C'_*) \leftarrow \text{SplitLast}(C)$ 
42:    $C_* \leftarrow \text{left}_{\text{res}_C}(C'_*)$ 
43:    $C_0 \leftarrow N \parallel 0^{n-|N|}$ 
44:   for  $i \leftarrow 1$  to  $m$  do
45:     if  $i = 1$  then
46:        $T \leftarrow N \parallel \langle 4 \rangle_d \parallel 00$ 
47:        $M_i \leftarrow F_K^{-1,T,0,i}(C_i) \oplus T_A$ 
48:     else
49:        $T \leftarrow f_T(t, C_{i-2}, C_{i-1}) \parallel 01$ 
50:        $M_i \leftarrow F_K^{-1,T,0,i}(C_i)$ 
51:      $\Delta \leftarrow \Delta \oplus M_i$ 
52:   if  $m = 0$  then
53:      $T \leftarrow N \parallel \langle 4 \rangle_d \parallel 1 \parallel \text{pad}_C$ 
54:   else
55:      $T \leftarrow f_T(t, C_{m-1}, C_m) \parallel 1 \parallel \text{pad}_C$ 
56:    $(X, Z) \leftarrow F_K^{-1,T,0,b}(\tau); M'_* \leftarrow X \oplus \Delta$ 
57:   if  $\text{CheckLast}(M'_*, Z, C_*, \text{res}_C) = 0$  then
58:     return  $\perp$ 
59:   return  $M_1 \parallel \dots \parallel M_m \parallel \text{left}_{\text{res}_C}(M'_*)$ 

```

CheckLast(X, Z, C_*, ρ)

$\triangleright$ common to all modes; $1 \leq \rho \leq n$

```

60:    $B \leftarrow \text{left}_\rho(X \oplus Z)$ 
61:   if  $\rho < n$  then
62:      $\text{suffix} \leftarrow X[\rho + 1 : n]$ 
63:   else
64:      $\text{suffix} \leftarrow \varepsilon$ 
65:   return  $[B = C_* \wedge (\rho = n \vee \text{suffix} = 10^{n-\rho-1})]$ 

```

B Algorithm-Level Attack Procedures

The seven procedures below follow the attack numbering in the main text. For the AD attacks, choose $d = t - |N| - 2 \geq 3$ and satisfy the member-specific nonce restrictions. Every procedure receives the encryption oracle $\mathcal{O}_{\text{Enc}}$ and, when needed, the verification oracle $\mathcal{O}_{\text{Ver}}$ defined in Section 2.1; oracle calls do not give the adversary access to K . Attacks 1–5 repeat an encryption nonce; Attacks 6 and 7 use only one encryption query.

Algorithm 5 Attack 1: Two-query AD-free NM OAE prefix distinguisher

$\mathcal{D}_{\text{prefix}} \mathcal{O}_{\text{Enc}}$

- 1: Choose legal N and full blocks $X, Y \in \{0, 1\}^n$
- 2: $(C, \tau) \leftarrow \mathcal{O}_{\text{Enc}}(N, \varepsilon, X)$
- 3: $(C'_1 \parallel C'_2, \tau') \leftarrow \mathcal{O}_{\text{Enc}}(N, \varepsilon, X \parallel Y)$
- 4: **return** $[C \neq C'_1]$ ▷ one denotes the real world

Algorithm 6 Attack 2: Four-query nonce-misuse OAE distinguisher

$\mathcal{D}_{\text{rect}} \mathcal{O}_{\text{Enc}}$

- 1: Fix legal N and choose $X_0 \neq X_1, Y_0 \neq Y_1$ in $\{0, 1\}^n$
- 2: **for** $i \leftarrow 0$ **to** 1 **do**
- 3: **for** $j \leftarrow 0$ **to** 1 **do**
- 4: $(\varepsilon, \tau_{ij}) \leftarrow \mathcal{O}_{\text{Enc}}(N, X_i \parallel Y_j, \varepsilon)$
- 5: $R \leftarrow \tau_{00} \oplus \tau_{01} \oplus \tau_{10} \oplus \tau_{11}$
- 6: **return** $[R = 0^n]$ ▷ one denotes the real world

Algorithm 7 Attack 3: Three-query deterministic AD-only forgery

$\mathcal{A}_{\text{AD}} \mathcal{O}_{\text{Enc}}, \mathcal{O}_{\text{Ver}}$

- 1: Choose legal $N, X_0 \neq X_1$, and $Y_0 \neq Y_1$ in $\{0, 1\}^n$
- 2: $(\varepsilon, \tau_{00}) \leftarrow \mathcal{O}_{\text{Enc}}(N, X_0 \parallel Y_0, \varepsilon)$
- 3: $(\varepsilon, \tau_{01}) \leftarrow \mathcal{O}_{\text{Enc}}(N, X_0 \parallel Y_1, \varepsilon)$
- 4: $(\varepsilon, \tau_{10}) \leftarrow \mathcal{O}_{\text{Enc}}(N, X_1 \parallel Y_0, \varepsilon)$
- 5: $\tau^* \leftarrow \tau_{00} \oplus \tau_{01} \oplus \tau_{10}$
- 6: **return** $\mathcal{O}_{\text{Ver}}(N, X_1 \parallel Y_1, \varepsilon, \tau^*)$

Algorithm 8 Attack 4: Three-encryption nonempty NM forgery: Umbreon/Jolteon

$\mathcal{A}_{\text{UJ}} \mathcal{O}_{\text{Enc}}, \mathcal{O}_{\text{Ver}}$

- 1: Choose legal N , full AD blocks $X_0 \neq X_1, Y$, and $M \in \{0, 1\}^n$
- 2: $A_0 \leftarrow X_0 \parallel Y; A_1 \leftarrow X_1 \parallel Y$
- 3: $(\varepsilon, \tau_0) \leftarrow \mathcal{O}_{\text{Enc}}(N, A_0, \varepsilon)$
- 4: $(\varepsilon, \tau_1) \leftarrow \mathcal{O}_{\text{Enc}}(N, A_1, \varepsilon)$
- 5: $\delta \leftarrow \tau_0 \oplus \tau_1$
- 6: $(C, \tau) \leftarrow \mathcal{O}_{\text{Enc}}(N, A_0, M)$
- 7: $C' \leftarrow C \oplus \delta$
- 8: **return** $\mathcal{O}_{\text{Ver}}(N, A_1, C', \tau)$

Algorithm 9 Attack 5: Three-encryption nonempty NM forgery: Espeon

$\mathcal{A}_{\text{E}} \mathcal{O}_{\text{Enc}}, \mathcal{O}_{\text{Ver}}$

- 1: Choose legal N , full AD blocks $X_0 \neq X_1, Y$, and $M_1, M_2 \in \{0, 1\}^n$
- 2: $A_0 \leftarrow X_0 \parallel Y; A_1 \leftarrow X_1 \parallel Y$
- 3: $(\varepsilon, \tau_0) \leftarrow \mathcal{O}_{\text{Enc}}(N, A_0, \varepsilon)$
- 4: $(\varepsilon, \tau_1) \leftarrow \mathcal{O}_{\text{Enc}}(N, A_1, \varepsilon)$
- 5: $\delta \leftarrow \tau_0 \oplus \tau_1$
- 6: $(C_1 \parallel C_2, \tau) \leftarrow \mathcal{O}_{\text{Enc}}(N, A_0, M_1 \parallel M_2)$
- 7: $C'_2 \leftarrow C_2 \oplus \delta$
- 8: **return** $\mathcal{O}_{\text{Ver}}(N, A_1, C_1 \parallel C'_2, \tau)$

Algorithm 10 Attack 6: One-encryption nonce-respecting authenticity forgery for Jolteon

$\mathcal{A}_{\text{J}}^{\text{nr}} \mathcal{O}_{\text{Enc}}, \mathcal{O}_{\text{Ver}}$

- 1: Choose any legal nonce N , any legal AD A , and blocks $M_1, M_2 \in \{0, 1\}^n$
- 2: $(C_1 \parallel C_2, \tau) \leftarrow \mathcal{O}_{\text{Enc}}(N, A, M_1 \parallel M_2)$
- 3: $D \leftarrow M_1 \oplus C_1$
- 4: $C' \leftarrow C_2 \oplus D$
- 5: **return** $\mathcal{O}_{\text{Ver}}(N, A, C', \tau)$ ▷ accepts with probability 1

Algorithm 11 Attack 7: One-encryption NR AD-substitution forgery against Espeon

$\mathcal{A}_{\text{E}}^{\text{nr}} \mathcal{O}_{\text{Enc}}, \mathcal{O}_{\text{Ver}}$

- 1: Choose legal nonce N , distinct legal A_0, A_1 , and M with $0 < |M| \leq n$
- 2: $(C, \tau) \leftarrow \mathcal{O}_{\text{Enc}}(N, A_0, M)$
- 3: **return** $\mathcal{O}_{\text{Ver}}(N, A_1, C, \tau)$ ▷ accepts with probability 1

C Analyzed Material and Author Validation

Specification. All algorithm and proof references concern the supplied 20-page full version identified as ePrint 2023/1361 [BPA+23b], including its Fig. 9 and Theorem 3.1. The reviewed PDF has SHA-256 fingerprint

031392a79d1a1db439637b394e7d3b19eaf47f7640e90d9b2c7cd87a556aed28. This identifies the analyzed bytes rather than an unspecified revision at the ePrint URL.

Source and supporting checks. The implementation summary in Section 6.1 concerns repository commit

<https://github.com/KULeuven-COSIC/Eevee/commit/a8bb0731a75c6fd04e7a3f909513b9d9778e1017>. The supplied copies of `umbreon_core.c`, `jolteon_core.c`, and `espeon.c` have, respectively, the SHA-256 fingerprints

```
8308799ad3e47ccae323b47904566e677db34e42dc6f4fc5b8deefc1ecec042c
9879fb496869cfba7f7bdf011fad3e119ee2dd72c0efa680291132f18fce55e6
a938ca0a5385ea212e6afbd4c7170b348e21891163e823abe9f1fbd20361b084.
```

Author validation. The authors conducted local validation of the implementation observations reported in Section 6.1 on 15 October 2026. The unmodified routines were compiled across six instances and both ForkSkinny backends. The validation comprised 532 top-level cases, including the constructed forgeries and AD relations, normal-decryption checks, incorrect-tag rejection controls, and repeated executions where applicable. These finite executions supplement the source analysis and do not estimate the ideal-forkciphper advantages of Attacks 1 and 2. The C nonce sizes are fixed by the instances and are not substituted for the specified algorithms’ nonce or tweak parameters. Unresolved byte differences between the two 64-bit backends were observed but are not used as evidence for the attacks. None of the host executions constitutes a test of ARM firmware or an MPC deployment. The validation code and generated artifacts are not included in this version.

Reproduction. From the supplied workspace root, the checks can be rerun with

```
python3 eevee-code-validation/run_validation.py --backend both
python3 eevee-code-validation/verify_results.py
```

The report gives detailed input coverage, acceptance results, and unresolved byte differences between the two 64-bit backends. None of the host executions is a test of ARM firmware or an MPC deployment.