This article is an amalgamation of prior work of the same authors. Concretely, it combines the contents of eprint articles 2018/1040 (by Inoue and Minematsu [16]), 2018/1087 (by Poettering [31]), and 2018/1090 (by Iwata [20]) that appeared in November 2018 on closely related topics into a single edited document. This article should be seen as a successor of all three eprint articles. An extended abstract appears in the proceedings of CRYPTO 2019 [15] and is available via DOI 10.1007/978-3-030-26948-7_1 from the Springer website. This is the full version. It will appear in Journal of Cryptology and is available as entry 2019/311 in the IACR eprint archive.

Cryptanalysis of OCB2: Attacks on Authenticity and Confidentiality

Akiko Inoue[1], Tetsu Iwata[2], Kazuhiko Minematsu[1], and Bertram Poettering[3]

[1] NEC Corporation, Kawasaki, Japan,
a_inoue@nec.com, k-minematsu@nec.com
[2] Nagoya University, Nagoya, Japan, tetsu.iwata@nagoya-u.jp
[3] IBM Research – Zurich, Switzerland, poe@zurich.ibm.com

Abstract. We present practical attacks on OCB2. This mode of operation of a blockcipher was designed with the aim to provide particularly efficient and provably-secure authenticated encryption services, and since its proposal about 15 years ago it belongs to the top performers in this realm. OCB2 was included in an ISO standard in 2009.
An internal building block of OCB2 is the tweakable blockcipher obtained by operating a regular blockcipher in XEX* mode. The latter provides security only when evaluated in accordance with certain technical restrictions that, as we note, are not always respected by OCB2. This leads to devastating attacks against OCB2's security promises: We develop a range of very practical attacks that, amongst others, demonstrate universal forgeries and full plaintext recovery. We complete our report with proposals for (provably) repairing OCB2. As a direct consequence of our findings, OCB2 is currently in a process of removal from ISO standards. Our attacks do not apply to OCB1 and OCB3, and our privacy attacks on OCB2 require an active adversary.

Keywords: OCB2, Authenticated Encryption, Cryptanalysis, Forgery, Plaintext Recovery, XEX

1 Introduction

Authenticated encryption (AE) is a form of symmetric-key encryption that simultaneously protects the confidentiality and authenticity of messages. The primitive is widely accepted as a fundamental tool in practical cryptography, finding application in many settings, including in SSH and TLS.

Constructions of the AE primitive include the OCB family of blockcipher modes of operation. Its three members (OCB1, OCB2, OCB3) are celebrated for their beautiful and innovative architecture, and their almost unrivaled efficiency. In fact, the modes are fully parallelizable and thus effectively as efficient as the fastest known confidentiality-only modes. The first version (OCB1) was proposed at ACM CCS 2001 by Rogaway et al. [38], the second version (OCB2) at ASIACRYPT 2004 by Rogaway [34] (hereafter Rog04), and the third version (OCB3) at FSE 2011 by Krovetz and Rogaway [23]. While all three designs share roughly the same construction principles, differences to note include both the external interface (while OCB1 is a pure AE mode, its successors OCB2 and OCB3 are AEAD modes where encryption and decryption is performed with respect to an auxiliary associated-data input) and a core internal building block (while OCB1 and OCB3 are driven by look-up tables, OCB2 relies on the so-called powering-up construction).

Each version of OCB has received significant attention from researchers, standardization bodies, and the industry. In particular, OCB1 is listed in the IEEE 802.11 standard as an option for the protection of wireless networks, OCB2 was included in the ISO/IEC 19772:2009 [18] standard, and OCB3 is specified in document RFC 7253 [24] as an IETF Internet standard. Moreover, OCB3 is included in the final

portfolio of the CAESAR competition[4]. Various versions of OCB have been implemented in popular cryptographic libraries, including in Botan, BouncyCastle, LibTomCrypt, OpenSSL, and SJCL.

The security of (all versions of) OCB has been extensively studied. For each version, the designer(s) provided security reductions to the security of the underlying blockcipher, with additive birthday-bound tightness of roughly the form $O(\sigma^2/2^n)$, where σ indicates the number of processed blocks (message and associated data) and n is the block size of the cipher. Note that this bound formally becomes pointless if $\sigma = 2^{n/2}$ blocks are involved, and indeed Ferguson [12] and Sun et al. [41] showed collision attacks that get along with this many processed blocks, implying that the bound is tight. (The attacks do not seem to be practical, though, as they require processing 300 EB (exabytes) of data with a single key, assuming $n = 128$.) As discussed below, all further known attacks against the members of the OCB family are in relaxed security settings (e.g. involving nonce misuse), with the conclusion being that their security is widely believed to hold (up to the birthday bound, in classic security models).

In this article we invalidate this belief by presenting a series of attacks against OCB2. The most basic attack requires one encryption and one decryption (of short messages and ciphertexts, respectively) to create an existential forgery with success probability one. No heavy computation or large amount of memory is needed for this; rather, performing a couple of XOR computations is sufficient. The attack is independent of the blockcipher with which OCB2 is operated, including of its key length and its block length. Further, the message to which the forged ciphertext decrypts is strongly dependent on the message involved in the initial encryption query, so that most parts of it can be assumed to be known to, or influenced by, the adversary. Extended versions of our attack achieve forgeries for arbitrary messages (with full control over nonces and associated data), and full plaintext recovery, at the expense of a slight increase in the number of required encryption and decryption queries. Long story short: Our attacks on OCB2 are as critical as attacks on AE schemes can be.

We turn to technical details of our attacks. All members of the OCB family can be seen as modes of operation of a tweakable blockcipher (TBC, [25]): The message to be encrypted is chunked into blocks, and each message block is enciphered independently of the others using a tweak that reflects the position of the block in the message. Special tweaking rules are deployed for the last (possibly padded) message block and the checksum used for tag generation. In OCB2, the tweakable blockcipher itself is derived from an underlying regular blockcipher (e.g. AES) using the XEX* transform. The latter is a hybrid of XE ("XOR-encipher", $C = E_K(\Delta \oplus M)$) and XEX ("XOR-encipher-XOR", $C = \Delta \oplus E_K(\Delta \oplus M)$) where it can be decided on a per-evaluation basis which of the two is used. The flaw of OCB2 that we identify and exploit is located neither in the general method with which the AEAD scheme is constructed from a tweakable blockcipher nor in the XEX* primitive. The problem is rather hiding in the interplay between the former and a technical peculiarity of the latter: If XEX* is ever evaluated twice on the same input but in different modes (XE vs. XEX), it gives up on all security promises. While the corresponding access rule was already identified as necessary by Rog04, it was overlooked that OCB2 actually does not always satisfy it. Indeed, as we expose in this paper, an attacker can arrange that an XEX evaluation occurring when encrypting a regular message block and an XE evaluation occurring when decrypting a (padded) last block of an *unauthentic* ciphertext are on the same inputs. This issue, that was overlooked by the cryptographic community for the past 15 years, not only devalidates the formal security argument for OCB2 but ultimately leads to attacks that completely break the security of this primitive. While, as we prove, OCB2 can be fixed by replacing certain XE invocations by XEX invocations (with an associated cost of one additional XOR operation per encryption/decryption operation), the fixed version unfortunately loses backward compatibility with unmodified OCB2 implementations. We finally note that OCB1 and OCB3 do not combine the XE and XEX modes in the way OCB2 does, and we did not find them vulnerable to our attacks.

As our attacks are technical and fairly complex, we confirmed their effectiveness by implementing them: For our most relevant attacks we have C code that breaks the OCB2 reference implementation[5] with the reported high efficiency and success rate. As an example, a part of our code for the minimal forgery attack (Sec. 4.1) is shown in Appendix E.

[4] `https://competitions.cr.yp.to/caesar.html`

[5] by Krovetz, `http://web.cs.ucdavis.edu/~rogaway/ocb/code-2.0.htm`

1.1 Impact

OCB2 was standardized in ISO/IEC 19772:2009 about a decade ago [18]. As the scheme offers exceptional performance that is challenging to rival by purely AES-based constructions, it has to be assumed that industry widely picked up on it, ultimately incorporating the scheme into products. The consequences of our findings thus might be severe. We have been in contact with members of ISO/IEC SC 27 Working Group 2, which is responsible for the standard, to advise on the right interpretation of our results. The working group has issued a document [19] that acknowledges our findings and makes it clear that OCB2 should no longer be used. Moves are nearing completion to remove the scheme from the international standard.[6]

OCB2 was and possibly still is covered by Intellectual Property claims. While such claims don't necessarily manifest a noticeable obstacle for deployment in industry, for open source software development efforts they routinely are. As a consequence, a number of relevant open source crypto libraries do not have an implementation of OCB2 and are thus not affected by our findings. (An exception to this is Stanford's SJCL library[7], and we have communicated our findings to the SJCL team.) The lack of open implementations suggests that most affected parties have industrial background. By the very nature of (IND$ secure) encryption, spotting closed-source products that rely on OCB2 for security and now became vulnerable remains a challenge.

Some More Details on the Real-World Use of OCB2. As mentioned earlier, there are some software libraries that implement OCB. Specifically we found SJCL, OpenSSL, Botan, Bouncy Castle, Libgcrypt, and LibTomCrypt. However, with exception of SJCL, these libraries do not seem to support OCB2 but instead either OCB1 or OCB3.

Joplin[8] is a multi-platform application for taking notes. It uses OCB2 through SJCL according to the thread on the developer site on github[9]. In this thread, the developer of Joplin acknowledges to be aware of the use of OCB2 but communicates to have decided to wait for the reaction of the SJCL team.

1.2 Further Related Work

We already mentioned the attacks of Ferguson [12] and Sun et al. [41] that indicate the tightness of the birthday-bound claims for OCB. In scenarios where OCB is deployed in a somewhat sloppy way, e.g. where nonces are repeated (nonce-misuse setting) or where message fragments emerging from partially decrypted ciphertexts are leaked (release of unverified plaintext setting), attacks are identified by Andreeva et al. [1] and Ashur et al. [3]. Vaudenay and Vizár [42] studied all third-round CAESAR candidates, including OCB3 but not OCB2.

With the goal of better understanding the security of the OCB schemes, Aoki and Yasuda [2] show that relaxed assumptions on the underlying blockcipher are sufficient to obtain positive results. Note that our attacks are in conflict with their claims, indicating that some of their arguments have to be reconsidered; the authors of [2] confirmed this view to us.

Attacks in the *reforgeability setting* [8,13] deliver a series of existential forgeries with the specific property that creating the first forgery is the hardest part. In most cases the hardness notion is based on computation time. Also our attacks can be seen in the reforgeability setting, but with a different complexity measure: While the first OCB2 forgery is only existential and requires two queries (one encryption, one decryption), from just one more encryption query one can create hundreds of independent universal forgeries.

1.3 Organization and Contributions

We recall notions of tweakable blockciphers and authenticated encryption in Sec. 2. After specifying the OCB2 algorithms in Sec. 3, we present simple authenticity and confidentiality attacks against them

[6] ISO document Draft Amendment ISO/IEC 19772:2009/DAM 1:2019 lists OCB2 as a deprecated scheme. The document is currently available at `https://www.iso.org/standard/77459.html`.

[7] `http://bitwiseshiftleft.github.io/sjcl/`

[8] `https://joplinapp.org/`

[9] `https://github.com/laurent22/joplin/issues/943`

in Sec. 4. While these attacks achieve overwhelming advantages with respect to formal notions of unforgeability and indistinguishability, and thus make evident that OCB2 is *academically* broken, certain restrictions on the format of forged or distinguished messages remain. We hence develop, in Sec. 5 and Sec. 6, a set of advanced attacks (including raw blockcipher access, universal forgery, and arbitrary decryption) that break the scheme also in *real-world* settings. In Sec. 7 we explore which technical component of OCB2 is responsible for its insecurity; as many other schemes in symmetric cryptography use structures similar to those of OCB2, these reflections might also guide future cryptanalysis efforts. In Sec. 8 we survey the applicability of our attack strategies to related encryption modes, including to OCB1 and OCB3; however we do not identify any further weaknesses. Finally, in Sec. 9 we consider approaches to repair OCB2.

2 Preliminaries

2.1 Notation

If A is a finite set we write $a \xleftarrow{\$} A$ for the operation of picking an element of A uniformly at random and assigning it to the variable a. If B, B' are set variables we write $B \xleftarrow{\cup} B'$ as shorthand for $B \leftarrow B \cup B'$.

Strings and Padding. Let $\{0,1\}^*$ be the set of all binary strings, including the empty string ε. The bit length of $X \in \{0,1\}^*$ is denoted by $|X|$, and in particular we have $|\varepsilon| = 0$. We write $\{0,1\}^{\leq n}$ for $\bigcup_{l \in \{0,\ldots,n\}} \{0,1\}^l$, where $\{0,1\}^0 = \{\varepsilon\}$. The sequence of n zeros is denoted with 0^n, with the convention that $0^0 = 1^0 = \varepsilon$. The concatenation of two bit strings X and Y is written $X \,\|\, Y$, or XY when no confusion is possible. The XOR combination of two same-length bit strings X, Y is denoted $X \oplus Y$. We denote with $\mathtt{msb}_c(X)$ and $\mathtt{lsb}_c(X)$ the first and last $c \leq |X|$ bits of X, respectively.

For $X \in \{0,1\}^{\leq n}$ we define the zero padding, written as $X \,\|\, 0^*$, and the one-zero padding, written as $X \,\|\, 10^*$, as follows: If $|X| = n$ we let $X \,\|\, 0^* = X \,\|\, 10^* = X$, i.e., the padding is trivial. If $|X| < n$, we let $X \,\|\, 0^* = X \,\|\, 0^{n-|X|}$ and $X \,\|\, 10^* = X \,\|\, 10^{n-|X|-1}$.

For $X \in \{0,1\}^*$, the parsing of X into n-bit blocks is denoted by

$$(X[1], X[2], \ldots, X[m]) \xleftarrow{n} X,$$

where $m = |X|_n \stackrel{\text{def}}{=} \lceil |X|/n \rceil$ and $X[1] \,\|\, X[2] \,\|\, \ldots \,\|\, X[m] = X$ and $|X[i]| = n$ for $1 \leq i < m$ and $0 < |X[m]| \leq n$ when $|X| > 0$. When $|X| = 0$, we let $m = 1$ and $X[1] \leftarrow \varepsilon$.

2.2 (Tweakable) Blockciphers and Finite Fields

A tweakable blockcipher (TBC) [25] is a keyed function $\widetilde{E}\colon \mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \mathcal{M}$ such that for each $(K,T) \in \mathcal{K} \times \mathcal{T}$, the partial function $\widetilde{E}(K,T,\cdot)$ is a permutation of $\mathcal{M}$. Here, K is the key and T is a public value called tweak, and typically we have $\mathcal{M} = \{0,1\}^n$ where n is called the block length. (It is safe to assume $n = 128$ from here on.) A conventional blockcipher is a TBC where $\mathcal{T}$ consists of a singleton and is written as $E\colon \mathcal{K} \times \mathcal{M} \to \mathcal{M}$. The enciphering of $X \in \mathcal{M}$ under key $K \in \mathcal{K}$ and tweak $T \in \mathcal{T}$ is denoted, equivalently, $\widetilde{E}(K,T,X)$ or $\widetilde{E}_K(T,X)$ or $\widetilde{E}_K^T(X)$. For blockciphers we correspondingly write $E(K,X)$ or $E_K(X)$. The deciphering is written as $\widetilde{E}_K^{-1,T}(Y)$ for TBCs and $E_K^{-1}(Y)$ for blockciphers. For any $K \in \mathcal{K}$ and $T \in \mathcal{T}$, when $Y = \widetilde{E}_K^T(X)$ we have $\widetilde{E}_K^{-1,T}(Y) = X$.

When the key K used with a blockcipher or TBC invocation is obvious from the context, we may omit writing it. Moreover, for a mode of operation that uses a keyed blockcipher instance E_K in a black-box manner, we may treat E_K, instead of K, as the key (and correspondingly for a TBC $\widetilde{E}$).

Security Notions. Consider a TBC of the form $\widetilde{E}\colon \mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \mathcal{M}$. A tweakable uniform random permutation (TURP) for sets $\mathcal{T}, \mathcal{M}$ is an information-theoretic TBC that behaves like uniformly picked from all $\mathcal{T}$-tweaked permutations over $\mathcal{M}$ (i.e., like a uniformly picked function $f\colon \mathcal{T} \times \mathcal{M} \to \mathcal{M}$ such that $f(T,\cdot)$ is a permutation of $\mathcal{M}$ for all $T \in \mathcal{T}$). We denote TURP instances for $\widetilde{E}$ with $\widetilde{\mathsf{P}}$.

We define the Tweakable Pseudorandom Permutation (TPRP) advantage and the Tweakable Strong PRP (TSPRP) advantage of an adversary $\mathcal{A}$ as follows:

$$\mathbf{Adv}^{\mathsf{tprp}}_{\widetilde{E}}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr\left[\mathcal{A}^{\widetilde{E}_K} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\widetilde{\mathsf{P}}} \Rightarrow 1\right]$$
$$\mathbf{Adv}^{\mathsf{tsprp}}_{\widetilde{E}}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr\left[\mathcal{A}^{\widetilde{E}_K, \widetilde{E}_K^{-1}} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\widetilde{\mathsf{P}}, \widetilde{\mathsf{P}}^{-1}} \Rightarrow 1\right]$$

Here, the probabilities are over the uniform choice of key $K \in \mathcal{K}$ or $\widetilde{\mathsf{P}}$, and the randomness of the adversary. The adversaries perform chosen-plaintext attacks and chosen-ciphertext attacks, respectively, in both cases with chosen tweaks. (That is, they can query any (T, X) in the enciphering direction and any (T, Y) in the deciphering direction (if applicable), with freely chosen tweak T.)

For blockciphers $E\colon \mathcal{K} \times \mathcal{M} \to \mathcal{M}$ we analogously define the PRP advantage $\mathbf{Adv}^{\mathsf{prp}}_{E}(\mathcal{A})$ and SPRP advantage $\mathbf{Adv}^{\mathsf{sprp}}_{E}(\mathcal{A})$, using a URP P as information-theoretic reference point. (A URP uniformly distributes over all permutations over $\mathcal{M}$.)

Galois Fields. Following [34,21], bit strings $a \in \{0,1\}^n$ can be considered elements of $\mathrm{GF}(2^n)$, assuming a representation of the latter with a polynomial basis and seeing the individual bits of a as polynomial coefficients. The strings $0^{n-2}10$ and $0^{n-2}11$ correspond with the polynomials '$\mathtt{x}$' and '$\mathtt{x}+1$', and we denote these field elements with '2' and '3', respectively. It is common to refer to the multiplication of a field element with 2 (read: $\mathtt{x}$) as *doubling*. For instance, $2^i a$ denotes i-times doubling a. Standard calculation rules (for fields) apply; in particular we have $3a = 2a \oplus a$ and $2^i 3a = 3(2^i a) = 2^{i+1}a \oplus 2^i a$ for all i.

OCB2 is based on a blockcipher, and in the spirit of the above it considers the latter's domain $\mathcal{M} = \{0,1\}^n$ a Galois field. Concretely, a fixed block length of $n = 128$ is assumed (which matches that of AES), and as the (irreducible) reduction polynomial of the $\mathrm{GF}(2^n)$ representation the lexicographically-first *primitive* polynomial is used, which is $\mathtt{x}^{128} + \mathtt{x}^7 + \mathtt{x}^2 + \mathtt{x} + 1$. This choice implies that all non-zero elements of $\mathrm{GF}(2^n)$ are (cyclically) obtained by continuously doubling the element 2, and further that the doubling mapping $a \mapsto 2a$ can be efficiently implemented as $\mathtt{lsb}_n(a \ll 1)$ if $\mathtt{msb}_1(a) = 0$ and $\mathtt{lsb}_n(a \ll 1) \oplus (0^{120}10000111)$ if $\mathtt{msb}_1(a) = 1$, where $(a \ll 1)$ denotes the left-shift of a by one bit position. See [34] for more details on this representation.

2.3 AE and AEAD

For simplicity we refer with the term AE to both: schemes implementing (pure) Authenticated Encryption and schemes implementing Authenticated Encryption with Associated Data (AEAD) [33]. An AE scheme $\Pi = (\mathcal{E}, \mathcal{D})$ is defined over a key space $\mathcal{K}$, a nonce space $\mathcal{N}$, an associated data (AD) space $\mathcal{A}$, a message space $\mathcal{M}$, and a tag space $\mathcal{T} = \{0,1\}^\tau$ for some fixed tag length τ.[10] Here, AD is a part of the input to the encryption and decryption algorithms that is not encrypted but must be authenticated, typically encoding some kind of context information. Formally, the AEAD encryption algorithm is a function $\mathcal{E}\colon \mathcal{K}\times\mathcal{N}\times\mathcal{A}\times\mathcal{M} \to \mathcal{M}\times\mathcal{T}$, and the decryption algorithm is a function $\mathcal{D}\colon \mathcal{K}\times\mathcal{N}\times\mathcal{A}\times\mathcal{M}\times\mathcal{T} \to \mathcal{M}\cup\{\perp\}$, where symbol $\perp \notin \mathcal{M}$ is used to report verification failures.

To encrypt plaintext M with nonce N and associated data A under key K, compute $(C, T) \leftarrow \mathcal{E}_K(N, A, M)$ to produce ciphertext C and tag T. The tuple (N, A, C, T) is communicated to the receiver and the original message M recovered by computing $\mathcal{D}_K(N, A, C, T)$.

Security Notions. The security of AE is typically captured with two notions: privacy and authenticity. Following the definitions of [6,36], authenticity requires that ciphertexts (including nonce, associated data, and tag) cannot be manipulated or forged, while privacy requires that ciphertexts (including the tag) cannot be distinguished. More precisely, while [36, Sec. 3] defines privacy as the inability of a passive adversary (that cannot pose decryption queries) to distinguish ciphertext-tag pairs from random strings, [36, Sec. 6] gives a second definition that formalizes privacy against active adversaries (that can pose decryption queries). As noted in [36, Sec. 6], if authenticity is provided by a scheme, the two privacy notions are equivalent. Since the current article considers an AE scheme that does *not* provide

[10] We do not employ a dedicated symbol for the ciphertext space but instead use the symbol $\mathcal{M}$ for both messages and ciphertexts.

authenticity, we emphasize that for this scheme the equivalence of the two notions cannot be assumed, and in fact it does not hold. We correspondingly reproduce the two definitions separately.

We formalize privacy against passive attacks with a pair of games where a nonce-respecting adversary interacts with an oracle that is called on inputs (N, A, M) and either implements a keyed AEAD instance that returns the ciphertext $(C, T) = \mathcal{E}_K(N, A, M)$, or implements a random-bits oracle \$ that returns a uniformly random string of length $|M| + \tau$. Here, by writing nonce-respecting we mean that the adversary uses distinct nonces in its encryption queries. The privacy advantage of a nonce-respecting adversary $\mathcal{A}$ is defined as

$$\mathbf{Adv}^{\mathsf{priv}}_{\Pi}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr\left[\mathcal{A}^{\mathcal{E}_K(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot,\cdot)} \Rightarrow 1\right].$$

Privacy against active adversaries is defined similarly, but with an added decryption oracle that the adversary may query on arbitrary tuples (N, A, C, T) except those where (C, T) was returned by a $\mathcal{E}_K(N, A, \cdot)$ or $\$(N, A, \cdot)$ query before. There is no restriction on nonces for the decryption queries, and in particular nonces can be replayed from prior queries.[11] The corresponding advantage definition is

$$\mathbf{Adv}^{\mathsf{priv\text{-}cca}}_{\Pi}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr\left[\mathcal{A}^{\mathcal{E}_K(\cdot,\cdot,\cdot),\mathcal{D}_K(\cdot,\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot,\cdot),\mathcal{D}_K(\cdot,\cdot,\cdot,\cdot)} \Rightarrow 1\right].$$

With respect to the authenticity notion, we deem adversaries $\mathcal{A}$ with access to $\mathcal{E}_K$ and $\mathcal{D}_K$ oracles successful if they are effective with creating forgeries. Formally, the authenticity advantage is defined as

$$\mathbf{Adv}^{\mathsf{auth}}_{\Pi}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr\left[\mathcal{A}^{\mathcal{E}_K(\cdot,\cdot,\cdot),\mathcal{D}_K(\cdot,\cdot,\cdot,\cdot)} \text{ forges}\right],$$

where $\mathcal{A}$ forges if it receives a value $M \neq \perp$ from the $\mathcal{D}_K$ oracle, conditioned on it being nonce-respecting and not querying tuples (N, A, C, T) to the $\mathcal{D}_K$ oracle if it made a query (N, A, M) to $\mathcal{E}_K$ with result (C, T) before.

3 The OCB2 Mode of Operation

The OCB2 authenticated encryption scheme was first described in [34].[12] Like its predecessor OCB1 it is fully parallelizable and rate-1 (requiring one blockcipher invocation per message block), but it replaces a table-driven component of OCB1 with the 'powering-up' construction to compute a sequence of XEX masks by continuously doubling them. Further, in [34, Sec. 11] OCB2 was first defined as an AE mode without AD, and subsequently extended to an AEAD mode (then dubbed AEM). When AD is empty, the algorithm of AEM is identical to (AD-less) OCB2. When AD is non-empty, the tag is an XOR of a MAC of the AD and the tag of (AD-less) OCB2. The related PMAC construction [9] was identified as a particularly interesting option for processing the AD as it would allow sharing its blockcipher instance with that of the OCB2 encryption core.

Our specification of OCB2 is taken from [35, Fig. 3] and supports associated data.[13] The mode's key space $\mathcal{K}$ is that of the underlying blockcipher E, the latter is required to have block length $n = 128$ (in particular, AES is suitable), the nonce space is $\mathcal{N} = \{0,1\}^n$, the message space $\mathcal{M}$ and the AD space $\mathcal{A}$ are the set $\{0,1\}^*$ of strings of arbitrary length, and the tag space is $\mathcal{T} = \{0,1\}^\tau$ for any fixed parameter $\tau \leq n$. Note that, as mentioned in Sec. 2, we may simply write E to denote a keyed blockcipher instance $E_K \colon \{0,1\}^n \to \{0,1\}^n$.

The OCB2 algorithms $\mathcal{E}_E$ and $\mathcal{D}_E$ are detailed in Fig. 1. The algorithms are further illustrated in Fig. 2. In the code, for $X \in \{0,1\}^{\leq n}$, expression $\texttt{len}(X)$ denotes an n-bit encoding of $|X|$, $\text{PMAC}_E(A)$ denotes the PMAC of A computed with the (keyed) blockcipher instance E, and the field operations are with respect to the $\text{GF}(2^n)$ setup described in Sec. 2.2. The details of functions `len` and PMAC are not relevant for our attacks, so we omit their description here. (For completeness we reproduce them in Appendix B.)

[11] We clarify that the PRIV-CCA notion does not imply the AUTH notion, and in particular not AE. To see this, modify any PRIV-CCA secure scheme by augmenting the ciphertext space by one additional ciphertext that always decrypts to some fixed message, independently of the used key, nonce, and associated data. This modified scheme provides PRIV-CCA but not AUTH.

[12] In that paper the mode was actually referred to as OCB1; what we call OCB1 was referred to as OCB in [34].

[13] The PMAC version from [35] is slightly different from the initial version [9] in that it uses doublings for mask generation and is adapted to be computationally independent from the encryption part when combined with OCB2.

Algorithm $\mathcal{E}_E(N, A, M)$

1. $L \leftarrow E(N)$
2. $(M[1], \ldots, M[m]) \xleftarrow{n} M$
3. **for** $i \leftarrow 1$ **to** $m-1$
4. $\quad C[i] \leftarrow 2^i L \oplus E(2^i L \oplus M[i])$
5. $\text{Pad} \leftarrow E(2^m L \oplus \texttt{len}(M[m]))$
6. $C[m] \leftarrow M[m] \oplus \texttt{msb}_{|M[m]|}(\text{Pad})$
7. $\Sigma \leftarrow C[m] \,\|\, 0^* \oplus \text{Pad}$
8. $\Sigma \leftarrow M[1] \oplus \cdots \oplus M[m-1] \oplus \Sigma$
9. $T \leftarrow E(2^m 3L \oplus \Sigma)$
10. **if** $A \neq \varepsilon$ **then** $T \leftarrow T \oplus \text{PMAC}_E(A)$
11. $T \leftarrow \texttt{msb}_\tau(T)$
12. **return** (C, T)

Algorithm $\mathcal{D}_E(N, A, C, T)$

1. $L \leftarrow E(N)$
2. $(C[1], \ldots, C[m]) \xleftarrow{n} C$
3. **for** $i \leftarrow 1$ **to** $m-1$
4. $\quad M[i] \leftarrow 2^i L \oplus E^{-1}(2^i L \oplus C[i])$
5. $\text{Pad} \leftarrow E(2^m L \oplus \texttt{len}(C[m]))$
6. $M[m] \leftarrow C[m] \oplus \texttt{msb}_{|C[m]|}(\text{Pad})$
7. $\Sigma \leftarrow C[m] \,\|\, 0^* \oplus \text{Pad}$
8. $\Sigma \leftarrow M[1] \oplus \cdots \oplus M[m-1] \oplus \Sigma$
9. $T^* \leftarrow E(2^m 3L \oplus \Sigma)$
10. **if** $A \neq \varepsilon$ **then** $T^* \leftarrow T^* \oplus \text{PMAC}_E(A)$
11. $T^* \leftarrow \texttt{msb}_\tau(T^*)$
12. **if** $T = T^*$ **return** M
13. **else return** $\perp$

Fig. 1. Algorithms of OCB2. See Appendix B for the specifications of functions `len` and PMAC. Blockcipher E is implicitly parameterized with the AEAD key.

4 Basic Attacks on Authenticity and Confidentiality

We show that OCB2 provides neither authenticity nor confidentiality by specifying attacks against the formal definitions of these notions. We start with a minimal attack on unforgeability that gets along with a single encryption query to produce an existential forgery with probability 1. This attack, while effective, is rather limited with respect to the choice of involved parameters like message length and tag length. We thus proceed with giving a more general version that extends the basic attack in terms of these parameters. We then focus on the confidentiality of OCB2 and observe that our attacks against authenticity *effectively* also break the privacy of OCB2.

The attacks considered here neither produce universal forgeries nor serve for decrypting arbitrary ciphertexts. These more powerful attacks are described in Sec. 6.

4.1 Minimal Forgery Attack

We give the minimal example of our forgery attacks against OCB2. For simplicity, assume $\tau = n$, i.e., that tags have maximum length. Note that the attack is independent of both the AD processing function (PMAC) and the details of the length encoding function `len`. The following steps of our attack are also illustrated in Fig. 3 and specified in pseudocode in Fig. 4 (left).

1. Encrypt (N, A, M) where N is any nonce, $A = \varepsilon$ is empty, and M is the $2n$-bit message $M = M[1] \,\|\, M[2]$ where
$$M[1] = \texttt{len}(0^n)$$
and $M[2]$ is any n-bit block. The encryption oracle returns a pair (C, T) consisting of a $2n$-bit ciphertext $C = C[1] \,\|\, C[2]$ and a tag T.
2. Decrypt (N', A', C', T') with $|C'| = n$ such that
$$\begin{aligned} N' &= N, \\ A' &= \varepsilon, \\ C' &= C[1] \oplus \texttt{len}(0^n) \\ T' &= M[2] \oplus C[2] \end{aligned} \tag{1}$$

Note that $C' \neq C$ (as they have different lengths), so we have a successful forgery if (N', A', C', T') is indeed accepted by the decryption algorithm. To see that this is the case, observe first that by the encryption algorithm we have

$$\begin{aligned} C[1] &= 2L \oplus E(2L \oplus \texttt{len}(0^n)) \\ C[2] &= M[2] \oplus \text{Pad}, \end{aligned} \tag{2}$$

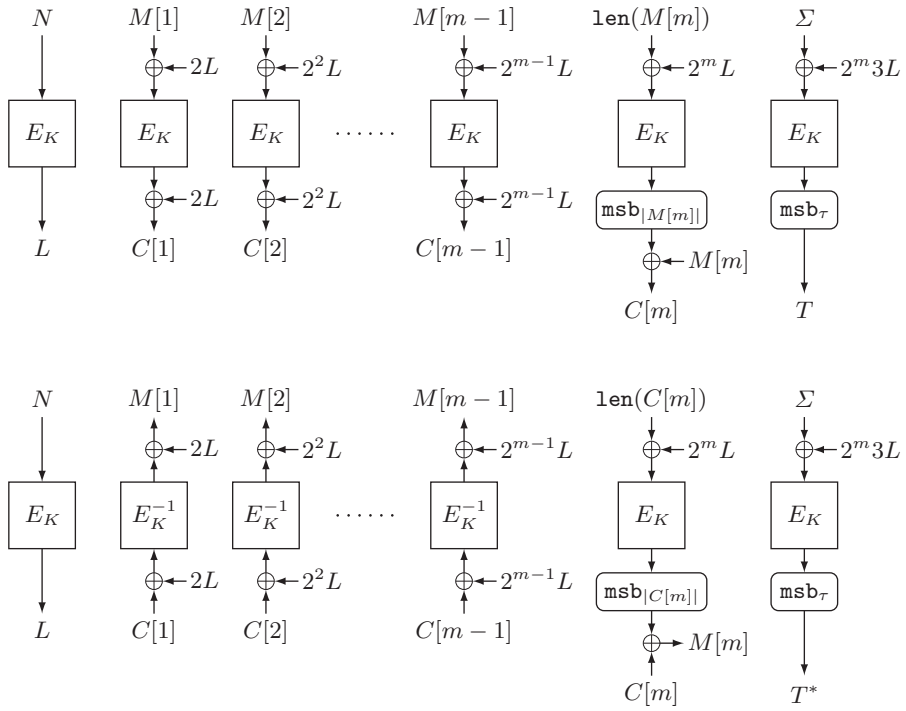

Fig. 2. OCB2 encryption and decryption in the case of empty AD.

where $L = E(N)$ and $\text{Pad} = E(2^2L \oplus \texttt{len}(0^n))$. Let Pad' and Σ' be the intermediate values computed during decryption. Then C' is decrypted to

$$\begin{aligned} M' &= C' \oplus \text{Pad}' \\ &= C' \oplus E(2L \oplus \texttt{len}(0^n)) \\ &= C[1] \oplus \texttt{len}(0^n) \oplus E(2L \oplus \texttt{len}(0^n)) \\ &= 2L \oplus E(2L \oplus \texttt{len}(0^n)) \oplus \texttt{len}(0^n) \oplus E(2L \oplus \texttt{len}(0^n)) \\ &= 2L \oplus \texttt{len}(0^n), \end{aligned}$$

and the tag is recovered as

$$\begin{aligned} T^* &= E(2 \cdot 3L \oplus \Sigma') \\ &= E(2 \cdot 3L \oplus C' \oplus \text{Pad}') \\ &= E(2 \cdot 3L \oplus M') \\ &= E(2 \cdot 3L \oplus 2L \oplus \texttt{len}(0^n)) \\ &= E(2^2L \oplus \texttt{len}(0^n)) \qquad (3) \\ &= \text{Pad} \\ &= T', \qquad (4) \end{aligned}$$

where (3) follows from the identity $2 \cdot 3L = 2^2L \oplus 2L$ and (4) follows from (1) and (2). The conclusion is: We have $T^* = T'$ and thus tuple (N', A', C', T') is (falsely) accepted as an authentic ciphertext. This breaks the authenticity of OCB2.

4.2 Forgeries for Longer Messages

The attack of Sec. 4.1 can be generalized, without increasing the number of encryption or decryption queries, to allow forging ciphertexts for arbitrarily long messages. The generalized attack further drops the requirement $A = \varepsilon$ for the encryption query, and relaxes the $\tau = n$ requirement on the tag length.

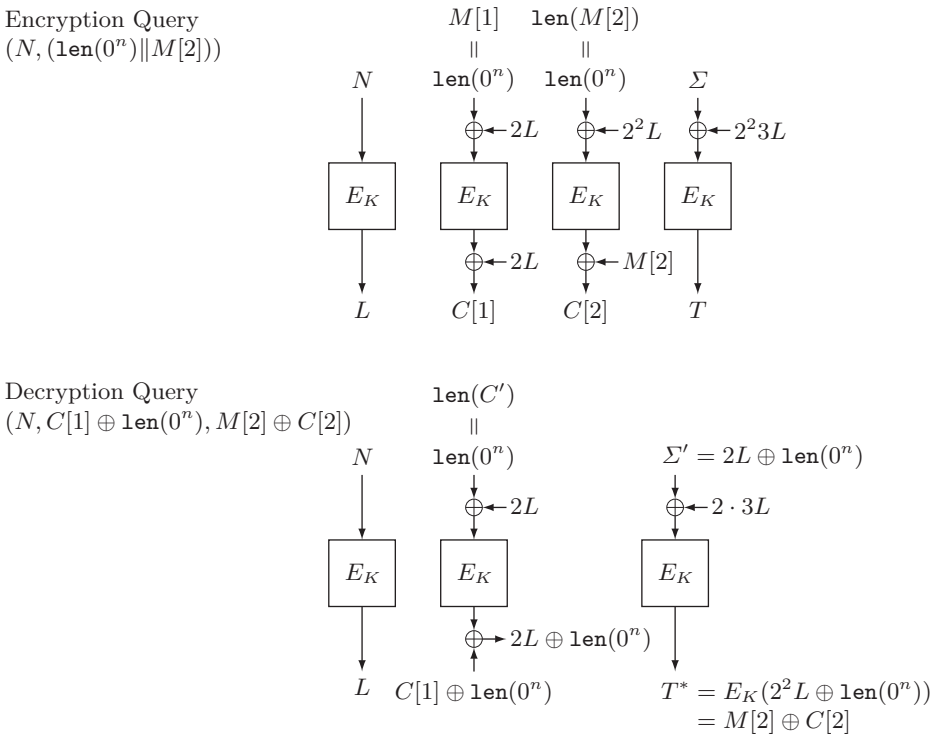

Fig. 3. Minimal forgery attack (see Sec. 4.1).

Intuitively, the latter two improvements are possible because A and τ exclusively influence the tag T returned by the encryption algorithm, and because the attack of Sec. 4.1 is independent of this value.

The attack described below requires encrypting a message $M = M[1] \| \cdots \| M[m]$ with only one special block: While for the penultimate block we require $M[m-1] = \texttt{len}(0^n) = 0^{120}10^7$ (fifteen null bytes followed by byte value 128, see also Appendix B), all other blocks may have arbitrary contents and the last block may be partial (i.e., $0 < |M[m]| \leq n$).[14] We believe this format is not too special and could naturally occur in applications, e.g., if plaintexts receive a length padding before being encrypted. The attack steps are as follows.

1. Encrypt (N, A, M) where N and A are arbitrary, $M = M[1] \| \cdots \| M[m-1] \| M[m]$ is an m-block message satisfying
$$M[m-1] = \texttt{len}(0^n),$$
and $M[m]$ is any s-bit string such that $\tau \leq s \leq n$. The encryption oracle returns a pair (C, T) where $C = C[1] \| \cdots \| C[m-1] \| C[m]$ and $|C[m]| = s$ and $|T| = \tau$.
2. Decrypt (N', A', C', T') where $N' = N$, $A' = \varepsilon$, and $C' = C'[1] \| \cdots \| C'[m-2] \| C'[m-1]$ has $m-1$ (full) blocks such that
$$C'[i] = C[i] \text{ for } 1 \leq i \leq m-2$$
$$C'[m-1] = C[m-1] \oplus \texttt{len}(M[m]) \oplus \sum_{j=1}^{m-2} M[j]$$
$$T' = \texttt{msb}_\tau(M[m] \oplus C[m]).$$

[14] The attack does not require knowledge of the contents of blocks $M[1], \ldots, M[m-2]$, but does depend on their sum $M[1] \oplus \cdots \oplus M[m-2]$.

To see that this tuple is accepted as authentic (and thus manifests a forgery), let $\overline{T}'$ be the reconstructed (untruncated) tag in the decryption query. We have

$$\begin{aligned}\overline{T}' &= E(2^{m-1}3L \oplus \Sigma') \\ &= E\Big(2^{m-1}3L \oplus \big(C'[m-1] \oplus \mathrm{Pad}'\big) \oplus \sum_{j=1}^{m-2} M'[j]\Big).\end{aligned}$$

As Pad' is computed as $\mathrm{Pad}' = E(2^{m-1}L \oplus \mathtt{len}(C'[m-1])) = 2^{m-1}L \oplus 2^{m-1}L \oplus E(2^{m-1}L \oplus \mathtt{len}(0^n)) = 2^{m-1}L \oplus C[m-1]$ and we have $M'[j] = M[j]$ for all $1 \le j < m-1$, we obtain

$$\overline{T}' = E\Big(2^{m-1}3L \oplus C'[m-1] \oplus (2^{m-1}L \oplus C[m-1]) \oplus \sum_{j=1}^{m-2} M[j]\Big).$$

By the identity $2^{m-1}3L \oplus 2^{m-1}L = 2^m L$ and the specification of $C'[m-1]$, this implies that

$$\begin{aligned}\overline{T}' &= E(2^m L \oplus \mathtt{len}(M[m])) \\ &= \mathrm{Pad} \\ &= M[m] \oplus C[m].\end{aligned}$$

This in particular means that $T^* = \mathtt{msb}_\tau(\overline{T}') = \mathtt{msb}_\tau(M[m] \oplus C[m]) = T'$, i.e., the forged ciphertext is accepted as authentic.

4.3 Confidentiality Attack

In Sec. 4.1 we have seen a basic attack that breaks the authenticity of OCB2. Perhaps surprisingly at first, the very same attack (formally) also breaks the privacy of the scheme. In the following we describe a corresponding two-query adversary against the PRIV-CCA notion that achieves a distinguishing advantage of almost 1.

Attacking the PRIV-CCA and PRIV-CV notions. The intuition behind our adversary is simple: It poses the same encryption and decryption queries as the adversary from Sec. 4.1, but then considers whether the value M' returned by the decryption oracle indicates that the ciphertext is valid or not. Precisely, if $M' \in \mathcal{M}$, it outputs $b = 1$; otherwise, if $M' = \perp$, it outputs $b = 0$. Note that if the adversary interacts with legit $\mathcal{E}$ and $\mathcal{D}$ oracles then the forgery will be successful (by what we proved in Sec. 4.1) and we have the $b = 1$ case. On the other hand, if it interacts with \$ and $\mathcal{D}$, the probability that $M' \neq \perp$ is only $2^{-\tau}$, and thus $b = 0$ is output with high probability.

We turn to describing this attack in the terms of formal definitions of confidentiality. In Sec. 2.3 we formalized the two notions PRIV and PRIV-CCA, where the former did not have a decryption oracle and targeted fully passive adversaries. We note that a variant of PRIV that provides a *ciphertext verification oracle* would interpolate between the two. We call this notion PRIV-CV (for ciphertext verification). The new oracle tries to decrypt any provided ciphertext and returns a bit, encoded as $\top/\perp$, that indicates whether the ciphertext is valid or not. Obviously, any adversary that breaks PRIV-CV in particular also breaks PRIV-CCA. We give the formal details of the above attack using the PRIV-CV formalism. The corresponding code is in Fig. 4 (right), where we denote the verification oracle with $\mathcal{V}$.

Attacking the IND-CCA notion. A different formalization of confidentiality is given by the IND-CCA notion [36]. It does not require that ciphertexts look like random strings but instead focuses on the bare semantic security aspect of encryption. It is easy to modify our above attack to be successful in the IND-CCA sense: In the classic left-or-right setting, the left message would be chosen according to our authenticity attack (e.g., $M_L = \mathtt{len}(0^n) \,\|\, 0^n$), while the right message would be chosen to be something unrelated (e.g., $M_R = 0^{2n}$). As above, the adversary would output $b = 1$ iff its forgery attempt is deemed valid.

Adversary $\mathcal{A}^{\mathcal{E}(\cdot,\cdot,\cdot),\mathcal{D}(\cdot,\cdot,\cdot,\cdot)}$	**Adversary** $\mathcal{A}^{\mathcal{E}/\$(\cdot,\cdot,\cdot),\mathcal{V}(\cdot,\cdot,\cdot,\cdot)}$
1. **Step 1:**	1. **Step 1:**
2. $M[1] \leftarrow \mathtt{len}(0^n)$	2. $M[1] \leftarrow \mathtt{len}(0^n)$
3. **Pick any** $M[2] \in \{0,1\}^n$	3. **Pick any** $M[2] \in \{0,1\}^n$
4. $M \leftarrow M[1] \,\|\, M[2]$	4. $M \leftarrow M[1] \,\|\, M[2]$
5. **Pick any** $N \in \{0,1\}^n$	5. **Pick any** $N \in \{0,1\}^n$
6. **Query** $(C,T) \leftarrow \mathcal{E}(N,\varepsilon,M)$	6. **Query** $(C,T) \leftarrow [\mathcal{E}/\$](N,\varepsilon,M)$
7. **Step 2:**	7. **Step 2:**
8. $C[1] \,\|\, C[2] \xleftarrow{n} C$	8. $C[1] \,\|\, C[2] \xleftarrow{n} C$
9. $C' \leftarrow C[1] \oplus \mathtt{len}(0^n)$	9. $C' \leftarrow C[1] \oplus \mathtt{len}(0^n)$
10. $T' \leftarrow M[2] \oplus C[2]$	10. $T' \leftarrow M[2] \oplus C[2]$
11. **Query** $M' \leftarrow \mathcal{D}(N,\varepsilon,C',T')$	11. **Query** $z \leftarrow \mathcal{V}(N,\varepsilon,C',T')$
12. **Stop**	12. **if** $z = \top$ **then** $b \leftarrow 1$ **else** $b \leftarrow 0$
	13. **Stop with** b

Fig. 4. Left: Minimal attack on authenticity. **Right:** Minimal attack on privacy (version with ciphertext verification oracle).

4.4 Relaxing the Attack Conditions

We can relax the conditions from Sec. 4.1 and Sec. 4.2 on the last two message blocks in the encryption query, namely $M[m-1] = \mathtt{len}(0^n)$ and $|M[m]| \geq \tau$, at the cost of a decreased success probability. First, the attacks also work for some values $M[m-1] \neq \mathtt{len}(0^n)$. For example, suppose the minimal case with $m = 2$ and $\tau = n$, where the adversary sets $M[m-1] = M[1] = \mathtt{len}(0^{n-i})$ for some $0 < i < n$. Then, the adversary creates the forgery attempt by using $(n-i)$-bit $C' = \mathtt{msb}_{n-i}(C[m-1] \oplus \mathtt{len}(0^n))$ and $T' = C[2] \oplus M[2]$. The forgery is accepted if $T' = T^*$ (the true tag) holds, which implies

$$2^2 L \oplus \mathtt{len}(0^n) = 2 \cdot 3L \oplus \Sigma',$$

where $\Sigma' = C' \,\|\, 0^i \oplus \mathrm{Pad}'$ and $\mathrm{Pad}' = E_K(2L \oplus \mathtt{len}(0^{n-i}))$. This equation is equivalent to $\mathtt{lsb}_i(E_K(2L \oplus \mathtt{len}(0^{n-i}))) = \mathtt{lsb}_i(2L \oplus \mathtt{len}(0^n))$, which holds with probability about $1/2^i$ assuming E_K is perfect. Hence, the forgery is still successful if i is small. If the adversary encrypts 2^i messages with different nonce values and mounts the above attack for each message, we can expect one successful forgery. The generalizations to long messages and to the case $\tau < n$ are immediate.

Second, when $j = \mathtt{len}(M[m]) < \tau$, the adversary only knows the first j bits of the true tag, i.e, $\mathtt{msb}_j(T^*) = \mathtt{msb}_j(\mathrm{Pad})$. By guessing the remaining $\tau - j$ bits of T^*, the attack will succeed with probability $1/2^{\tau-j}$. Thus, the attack is still practical when $\mathtt{len}(M[m])$ is close to τ.

5 Raw Blockcipher Access

The attacks that we demonstrated in Sec. 4 are powerful and general, but not universal. For instance, our attacks on authenticity succeeded with deriving non-authentic valid ciphertexts from authentic ones, but they did not provide full control over the forged message, nonce, and AD. Also our confidentiality attack could only distinguish encryptions, rather than recover plaintexts. We present correspondingly stronger attacks in Sec. 6. They are based on a toolbox of the three attack algorithms SamplePairs, VecEncipher, and VecDecipher that we present in this section.

Generally speaking, the best achievable result in symmetric cryptanalysis is key recovery: If the adversary obtains a full copy of the instance key, it is on par with the regular participants and can do everything they can do. For the wide range of modes of operation of blockciphers that reduce the role of the blockcipher to that of a privately accessible random permutation (this includes the three members of the OCB family), a key recovery attack cannot be expected to exist. This holds even for the weakest such candidate, simply because a mode that does not get in contact with the key, also cannot leak it. The situation becomes different, however, when slightly changing the point of perspective: If we see such modes as being *keyed with a private permutation* rather than with a blockcipher key,[15] a key recovery

[15] See Fig. 1 for an example: Our notation $\mathcal{E}_E(N,A,M)$ suggests that the mode's key is the access to E and its inverse; the key K does not appear at this level of abstraction.

attack would be one where the adversary, through the attack, obtains unrestricted access to the private permutation. This is what we develop in this section: Algorithms VecEncipher and VecDecipher leverage on access to a keyed OCB2 instance to provide unrestricted (bidirectional) access to the underlying blockcipher instance E_K.

Assuming a fixed blockcipher instance $E = E_K$, we refer to any pair $(X, Y) \in \{0,1\}^n \times \{0,1\}^n$ satisfying $E(X) = Y$ as an *input-output pair* or *mapping* of the blockcipher. The regular deployment of OCB2 does not expose such pairs. (This is not coincidental as the XEX* construction becomes insecure when such pairs become public.) However, as we observe and explore in the following, the decrypted message resulting from a successful forgery against OCB2 *does* leak one or more input-output pairs. The task of our SamplePairs algorithm is to gather such pairs and store them in a set variable $\mathbb{E}$. Our VecEncipher and VecDecipher algorithms, which emulate direct blockcipher access, will consult this set, for instance when they are in need of a fresh nonce N with an a priori known value $L = E(N)$ (see line 1 in Fig. 1). After the set variable has been initialized as per $\mathbb{E} \leftarrow \emptyset$, each invocation of one of the three algorithms populates it with new elements. At any point the invariants $\mathbb{E} \subseteq \{0,1\}^n \times \{0,1\}^n$ and $(X, Y) \in \mathbb{E} \Rightarrow E(X) = Y$ hold.

5.1 Extracting Random Blockcipher Mappings

We develop a procedure that, on input an integer m, performs a specific OCB2 forgery attack and extracts roughly m input-output pairs from the result. As our procedure does not control the points X, Y for which it finds the pairs we refer to the process as 'random mapping extraction'.

Recall that in our authenticity attack from Sec. 4.1 the adversary learns the value $M' = 2L \oplus \mathtt{len}(0^n)$ and thus $E(N) = L = (M' \oplus \mathtt{len}(0^n))/2$ from the forgery. Note that the pair (N, L) is the first example of an extracted input-output pair. In fact, inspection of the OCB2 algorithms in Fig. 1 shows that also $(2L \oplus \mathtt{len}(0^n), 2L \oplus C[1])$ and $(2^2 L \oplus \mathtt{len}(0^n), C[2] \oplus M[2])$ are input-output pairs of E. In addition, but only if tags are not truncated, we can obtain one more such pair from Σ and T.

Similar observations hold for our long-message forgery attack of Sec. 4.2, and the number of extractable input-output pairs is even higher (linear in the length of the message). Our SamplePairs procedure, specified in Fig. 5 and illustrated in Fig. 6, mechanizes the input-output pair gathering by crafting, in the spirit of Sec. 4.2, a forgery for a long all-zero message. Precisely, the procedure takes on input a value $m \geq 2$ and extracts at least $m + 1$ input-output pairs[16] from an invocation of its $\mathcal{E}$ and $\mathcal{D}$ oracles. One such pair is extracted from the nonce N and corresponding value L; a total of $m - 1$ pairs are extracted from all but the last message block; one pair is extracted from the padding used to encrypt the last block; and, if tags are not shortened, one more pair is extracted from the checksum and tag. The correctness of the procedure follows from inspection and the correctness of our attack from Sec. 4.2.

5.2 Extracting Specific Blockcipher Mappings

Once a non-empty set $\mathbb{E}$ is obtained with the SamplePairs procedure, we can implement a second procedure that takes an arbitrary vector $(X_1, X_2, \ldots)$ of blockcipher inputs and returns the vector $(Y_1, Y_2, \ldots)$ such that $E(X_i) = Y_i$ for all i. The underlying idea is to pick from $\mathbb{E}$ a random input-output pair (N, L), to use N as a (hopefully fresh) nonce in an encryption query of a message M, and to exploit the a priori knowledge of value L (that would normally remain hidden) to carefully prepare message M such that the blockcipher invocations induced by the encryption process coincide exactly with the points X_i. The corresponding values Y_i can then be extracted from the ciphertext.[17]

The specification of the corresponding VecEncipher procedure is in Fig. 7. See Fig. 8 for an illustration. The nonce generation in line 2 assumes that set $\mathbb{E}$ was populated before by at least one invocation of procedure SamplePairs. The likely most interesting detail of the procedure is that while the first $m - 1$ values X_i are embedded directly into (the first $m - 1$ blocks of) the message M, the one remaining

[16] The number of pairs can be fewer than $m + 1$ if collisions occur. This happens, however, only with negligible probability.

[17] The technique of first learning value L in order to then attack the security of OCB was already explored in prior work. While Ferguson [12] recovered L from collisions arising during the encryption of very long messages, Vaudenay and Vizár [42] achieved L-recovery in a setting that is not nonce-respecting. Notably, [42] observed that learning L suffices to recover arbitrary blockcipher mappings.

Procedure $\mathsf{SamplePairs}^{\mathcal{E}(\cdot,\cdot,\cdot),\mathcal{D}(\cdot,\cdot,\cdot,\cdot)}(m)$

1. **Global variable:** $\mathbb{E}$
2. $M[1,\ldots,m-2,m] \leftarrow 0^n$
3. $M[m-1] \leftarrow \mathtt{len}(0^n)$
4. $M \leftarrow M[1] \,\|\, \ldots \,\|\, M[m]$
5. $N \xleftarrow{\$} \{0,1\}^n$
6. $(C,T) \leftarrow \mathcal{E}(N,\varepsilon,M)$
7. $C[1] \,\|\, \ldots \,\|\, C[m] \xleftarrow{n} C$
8. $C[m-1] \leftarrow C[m-1] \oplus \mathtt{len}(0^n)$
9. $C' \leftarrow C[1] \,\|\, \ldots \,\|\, C[m-1]$
10. $T' \leftarrow \mathtt{msb}_\tau(C[m])$
11. $M' \leftarrow \mathcal{D}(N,\varepsilon,C',T')$
12. $M'[1] \,\|\, \ldots \,\|\, M'[m-1] \xleftarrow{n} M'$
13. $L \leftarrow 2^{-(m-1)}(M'[m-1] \oplus \mathtt{len}(0^n))$
14. **for** $i \leftarrow 1$ **to** $m-1$
15. $\quad (X_i, Y_i) \leftarrow (2^i L \oplus M[i], 2^i L \oplus C[i])$
16. $X_m \leftarrow 2^m L \oplus \mathtt{len}(0^n)$
17. $Y_m \leftarrow C[m]$
18. $\mathbb{E} \xleftarrow{\cup} \{(N,L)\}$
19. $\mathbb{E} \xleftarrow{\cup} \{(X_1,Y_1),\ldots,(X_m,Y_m)\}$
20. **if** $\tau = n$ **then**
21. $\quad X_T \leftarrow 2^m 3L \oplus \mathtt{len}(0^n)$
22. $\quad Y_T \leftarrow T$
23. $\quad \mathbb{E} \xleftarrow{\cup} \{(X_T,Y_T)\}$
24. **return**

Fig. 5. Extraction of a random collection of at least $m+1$ pairs (X_i, Y_i) such that $E(X_i) = Y_i$ for all i.

value X_m is only implicitly embedded: We carefully choose the last message block $M[m]$ such that the checksum $\Sigma = M[1] \oplus \cdots \oplus M[m]$ used to derive the authentication tag is such that the tag is computed as $T = E(X_m)$. Observe that the full T, and thus Y_m, is visible to the adversary only if $\tau = n$, i.e., if the tag is not truncated. Correspondingly, our procedure translates X_m to Y_m only in this case. Otherwise, if $\tau < n$, only for $X_1, \ldots, X_{m-1}$ the corresponding value Y_i is identified and returned. Note that we feed back all extracted pairs (X_i, Y_i) into the set $\mathbb{E}$, giving more choice to pick a fresh nonce in line 2 of a later invocation of VecEncipher.

5.3 Extracting Specific Blockcipher Inverse Mappings

We next present our third procedure: VecDecipher. It takes an arbitrary vector $(Y_1, Y_2, \ldots)$ of blockcipher outputs and returns the vector $(X_1, X_2, \ldots)$ of blockcipher inputs such that $E(X_i) = Y_i$ for all i, i.e., this procedure extracts specific blockcipher *inverse* mappings. The idea underlying the procedure is similar to that of VecEncipher: We pick from set $\mathbb{E}$ an input-output pair (N, L), and prepare ciphertext C and tag T such that the inverse blockcipher invocations induced by the decryption process are on the points Y_i. The corresponding values X_i can be obtained from the recovered message.

We specify the VecDecipher procedure in Fig. 9 and provide an illustration in Fig. 10. In the following we explain the details, always assuming $L = E(N)$. For any given vector $(Y_1, \ldots, Y_{m'})$, consider the message $M' := (M'[1], \ldots, M'[m])$ of $m := 2m'+1$ blocks (implicitly) defined as

$$M'[i] := \begin{cases} 2^i L \oplus X_{\lceil i/2 \rceil} & \text{for } i = 1,3,5,\ldots,2m'-1, \\ 2^i L \oplus X_{i/2} & \text{for } i = 2,4,6,\ldots,2m', \\ S \oplus 2^{2m'+1} 3L \oplus N & \text{for } i = 2m'+1, \end{cases}$$

where $S := (2^1 + 2^2 + \cdots + 2^{2m'})L$ and $(X_1, \ldots, X_{m'})$ is the target vector we would like to extract. Encrypting $M' = (M'[1], \ldots, M'[2m'+1])$ in the configuration (N, ε, M') results in a pair (C, T) consisting of a ciphertext $C = (C[1], \ldots, C[2m'+1])$ and a tag T. See Fig. 10. Our approach is to construct (C, T)

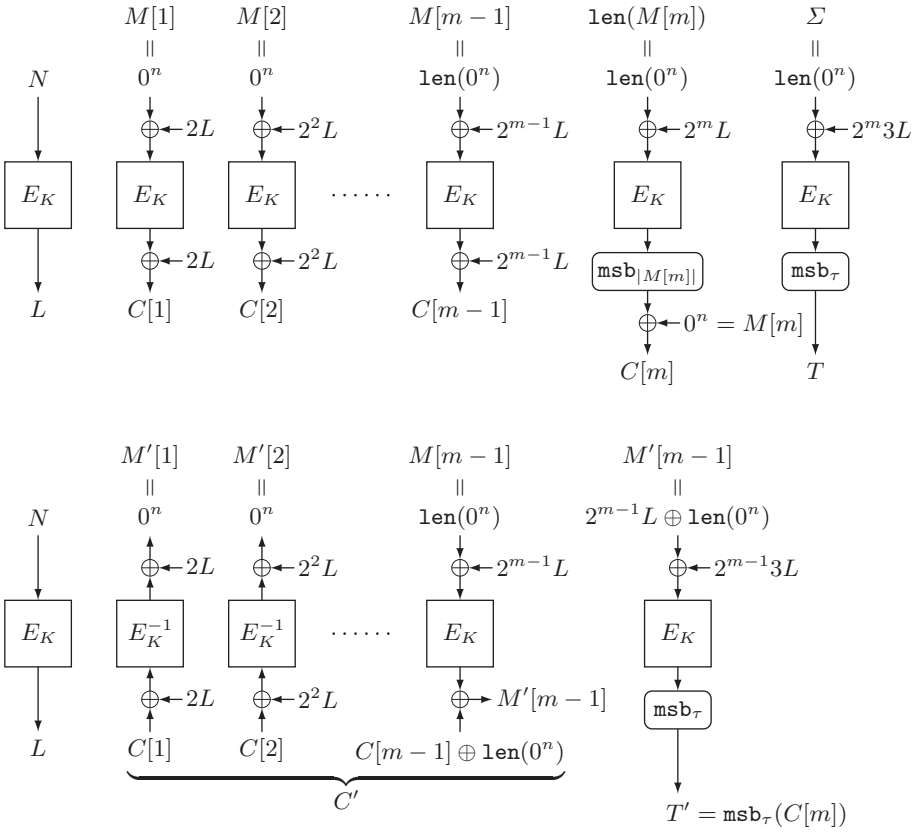

Fig. 6. Top: Line 2–6 of SamplePairs. **Bottom:** Line 7–11 of SamplePairs. In the bottom figure, $M'[m-1]$ is $2^{m-1}L \oplus \mathtt{len}(0^n)$, and note that $2^{m-1}L \oplus 2^{m-1}3L = 2^m L$ holds.

from $(Y_1, Y_2, \ldots)$ and other known information, and to let the decryption oracle, on input (N, ε, C, T), recover M' for us, from which, in turn, we recover the vector $(X_1, X_2, \ldots)$.

Observe that the message M' is defined to meet the following three conditions:

- Knowledge of M' is sufficient to obtain $(X_1, \ldots, X_{m'})$. This condition is indeed met as we know the value of L.
- We can compute all ciphertext components $C[1], \ldots, C[2m'+1]$. Also this condition is met: The values of $C[1], \ldots, C[2m']$ can be derived directly from the inputs $(Y_1, \ldots, Y_{m'})$, and the last block $C[2m'+1]$ can be derived from $E(2^m L \oplus \mathtt{len}(0^n))$. (Note that computing the last block typically requires invoking the VecEncipher procedure of Fig. 7 as a subroutine.)
- We can compute the tag T. To verify this condition, first consider that the checksum of M' is the known value $\Sigma' = 2^m 3L \oplus N$. This is so as $M'[1] \oplus \cdots \oplus M'[2m'] = S$ (as the contributions of the individual X_i terms cancel out in the sum), and thus

$$\Sigma' = M'[1] \oplus \cdots \oplus M'[2m'] \oplus M'[2m'+1] = S \oplus (S \oplus 2^m 3L \oplus N) = 2^m 3L \oplus N.$$

  To complete the argument, note that $\Sigma' = 2^m 3L \oplus N$ implies that $T = L$.

We see that we can first put together the correct encryption (C, T) of inputs (N, ε, M'), and then extract the vector $(X_1, \ldots, X_{m'})$ from the result of a corresponding decryption query. The precise steps are worked out in procedure VecDecipher in Fig. 9.

6 Universal Forgery and Decryption

In this section we target the most powerful goals of encryption scheme cryptanalysis: We contribute a universal forgery attack and a full plaintext recovery attack for arbitrary ciphertexts.

Procedure $\mathsf{VecEncipher}^{\mathcal{E}(\cdot,\cdot,\cdot)}(X_1, \ldots, X_{m-1}, X_m)$

1. **Global variable:** $\mathbb{E}$
2. $(N, L) \xleftarrow{\$} \mathbb{E}$
3. **for** $i \leftarrow 1$ **to** $m-1$
4. $\quad M[i] \leftarrow 2^i L \oplus X_i$
5. $\Sigma \leftarrow 2^m 3L \oplus X_m$
6. $M[m] \leftarrow M[1] \oplus \ldots \oplus M[m-1] \oplus \Sigma$
7. $M \leftarrow M[1] \,\|\, \ldots \,\|\, M[m]$
8. $(C, T) \leftarrow \mathcal{E}(N, \varepsilon, M)$
9. $C[1] \,\|\, \ldots \,\|\, C[m] \xleftarrow{n} C$
10. **for** $i \leftarrow 1$ **to** $m-1$
11. $\quad Y_i \leftarrow 2^i L \oplus C[i]$
12. $X' \leftarrow 2^m L \oplus \mathtt{len}(0^n)$
13. $Y' \leftarrow M[m] \oplus C[m]$
14. $\mathbb{E} \xleftarrow{\cup} \{(X_1, Y_1), \ldots, (X_{m-1}, Y_{m-1})\}$
15. $\mathbb{E} \xleftarrow{\cup} \{(X', Y')\}$
16. **if** $\tau = n$ **then**
17. $\quad Y_m \leftarrow T$
18. $\quad \mathbb{E} \xleftarrow{\cup} \{(X_m, Y_m)\}$
19. **return** $(Y_1, \ldots, Y_{m-1}, Y_m)$

Fig. 7. Look-up, given values $X_1, \ldots, X_{m-1}$, of values $Y_1, \ldots, Y_{m-1}$ such that $E(X_i) = Y_i$ for all i. If $\tau = n$ (gray part), even one more mapping $X_m \to Y_m$ can be processed. (If $\tau < n$, use any value for X_m in line 5, e.g., $X_m = 0^n$.) To ensure that a fresh nonce can be picked in line 2, procedure VecEncipher may only be invoked after SamplePairs has been.

6.1 Universal Forgeries

In a universal forgery attack the adversary chooses any $N^\$ \in \mathcal{N}$, any $M^\$ \in \mathcal{M}$, and any $A^\$ \in \mathcal{A}$, and creates a forgery $(C^\$, T^\$)$ such that the decryption algorithm $\mathcal{D}_K(N^\$, A^\$, C^\$, T^\$)$ returns $M^\$$. We present a universal forgery attack for OCB2 that is based on the two sub-routines SamplePairs and VecEncipher that we described in Sec. 5. In fact, given these algorithms it is actually immediate to compute forgeries on any combination of nonce N, message M, and AD A: It simply suffices to execute OCB2's encryption algorithm $\mathcal{E}_E$ from Fig. 1 on input N, A, M, emulating all blockcipher evaluations with invocations of VecEncipher. The resulting forgeries are perfect and fresh (non-trivial). Note further that OCB2 is parallelizable, that is, most of the blockcipher evaluations of an encryption operation happen concurrently of each other. This property makes forging very efficient (in terms of the number of required encryption queries), as all concurrent enciphering operations can be batch-processed with a single VecEncipher call.

When closely looking at the details it however becomes apparent that universally forging cannot be performed with a single VecEncipher invocation. As a matter of fact, not all enciphering operations related to an encryption are concurrent: In OCB2's $\mathcal{E}_E$ algorithm, tag T is computed by enciphering a value dependent on Pad which is a blockcipher output by itself. These computations cannot be parallelized, and it becomes clear that universal forging requires at least two succeeding VecEncipher invocations. A similar observation can be made for the PMAC algorithm (see Fig. 15 in Appendix B) where the finalization step requires enciphering an intermediate sum that is computed by adding up outputs of other enciphering operations. The latter, in turn, depend on the value $E(0^n)$, so the minimal number of VecEncipher invocations increases to three. (Of course $E(0^n)$ could be cached from a prior forgery but a worst-case analysis cannot assume that.)

We complete this discussion by showing that three VecEncipher invocations are sufficient in all cases. We do this by describing the full set of instructions to compute a forgery $(C^\$, T^\$)$ for input data $N^\$, M^\$, A^\$$.

The attack successively calls SamplePairs and VecEncipher. The first call is used to retrieve $E(N^\$)$ and $E(0^n)$, the second is used to retrieve the encipherings needed for encrypting $M^\$$ and PMAC-ing $A^\$$ *except the tag and the last AD block*, and the third is used for processing the tag and the last AD block. Specifically, the steps for the universal forgery are as follows:

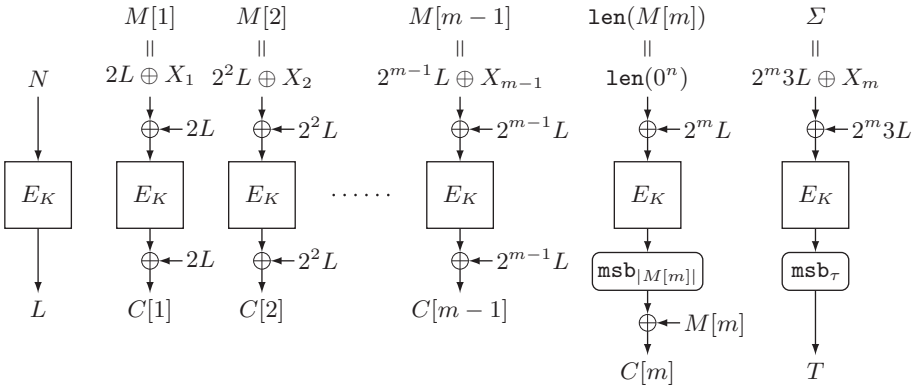

Fig. 8. Line 2–8 of VecEncipher. Note that $M[m]$ is $M[1] \oplus \cdots \oplus M[m-1] \oplus \Sigma$.

Procedure $\mathsf{VecDecipher}^{\mathcal{E}(\cdot,\cdot,\cdot),\mathcal{D}(\cdot,\cdot,\cdot,\cdot)}(Y_1, \ldots, Y_{m'})$

1. **Global variable:** $\mathbb{E}$
2. $m \leftarrow 2m' + 1$
3. Fix any $(N, L) \in \mathbb{E}$
4. $S \leftarrow 0^n$
5. **for** $i \leftarrow 1$ **to** $m - 1$
6. $\quad S \leftarrow S \oplus 2^i L$
7. $\quad C[i] \leftarrow 2^i L \oplus Y_{\lceil i/2 \rceil}$
8. Pad $\leftarrow E(2^m L \oplus \texttt{len}(0^n))$
9. $C[m] \leftarrow S \oplus 2^m 3L \oplus N \oplus \text{Pad}$
10. $C \leftarrow C[1] \| \ldots \| C[m]$
11. $T \leftarrow \texttt{msb}_\tau(L)$
12. $M' \leftarrow \mathcal{D}(N, \varepsilon, C, T)$
13. $M'[1] \| \ldots \| M'[m] \xleftarrow{n} M'$
14. **for** $i \leftarrow 1$ **step** 2 **to** $m - 2$
15. $\quad X_{\lceil i/2 \rceil} \leftarrow 2^i L \oplus M'[i]$
16. $\mathbb{E} \xleftarrow{\cup} \{(X_1, Y_1), \ldots, (X_{m'}, Y_{m'})\}$
17. **return** $(X_1, \ldots, X_{m'})$

Fig. 9. Look-up, given values $Y_1, \ldots, Y_{m'}$, of values $X_1, \ldots, X_{m'}$ such that $E(X_i) = Y_i$ for all i. The invocation of E in line 8 shall be implemented via a VecEncipher invocation. (Note that the input $2^m L \oplus \texttt{len}(0^n)$ to the latter is independent of $Y_1, \ldots, Y_{m'}$, meaning that the result Pad might be cacheable across invocations of VecDecipher.)

1. The adversary invokes SamplePairs(2). With overwhelming probability, we assume the nonce sampled in SamplePairs(2), N', is different from $N^\$$. Then she obtains a set of distinct pairs written as $\mathbb{E} = \{(N', L'), (X', Y'), (X'', Y'')\}$.
2. If $(N^\$, E_K(N^\$)), (0^n, E_K(0^n)) \in \mathbb{E}$, she goes to the next step. Otherwise, she invokes VecEncipher$(N^\$, 0^n, 0^n)$ and obtains $L \coloneqq E_K(N^\$)$ and $V \coloneqq 3^2 E_K(0^n)$.
3. Let

$$\begin{aligned} X_i &\coloneqq M^\$[i] \oplus 2^i L \text{ for } 1 \le i \le m-1, \\ X_m &\coloneqq \texttt{len}(M^\$[m]) \oplus 2^m L, \\ X_i^A &\coloneqq A^\$[i] \oplus 2^i V, \text{ for } 1 \le i \le a-1, \end{aligned}$$

   where $(M^\$[1], \ldots, M^\$[m]) \xleftarrow{n} M^\$$ and $(A^\$[1], \ldots, A^\$[a]) \xleftarrow{n} A^\$$. She obtains $Y_i = E_K(X_i)$ $(1 \le i \le m)$ and $Y_i^A = E_K(X_i^A)$ $(1 \le i \le a-1)$ by performing VecEncipher$(X_1, \ldots, X_m, X_1^A, \ldots, X_{a-1}^A, 0^n)$.
4. Let $X_{m+1} \coloneqq \Sigma^\$ \oplus 2^m \cdot 3L$, where

$$\Sigma^\$ = M^\$[1] \oplus \ldots \oplus M^\$[m-1] \oplus (M^\$[m] \| \texttt{lsb}_{n-|M^\$[m]|}(Y_m)).$$

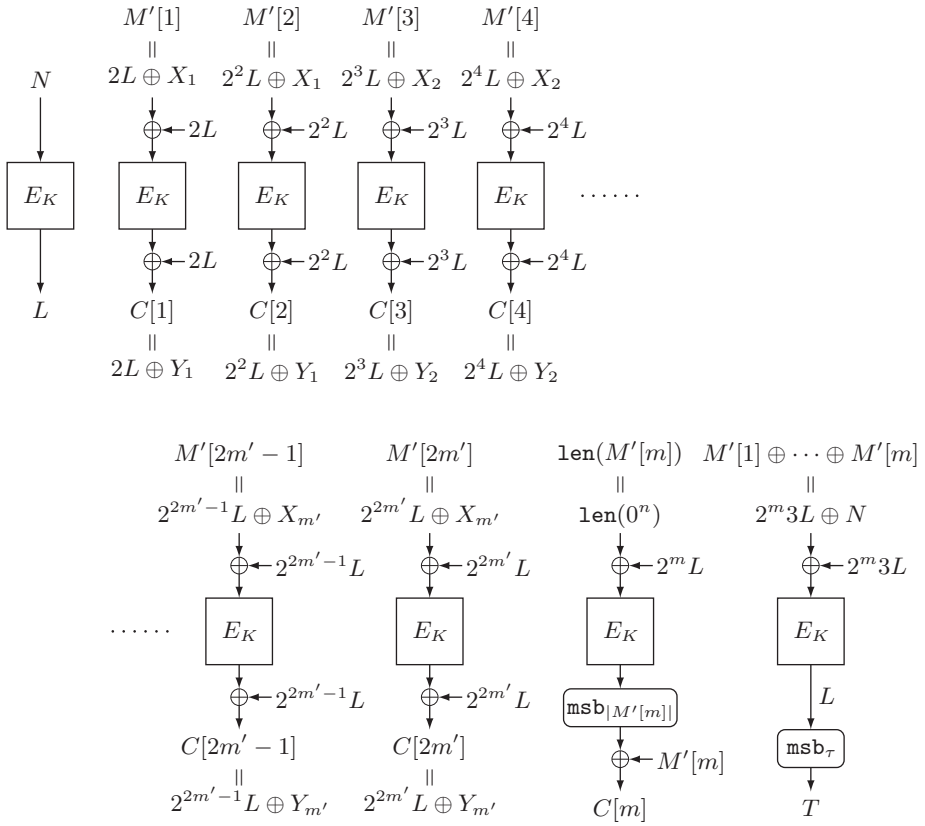

Fig. 10. The encryption process of (N, ε, M') in VecDecipher, where $M' = (M'[1], \ldots, M'[m])$ and $m = 2m' + 1$. Note that $M'[m] = S \oplus 2^m 3L \oplus N$, where $S = (2^1 + 2^2 + \cdots + 2^{2^{m'}})L$. Observe that the only unknown block in (C, T) is $C[m]$, which can be computed with VecEncipher. It follows that the decryption oracle returns M' for a query (N, ε, C, T).

If $|A^\$[a]| = n$, let $X_a^A \coloneqq \sum_{i=1}^{a-1} Y_i^A \oplus A^\$[a] \oplus 2^a \cdot 3V$ and else, $X_a^A \coloneqq \sum_{i=1}^{a-1} Y_i^A \oplus (A^\$[a] \,\|\, 10^*) \oplus 2^a \cdot 3^2 V$. She obtains $Y_{m+1} = E_K(X_{m+1})$ and $Y_a^A = E_K(X_a^A)$ by calling VecEncipher$(X_{m+1}, X_a^A, 0^n)$.

5. She creates $(N^\$, A^\$, C^\$, T^\$)$, where

$$C^\$ = (Y_1 \oplus 2L) \,\|\, \cdots \,\|\, (Y_{m-1} \oplus 2^{m-1}L) \,\|\, (\mathtt{msb}_{|M^\$[m]|}(Y_m) \oplus M^\$[m]),$$
$$T^\$ = \mathtt{msb}_\tau(Y_{m+1} \oplus Y_a^A).$$

This tuple $(N^\$, A^\$, C^\$, T^\$)$ will be accepted as valid by $\mathcal{D}$, with return value $M^\$$.

6.2 Plaintext Recovery

We consider an attack setting that closely follows [28]: Fix a triple consisting of a nonce N^*, associated data A^*, and a plaintext M^* according to any distribution such that the adversary does not have full a priori knowledge of M^*. Let a challenger pick a key K uniformly at random from the key space and compute the encryption $(C^*, T^*) \leftarrow \mathcal{E}_K(N^*, A^*, M^*)$. The adversary obtains a copy of (N^*, A^*, C^*, T^*) and the attack goal is to accurately recover the unknown parts of message M^*. The adversary is assisted by encryption and decryption oracles, where three conditions have to be met: No two encryption queries may use the same nonce (nonce-respecting property), an encryption query with nonce N^* may not be posed (variant of nonce-respecting property), and the tuple (N^*, A^*, C^*, T^*) may not be queried for decryption.

We present two plaintext recovery attacks against OCB2. The first attack uses the SamplePairs, VecEncipher, and VecDecipher procedures from Sec. 5 to directly emulate the decryption process of

(N^*, A^*, C^*, T^*). The second plaintext recovery attack employs SamplePairs and VecEncipher, plus a single decryption query. While the first attack works for any challenge (N^*, A^*, C^*, T^*), the second attack requires that M^* and thus C^* have a length of at least three blocks. On the other hand, while both attacks are quite efficient, the first attack requires decryption queries on longer ciphertexts.

In both scenarios, we first recover $L^* := E_K(N^*)$. This can be done by using SamplePairs and VecEncipher as follows: The adversary first calls SamplePairs(2), and with overwhelming probability, we assume nonce N' sampled in SamplePairs(2) is different from N^*. Then she obtains a set of distinct pairs $\mathbb{E} = \{(N', L'), (X', Y'), (X'', Y'')\}$. If $(N^*, E_K(N^*)) \in \mathbb{E}$, then we have L^*. Otherwise, she invokes VecEncipher$(N^*, 0^n)$ to obtain L^*.

Let m^* be the number of blocks of C^*. The first attack faithfully emulates the decryption process of (N^*, A^*, C^*, T^*). That is, to recover M^* from the challenge and L^*, it is enough to compute $2^i L^* \oplus E_K^{-1}(2^i L^* \oplus C^*[i])$ for $1 \le i \le m^* - 1$ to obtain $M^*[1], \ldots, M^*[m^* - 1]$, and $E_K(2^{m^*} L \oplus \mathtt{len}(C^*[m^*]))$ to compute $M^*[m^*]$. See the bottom figure in Fig. 2. The first $m^* - 1$ blockcipher decryption calls can be performed with VecDecipher procedure, and the last blockcipher encryption call can be performed with VecEncipher procedure. The entire attack can be formalized as follows:

1. From the challenge (N^*, A^*, C^*, T^*), compute $L^* := E_K(N^*)$ as described above. We need one call to SamplePairs and one call to VecEncipher.
2. Invoke VecDecipher$(2L^* \oplus C^*[1], \ldots, 2^{m^*-1} L^* \oplus C^*[m^* - 1])$ to obtain $X_1, \ldots, X_{m^*-1}$, where $X_i = E_K^{-1}(2^i L^* \oplus C^*[i])$. Perform VecEncipher$(2^{m^*} L \oplus \mathtt{len}(C^*[m^*]), 0^n)$ to obtain $\mathrm{Pad}^* = E_K(2^{m^*} L \oplus \mathtt{len}(C^*[m^*]))$.
3. She creates $M^* = (M^*[1], \ldots, M^*[m^*])$, where $M^*[i] := 2^i L^* \oplus X_i$ for $1 \le i \le m^* - 1$, and $M^*[m^*] := \mathtt{msb}_{|C^*[m^*]|}(\mathrm{Pad}^*) \oplus C^*[m^*]$.

We see that M^* is the target message to recover, and the attack succeeds with an overwhelming probability. Therefore, it is possible to mount a plaintext recovery attack against any challenge (N^*, A^*, C^*, T^*).

We next present another attack to recover M^* by using SamplePairs, VecEncipher, and a decryption query. The attack requires that M^* (and hence C^*) consist of at least three blocks. The attack works as follows:

1. The adversary first recovers $L^* = E_K(N^*)$ for the challenge (N^*, A^*, C^*, T^*). This step needs one call to SamplePairs and one call to VecEncipher.
2. Then she modifies C^* to make a decryption query. Specifically, let $C^* = (C^*[1], \ldots, C^*[m^*])$ be the challenge ciphertext broken into blocks, and we first fix two distinct indices $j, k \in \{1, \ldots, m^* - 1\}$. Note that we are assuming that M^* is long and $m^* \ge 3$. The adversary then defines $C^\$ = (C^\$[1], \ldots, C^\$[m^*])$ as follows:
   - $C^\$[i] := C^*[i]$ for $i \in \{1, \ldots, m^*\} \setminus \{j, k\}$
   - $C^\$[j] := C^*[k] \oplus 2^k L^* \oplus 2^j L^*$
   - $C^\$[k] := C^*[j] \oplus 2^k L^* \oplus 2^j L^*$
3. Next, the adversary makes a decryption query $(N^*, A^*, C^\$, T^*)$. Note that this is almost the same as the challenge query, but the j-th and k-th blocks of C^* are masked and swapped. We have $C^\$ = C^*$ only with a negligible probability (e.g., if $C^*[j] = C^*[k]$ and $L^* = 0^n$). We see that the query will be accepted since the checksum remains the same. Thus, the adversary obtains $M^\$$.
4. The target of the attack, M^*, is obtained by swapping the j-th and k-th blocks of $M^\$$ and making necessary modifications. Precisely, from $M^\$ = (M^\$[1], \ldots, M^\$[m^*])$, the adversary obtains $M^* = (M^*[1], \ldots, M^*[m^*])$ as follows:
   - $M^*[i] := M^\$[i]$ for $i \in \{1, \ldots, m^*\} \setminus \{j, k\}$
   - $M^*[j] := M^\$[k] \oplus 2^k L^* \oplus 2^j L^*$
   - $M^*[k] := M^\$[j] \oplus 2^k L^* \oplus 2^j L^*$

See Fig. 11 for the encryption process of (N^*, A^*, M^*) and the decryption process of $(N^*, A^*, C^\$, T^*)$.

We note that in the first attack, the decryption query (within VecDecipher) has a ciphertext of $2m^* - 1$ blocks, while it is only m^* blocks in the second attack.

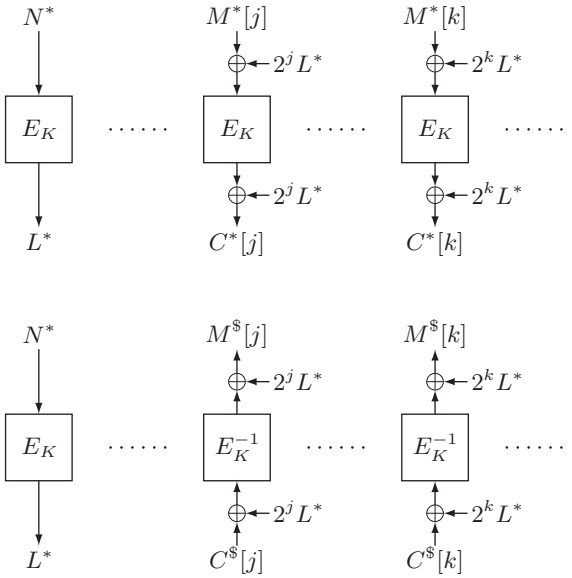

Fig. 11. Top: The encryption process of (N^*, A^*, M^*). **Bottom:** The decryption process of $(N^*, A^*, C^\$, T^*)$. In the bottom figure, we have $C^\$[j] = C^*[k] \oplus 2^kL^* \oplus 2^jL^*$ and $C^\$[k] = C^*[j] \oplus 2^kL^* \oplus 2^jL^*$, and it follows that $M^*[j] = M^\$[k] \oplus 2^kL^* \oplus 2^jL^*$ and $M^*[k] = M^\$[j] \oplus 2^kL^* \oplus 2^jL^*$. We see that the checksum remains the same.

7 Design Flaw of OCB2

The root of the flaw in OCB2 is in the instantiation of AE using XEX*. Here, XEX* is a mode of operation that constructs a TBC based on a blockcipher, and in fact it combines two modes, XEX and XE. For blockcipher E_K, let

$$\mathrm{XEX}_E^{N,i,j}(X) \stackrel{\text{def}}{=} E(2^iL \oplus X) \oplus 2^iL,$$
$$\mathrm{XE}_E^{N,i,j}(X) \stackrel{\text{def}}{=} E(2^i3^jL \oplus X),$$

where $L = E(N)$ for nonce N, for $i = 1, 2, \ldots$ and $j = 0, 1, \ldots$. Here, j is always set to 0 for XEX. XEX* unifies them by introducing one bit b to the tweak. That is,

$$\mathrm{XEX}_E^{*,b,N,i,j}(X) = \begin{cases} \mathrm{XEX}_E^{N,i,j}(X) & \text{if } b = 1; \\ \mathrm{XE}_E^{N,i,j}(X) & \text{if } b = 0. \end{cases}$$

Decryption is trivially defined, and is never invoked when $b = 0$ (for security; see Definition 1). Rog04 refers b to *tag*; not to be confused with the tag in the global interface of AE.

Suppose an encryption query of the form (N, A, M), where $A = \varepsilon$ and M is parsed as $(M[1], \ldots, M[m])$, is given to OCB2. It encrypts M by using $\mathrm{XEX}_E^{*,1,N,i,0}$ for $M[i]$ with $i = 1, \ldots, m-1$, and $\mathrm{XEX}_E^{*,0,N,m,0}$ for $M[m]$. The checksum, Σ, is encrypted by $\mathrm{XEX}_E^{*,0,N,m,1}$ to create the (untruncated) tag.

In the proof of OCB2, we first apply the standard conversion from computational to information-theoretic security [5] and focus on the security of OCB2 instantiated by an n-bit uniform random permutation (URP), P, denoted by OCB2$_\mathsf{P}$. Then, the proof of OCB2$_\mathsf{P}$ has two main steps: the indistinguishability of XEX$^*_\mathsf{P}$, and the privacy and authenticity of AE[18] which replaces XEX$^*_\mathsf{P}$ in OCB2$_\mathsf{P}$ with an ideal primitive, a tweakable random permutation $\widetilde{\mathsf{P}}$. The latter step is not relevant to our attacks.

For the first step, Rog04 proved that XEX$^*_\mathsf{P}$ is indistinguishable from $\widetilde{\mathsf{P}}$ for any adversary who queries to both encryption and decryption of XEX$^*_\mathsf{P}$ and respects the semantics of tag b. More precisely, the conditions for the adversary are as follows.

Definition 1. *We say an adversary querying* XEX* *is tag-respecting when*

[18] An equivalent mode for OCB3 is called ΘCB3 [23].

Algorithm $\Theta\text{CB2}.\mathcal{E}_{\widetilde{E}}(N, A, M)$	**Algorithm** $\Theta\text{CB2}.\mathcal{D}_{\widetilde{E}}(N, A, C, T)$
1. $(M[1], \ldots, M[m]) \xleftarrow{n} M$	1. $(C[1], \ldots, C[m]) \xleftarrow{n} C$
2. **for** $i = 1$ **to** $m - 1$	2. **for** $i = 1$ **to** $m - 1$
3. $\quad C[i] \leftarrow \widetilde{E}^{*,1,N,i,0}(M[i])$	3. $\quad M[i] \leftarrow (\widetilde{E}^{*,1,N,i,0})^{-1}(C[i])$
4. $\text{Pad} \leftarrow \widetilde{E}^{*,0,N,m,0}(\texttt{len}(M[m]))$	4. $\text{Pad} \leftarrow \widetilde{E}^{*,0,N,m,0}(\texttt{len}(C[m]))$
5. $C[m] \leftarrow M[m] \oplus \texttt{msb}_{\lvert M[m]\rvert}(\text{Pad})$	5. $M[m] \leftarrow C[m] \oplus \texttt{msb}_{\lvert C[m]\rvert}(\text{Pad})$
6. $\Sigma \leftarrow C[m] \,\|\, 0^* \oplus \text{Pad}$	6. $\Sigma \leftarrow C[m] \,\|\, 0^* \oplus \text{Pad}$
7. $\Sigma \leftarrow M[1] \oplus \cdots \oplus M[m-1] \oplus \Sigma$	7. $\Sigma \leftarrow M[1] \oplus \cdots \oplus M[m-1] \oplus \Sigma$
8. $T \leftarrow \widetilde{E}^{*,0,N,m,1}(\Sigma)$	8. $T^* \leftarrow \widetilde{E}^{*,0,N,m,1}(\Sigma)$
9. **return** (C, T)	9. **if** $T = T^*$ **return** M
	10. **else return** $\perp$

Fig. 12. Algorithms of ΘCB2. For simplicity, $\tau = n$ and $A = \varepsilon$.

1. $\text{XEX}^{*,0,N,i,j}$ *is only queried in encryption queries for any* (N, i, j)*;*
2. *Once* $\text{XEX}^{*,b,N,i,j}$ *is queried in either encryption or decryption, then it is not allowed to query* $\text{XEX}^{*,1-b,N,i,j}$*, for any* (N, i, j)*.*

Let $\Theta\text{CB2}_{\widetilde{E}}$ be the mode of operations of TBC $\widetilde{E}_K$ which has the same interface as XEX^*_E. The pseudocode is shown in Fig. 12. Then, $\Theta\text{CB2}_{\text{XEX}^*_E}$ is equivalent to OCB2_E.

Let $\widetilde{\mathsf{P}}$ be TURP which has the same interface as XEX^*. Rog04 showed that, for any privacy-adversary $\mathcal{A}$ and authenticity-adversary $\mathcal{A}_\pm$,

$$\mathbf{Adv}^{\mathsf{priv}}_{\text{OCB2}_{\widetilde{\mathsf{P}}}}(\mathcal{A}) = \mathbf{Adv}^{\mathsf{priv}}_{\Theta\text{CB2}_{\text{XEX}^*_{\widetilde{\mathsf{P}}}}}(\mathcal{A}) \le \mathbf{Adv}^{\mathsf{tprp}}_{\text{XEX}^*_{\widetilde{\mathsf{P}}}}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{priv}}_{\Theta\text{CB2}_{\widetilde{\mathsf{P}}}}(\mathcal{A}), \tag{5}$$

$$\mathbf{Adv}^{\mathsf{auth}}_{\text{OCB2}_{\widetilde{\mathsf{P}}}}(\mathcal{A}_\pm) = \mathbf{Adv}^{\mathsf{auth}}_{\Theta\text{CB2}_{\text{XEX}^*_{\widetilde{\mathsf{P}}}}}(\mathcal{A}_\pm) \le \mathbf{Adv}^{\mathsf{tsprp}}_{\text{XEX}^*_{\widetilde{\mathsf{P}}}}(\mathcal{B}_\pm) + \mathbf{Adv}^{\mathsf{auth}}_{\Theta\text{CB2}_{\widetilde{\mathsf{P}}}}(\mathcal{A}_\pm) \tag{6}$$

hold for some CPA-adversary $\mathcal{B}$ and CCA-adversary $\mathcal{B}_\pm$, which are tag-respecting and can simulate the privacy and the authenticity games involving $\Theta\text{CB2}_{\text{XEX}^*_{\widetilde{\mathsf{P}}}}$ and $\mathcal{A}$ and $\mathcal{A}_\pm$, respectively. From Rog04, we have

$$\mathbf{Adv}^{\mathsf{tprp}}_{\text{XEX}^*_{\widetilde{\mathsf{P}}}}(\mathcal{B}) \le \frac{4.5q^2}{2^n} \text{ and } \mathbf{Adv}^{\mathsf{tsprp}}_{\text{XEX}^*_{\widetilde{\mathsf{P}}}}(\mathcal{B}_\pm) \le \frac{9.5q^2}{2^n} \tag{7}$$

for any $\mathcal{B}$ and $\mathcal{B}_\pm$ that are tag-respecting and use at most q queries.[19] The last terms of (5) and (6) are proved to be almost ideally small: zero for privacy and $2^{n-\tau}/(2^n - 1)$ for authenticity with single decryption query.

The privacy bound is obtained from the first inequality of (7) and (5). However, to derive the authenticity bound, we need to identify $\mathcal{B}_\pm$ that can simulate $\mathcal{A}_\pm$, where $\mathcal{A}_\pm$ must compute the decryption of ΘCB2, even with single decryption query. Depending on $\mathcal{A}_\pm$, there are cases that no tag-respecting $\mathcal{B}_\pm$ can simulate $\mathcal{A}_\pm$. For example, let us assume that $\mathcal{A}_\pm$ first queries (N, A, M) of $|M| = 2n$ to the encryption oracle and then queries (N', A', C', T') to the decryption oracle, where $N' = N$, $A' = \varepsilon$ and $|C'| = n$, where the attack in Sec. 4.1 is an example case. Then, $\mathcal{B}_\pm$ who simulates $\mathcal{A}_\pm$ first queries to $\text{XEX}^{*,1,N,1,0}$ and $\text{XEX}^{*,0,N,2,0}$ and $\text{XEX}^{*,0,N,2,1}$. For the second query, it queries to $\text{XEX}^{*,0,N,1,0}$ and $\text{XEX}^{*,0,N,1,1}$. Thus both $\text{XEX}^{*,1,N,1,0}$ and $\text{XEX}^{*,0,N,1,0}$ are queried, which implies a violation of the second condition of Definition 1. Consequently, the authenticity proof of Rog04 does not work, hence our attacks. At the same time, this also implies that the privacy (confidentiality) attack *under CPA*, i.e. distinguishing the ciphertext from random using only encryption queries, is not possible. This shows a sharp difference between CPA and CCA queries, where the latter easily break confidentiality (see Sec. 4.3).

8 Applicability to Related Schemes

Other OCB Versions. To the best of our knowledge, our attacks do not apply to OCB1 and OCB3. For OCB1, the last block is encrypted by XE with a clearly separated mask. For OCB3, the last block is

[19] We note that the constant 9.5 in $\mathbf{Adv}^{\mathsf{tsprp}}_{\text{XEX}^*_{\widetilde{\mathsf{P}}}}(\mathcal{B}_\pm)$ in (7) was improved to 4.5 in [29].

encrypted by XEX when it is n bits long, and otherwise by XE with a mask separated from those used by XEX.

Designs based on OCB. We have not found other AE algorithms based on OCB that could be affected by our attacks. OTR [27] is an inverse-free (for the absence of the blockcipher decryption in the scheme) parallelizable AE mode that has a similar structure as OCB. As it only uses XE for the whole process, it is safe from our attacks. OPP [14] is a permutation-based AE based on OCB. It always uses XEX, or more precisely, a variant of XPX [26], because otherwise an offline permutation inverse query easily breaks the scheme. It is safe because of this consistent use of XPX. ZOCB [4] is a TBC-based AE whose structure is similar to ΘCB2. Unlike ΘCB2, it utilizes mask values applied to the underlying TBC, for a combined, faster AD processing than ΘCB2. This makes ZOCB also similar to OCB2. It adopts (a variant of) XEX for the last message block, hence our attacks do not apply. Finally, OCB-hc [17] is a revised version of OCB2 that has a smaller state size. It is safe since it adopts XEX for the last message block.

9 Fixing OCB2

In this section, we discuss several ways to prevent our attacks in practice. In principle, each of our suggestions would require its own formal security analysis. However, we provide one only for the "XEX for the last plaintext block" fix presented in Sec. 9.1 and the "XE for the last message block" fix presented in Sec. 9.2. Our other proposals intuitively lead to a secure scheme, however, without conducting further research we cannot fully vouch for their security because these proposals do not allow the proof strategy that we adopt for OCB2f.[20] That is, the abstraction by TBC cannot work anymore.

Always using AD. Our forgery attacks from Sec. 4 have the property that the AD of the forgeries have to be the empty string. This was unavoidable as for $A \neq \varepsilon$ we would have had to predict $\mathrm{PMAC}_E(A)$ but we are not aware of a way to do so. (Of course, if we could use the VecEncipher algorithm of Sec. 5.2 then computing PMAC values is not a challenge; however, VecEncipher can only be invoked after SamplePairs, and the latter implicitly conducts a forgery with $A = \varepsilon$.) Overall we note that a forgery with $A = \varepsilon$ is a key component of *all* our attacks on OCB2. This observation immediately suggests a fix: If the involved users agree that all encryption/decryption operations are with respect to a non-empty AD, then it seems (to us) that all problems go away. An easy way to implement this strategy generically is to prepend a fixed string (e.g. the single letter "A" or the all-zero block 0^n) to every occurring AD (including the empty AD).

Always using PMAC. Recall from line 10 of $\mathcal{E}_E$ and line 10 of $\mathcal{D}_E$ in Fig. 1 that $\mathrm{PMAC}_E(A)$ is XOR-ed into the tag only if $A \neq \varepsilon$. We discuss the case that this condition is removed, and $\mathrm{PMAC}_E(A)$ is *always* XOR-ed into the tag, also when $A = \varepsilon$. An initial analysis of the PMAC algorithm (see Fig. 15 in Appendix B) shows that the value $\mathrm{PMAC}_E(\varepsilon)$ is unpredictable, and also cannot be replayed from other ciphertexts, so that also this modification of OCB2 promises to be a secure candidate.

Counter-cryptanalysis. The two countermeasures just discussed require that the code of both the sender and the receiver would have to be adapted. It might be impossible to do so for instance if OCB2 is included in already shipped products that cannot be updated remotely. In such settings the following two options might be interesting: The sender is modified to never encrypt a message where the second-last block is $\mathtt{len}(0^n)$ while the receiver remains unchanged, or the sender remains unchanged and the receiver is modified to never decrypt to a message where the last block would be of the form $2^m L \oplus \mathtt{len}(0^n)$.[21] While such changes would (marginally) influence the correctness of the encryption scheme, they seem to make our attacks impossible. To patch a live system this might be a viable option.

[20] Recent work on AEAD combiners [32] suggests operating multiple AEAD schemes and combining their results, with the effect that the result is secure if at least one of the ingredient schemes is. This approach might be interesting if the unproven methods proposed here are used.

[21] We caution that this change might not be sufficient. Our results from Sec. 4.4 indicate that more plaintexts and ciphertexts have to be rejected: on the encryptor's side all messages with $M[m-1] = \mathtt{len}(0^{n-s})$ for some $s = 1, \ldots, n$, and on the decryptor's side all ciphertexts that would result, when decrypted, in $M^*[m-1] = \mathtt{len}(0^{n-s})$ for some $s = 1, \ldots, n$. We are still investigating which conditions would be necessary/sufficient for security.

USE OF XEX$^+$. Minematsu and Matsushima [29] proposed an extension of XEX* called XEX$^+$. The latter allows to use plain blockcipher calls in combination with XEX and XE. The authors in particular suggest how to use XEX$^+$ to instantiate a variant of OCB, where the last message block is encrypted by an unmasked blockcipher. This variant of OCB is not affected by our attacks and provably secure.

9.1 XEX for The Last Message Block

Recall that the vulnerabilities of OCB2 stem from a bad interaction of the XE and XEX components in XEX* and the fact that XE is used for the last block of encryption. A simple way to fix OCB2 is to use XEX also for the last block. We call the resulting scheme OCB2f. Its pseudocode is obtained by changing line 5 of $\mathcal{E}_E$ in Fig. 1 to

$$\text{Pad} \leftarrow 2^m L \oplus E(2^m L \oplus \mathtt{len}(M[m]))$$

and line 5 of $\mathcal{D}_E$ to

$$\text{Pad} \leftarrow 2^m L \oplus E(2^m L \oplus \mathtt{len}(C[m])).$$

Similarly to OCB2, OCB2f is a mode of XEX*, since the tweak spaces of XE and XEX in OCB2f are distinct. Specifically, we define $\Theta\text{CB2f}_{\widetilde{E}}$ as a mode obtained by changing $\widetilde{E}^{*,0,N,m,0}$ to $\widetilde{E}^{*,1,N,m,0}$ in line 4 of $\Theta\text{CB2}.\mathcal{E}_{\widetilde{E}}$ and in line 4 of $\Theta\text{CB2}.\mathcal{D}_{\widetilde{E}}$ in Fig. 12. Then $\Theta\text{CB2f}_{\widetilde{E}}$ is equivalent to OCB2f_E if $\widetilde{E}_K$ is XEX^*_E. To handle the case of non-empty AD, we also define $\mathbb{PMAC}_{\widetilde{E}}$ as a mode of TBC $\widetilde{E}_K$ defined in the same way as ΘCB2 (see Fig. 15 in Appendix B). As Fig. 15 shows, $\mathbb{PMAC}_{\text{XEX}^*_E}$ is equivalent to PMAC_E . We finally add

$$\begin{array}{l}\text{if } A \neq \varepsilon \text{ then } T \leftarrow \mathtt{msb}_\tau(T \oplus \mathbb{PMAC}_{\widetilde{E}}(A))\\ \text{else } T \leftarrow \mathtt{msb}_\tau(T)\end{array}$$

after line 8 in $\Theta\text{CB2}.\mathcal{E}_{\widetilde{E}}$ in Fig. 12 and

$$\begin{array}{l}\text{if } A \neq \varepsilon \text{ then } T^* \leftarrow \mathtt{msb}_\tau(T^* \oplus \mathbb{PMAC}_{\widetilde{E}}(A))\\ \text{else } T^* \leftarrow \mathtt{msb}_\tau(T^*)\end{array}$$

after line 8 in $\Theta\text{CB2}.\mathcal{D}_{\widetilde{E}}$ to make it AEAD. We prove the security of OCB2f using a hybrid argument involving ΘCB2f. To simplify the argument, we also define $\Theta\text{CB2f}'$ by converting $\mathbb{PMAC}_{\widetilde{E}}$ in ΘCB2f to a URF (uniform random function) $\mathsf{R} : \{0,1\}^* \to \{0,1\}^n$. The security bounds of OCB2f are the same as those claimed for OCB2:

Theorem 1. *Let $\mathcal{A}$ and $\mathcal{A}_\pm$ denote the adversaries against AEAD in the privacy and authenticity games. We assume $\mathcal{A}_\pm$ uses q_v decryption queries. We have*

$$\begin{aligned}\mathbf{Adv}^{\mathsf{priv}}_{\text{OCB2f}_\mathsf{P}}(\mathcal{A}) &= \mathbf{Adv}^{\mathsf{priv}}_{\Theta\text{CB2f}_{\text{XEX}^*_\mathsf{P}}}(\mathcal{A}) \le \frac{5\sigma^2_{\text{priv}}}{2^n},\\ \mathbf{Adv}^{\mathsf{auth}}_{\text{OCB2f}_\mathsf{P}}(\mathcal{A}_\pm) &= \mathbf{Adv}^{\mathsf{auth}}_{\Theta\text{CB2f}_{\text{XEX}^*_\mathsf{P}}}(\mathcal{A}_\pm) \le \frac{5\sigma^2_{\text{auth}}}{2^n} + \frac{4q_v}{2^\tau},\end{aligned}$$

where σ_{priv} and σ_{auth} are the number of queried blocks (the number of invocations of XEX**) in the privacy game and the authenticity game, respectively.*

Intuitively, the security of OCB2f holds because (1) OCB2f is ΘCB2f using $\widetilde{E}$ instantiated by XEX*, and (2) ΘCB2f and $\Theta\text{CB2f}'$ are indistinguishable (up to collision), and (3) $\Theta\text{CB2f}'$ in the privacy and authenticity games do not force the adversary to violate the access rules (Definition 1). Combining the known bounds of XEX* and $\mathbb{PMAC}_{\widetilde{E}}$ and the proofs of $\Theta\text{CB2}_{\widetilde{\mathsf{P}}}$ with minor changes gives the desired results. A full proof is given in Appendix C.

9.2 XE for The Last Message Block

In Sec. 9.1, we use XEX for the last block instead of XE. This simple fix, however, does not respect (what we believe) the original concept of OCB2, that is, the TBC should avoid the use of outer XOR of mask when it is not needed. Here, we propose another way to fix OCB2 that uses XE for the last block, but with a different constant from the original to thwart attacks. As we have a large number of possible options, listing all of them is rather impractical. Instead, we present several representative options that require minimum modifications to the original algorithms.

To start with, let us redefine XEX* as follows:

$$\mathrm{XEX}^{*,b,N,i_1,i_2,i_3}_E(X) = \begin{cases} E(\alpha_1^{i_1}\alpha_2^{i_2}\alpha_3^{i_3}L \oplus X) \oplus \alpha_1^{i_1}\alpha_2^{i_2}\alpha_3^{i_3}L & \text{if } b = 1; \\ E(\alpha_1^{i_1}\alpha_2^{i_2}\alpha_3^{i_3}L \oplus X) & \text{if } b = 0, \end{cases}$$

where $L = E(N)$, $\alpha_1, \alpha_2, \alpha_3 \in (\mathrm{GF}(2^n))^\times$, and $(i_1, i_2, i_3) \in \mathbb{I}_1 \times \mathbb{I}_2 \times \mathbb{I}_3 \subset \mathbb{Z}^3$. The specification of XEX* in OCB2 uses $\alpha_1 = 2$ and $\alpha_2 = 3$, and there is no α_3. The proper definitions of the parameters α_1, α_2, α_3 and $\mathbb{I}_1$, $\mathbb{I}_2$, $\mathbb{I}_3$ depend on n and the primitive polynomial defining $\mathrm{GF}(2^n)$. In any case, the mapping $\Phi_{\alpha_1,\alpha_2,\alpha_3} : \mathbb{I}_1 \times \mathbb{I}_2 \times \mathbb{I}_3 \to (\mathrm{GF}(2^n))^\times$, where

$$\Phi_{\alpha_1,\alpha_2,\alpha_3}(i_1, i_2, i_3) := \alpha_1^{i_1}\alpha_2^{i_2}\alpha_3^{i_3}, \tag{8}$$

must be injective. We also need to exclude $(i_1, i_2, i_3) = (0, 0, 0)$ to thwart a chosen-ciphertext attack against XEX, as in the case of the original XEX*.

OCB2 can be fixed by properly setting the above index vector (i_1, i_2, i_3) used for the last message block and the tag generation, where both use $b = 0$ (XE). We detail possible instantiations for some concrete values of n.

Case of $n = 128$. Rog04 showed that, when $\mathrm{GF}(2^{128}) = \mathrm{GF}(2)[x]/(x^{128}+x^7+x^2+x+1)$, the index set $(\alpha_1, \alpha_2, \alpha_3) = (2, 3, 7)$ and the domain $\mathbb{I}_1 \times \mathbb{I}_2 \times \mathbb{I}_3 = \{-2^{108}, \ldots, 2^{108}\} \times \{-2^7, \ldots, 2^7\} \times \{-2^7, \ldots, 2^7\}$ yield a secure instance of XEX*. Therefore, we can change $\mathcal{E}_E$ in Fig. 1 as follows:

$$\begin{aligned} &\text{line 5: Pad} \leftarrow E(2^{m-1}3L \oplus \mathtt{len}(M[m])), \\ &\text{line 9: } T \leftarrow E(2^{m-1}3 \cdot 7L \oplus \Sigma). \end{aligned}$$

Alternatively, we could use Pad $\leftarrow E(2^{m-1}7L \oplus \mathtt{len}(M[m]))$ in line 5. $\mathcal{D}_E$ is changed accordingly.

Case of $n = 64$. When $\mathrm{GF}(2^{64}) = \mathrm{GF}(2)[x]/(x^{64}+x^4+x^3+x+1)$, the index set $(\alpha_1, \alpha_2, \alpha_3) = (2, 3, 11)$ and the domain $\mathbb{I}_1 \times \mathbb{I}_2 \times \mathbb{I}_3 = \{-2^{44}, \ldots, 2^{44}\} \times \{-2^7, \ldots, 2^7\} \times \{-2^7, \ldots, 2^7\}$ yield a secure setting as shown by Rog04. The element $7 \in \mathrm{GF}(2^{64})$ cannot be used because $2^{64} = 3^2 \cdot 7$ holds over $\mathrm{GF}(2^{64})$. Similarly to the case of $n = 128$, we can change $\mathcal{E}_E$ in Fig. 1 as follows:

$$\begin{aligned} &\text{line 5: Pad} \leftarrow E(2^{m-1}3L \oplus \mathtt{len}(M[m])), \\ &\text{line 9: } T \leftarrow E(2^{m-1}3 \cdot 11L \oplus \Sigma). \end{aligned}$$

Alternatively, we could use Pad $\leftarrow E(2^{m-1}11L \oplus \mathtt{len}(M[m]))$ in line 5, and the decryption is changed accordingly.

Generalizing the above two cases, we define a fix of OCB2, called OCB2ff, in Fig. 13. Assuming $|\mathbb{I}_1|$ is sufficiently large, OCB2ff needs at least $\{0, 1, 2, 3, 4\} \times \{0, 1\} \subseteq \mathbb{I}_2 \times \mathbb{I}_3$. Similarly to OCB2f, OCB2ff can be interpreted as a mode of TBC, which we call ΘCB2ff as described in Fig. 14. ΘCB2ff equals to OCB2ff when the inner TBC $\widetilde{E}$ is XEX*. Since the tweak spaces of XE and XEX in OCB2ff are disjoint, the adversary against XEX* can simulate the privacy and authenticity games of ΘCB2ff without violating the tag-respecting rule. Therefore, we can prove the security of OCB2ff by using the hybrid argument. We remark that the proof is similar to that of OCB2f, although it is not identical due to the difference in the last block encryption; in OCB2f, the last block encryption is not explicitly separated (by tweaks) from the non-last block encryptions, while OCB2ff explicitly separates them.

The security bounds are the same as OCB2f. We present them for completeness:

Algorithm OCB2ff.$\mathcal{E}_E(N, A, M)$

1. $L \leftarrow E(N)$
2. $(M[1], \ldots, M[m]) \xleftarrow{n} M$
3. **for** $i \leftarrow 1$ **to** $m-1$
4. $\quad C[i] \leftarrow \alpha_1^i L \oplus E(\alpha_1^i L \oplus M[i])$
5. $\mathrm{Pad} \leftarrow E(\alpha_1^{m-1} \cdot \alpha_2 L \oplus \mathtt{len}(M[m]))$
6. $C[m] \leftarrow M[m] \oplus \mathtt{msb}_{|M[m]|}(\mathrm{Pad})$
7. $\Sigma \leftarrow C[m] \,\|\, 0^* \oplus \mathrm{Pad}$
8. $\Sigma \leftarrow M[1] \oplus \cdots \oplus M[m-1] \oplus \Sigma$
9. $T \leftarrow E(\alpha_1^{m-1} \cdot \alpha_2 \cdot \alpha_3 L \oplus \Sigma)$
10. **if** $A \neq \varepsilon$ **then** $T \leftarrow T \oplus \mathrm{PMAC}_E(A)$
11. $T \leftarrow \mathtt{msb}_\tau(T)$
12. **return** (C, T)

Algorithm OCB2ff.$\mathcal{D}_E(N, A, C, T)$

1. $L \leftarrow E(N)$
2. $(C[1], \ldots, C[m]) \xleftarrow{n} C$
3. **for** $i \leftarrow 1$ **to** $m-1$
4. $\quad M[i] \leftarrow \alpha_1^i L \oplus E^{-1}(\alpha_1^i L \oplus C[i])$
5. $\mathrm{Pad} \leftarrow E(\alpha_1^{m-1} \cdot \alpha_2 L \oplus \mathtt{len}(C[m]))$
6. $M[m] \leftarrow C[m] \oplus \mathtt{msb}_{|C[m]|}(\mathrm{Pad})$
7. $\Sigma \leftarrow C[m] \,\|\, 0^* \oplus \mathrm{Pad}$
8. $\Sigma \leftarrow M[1] \oplus \cdots \oplus M[m-1] \oplus \Sigma$
9. $T^* \leftarrow E(\alpha_1^{m-1} \cdot \alpha_2 \cdot \alpha_3 L \oplus \Sigma)$
10. **if** $A \neq \varepsilon$ **then** $T^* \leftarrow T^* \oplus \mathrm{PMAC}_E(A)$
11. $T^* \leftarrow \mathtt{msb}_\tau(T^*)$
12. **if** $T = T^*$ **return** M
13. **else return** $\perp$

Fig. 13. Algorithms of OCB2ff.

Algorithm ΘCB2ff.$\mathcal{E}^{\widetilde{E}}(N, A, M)$

1. $(M[1], \ldots, M[m]) \xleftarrow{n} M$
2. **for** $i = 1$ **to** $m-1$
3. $\quad C[i] \leftarrow \widetilde{E}^{*,1,N,i,0,0}(M[i])$
4. $\mathrm{Pad} \leftarrow \widetilde{E}^{*,0,N,m-1,1,0}(\mathtt{len}(M[m]))$
5. $C[m] \leftarrow M[m] \oplus \mathtt{msb}_{|M[m]|}(\mathrm{Pad})$
6. $\Sigma \leftarrow C[m] \,\|\, 0^* \oplus \mathrm{Pad}$
7. $\Sigma \leftarrow M[1] \oplus \cdots \oplus M[m-1] \oplus \Sigma$
8. $T \leftarrow \widetilde{E}^{*,0,N,m-1,1,1}(\Sigma)$
9. **if** $A \neq \varepsilon$ **then** $T \leftarrow T \oplus \mathbb{PMAC}^{\widetilde{E}}(A)$
10. $T \leftarrow \mathtt{msb}_\tau(T)$
11. **return** (C, T)

Algorithm ΘCB2ff.$\mathcal{D}^{\widetilde{E}}(N, A, C, T)$

1. $(C[1], \ldots, C[m]) \xleftarrow{n} C$
2. **for** $i = 1$ **to** $m-1$
3. $\quad M[i] \leftarrow (\widetilde{E}^{*,1,N,i,0,0})^{-1}(C[i])$
4. $\mathrm{Pad} \leftarrow \widetilde{E}^{*,0,N,m-1,1,0}(\mathtt{len}(C[m]))$
5. $M[m] \leftarrow C[m] \oplus \mathtt{msb}_{|C[m]|}(\mathrm{Pad})$
6. $\Sigma \leftarrow C[m] \,\|\, 0^* \oplus \mathrm{Pad}$
7. $\Sigma \leftarrow M[1] \oplus \cdots \oplus M[m-1] \oplus \Sigma$
8. $T^* \leftarrow \widetilde{E}^{*,0,N,m-1,1,1}(\Sigma)$
9. **if** $A \neq \varepsilon$ **then** $T^* \leftarrow T^* \oplus \mathbb{PMAC}^{\widetilde{E}}(A)$
10. $T^* \leftarrow \mathtt{msb}_\tau(T^*)$
11. **if** $T = T^*$ **return** M
12. **else return** $\perp$

Fig. 14. Algorithms of ΘCB2ff. $\widetilde{E}$ is any TBC which has the same arguments as XEX*.

Theorem 2. *Let $\mathcal{A}$ and $\mathcal{A}_\pm$ denote the adversaries against AEAD in the privacy and authenticity games. We assume $\mathcal{A}_\pm$ uses q_v decryption queries. We have*

$$\mathbf{Adv}^{\mathsf{priv}}_{\mathrm{OCB2ff}_\mathsf{P}}(\mathcal{A}) = \mathbf{Adv}^{\mathsf{priv}}_{\Theta\mathrm{CB2ff}_{\mathrm{XEX}^*_\mathsf{P}}}(\mathcal{A}) \leq \frac{5\sigma^2_{\mathrm{priv}}}{2^n},$$
$$\mathbf{Adv}^{\mathsf{auth}}_{\mathrm{OCB2ff}_\mathsf{P}}(\mathcal{A}_\pm) = \mathbf{Adv}^{\mathsf{auth}}_{\Theta\mathrm{CB2ff}_{\mathrm{XEX}^*_\mathsf{P}}}(\mathcal{A}_\pm) \leq \frac{5\sigma^2_{\mathrm{auth}}}{2^n} + \frac{4q_v}{2^\tau},$$

where σ_{priv} and σ_{auth} are the number of queried blocks (the number of invocations of XEX**) in the privacy game and the authenticity game, respectively.*

A proof is presented in Appendix D.

Further Options. We close this section by briefly describing some possible options when the tweak set does not satisfy the condition for OCB2ff. Unlike the previous options, we need to do more than changing two lines. First, if the function Φ in (8) does not have the third element α_3 and $\{0, 1, 2, 3, 4, 5\} \subseteq \mathbb{I}_2$ holds, where the latter condition is to implement PMAC, we modify the algorithm of OCB2ff.$\mathcal{E}_E$ in Fig. 13 as follows:

$$\text{line 9: } T \leftarrow E(\alpha_1^{m-1} \cdot \alpha_2^2 L \oplus \Sigma).$$

We also modify the decryption accordingly, and moreover, the algorithm of PMAC_E in Fig. 15 should be changed as follows:

$$\begin{aligned}
&\text{line 2: } V \leftarrow \alpha_2^3 E(0^n),\\
&\text{line 5: } S \leftarrow S \oplus E(\alpha_1^i V \oplus A[i]),\\
&\text{line 8: } Q \leftarrow E(\alpha_1^{a-1} \cdot \alpha_2 V \oplus S),\\
&\text{line 9: } \textbf{else } Q \leftarrow E(\alpha_1^{a-1} \cdot \alpha_2^2 V \oplus S).
\end{aligned}$$

With these modifications, the security can be proved in the same manner as OCB2ff, at the cost of additional change to PMAC. This fix works for $n = 64$ and $n = 128$ by using the concrete examples by Rog04. However, to the best of our knowledge, it does not work with other masking functions for $n \notin \{64, 128\}$ known in literature. For example, Minalpher [39] (a candidate of CAESAR competition) defines a masking function for $n = 256$, and it does not have the third element and $\mathbb{I}_2 = \{0, 1, 2\}$. In such a case, we can compensate for the lack of the elements of $\mathbb{I}_2$ by excluding 0^n from the possible nonce values. This separates the AE part from the MAC part in terms of the tweaks, and it allows to instantiate a fix of OCB2 with a masking function with two elements α_1, α_2 and $\mathbb{I}_2 = \{0, 1, 2\}$.

In addition, we can interpret α_1, α_2, α_3 as maps on $\{0,1\}^n$. When $n = 1024$, Granger et al. [14] proposed word-based masking functions to define a permutation-based analog of OCB. Suppose that $(x_0, \ldots, x_{15}) \in \{0,1\}^{1024}$, where $x_0, \ldots, x_{15} \in \{0,1\}^{64}$. Let $\alpha : \{0,1\}^{1024} \to \{0,1\}^{1024}$ be the linear map such that

$$(x_0, \ldots, x_{15}) \mapsto (x_1, x_2, \ldots, x_{15}, (x_0 \lll 53)) \oplus (x_5 \ll 13)),$$

where $(x \lll \ell)$ denotes the left-rotation of x by ℓ bits and $(x \ll \ell)$ denotes the left-shift of x by ℓ bits. Accordingly, a binary 1024×1024 matrix M such that $\alpha(x) = M \cdot x$ can be defined. Granger et al. defined the mask functions and the corresponding domains as follows.

$$\begin{aligned}
&\alpha_1^{i_1}(x) := M^{i_1} \cdot x, \quad \alpha_2^{i_2}(x) := (M + I)^{i_2} \cdot x, \quad \alpha_3^{i_3}(x) := (M^2 + M + I)^{i_3} \cdot x,\\
&(i_1, i_2, i_3) \in \mathbb{I}_1 \times \mathbb{I}_2 \times \mathbb{I}_3 = \{0, 1, \ldots, 2^{1020} - 1\} \times \{0, 1, 2, 3\} \times \{0, 1\},\\
&\Phi_{\alpha_1, \alpha_2, \alpha_3} : (i_1, i_2, i_3) \mapsto \alpha_1^{i_1} \circ \alpha_2^{i_2} \circ \alpha_3^{i_3},
\end{aligned}$$

where I denotes the identity matrix. Although the above masking function cannot be used to instantiate OCB2ff since $\{0, 1, 2, 3, 4\} \nsubseteq \mathbb{I}_2$, we can fix OCB2 with this masking function in a similar fashion, say by seeing the pair (α_2, α_3) as the second element of the tweak that has $4 \times 2 = 8$ variations and adopting the two-element solution described earlier.

Finally, Elephant family [7] from the round 2 of NIST Lightweight Cryptography Standardization project[22] also introduces a set of word-based masking functions for $n \in \{160, 176, 200\}$. Their masking function does not have the third element α_3 and $\mathbb{I}_2 = \{0, 1, 2\}$, thus we have to use the same technique as in the case of Minalpher.

10 Conclusions

We have presented practical forgery and decryption attacks against OCB2, a high-profile ISO-standard authenticated encryption scheme. This was possible due to the discrepancy between the proof of OCB2 and the actual construction, in particular the interpretation of OCB2 as a mode of a TBC which combines XEX and XE. While the latest OCB version, OCB3, has a superior software performance than the previous versions, and is clearly recommended by the designers, we believe OCB2 is still quite influential for its simple description and the sophisticated modular design based on a TBC. Our attacks show that, while the approach introduced by Rog04 is invaluable, we could not directly derive a secure AE from it without applying a fix.

We comment that, due to errors in proofs, 'provably-secure schemes' sometimes still can be broken, or schemes remain secure but nevertheless the proofs need to be fixed. Even if we limit our focus to AE, we have many examples for this, such as NSA's Dual CTR [37,11], EAX-prime [28], GCM [22], and some of the CAESAR submissions [30,10,40]. We believe our work emphasizes the need for quality of security proofs, and their active verification.

[22] `https://csrc.nist.gov/Projects/Lightweight-Cryptography`

Acknowledgments

The authors would like to thank Phil Rogaway for his response to our findings. We further thank officials of ISO SC 27, in particular Chris Mitchell, for feedback and suggestions. We also would like to thank the reviewers of the Journal of Cryptology, and the reviewers of CRYPTO 2019, for their useful comments and thorough reviews. We further appreciate the comments of Eamonn Postlethwaite on an early version of this article.

References

1. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., Yasuda, K.: How to securely release unverified plaintext in authenticated encryption. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part I. LNCS, vol. 8873, pp. 105–125. Springer, Heidelberg, Germany, Kaoshiung, Taiwan, R.O.C. (Dec 7–11, 2014)
2. Aoki, K., Yasuda, K.: The security of the OCB mode of operation without the SPRP assumption. In: Susilo, W., Reyhanitabar, R. (eds.) ProvSec 2013. LNCS, vol. 8209, pp. 202–220. Springer, Heidelberg, Germany, Melaka, Malaysia (Oct 23–25, 2013)
3. Ashur, T., Dunkelman, O., Luykx, A.: Boosting authenticated encryption robustness with minimal modifications. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part III. LNCS, vol. 10403, pp. 3–33. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2017)
4. Bao, Z., Guo, J., Iwata, T., Minematsu, K.: ZOCB and ZOTR: Tweakable blockcipher modes for authenticated encryption with full absorption. IACR Trans. Symm. Cryptol. 2019(2), 1–54 (2019)
5. Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A concrete security treatment of symmetric encryption. In: 38th FOCS. pp. 394–403. IEEE Computer Society Press, Miami Beach, Florida (Oct 19–22, 1997)
6. Bellare, M., Rogaway, P., Wagner, D.: The EAX mode of operation. In: Roy, B.K., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 389–407. Springer, Heidelberg, Germany, New Delhi, India (Feb 5–7, 2004)
7. Beyne, T., Chen, Y.L., Dobraunig, C., Mennink, B.: Elephant. Tech. rep., National Institute of Standards and Technology (2019), available at `https://csrc.nist.gov/Projects/lightweight-cryptography/round-2-candidates`
8. Black, J., Cochran, M.: MAC reforgeability. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 345–362. Springer, Heidelberg, Germany, Leuven, Belgium (Feb 22–25, 2009)
9. Black, J., Rogaway, P.: A block-cipher mode of operation for parallelizable message authentication. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 384–397. Springer, Heidelberg, Germany, Amsterdam, The Netherlands (Apr 28 – May 2, 2002)
10. Bost, R., Sanders, O.: Trick or tweak: On the (in)security of OTR's tweaks. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part I. LNCS, vol. 10031, pp. 333–353. Springer, Heidelberg, Germany, Hanoi, Vietnam (Dec 4–8, 2016)
11. Donescu, P., Gligor, V.D., Wagner, D.: A note on NSA's Dual Counter Mode of encryption (2001), `http://www.cs.berkeley.edu/~daw/papers/dcm-prelim.ps`
12. Ferguson, N.: Collision attacks on OCB. Comments to NIST (2002), `https://csrc.nist.gov/CSRC/media/Projects/Block-Cipher-Techniques/documents/BCM/Comments/general-comments/papers/Ferguson.pdf`
13. Forler, C., List, E., Lucks, S., Wenzel, J.: Reforgeability of authenticated encryption schemes. In: Pieprzyk, J., Suriadi, S. (eds.) ACISP 17, Part II. LNCS, vol. 10343, pp. 19–37. Springer, Heidelberg, Germany, Auckland, New Zealand (Jul 3–5, 2017)
14. Granger, R., Jovanovic, P., Mennink, B., Neves, S.: Improved masking for tweakable blockciphers with applications to authenticated encryption. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part I. LNCS, vol. 9665, pp. 263–293. Springer, Heidelberg, Germany, Vienna, Austria (May 8–12, 2016)
15. Inoue, A., Iwata, T., Minematsu, K., Poettering, B.: Cryptanalysis of OCB2: Attacks on authenticity and confidentiality. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part I. LNCS, vol. 11692, pp. 3–31. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2019)
16. Inoue, A., Minematsu, K.: Cryptanalysis of OCB2. Cryptology ePrint Archive, Report 2018/1040 (2018), `https://eprint.iacr.org/2018/1040`
17. Inoue, A., Minematsu, K.: Parallelizable authenticated encryption with small state size. In: Paterson, K.G., Stebila, D. (eds.) SAC 2019. LNCS, vol. 11959, pp. 618–644. Springer, Heidelberg, Germany, Waterloo, ON, Canada (Aug 12–16, 2019)
18. ISO: Information Technology - Security techniques - Authenticated encryption, ISO/IEC 19772:2009. International Standard ISO/IEC 19772 (2009)
19. ISO/IEC JTC 1/SC 27: STATEMENT ON OCB2.0 – Major weakness found in a standardised cipher scheme (2019-01-09, press release), `https://www.din.de/blob/321470/da3d9bce7116deb510f6aded2ed0b4df/20190107-press-release-19772-2009-1st-ed-ocb2-0-data.pdf`

20. Iwata, T.: Plaintext recovery attack of OCB2. Cryptology ePrint Archive, Report 2018/1090 (2018), https://eprint.iacr.org/2018/1090
21. Iwata, T., Kurosawa, K.: OMAC: One-key CBC MAC. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 129–153. Springer, Heidelberg, Germany, Lund, Sweden (Feb 24–26, 2003)
22. Iwata, T., Ohashi, K., Minematsu, K.: Breaking and repairing GCM security proofs. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 31–49. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2012)
23. Krovetz, T., Rogaway, P.: The software performance of authenticated-encryption modes. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 306–327. Springer, Heidelberg, Germany, Lyngby, Denmark (Feb 13–16, 2011)
24. Krovetz, T., Rogaway, P.: The OCB Authenticated-Encryption Algorithm. IRTF RFC 7253 (2014)
25. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 31–46. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2002)
26. Mennink, B.: XPX: Generalized tweakable Even-Mansour with improved security guarantees. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 64–94. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2016)
27. Minematsu, K.: Parallelizable rate-1 authenticated encryption from pseudorandom functions. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 275–292. Springer, Heidelberg, Germany, Copenhagen, Denmark (May 11–15, 2014)
28. Minematsu, K., Lucks, S., Morita, H., Iwata, T.: Attacks and security proofs of EAX-prime. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 327–347. Springer, Heidelberg, Germany, Singapore (Mar 11–13, 2014)
29. Minematsu, K., Matsushima, T.: Generalization and Extension of XEX* Mode. IEICE Transactions 92-A(2), 517–524 (2009)
30. Nandi, M.: Forging attacks on two authenticated encryption schemes COBRA and POET. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part I. LNCS, vol. 8873, pp. 126–140. Springer, Heidelberg, Germany, Kaoshiung, Taiwan, R.O.C. (Dec 7–11, 2014)
31. Poettering, B.: Breaking the confidentiality of OCB2. Cryptology ePrint Archive, Report 2018/1087 (2018), https://eprint.iacr.org/2018/1087
32. Poettering, B., Rösler, P.: Combiners for AEAD. IACR Trans. Symm. Cryptol. 2020(1), 121–143 (2020), https://doi.org/10.13154/tosc.v2020.i1.121-143
33. Rogaway, P.: Authenticated-encryption with associated-data. In: Atluri, V. (ed.) ACM CCS 2002. pp. 98–107. ACM Press, Washington, DC, USA (Nov 18–22, 2002)
34. Rogaway, P.: Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 16–31. Springer, Heidelberg, Germany, Jeju Island, Korea (Dec 5–9, 2004)
35. Rogaway, P.: Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. Full version of [34] (2004), available from http://www.cs.ucdavis.edu/~rogaway/papers/
36. Rogaway, P.: Nonce-based symmetric encryption. In: Roy, B.K., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 348–359. Springer, Heidelberg, Germany, New Delhi, India (Feb 5–7, 2004)
37. Rogaway, P.: On the Role Definitions in and Beyond Cryptography. In: ASIAN. LNCS, vol. 3321, pp. 13–32. Springer (2004)
38. Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB: A block-cipher mode of operation for efficient authenticated encryption. In: Reiter, M.K., Samarati, P. (eds.) ACM CCS 2001. pp. 196–205. ACM Press, Philadelphia, PA, USA (Nov 5–8, 2001)
39. Sasaki, Y., Todo, Y., Aoki, K., Naito, Y., Sugawara, T., Murakami, Y., Matsui, M., Hirose, S.: Minalpher (A submission to CAESAR), https://info.isl.ntt.co.jp/crypt/minalpher/files/minalpherv11.pdf
40. Schroé, W., Mennink, B., Andreeva, E., Preneel, B.: Forgery and subkey recovery on CAESAR candidate iFeed. In: Dunkelman, O., Keliher, L. (eds.) SAC 2015. LNCS, vol. 9566, pp. 197–204. Springer, Heidelberg, Germany, Sackville, NB, Canada (Aug 12–14, 2016)
41. Sun, Z., Wang, P., Zhang, L.: Collision attacks on variant of OCB mode and its series. In: Inscrypt. LNCS, vol. 7763, pp. 216–224. Springer (2012)
42. Vaudenay, S., Vizár, D.: Can caesar beat galois? - Robustness of CAESAR candidates against nonce reusing and high data complexity attacks. In: Preneel, B., Vercauteren, F. (eds.) ACNS 18. LNCS, vol. 10892, pp. 476–494. Springer, Heidelberg, Germany, Leuven, Belgium (Jul 2–4, 2018)

A Brief History of This Paper

A frequent question we have received is how we came to find the flaws, and how they lead to the devastating attacks. The current article is based on three prior ones [16,31,20] that appeared in late 2018 on the IACR ePrint archive. That OCB2 might be flawed was first identified by the authors of [16] when

Algorithm $\mathrm{PMAC}_E(A)$	**Algorithm** $\mathbb{PMAC}_{\widetilde{E}}(A)$
1. $S \leftarrow 0^n$	1. $S \leftarrow 0^n$
2. $V \leftarrow 3^2 E(0^n)$	2. $(A[1], \ldots, A[a]) \xleftarrow{n} A$
3. $(A[1], \ldots, A[a]) \xleftarrow{n} A$	3. **for** $i \leftarrow 1$ **to** $a-1$
4. **for** $i \leftarrow 1$ **to** $a-1$	4. $\quad S \leftarrow S \oplus \widetilde{E}^{*,0,0^n,i,2}(A[i])$
5. $\quad S \leftarrow S \oplus E(2^i V \oplus A[i])$	5. $S \leftarrow S \oplus A[a] \,\|\, 10^*$
6. $S \leftarrow S \oplus A[a] \,\|\, 10^*$	6. **if** $\lvert A[a]\rvert = n$
7. **if** $\lvert A[a]\rvert = n$	7. $\quad Q \leftarrow \widetilde{E}^{*,0,0^n,a,3}(S)$
8. $\quad Q \leftarrow E(2^a 3 V \oplus S)$	8. **else** $Q \leftarrow \widetilde{E}^{*,0,0^n,a,4}(S)$
9. **else** $Q \leftarrow E(2^a 3^2 V \oplus S)$	9. **return** Q
10. **return** Q	

Fig. 15. Left: The algorithm PMAC_E for the use in OCB2. **Right:** A TBC-based PMAC, $\mathbb{PMAC}_{\widetilde{E}}$.

they re-examined the proofs of OCB2 for educational purposes and searched for potential improvements. Instead they came to find a seemingly tiny crack in the proof that they first tried to fix as they strongly believed OCB2 was a secure design, but after several tries they ended up with existential and (near-)universal forgeries. Only two weeks after these findings became public (in [16]), the author of the second ePrint article [31] announced an IND-CCA vulnerability and first steps towards plaintext recovery, and again three days later, the author of the third ePrint article [20] announced full plaintext recovery. This series of happenings is a good example of "attacks only get better" and how seemingly minor error conditions can rapidly grow to nullify the security of a renowned scheme.

B Left-out Details of OCB2

We complete our OCB2 description from Sec. 3 by specifying the details of the PMAC and `len` functions. For the former see Fig. 15. The latter takes a string $X \in \{0,1\}^{\leq n}$ and encodes its lengths $|X|$ as per $\mathtt{len}(X) = 0^{n-8} \| \ell_X$, where ℓ_X denotes the standard binary encoding of $|X|$. For example, $\mathtt{len}(0^n)$ for $n = 128$ is $0^{120}10^7$.

C Proof of OCB2f

We first prove the privacy bound. Let $\mathbf{Adv}^{\mathsf{cpa}}_{\mathbb{PMAC}_{\widetilde{\mathsf{P}}},\mathsf{R}}(\mathcal{A})$ denote the PRF advantage of $\mathbb{PMAC}_{\widetilde{\mathsf{P}}}$ for CPA-adversary $\mathcal{A}$. Rog04 showed that it is bounded by $0.5q^2/2^n$ for $\mathcal{A}$ with q queries. Then we have

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{priv}}_{\mathrm{OCB2f}_{\mathsf{P}}}(\mathcal{A}) &= \mathbf{Adv}^{\mathsf{priv}}_{\Theta\mathrm{CB2f}_{\mathrm{XEX}^*_{\mathsf{P}}}}(\mathcal{A}) \\
&\leq \mathbf{Adv}^{\mathsf{tprp}}_{\mathrm{XEX}^*_{\mathsf{P}}}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{priv}}_{\Theta\mathrm{CB2f}_{\widetilde{\mathsf{P}}}}(\mathcal{A}) \\
&\leq \mathbf{Adv}^{\mathsf{tprp}}_{\mathrm{XEX}^*_{\mathsf{P}}}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{cpa}}_{\mathbb{PMAC}_{\widetilde{\mathsf{P}},\mathsf{R}}}(\mathcal{C}) + \mathbf{Adv}^{\mathsf{priv}}_{\Theta\mathrm{CB2f}'_{\widetilde{\mathsf{P}}}}(\mathcal{A}) \qquad (9) \\
&\leq \frac{4.5\sigma^2_{\mathrm{priv}}}{2^n} + \frac{0.5\sigma^2_{\mathrm{priv}}}{2^n} + 0,
\end{aligned}$$

for $\mathcal{B}$ and $\mathcal{C}$ using σ_{priv} CPA-queries. Here, the first inequality follows from the fact that possible tweak values of XE and XEX in OCB2f are distinct and $\mathcal{A}$ is tag-respecting. The last inequality follows from (7) and the aforementioned bound of $\mathbb{PMAC}$, and that the last term of (9) is zero as well as ΘCB2 as shown by Rog04.

For the authenticity bound, we have

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{auth}}_{\mathrm{OCB2f}_{\mathsf{P}}}(\mathcal{A}_{\pm}) &= \mathbf{Adv}^{\mathsf{auth}}_{\Theta\mathrm{CB2f}_{\mathrm{XEX}^*_{\mathsf{P}}}}(\mathcal{A}_{\pm}) \\
&\leq \mathbf{Adv}^{\mathsf{tsprp}}_{\mathrm{XEX}^*_{\mathsf{P}}}(\mathcal{B}_{\pm}) + \mathbf{Adv}^{\mathsf{cpa}}_{\mathbb{PMAC}_{\widetilde{\mathsf{P}},\mathsf{R}}}(\mathcal{C}) + \mathbf{Adv}^{\mathsf{auth}}_{\Theta\mathrm{CB2f}'_{\widetilde{\mathsf{P}}}}(\mathcal{A}_{\pm}) \qquad (10) \\
&\leq \frac{4.5\sigma^2_{\mathrm{auth}}}{2^n} + \frac{0.5\sigma^2_{\mathrm{auth}}}{2^n} + q_v\left(\frac{2}{2^\tau} + \frac{2}{2^n}\right)
\end{aligned}$$

for $\mathcal{B}_\pm$ using σ_{auth} CCA-queries and $\mathcal{C}_\pm$ using σ_{auth} CPA-queries. Here, the first inequality follows from the same reason as the case of privacy bound, and the second follows from an improved CCA bound of XEX* [29] and $\mathbb{PMAC}$ bound.

All that remains is to prove that $\mathbf{Adv}^{\mathsf{auth}}_{\Theta\mathrm{CB2}[\widetilde{\mathsf{P}}]}(\mathcal{A}_\pm)$ (the last term of (10)) is at most $q_v(2/2^n+2/2^\tau)$. We first prove the case of $q_v = 1$. Let $(N_i, A_i, M_i, C_i, T_i)$ for $i = 1, \ldots, q$ be the tuples obtained by q encryption queries and let (N', A', C', T') be the decryption query. Let T^* denote the valid tag corresponding to (N', A', C'). Without loss of generality, we assume the decryption query is made after all the encryption queries. We need to consider the following cases for (N', A', C', T').

1. $N' \notin \{N_1, \ldots, N_q\}$
   Since the tag (T^*) computation is done by TURP taking a new tweak, T^* is completely random. Thus the forging probability is $1/2^\tau$.
2. $1 \le \exists\alpha \le q$, $N' = N_\alpha$, $C' = C_\alpha$, $A' \neq A_\alpha$
   Let $|C'|_n = |C_\alpha|_n = m_c$. We have sub-cases:

   (a) Let $A' = \varepsilon$ and $A_\alpha \neq \varepsilon$. Then

   $$T^* = \mathtt{msb}_\tau(\widetilde{\mathsf{P}}^{*,0,N_\alpha,m_c,1}(\Sigma_\alpha)) = T_\alpha \oplus \mathtt{msb}_\tau(\mathsf{R}(A_\alpha))$$

   holds. The adversary has to guess $\mathtt{msb}_\tau(\mathsf{R}(A_\alpha))$ to forge T^*. Thus the forging probability is $1/2^\tau$.

   (b) Let $A' \neq \varepsilon$, $A' = A_j$ for some $j \in \{1, \ldots, q\}$ and $A_j \neq A_\alpha$. We have

   $$T^* = \mathtt{msb}_\tau(\widetilde{\mathsf{P}}^{*,0,N_\alpha,m_c,1}(\Sigma_\alpha) \oplus \mathsf{R}(A_j)).$$

   To correctly guess T^*, the adversary needs to guess $\mathtt{msb}_\tau(\mathsf{R}(A_j))$, thus the forging probability is $1/2^\tau$.

   (c) Let $A' \neq \varepsilon$ and $A' \notin \{A_1, \ldots, A_q\}$. We have

   $$T^* = \mathtt{msb}_\tau(\widetilde{\mathsf{P}}^{*,0,N_\alpha,m_c,1}(\Sigma_\alpha) \oplus \mathsf{R}(A')).$$

   The same as (b): the adversary needs to guess $\mathtt{msb}_\tau(\mathsf{R}(A'))$, thus the forging probability is $1/2^\tau$.
3. $1 \le \exists\alpha \le q$, $N' = N_\alpha$, $C' \neq C_\alpha$, $|C'|_n = |C_\alpha|_n = m_c$
   In this case, the adversary learns $T_\alpha = \mathtt{msb}_\tau(\widetilde{\mathsf{P}}^{*,0,N',m_c,1}(\Sigma_\alpha) \oplus \mathsf{R}(A_\alpha))$ from encryption queries. The TURP $\widetilde{\mathsf{P}}^{*,0,N',m_c,1}$ is invoked again at the decryption query to produce T^*. The case $A' \neq A_\alpha$ can be analyzed in the same way as Case 2. Thus we assume $A' = A_\alpha$ in this case. Let Σ^* and M^* denote the valid values of the checksum and plaintext corresponding to (N', C'). We define $Z_\alpha := (N_\alpha, A_\alpha, M_\alpha, C_\alpha, T_\alpha)$, and we can bound the forging probability as follows:

   $$\Pr[T' = T^*|Z_\alpha] \le \Pr[T' = T^*|\Sigma^* \neq \Sigma_\alpha, Z_\alpha] + \Pr[\Sigma^* = \Sigma_\alpha|Z_\alpha] \tag{11}$$

   (a) For $1 \le \exists i \le m_c$ and $\forall j \in \{1, \ldots, m_c\} \setminus \{i\}$, we assume $C'[i] \neq C_\alpha[i]$, $C'[j] = C_\alpha[j]$. We obtain $\Sigma^* \neq \Sigma_\alpha$ because $M^*[i] \neq M_\alpha[i]$ holds, and $M^*[j] = M_\alpha[j]$ holds for all $j \in \{1, \ldots, m_c\} \setminus \{i\}$. From (11), we obtain the forging probability as follows.

   $$\begin{aligned}\Pr[T' = T^*|Z_\alpha] &\le \Pr[T' = T^*|\Sigma^* \neq \Sigma_\alpha, Z_\alpha] + \Pr[\Sigma^* = \Sigma_\alpha|Z_\alpha] \\ &\le \frac{2^{n-\tau}}{2^n - 1} \le \frac{2}{2^\tau}.\end{aligned}$$

   (b) For $1 \le \exists i < \exists j \le m_c$, we assume $C'[i] \neq C_\alpha[i]$ and $C'[j] \neq C_\alpha[j]$. We also define $\Delta M[i] := M^*[i] \oplus M_\alpha[i] \neq 0^n$ and $\Delta M[j] := M^*[j] \oplus M_\alpha[j] \neq 0^n$. As described above, we have

   $$\begin{aligned}&\Pr[T' = T^*|Z_\alpha] \\ &\le \Pr[T' = T^*|\Sigma^* \neq \Sigma_\alpha, Z_\alpha] + \Pr[\Sigma^* = \Sigma_\alpha|Z_\alpha], \\ &\le \Pr[T' = T^*|\Sigma^* \neq \Sigma_\alpha, Z_\alpha] + \max_{\forall\delta\in\{0,1\}^n} \Pr[\Delta M[i] \oplus \Delta M[j] = \delta|Z_\alpha], \\ &\le \frac{2^{n-\tau}}{2^n - 1} + \frac{1}{2^n - 1} = \frac{2^{n-\tau} + 1}{2^n - 1} \le \frac{2}{2^\tau} + \frac{2}{2^n}.\end{aligned}$$

4. $N' = N_\alpha$, $C' \neq C_\alpha$, $|C'|_n < |C_\alpha|_n$
   Let $|C'|_n = m_{c'}$ and $|C_\alpha|_n = m_c$. In this case, unlike ΘCB2, $\widetilde{\mathsf{P}}^{*,1,N',m_{c'},0}$ is invoked to decrypt $C'[m_{c'}]$, while $\widetilde{\mathsf{P}}^{*,1,N',m_{c'},0}$ is also used in an encryption query. Since the adversary knows $\widetilde{\mathsf{P}}^{*,1,N',m_{c'},0}(M_\alpha[m_{c'}])$, she can manipulate the last block of plaintext $M^*[m_{c'}]$ if $M_\alpha[m_{c'}] = \mathtt{len}(C'[m_{c'}])$. Then she can control the value of Σ^*. Nevertheless, she has to guess T^* without access to $\widetilde{\mathsf{P}}^{*,0,N',m_{c'},1}$, which has never been invoked in encryption, since $m_{c'} \neq m_c$. Therefore, the forging probability is $1/2^\tau$.
5. $N' = N_\alpha$, $C' \neq C_\alpha$, $|C'|_n > |C_\alpha|_n$
   As above, the forging probability is $1/2^\tau$.

From these five cases, the bound is $2/2^\tau + 2/2^n$ for $q_v = 1$. Finally, we apply the standard conversion from single to multiple decryption queries [5] and obtain the bound $q_v\,(2/2^\tau + 2/2^n)$ for $q_v \geq 1$. This concludes the proof. □

D Proof of OCB2ff

For the privacy bound, we have

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{priv}}_{\mathrm{OCB2ff}_{\mathsf{P}}}(\mathcal{A}) &= \mathbf{Adv}^{\mathsf{priv}}_{\Theta\mathrm{CB2ff}_{\mathrm{XEX}^*_{\mathsf{P}}}}(\mathcal{A}) \\
&\leq \mathbf{Adv}^{\mathsf{tprp}}_{\mathrm{XEX}^*_{\mathsf{P}}}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{priv}}_{\Theta\mathrm{CB2ff}_{\widetilde{\mathsf{P}}}}(\mathcal{A}) \\
&\leq \mathbf{Adv}^{\mathsf{tprp}}_{\mathrm{XEX}^*_{\mathsf{P}}}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{cpa}}_{\mathbb{PMAC}_{\widetilde{\mathsf{P}},\mathsf{R}}}(\mathcal{C}) + \mathbf{Adv}^{\mathsf{priv}}_{\Theta\mathrm{CB2ff}'_{\widetilde{\mathsf{P}}}}(\mathcal{A}) \qquad (12) \\
&\leq \frac{4.5\sigma^2_{\mathrm{priv}}}{2^n} + \frac{0.5\sigma^2_{\mathrm{priv}}}{2^n} + 0,
\end{aligned}$$

for $\mathcal{B}$ and $\mathcal{C}$ using σ_{priv} CPA-queries. Here, the first inequality follows since possible tweak values of XE and XEX in OCB2ff are distinct and $\mathcal{A}$ is tag-respecting. The last inequality follows from (7) and the bound of $\mathbb{PMAC}$, and the last term of (12) is zero with the same reasoning as ΘCB2 shown by Rog04.

For the authenticity bound, we have

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{auth}}_{\mathrm{OCB2ff}_{\mathsf{P}}}(\mathcal{A}_\pm) &= \mathbf{Adv}^{\mathsf{auth}}_{\Theta\mathrm{CB2ff}_{\mathrm{XEX}^*_{\mathsf{P}}}}(\mathcal{A}_\pm) \\
&\leq \mathbf{Adv}^{\mathsf{tsprp}}_{\mathrm{XEX}^*_{\mathsf{P}}}(\mathcal{B}_\pm) + \mathbf{Adv}^{\mathsf{cpa}}_{\mathbb{PMAC}_{\widetilde{\mathsf{P}},\mathsf{R}}}(\mathcal{C}) + \mathbf{Adv}^{\mathsf{auth}}_{\Theta\mathrm{CB2ff}'_{\widetilde{\mathsf{P}}}}(\mathcal{A}_\pm) \qquad (13) \\
&\leq \frac{4.5\sigma^2_{\mathrm{auth}}}{2^n} + \frac{0.5\sigma^2_{\mathrm{auth}}}{2^n} + q_v\left(\frac{2}{2^\tau} + \frac{2}{2^n}\right)
\end{aligned}$$

for $\mathcal{B}_\pm$ using σ_{auth} CCA-queries and $\mathcal{C}_\pm$ using σ_{auth} CPA-queries. Note that the algorithm of $\Theta\mathrm{CB2ff}'$ is obtained by replacing $\mathbb{PMAC}$ of ΘCB2ff with a URF R. Then the first inequality follows from the same reason as the case of privacy bound, and the second follows from an improved CCA bound of XEX^* [29] and $\mathbb{PMAC}$ bound.

It remains is to prove that $\mathbf{Adv}^{\mathsf{auth}}_{\Theta\mathrm{CB2ff}'_{\widetilde{\mathsf{P}}}}(\mathcal{A}_\pm)$ (the last term of (13)) is at most $q_v(2/2^n + 2/2^\tau)$. We first prove the case of $q_v = 1$. We use the same case analysis as ΘCB2f.

1. $N' \notin \{N_1, \ldots, N_q\}$
   Since the tag (T^*) computation is done by TURP taking a new tweak, T^* is completely random. Thus the forging probability is $1/2^\tau$.
2. $1 \leq \exists\alpha \leq q$, $N' = N_\alpha$, $C' = C_\alpha$, $A' \neq A_\alpha$
   Let $|C'|_n = |C_\alpha|_n = m_c$. The analysis of this case is almost the same as that of Case 2 in Appendix C except for the indices of tweaks. We have sub-cases:

   (a) Let $A' = \varepsilon$ and $A_\alpha \neq \varepsilon$. Then

   $$T^* = \mathtt{msb}_\tau(\widetilde{\mathsf{P}}^{*,0,N_\alpha,m_c-1,1,1}(\Sigma_\alpha)) = T_\alpha \oplus \mathtt{msb}_\tau(\mathsf{R}(A_\alpha))$$

   holds. The adversary has to guess $\mathtt{msb}_\tau(\mathsf{R}(A_\alpha))$ to forge T^*. Thus the forging probability is $1/2^\tau$.

(b) Let $A' \neq \varepsilon$, $A' = A_j$ for some $j \in \{1, \ldots, q\}$ and $A_j \neq A_\alpha$. We have

$$T^* = \mathtt{msb}_\tau(\widetilde{\mathsf{P}}^{*,0,N_\alpha,m_c-1,1,1}(\Sigma_\alpha) \oplus \mathsf{R}(A_j)).$$

To correctly guess T^*, the adversary needs to guess $\mathtt{msb}_\tau(\mathsf{R}(A_j))$, thus the forging probability is $1/2^\tau$.

(c) Let $A' \neq \varepsilon$ and $A' \notin \{A_1, \ldots, A_q\}$. We have

$$T^* = \mathtt{msb}_\tau(\widetilde{\mathsf{P}}^{*,0,N_\alpha,m_c-1,1,1}(\Sigma_\alpha) \oplus \mathsf{R}(A')).$$

The same as (b): the adversary needs to guess $\mathtt{msb}_\tau(\mathsf{R}(A'))$, thus the forging probability is $1/2^\tau$.

3. $1 \leq \exists\alpha \leq q$, $N' = N_\alpha$, $C' \neq C_\alpha$, $|C'|_n = |C_\alpha|_n = m_c$
The analysis of this case is exactly the same as that of Case 3 in Appendix C. Thus the forging probability can be evaluated as $2/2^\tau + 2/2^n$.
4. $N' = N_\alpha$, $C' \neq C_\alpha$, $|C'|_n < |C_\alpha|_n$
Let $|C'|_n = m_{c'}$ and $|C_\alpha|_n = m_c$. In the proof of ΘCB2f, we had to take care of the fact that the adversary can control the last block of M^* because the last block of C' is decrypted by TURP which has been invoked in the α-th encryption query. In the case of ΘCB2ff, however, we do not have to care such a case since TURP to decrypt $C'[m_{c'}]$ takes a new tweak and the adversary cannot control $M^*[m_{c'}]$ at all. Moreover, she has no information about $\widetilde{\mathsf{P}}^{*,1,N',m_{c'}-1,1,1}$ which produces T^*, since $m_{c'} \neq m_c$. Therefore, the forging probability is $1/2^\tau$.
5. $N' = N_\alpha$, $C' \neq C_\alpha$, $|C'|_n > |C_\alpha|_n$
As above, the forging probability is $1/2^\tau$.

From these five cases, the bound is $2/2^\tau + 2/2^n$ for $q_v = 1$. Finally, we apply the standard conversion from single to multiple decryption queries [5] and obtain the bound $q_v\,(2/2^\tau + 2/2^n)$ for $q_v \geq 1$. This concludes the proof. □

E Code Example for Minimal Forgery Attack

1. Retrieve OCB2 reference code from `http://web.cs.ucdavis.edu/~rogaway/ocb/code-2.0.htm` and AES reference code (`rijndael-alg-fst.c`).
2. Change the `main` routine of `ocb.c` to the following snippet:

```
int
main(void)
{
   block nbits = {0};
   block N = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
   block K = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
   byte Hatk[0]; /* any */
   byte Matk[32] = {0};
   byte Catk[32] = {0};
   byte Hforge[0]; /* must be empty */
   byte Cforge[16] = {0};
   byte Mforge[16] = {0};
   block T,Tatk,Tforge;
   int res;
   ocb_state *state;

   /* Test for the minimal attack */
   printf("Test for minimal attack \n");
   state = ocb_init((byte *)"abcdefghijklmnop",
   sizeof(T),sizeof(N),AES128);
   memset(nbits,0,sizeof(block));
   nbits[sizeof(block)-1] = 16 * 8; /* 128 bits */
```

```
    memcpy(Matk,nbits,sizeof(block));
    printf("Encryption query:\n");
    pbuf(N,16, "  Nonce");
    pbuf(Matk,32, "  Plaintext");
    pbuf(Hatk,sizeof(Hatk), "  AD");
    ocb_provide_header(state,Hatk,sizeof(Hatk));
    ocb_encrypt(state,N,Matk,sizeof(Matk),Catk,Tatk);
    pbuf(Catk,32, "  Ciphertext");
    pbuf(Tatk,16, "  Tag");
    printf("Decryption query (forgery):\n");
    memcpy(Cforge, Catk, 16);
    xor_block(Cforge, Cforge, nbits);
    pbuf(N,16, "  Forged Nonce (the same as encryption)");
    pbuf(Hforge, sizeof(Hforge), "  Forged AD (empty)");
    pbuf(Cforge,16, "  Forged Ciphertext");
    memcpy(Tforge, Matk+16, 16);
    xor_block(Tforge, Tforge, Catk+16);
    pbuf(Tforge, 16, "  Forged Tag");
    ocb_provide_header(state,Hforge,sizeof(Hforge));
    res = ocb_decrypt(state,N,Cforge,sizeof(Cforge),Tforge,Mforge);
    ocb_zeroize(state);
    printf("Tags match: %i.\n", res); /* 1 is "matched" */
    pbuf(Mforge, sizeof(Mforge), "  Forged Plaintext");
    return 0;
}
```