Cryptanalysis of OCB2: Attacks on Authenticity and Confidentiality

Akiko Inoue[1], Tetsu Iwata[0000−0002−4729−0979]2, Kazuhiko Minematsu[0000−0002−3427−6772]1, and Bertram Poettering[0000−0001−6525−5141]3

[1] NEC Corporation, Kawasaki, Japan,
a-inoue@cj.jp.nec.com, k-minematsu@ah.jp.nec.com
[2] Nagoya University, Nagoya, Japan, tetsu.iwata@nagoya-u.jp
[3] Royal Holloway, University of London, London, United Kingdom and IBM Research Zurich, 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. To our understanding, 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. [34], the second version (OCB2)

at ASIACRYPT 2004 by Rogaway [30] (hereafter Rog04), and the third version (OCB3) at FSE 2011 by Krovetz and Rogaway [20]. 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 [15] standard, and OCB3 is specified as document RFC 7253 [21] 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 [10] and Sun et al. [36] 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 to craft the forgery. The attack is independent of the blockcipher E over which OCB2 is defined, including of its key and block length. Further, the message to which the forged ciphertext decrypts is strongly dependent on the message involved in the first 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 (including 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 could ever be.

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

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, [22]): For encrypting a message consisting of one or multiple blocks, 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. We emphasize that the flaw of OCB2 that we identify and exploit is located neither in the general method the AEAD scheme is constructed from a tweakable blockcipher nor in the security of the XEX* primitive. The problem is rather hidden 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. As it turns out, OCB2 can be provably fixed by replacing certain XE invocations by XEX invocations. While the price to pay for this is minor (one additional XOR operation per encryption/decryption operation), unfortunately the fixed version loses backward compatibility with (unmodified) OCB2 implementations.

Our attacks are technical and fairly complex, so we confirmed their effectivity 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. 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 affected by our attacks.

1.1 Impact

OCB2 has been standardized in ISO/IEC 19772:2009 for about a decade [15]. As the scheme offers exceptional performance that was and still is challenging to rival for AES-based constructions, it has to be assumed that industry has widely picked up on it, ultimately incorporating the scheme into products. The consequences of this might be severe. We have thus 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 findings. The working group has issued

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

a document [16] 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.

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[6]). The lack of open implementations suggests that most affected parties have industrial background. By the very nature of (IND$ secure) encryption, spotting products that rely on OCB2 for security and now became vulnerable remains a challenge.

1.2 Further Related Work

Besides the already mentioned attacks by Ferguson and Sun et al. (that show tightness of the birthday bound claimed for OCB), the following analyses in less classic attack settings have been conducted: Attacks in scenarios where the AE scheme is deployed in a somewhat sloppy way, e.g. where nonces are repeated (nonce-misuse setting) or where message fragments emerging from partially decrypted (possibly invalid) ciphertexts are leaked (release of unverified plaintext setting) are proposed by Andreeva et al. [1] and Ashur et al. [3]. In the same vein, but also considering attacks against the birthday bound of security claims, Vaudenay and Vizár [37] studied all third-round CAESAR candidates, including OCB3.

Not with the aim of breaking a particular version of OCB, but with the goal of better understanding the security of the schemes by refining the set of necessary security requirements on the underlying blockcipher, Aoki and Yasuda [2] show that relaxed assumptions are sufficient to establish the security of OCB. (Note that our attacks are in conflict with their claims on OCB2, indicating that their security arguments have to be reconsidered; the authors of [2] confirmed this view to us.)

Attacks in the *reforgeability setting* [6,11] deliver a series of existential forgeries with the specific property that creating the first forgery is the hardest part. While in most cases hardness is measured in terms of computation time, also our attacks can be seen in the reforgeability setting, but with a different complexity measure: While crafting the first OCB2 forgery requires two queries (one encryption, one decryption) and the forgery is only existential, all further forgeries can be universal (on arbitrary messages and associated data), and only require one further query (encryption). In fact, one can create hundreds of universal forgeries from the second encryption query.

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

1.3 Organization and Contributions

We recall notions of tweakable blockciphers and authenticated encryption in Section 2. After specifying the OCB2 algorithms in Section 3 we present simple authenticity and confidentiality attacks against them in Section 4. While the latter 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 Section 5, a set of advanced attacks (including universal forgery and arbitrary decryption) that break the scheme also in most *real-world* settings. In Section 6 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 attempts of such schemes. In Section 7 we survey the applicability of our attack strategies to related encryption modes, including to OCB1 and OCB3; however we do not identify any further weak candidate. Finally, in Section 8 we consider a couple of ways to repair OCB2.

2 Preliminaries

2.1 Notations

If A is a 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 sets 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 $|\varepsilon| = 0$. The sequence of c zeros is denoted with 0^c, with the convention that $0^0 = \varepsilon$. 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, n with $|X| \leq n$ we define the zero padding, $X \,\|\, 0^*$, and the one-zero padding, $X \,\|\, 10^*$. Both are X when $|X| = n$. They are $X \,\|\, 0^* = X \,\|\, 0^{n-|X|}$ and $X \,\|\, 10^* = X \,\|\, 10^{n-|X|-1}$, respectively, when $0 \leq |X| < n$.

For $X \in \{0,1\}^*$, we also define the parsing of a string into n-bit blocks denoted by

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

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

2.2 (Tweakable) Blockciphers

A tweakable blockcipher (TBC) [22] 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 with $\mathcal{T}$ being a singleton, and specifically 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 $T\in\mathcal{T}$ and $K\in\mathcal{K}$, 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 depends on a keyed blockcipher instance E_K, we may treat E_K as the key and write Mode_E (and correspondingly for a TBC $\widetilde{E}$).

Security of (Tweakable) Blockciphers. 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 distributed over 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 $f(T,*)$ is a permutation over $\mathcal{M}$ 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:

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

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

Here, the adversaries perform chosen-plaintext attacks and chosen-ciphertext attacks, respectively, and in both cases with chosen tweak. (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 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 [30,18], 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 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.

In the spirit of the above, OCB2 considers the domain $\mathcal{M} = \{0,1\}^n$ of the blockcipher it is based on a Galois field. More precisely, the fixed block length $n = 128$ is assumed (which matches 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. For more details on this representation, see [30].

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) [29]. An AE scheme $\Pi = (\Pi.\mathcal{E}, \Pi.\mathcal{D})$ 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. The understanding of AD is that it is an input to the encryption and decryption algorithms that is not to be kept confidential; rather it reflects the context in which the encryption happens and is authenticated along with the encrypted message.[7] Formally, the AEAD encryption algorithm is a function $\Pi.\mathcal{E}\colon \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{M} \times \mathcal{T}$, and the decryption (incl. verification) algorithm is a function $\Pi.\mathcal{D} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \times \mathcal{T} \to \mathcal{M} \cup \{\perp\}$, where symbol $\perp$ is used to report verification failures.

To encrypt plaintext M with nonce N, associated data A, and key K, compute $(C, T) \leftarrow \Pi.\mathcal{E}_K(N, A, M)$ to produce ciphertext C and tag T. The tuple (N, A, C, T) is communicated to the receiver[8] and the original message M recovered by computing $\Pi.\mathcal{D}_K(N, A, C, T)$.

Security Notions. The security of AE is typically captured with two notions: privacy and authenticity. Following the definitions of [5,32], authenticity requires that ciphertexts (including nonce, associated data, and tag) cannot be forged, and privacy requires their indistinguishability (including the tag). More precisely, while [32, Sec. 3] defines privacy as the inability of a passive adversary to distinguish ciphertext-tag pairs from random strings, [32, Sec. 6] gives a second definition that formalizes privacy against active adversaries (that can pose decryption queries). As noted in [32, Sec. 6], if authenticity is provided by

[7] For example, if network payloads are to be encrypted, it is useful to include network header information in the AD.

[8] In many practical cases, receivers can reproduce N and/or A by themselves so that these values do not need to be transmitted.

a scheme, the two privacy notions turn out to be equivalent. Since the current article considers an AE scheme that does *not* provide authenticity, we emphasize that for this scheme the equivalence of the two notions cannot be assumed (and in fact they differ!). 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 picked string of length $|M| + \tau$. The privacy advantage of an adversary $\mathcal{A}$ is defined as

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

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. The corresponding advantage definition is

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

where the probabilities are over the random choice $K \xleftarrow{\$} \mathcal{K}$.

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

$$\mathbf{Adv}^{\mathsf{auth}}_{\Pi}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\Pi.\mathcal{E}_K(\cdot,\cdot,\cdot),\Pi.\mathcal{D}_K(\cdot,\cdot,\cdot,\cdot)} \text{ forges }\right], \tag{1}$$

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

3 The OCB2 Mode of Operation

The OCB2 authenticated encryption scheme was initially, in [30], described as a pure (nonce-based) AE mode without support for AD processing.[9] Like its predecessor OCB1 it is fully parallelizable and rate-1 (requiring one blockcipher invocation per message block), but it replaced the table-driven design of OCB1 with the 'powering-up' construction to compute a sequence of tweaks by continuously doubling them. Further, in [30, Sec. 11] it was suggested that the OCB2 AE mode can be generalized into an AEAD mode (dubbed AEM) by XOR-ing, in all cases where the AD is non-empty, a MAC of the AD into the authentication tag of OCB2. The OCB2-related PMAC construction was identified as a

[9] In that paper the mode was actually referred to as OCB1; what we call OCB1 was referred to as OCB in [30].

Algorithm OCB2.$\mathcal{E}_E(N, A, M)$

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. Pad $\leftarrow E(2^m L \oplus \mathtt{len}(M[m]))$
6. $C[m] \leftarrow M[m] \oplus \mathtt{msb}_{|M[m]|}(\text{Pad})$
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. $T \leftarrow \mathtt{msb}_\tau(T)$
12. **return** (C, T)

Algorithm OCB2.$\mathcal{D}_E(N, A, C, T)$

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. Pad $\leftarrow E(2^m L \oplus \mathtt{len}(C[m]))$
6. $M[m] \leftarrow C[m] \oplus \mathtt{msb}_{|C[m]|}(\text{Pad})$
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. $T^* \leftarrow \mathtt{msb}_\tau(T^*)$
12. **if** $T = T^*$ **return** M
13. **else return** $\perp$

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

particularly interesting option as it would allow sharing its blockcipher instance with that of the OCB2 encryption core.[10]

Our specification of OCB2 is taken from [31, Fig. 3] and supports associated data. The 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 sets of strings of arbitrary length, and the tag space is $\mathcal{C} = \{0,1\}^\tau$ for any fixed parameter $\tau \leq n$.

The OCB2 algorithms $\mathcal{E}_E$ and $\mathcal{D}_E$ are detailed in Fig. 1 (and algorithm $\mathcal{E}_E$ is further illustrated in Fig. 2). In the code, for $X \in \{0,1\}^{\leq n}$, expression $\mathtt{len}(X)$ 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 ultimately not relevant for our attacks, so we omit their description here. (For completeness we reproduce them in Appendix B.)

4 Basic Attacks

We prove by example that, in the formal sense, OCB2 provides neither authenticity nor confidentiality. We start with specifying a minimal attack on unforgeability that gets along with a single encryption query to produce an existential forgery with probability 1. This attack, while formally valid, is rather limited

[10] The PMAC version from [31] is slightly different from the initial version [7] in that it uses doublings for mask generation and was further adapted to be computationally independent from the encryption part when combined with OCB2.

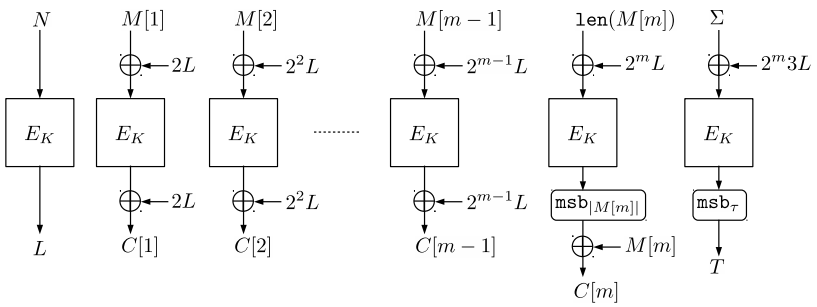

Fig. 2. OCB2 encryption for the case of empty AD.

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 (requiring one encryption and one decryption query).

The attacks considered here neither craft universal forgeries nor decrypt arbitrary ciphertexts. These more powerful attacks are described in Section 5.

4.1 Minimal Forgery

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 $\texttt{len}$.

The following steps of our attack are also illustrated in Fig. 3 and specified in pseudocode in Fig. 4.

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 the 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

$$\begin{aligned} N' &= N, \\ A' &= \varepsilon, \\ C' &= C[1] \oplus \texttt{len}(0^n) \\ T' &= M[2] \oplus C[2] \end{aligned} \tag{2}$$

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

$$\begin{aligned} C[1] &= 2L \oplus E(2L \oplus \mathtt{len}(0^n)) \\ C[2] &= M[2] \oplus \mathrm{Pad}, \end{aligned} \tag{3}$$

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

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

and the tag is recovered as

$$\begin{aligned} T^* &= E(2 \cdot 3L \oplus \Sigma') \\ &= E(2 \cdot 3L \oplus C' \oplus \mathrm{Pad}') \\ &= E(2 \cdot 3L \oplus M') \\ &= E(2 \cdot 3L \oplus 2L \oplus \mathtt{len}(0^n)) \\ &= E(2^2 L \oplus \mathtt{len}(0^n)) \qquad (4) \\ &= \mathrm{Pad} \\ &= T', \qquad (5) \end{aligned}$$

where (4) follows from the identity $2 \cdot 3L = 2^2 L \oplus 2L$ and (5) follows from (2) and (3). The conclusion is: We have $T^* = T'$ and thus tuple (N', A', C', T') is falsely accepted as an authentic ciphertext.

4.2 Forgery of Longer Messages

We next show that the attack of Section 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 for $A = \varepsilon$ for the encryption query, and relaxes the $\tau = n$ requirement for the tag length.

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

   $$M[m-1] = \mathtt{len}(0^n),$$

   and $M[m]$ is any s-bit string such that $\tau \leq s \leq n$. The encryption oracle returns a pair (C, T) where $C = C[1] \,\|\, \cdots \,\|\, C[m-1] \,\|\, C[m]$.

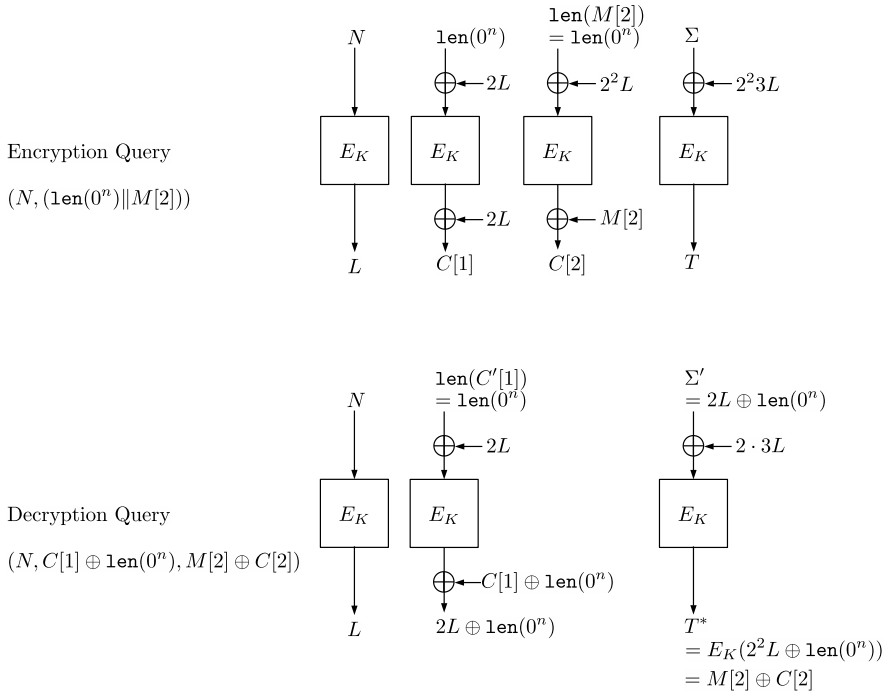

Fig. 3. Minimal forgery attack (see Sec. 4.1).

2. Decrypt (N', A', C', T') where $N' = N$, $A' = \varepsilon$, and $C' = C'[1] \,\|\, \cdots \,\|\, C'[m-2] \,\|\, C'[m-1]$ has $m-1$ blocks such that

$$\begin{aligned} C'[i] &= C[i] \text{ for } 1 \le i \le m-2 \\ C'[m-1] &= \sum_{i=1}^{m-2} M[i] \oplus C[m-1] \oplus \texttt{len}(M[m]) \\ T' &= \texttt{msb}_\tau(M[m] \oplus C[m]). \end{aligned}$$

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

$$\begin{aligned} \overline{T}' &= E(\Sigma' \oplus 3 \cdot 2^{m-1} L) \\ &= E\left(\sum_{i=1}^{m-2} M'[i] \oplus C'[m-1] \oplus \mathrm{Pad}' \oplus 3 \cdot 2^{m-1} L\right) \\ &= E\left(\sum_{i=1}^{m-2} M[i] \oplus C'[m-1] \oplus C[m-1] \oplus 2^{m-1} L \oplus 3 \cdot 2^{m-1} L\right), \end{aligned} \quad (6)$$

where $M'[i] = M[i]$ is the i-th decrypted plaintext block, and $\mathrm{Pad}' = C[m-1] \oplus 2^{m-1}L$. Since $2^{m-1}L \oplus 3 \cdot 2^{m-1}L = 2^m L$, the last term of (6) is further expanded as

$$
\begin{aligned}
& E\left(\sum_{i=1}^{m-2} M[i] \oplus C'[m-1] \oplus C[m-1] \oplus 2^m L\right) \\
& = E\left(\sum_{i=1}^{m-2} M[i] \oplus \left(\sum_{i=1}^{m-2} M[i] \oplus C[m-1] \oplus \mathtt{len}(M[m])\right) \oplus C[m-1] \oplus 2^m L\right) \\
& = E\left(\mathtt{len}(M[m]) \oplus 2^m L\right) \\
& = \mathrm{Pad}\ .
\end{aligned}
$$

Finally, we have

$$
\begin{aligned}
T^* &= \mathtt{msb}_\tau(\overline{T}') \\
&= \mathtt{msb}_\tau(\mathrm{Pad}) \\
&= \mathtt{msb}_\tau(M[m] \oplus C[m]) \quad (\because \tau \le |M[m]| \le n) \\
&= T'\ .
\end{aligned}
$$

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. More precisely, we describe a two-query adversary against the PRIV-CCA notion that achieves a distinguishing advantage of almost 1.

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

Attacking the PRIV-CV notion. In Sec. 2.3 we formalized the privacy notions PRIV and PRIV-CCA, where the former did not have a decryption oracle and targeted fully passive adversaries. We note that a version that is like the plain PRIV notion but adds 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 $\top/\perp$) indicating whether the ciphertext was valid. It is not hard to see that our attack against PRIV-CCA is actually an attack against PRIV-CV. (Note that this increases its applicability and thus makes it more powerful.) We give the full details of the attack in Fig. 4, where we denote the verification oracle with $\mathcal{V}$.

Adversary $\mathcal{A}^{\mathcal{E}(\cdot,\cdot,\cdot),\mathcal{D}(\cdot,\cdot,\cdot,\cdot)}$

1. **Step 1:**
2. $M[1] \leftarrow \mathtt{len}(0^n)$
3. **Pick any** $M[2] \in \{0,1\}^n$
4. $M \leftarrow M[1] \,\|\, M[2]$
5. **Pick any** $N \in \{0,1\}^n$
6. **Query** $(C,T) \leftarrow \mathcal{E}(N,\varepsilon,M)$
7. **Step 2:**
8. $C[1] \,\|\, C[2] \xleftarrow{n} C$
9. $C' \leftarrow C[1] \oplus \mathtt{len}(0^n)$
10. $T' \leftarrow M[2] \oplus C[2]$
11. **Query** $M' \leftarrow \mathcal{D}(N,\varepsilon,C',T')$
12. **Stop**

Adversary $\mathcal{A}^{\mathcal{E}/\$(\cdot,\cdot,\cdot),\mathcal{V}(\cdot,\cdot,\cdot,\cdot)}$

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

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

Attacking the IND-CCA notion. A different formalization of confidentiality is given by the IND-CCA notion [32] which does not require that ciphertexts look like random strings but 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, while the right message would be chosen to be the all-zero message (of the same length). As above, the adversary would output $b = 1$ iff its forgery attempt is deemed valid.

4.4 Observations

The attacks of Sec. 4.1 and 4.2 can be extended to several directions.

Truncated Tag. Since the tag T returned by the first encryption query is not needed by our attacks, they also work if AD is chosen non-empty (for the encryption query). However, the decryption query needs the empty AD. For the same reason, our attacks also work when $\tau < n$; we just set $T' = \mathtt{msb}_\tau(M[2] \oplus C[2])$ and the forgery will be accepted with probability one.

Almost-Arbitrary Message. Most of the blocks of the message involved in the first encryption query can be freely chosen by the attacker. Only the last but one block requires a special format: $\mathtt{len}(0^n) = 0^{120}10^7$ (see Appendix B). This format is not too special and could even occur naturally, e.g. if plaintexts receive a certain padding before being encrypted.

The condition on $M[m-1]$. Our attacks also work for some values $M[m-1]$ that differ from $\mathtt{len}(0^n)$. When $M[m-1] = \mathtt{len}(0^{n-s})$ for some $0 < s < n$,

by making $(n-s)$-bit $C'[m-1] = \mathtt{msb}_{n-s}(C[m-1]) \oplus \mathtt{msb}_{n-s}(\mathtt{len}(0^n))$ the forgery is still successful if s is small. In more detail, the success probability is $1/2^s$ which is the probability that $\mathtt{lsb}_s(\text{Pad})$ equals to $\mathtt{lsb}_s(2L \oplus \mathtt{len}(0^n))$. If we create 2^s forgeries for 2^s encryption queries, there will be at least one successful forgery with a high probability.

When $\mathtt{len}(M[m]) < \tau$, the adversary can forge T^* with probability $1/2^{\tau - \mathtt{len}(M[m])}$, since the adversary only knows $\mathtt{msb}_{\mathtt{len}(M[m])}(\text{Pad})$ and has to guess the remaining $(\tau - \mathtt{len}(M[m]))$ bits.

5 Advanced Attacks

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

5.1 Universal Forgeries

In a universal forgery attack the adversary freely chooses any $M^\$ \in \{0,1\}^*$, any $A^\$ \in \{0,1\}^*$, and any $N^\$ \in \{0,1\}^n$, and creates a forgery $(C^\$, T^\$)$ such that OCB2.$\mathcal{D}_E(N^\$, A^\$, C^\$, T^\$)$ returns $M^\$$. We present a universal forgery attack for OCB2 that is based on two sub-routines that we describe first.

Extracting random blockcipher mappings. Given a fixed blockcipher instance $E = E_K$, we refer to any pair $(X, Y) \in (\{0,1\}^n)^2$ satisfying $E(X) = Y$ as an *input-output pair* or *mapping* of the blockcipher. Note that the regular deployment of OCB2 does not expose such pairs. (This is not a coincidence as the XEX* construction becomes insecure when such pairs become public.) However, as we observe and explore in the following, if forged OCB2 ciphertexts surface and are decrypted then the resulting messages *do* leak one or more input-output pairs. We develop pseudocode for a procedure that, on input an integer m, performs a specific OCB2 forgery and extracts roughly m-many input-output pairs from the result. As our procedure does not control the positions X, Y for which it finds the pairs we refer to the process as 'random mapping extraction'.

Recall that in our authenticity attack from Section 4.1 the adversary learns 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 (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 $(2^2L \oplus \mathtt{len}(0^n), C[2] \oplus M[2])$ are input-output pairs of E. (In addition, but only if $\tau = n$, we can obtain one more such pair from Σ and T; however, for generality we ignore this observation in the following.)

Similar observations hold for our long-message forgery attack of Section 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 (left), mechanizes the input-output pair gathering by crafting, in the spirit of Section 4.2, a forgery for a long all-zero message. More precisely, the procedure

takes on input a value $m \geq 2$ and extracts $m+1$ input-output pairs[11], assuming it is provided with access to $\mathcal{E}$ and $\mathcal{D}$ oracles. (Again we ignore the extra pair obtainable when $\tau = n$.) The resulting pairs (X, Y) are collected in a global set $\mathbb{E}$. While the latter is shared with other algorithms that we describe below, the set can be characterized by the implication $(X, Y) \in \mathbb{E} \Rightarrow E(X) = Y$ (for one fixed blockcipher key K).

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.

The specification of the corresponding Encipher procedure is in Fig. 5 (right). 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 value X_m is only implicitly embedded: We carefully choose the last message block $M[m]$ such that the sum $\Sigma = M[1] \oplus \ldots \oplus M[m]$ 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 Encipher.

Universal Forgery Attack. Note that with the development of the Encipher algorithm it became trivial to compute forgeries on any combination of nonce N, message M, and AD A: It simply suffices to execute OCB2's encryption algorithm $\mathcal{E}$ from Fig. 1 on input N, A, M, emulating all blockcipher evaluations with invocations of Encipher. The resulting forgeries are perfect. Note further that OCB2 is parallelizable, i.e., 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 Encipher call.

When closely looking at the details it however becomes apparent that universally forging cannot be performed with a single Encipher invocation. As a matter

[11] The number of pairs can be fewer than $m+1$ when collisions occur, however this event has a negligible probability.

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

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. $\quad (X_i,Y_i) \leftarrow (2^iL \oplus M[i], 2^iL \oplus C[i])$
16. $X_m \leftarrow 2^mL \oplus \mathtt{len}(0^n)$
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. $\quad X_T \leftarrow \mathtt{len}(0^n) \oplus 2^m3L$
22. $\quad Y_T \leftarrow T$
23. $\quad \mathbb{E} \xleftarrow{\cup} \{(X_T,Y_T)\}$
24. **return**

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

1. **Global variable:** $\mathbb{E}$
2. $(N,L) \xleftarrow{\$} \mathbb{E}$
3. **for** $i \leftarrow 1$ **to** $m-1$**:**
4. $\quad M[i] \leftarrow 2^iL \oplus X_i$
5. $\Sigma \leftarrow 2^m3L \oplus X_m$
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. $\quad Y_i \leftarrow 2^iL \oplus C[i]$
12. $X' \leftarrow 2^mL \oplus \mathtt{len}(0^n)$
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. 5. Left: Procedure that generates a random collection of $m+1$ pairs (X_i,Y_i) such that $E(X_i) = Y_i$ for all i. If $\tau = n$ (gray part) then this is improved to $m+2$ pairs. **Right:** Procedure that given $X_1,\ldots,X_{m-1}$ finds $Y_1,\ldots,Y_{m-1}$ such that $E(X_i) = Y_i$ for all i. If $\tau = n$ (gray part) then 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., $X_m = 0$.) **Both:** The procedures share a common set variable $\mathbb{E}$ that is assumed to initially be empty. Procedure Encipher may only be invoked after SamplePairs has been (this is to ensure well-defined behavior in line 2 of the former).

of fact, not all enciphering operations related to an encryption are concurrent: In OCB2's $\mathcal{E}$ 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 Encipher invocations. A similar observation can be made for the PMAC algorithm (see Fig. 9 in Appendix) where the finalization step requires enciphering an intermediate sum that is computed by adding up outputs of other enciphering operations. The latter, by themselves depend on the value $E(0^n)$, so the minimal number of Encipher 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 Encipher 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 Encipher. The first call is to obtain $E(N^\$)$ and $E(0^n)$, the second is to obtain those needed for encryption of $M^\$$ and PMAC of $A^\$$ *except the tag and the last AD block*, and the third is for the tag and the last AD block. Specifically, the steps for the universal forgery are as follows:

1. The adversary performs SamplePairs(2). With overwhelming probability, we assume 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 performs
   Encipher$(N^\$, 0^n, 0^n)$ and obtains $L := E_K(N^\$)$ and $V := 3^2 E_K(0^n)$.
3. Let

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

   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 Encipher$(X_1, \cdots, X_m, X_1^A, \cdots, X_{a-1}^A, 0^n)$.
4. Let $X_{m+1} := \Sigma^\$ \oplus 2^m \cdot 3L$, where

$$\Sigma^\$ = \bigoplus_{i=1}^{m-1} M^\$[i] \oplus (M^\$[m] \,\|\, \mathtt{lsb}_{n-|M^\$[m]|}(Y_m)).$$

   If $|A^\$[a]| = n$, let $X_a^A := \Sigma_{i=1}^{a-1} Y_i^A \oplus A^\$[a] \oplus 2^a \cdot 3V$ and else, $X_a^A := \Sigma_{i=1}^{a-1} Y_i^A \oplus \mathtt{ozp}(A^\$[a]) \oplus 2^a \cdot 3^2 V$. She obtains $Y_{m+1} = E_K(X_{m+1})$ and $Y_a^A = E_K(X_a^A)$ by calling Encipher$(X_{m+1}, X_a^A, 0^n)$.
5. She creates $(N^\$, A^\$, C^\$, T^\$)$, where

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

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

5.2 Plaintext Recovery

Security Model of Plaintext Recovery Attack. We consider an attack model that closely follows [25]. A challenger has a secret key K. Let (C^*, T^*) be

the encryption of (N^*, A^*, M^*), where a nonce N^*, associated data A^*, and a plaintext M^* are arbitrarily chosen by the challenger.

Then (N^*, A^*, C^*, T^*) is given to the adversary as a challenge. She has access to the encryption and decryption oracles, and the goal is to recover M^*. She cannot use N^* as a nonce in encryption queries (as N^* was already used in encryption to generate the challenge). Also, the adversary is nonce-respecting and hence cannot repeat the same nonce in encryption queries. To avoid a trivial win, she cannot use the challenge (N^*, A^*, C^*, T^*) in decryption queries.

Plaintext Recovery Attack. (C^*, T^*) is the encryption of (N^*, A^*, M^*), and (N^*, A^*, C^*, T^*) is given to the adversary as a challenge. We first make an assumption that M^* is long and C^* has many blocks (for instance 3 or more blocks), and the goal is to recover M^* (We will later show how to recover short plaintexts).

We first recover $L^* := E_K(N^*)$. This can be done by using $\mathsf{SamplePairs}$ and $\mathsf{Encipher}$ as follows: The adversary first calls $\mathsf{SamplePairs}(2)$, and with overwhelming probability, we assume nonce N' sampled in $\mathsf{SamplePairs}(2)$ 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 performs $\mathsf{Encipher}(N^*, 0^n)$ and obtains L^* from the first block of the output of $\mathsf{Encipher}(N^*, 0^n)$.

With the knowledge of L^*, we modify 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 $m^* \geq 3$. We then define $C^\$ = (C^\$[1], \ldots, C^\$[m^*])$ as follows:

- $C^\$[i] := C^*[i]$ for $i \in \{1, \ldots, m^*\} \setminus \{j, k\}$
- $C^\$[j] := C^*[k] \oplus 2^k L^* \oplus 2^j L^*$
- $C^\$[k] := C^*[j] \oplus 2^k L^* \oplus 2^j L^*$

Next, the adversary makes a decryption query $(N^*, A^*, C^\$, T^*)$, i.e, this is almost the same as the challenge, but the j-th and k-th blocks of C^* are modified. This step can fail 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, and the adversary obtains $M^\$$. The goal of the attack, M^*, is obtained by swapping the j-th and k-th blocks of $M^\$$ and making necessary modifications. More precisely, from $M^\$ = (M^\$[1], \ldots, M^\$[m^*])$, we obtain $M^* = (M^*[1], \ldots, M^*[m^*])$ 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. 6 for the encryption process of (N^*, A^*, M^*) and the decryption process of $(N^*, A^*, C^\$, T^*)$.

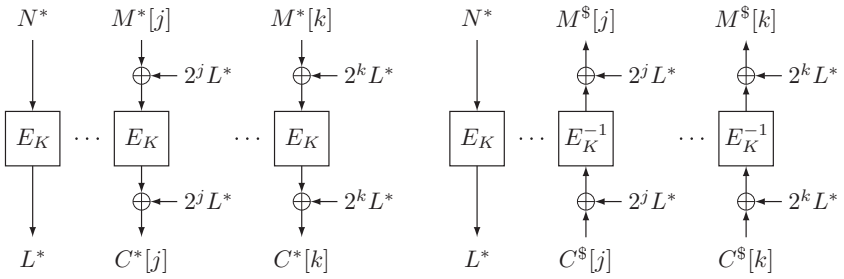

Fig. 6. Left: The encryption process of (N^*, A^*, M^*). **Right:** The decryption process of $(N^*, A^*, C^\$, T^*)$. In the right figure, we have $C^\$[j] = C^*[k] \oplus 2^kL^* \oplus 2^jL^*$ and $C^\$[k] = C^*[j] \oplus 2^kL^* \oplus 2^jL^*$, and it follows that $M^*[j] = M^\$[k] \oplus 2^kL^* \oplus 2^jL^*$ and $M^*[k] = M^\$[j] \oplus 2^kL^* \oplus 2^jL^*$. We see that the checksum remains the same.

Emulating Blockcipher Decryption. We show that, for any Y^*, the adversary can compute $X^* = E_K^{-1}(Y^*)$. This complements the extraction of a specific blockcipher mapping in Section 5.1, and this will be useful in the plaintext recovery for plaintexts of two blocks.

The adversary first calls $\mathsf{SamplePairs}(2)$, and let N be the nonce sampled in the call. Then she obtains $\mathbb{E} = \{(N, L), (X[1], Y[1]), (X[2], Y[2])\}$.

Let $(N', L') = (X[1], Y[1])$, where we assume that $N' \neq N$, and define

$$M^* = (X^* \oplus 2L', X^* \oplus 2^2L', 0^n) \in \{0,1\}^{3n}.$$

The approach we take is to compute C^* and T^* under the nonce N' and empty A^*, and make a decryption query (N', A^*, C^*, T^*). The adversary obtains M^*, and X^* can be obtained in an obvious way.

The observation here is that the checksum of M^* is $\Sigma^* := 2L' \oplus 2^2L'$, which is independent of X^*, and we know all the blockcipher input values to compute C^* and T^*. See Fig. 7 for the encryption process of (N', A^*, M^*). We need to derive the values of $C^*[3]$ and T^* in Fig. 7. This can be done by calling $\mathsf{Encipher}(X[1], X[2], 0^n)$, where $X[1] = \mathtt{len}(0^n) \oplus 2^3L'$ and $X[2] = 2L' \oplus 2^2L' \oplus 2^33L'$. From the output $(Y[1], Y[2], Y[3])$ of $\mathsf{Encipher}(X[1], X[2], 0^n)$, $C^*[3]$ is $Y[1]$ and T^* is $\mathtt{msb}_\tau(Y[2])$.

The final step is to make a decryption query (N', A^*, C^*, T^*), where A^* is empty, $C^* = (Y^* \oplus 2L', Y^* \oplus 2^2L', C^*[3])$, and $C^*[3]$ and T^* are obtained as above. The query will be accepted, and the oracle returns $M^* = (X^* \oplus 2L', X^* \oplus 2^2L', 0^n)$. The adversary can compute X^* from the knowledge of L', and we see that the entire process succeeds with an overwhelming probability.

Plaintext Recovery Attack (Short Plaintext). Here, we show that the plaintext recovery is possible even for short plaintexts. We first consider the case where $M^* = (M^*[1], M^*[2])$ is the target plaintext of two blocks. Let (N^*, A^*, C^*, T^*) be a challenge, where $C^* = (C^*[1], C^*[2])$ has two blocks. $L^* := E_K(N^*)$ can be recovered as in case for the plaintext recovery for long

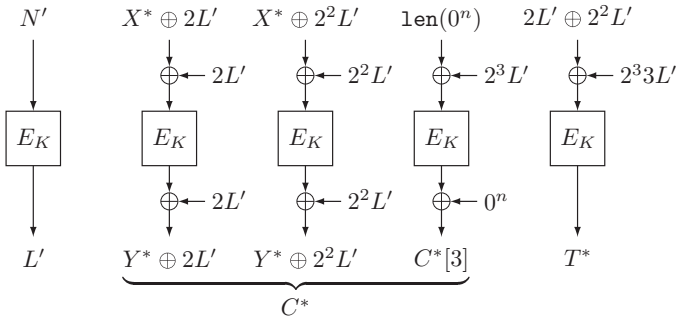

Fig. 7. The encryption process of (N', A^*, M^*). $C^*[3]$ and T^* are unknown.

plaintexts. We can then compute $\mathrm{Pad}^* := E_K(\mathtt{len}(C^*[2]) \oplus 2^2L^*)$ by calling $\mathsf{Encipher}(\mathtt{len}(C^*[2]) \oplus 2^2L^*, 0^n)$, and $M^*[2]$ can be obtained as $\mathtt{msb}_{|C^*[2]|}(\mathrm{Pad}^*) \oplus C^*[2]$. To recover $M^*[1]$, we need to compute $E_K^{-1}(C^*[1] \oplus 2L^*) \oplus 2L^*$, which can be done with the emulation of the blockcipher decryption we have just described.

When the target plaintext $M^* = M^*[1]$ has one block, we first recover $L^* := E_K(N^*)$, and then compute $\mathrm{Pad}^* := E_K(\mathtt{len}(C^*[1]) \oplus 2L^*)$ by calling $\mathsf{Encipher}(\mathtt{len}(C^*[1]) \oplus 2L^*, 0^n)$. This gives $M^*[1] = \mathtt{msb}_{|C^*[1]|}(\mathrm{Pad}^*) \oplus C^*[1]$.

Therefore, it is possible to mount a plaintext recovery attack against any challenge (N^*, A^*, C^*, T^*).

6 Design Flaw of OCB2

The root of the flaw in OCB2 is in the instantiation of AE using XEX*. For blockcipher E_K, let

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

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 $b = 0$. Rog04 refers b to *tag*; not to be confused with the tag in the global interface of AE.

Suppose an encryption query (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 (untrancated) tag.

In the proof of OCB2, we first apply the standard conversion from computational to information theoretic security [4] and focus on the security of OCB2 instantiated by an n-bit uniform random permutation (URP), P, denoted by $\mathrm{OCB2}_{\mathsf{P}}$. Then, the proof of $\mathrm{OCB2}_{\mathsf{P}}$ has two main steps: the indistinguishability of $\mathrm{XEX}^*_{\mathsf{P}}$, and the privacy and authenticity of AE[12] which replaces $\mathrm{XEX}^*_{\mathsf{P}}$ in $\mathrm{OCB2}_{\mathsf{P}}$ 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 $\mathrm{XEX}^*_{\mathsf{P}}$ is indistinguishable from $\widetilde{\mathsf{P}}$ for any adversary who queries to both encryption and decryption of $\mathrm{XEX}^*_{\mathsf{P}}$ 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*

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

Let $\Theta\mathrm{CB2}^{\widetilde{E}}$ be the mode of operations of TBC $\widetilde{E}_K$ which has the same interface as XEX^*_E. The pseudocode is shown in Fig. 8. Then, $\Theta\mathrm{CB2}_{\mathrm{XEX}^*_E}$ is equivalent to $\mathrm{OCB2}_E$.

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 $\mathcal{A}_{\pm}$,

$$\mathbf{Adv}^{\mathsf{priv}}_{\mathrm{OCB2}_{\mathsf{P}}}(\mathcal{A}) = \mathbf{Adv}^{\mathsf{priv}}_{\Theta\mathrm{CB2}_{\mathrm{XEX}^*_{\mathsf{P}}}}(\mathcal{A}) \le \mathbf{Adv}^{\mathsf{tprp}}_{\mathrm{XEX}^*_{\mathsf{P}}}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{priv}}_{\Theta\mathrm{CB2}^{\widetilde{\mathsf{P}}}}(\mathcal{A}), \tag{7}$$

$$\mathbf{Adv}^{\mathsf{auth}}_{\mathrm{OCB2}_{\mathsf{P}}}(\mathcal{A}_{\pm}) = \mathbf{Adv}^{\mathsf{auth}}_{\Theta\mathrm{CB2}_{\mathrm{XEX}^*_{\mathsf{P}}}}(\mathcal{A}_{\pm}) \le \mathbf{Adv}^{\mathsf{tsprp}}_{\mathrm{XEX}^*_{\mathsf{P}}}(\mathcal{B}_{\pm}) + \mathbf{Adv}^{\mathsf{auth}}_{\Theta\mathrm{CB2}^{\widetilde{\mathsf{P}}}}(\mathcal{A}_{\pm}) \tag{8}$$

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 $\Theta\mathrm{CB2}_{\mathrm{XEX}^*_{\mathsf{P}}}$ and $\mathcal{A}$ and $\mathcal{A}_{\pm}$, respectively. From Rog04, we have

$$\mathbf{Adv}^{\mathsf{tprp}}_{\mathrm{XEX}^*_{\mathsf{P}}}(\mathcal{B}) \le \frac{4.5q^2}{2^n}, \text{ and } \mathbf{Adv}^{\mathsf{tsprp}}_{\mathrm{XEX}^*_{\mathsf{P}}}(\mathcal{B}_{\pm}) \le \frac{9.5q^2}{2^n} \tag{9}$$

for any $\mathcal{B}$ and $\mathcal{B}_{\pm}$ that are tag-respecting and use at most q queries. The last terms of (7) and (8) 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 (9) and (7). 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 $\Theta\mathrm{CB2}$, even with single decryption query [13].

[12] An equivalent mode for OCB3 is called $\Theta\mathrm{CB3}$ [20].

[13] Rog04 defines the authenticity notion in the game that the adversary queries to the encryption oracle then outputs a query to the decryption oracle, but the response is not returned. The decryption oracle is not involved in the game and the success or failure of the forgery is determined outside the game. This definition itself is essentially the same as Equation (1), and has no problem. However, because the adversary's final output does not tell whether the adversary wins or loses, we do not know how to apply a hybrid argument of (8) using this definition.

Algorithm $\Theta\text{CB2}.\mathcal{E}^{\widetilde{E}}(N, A, M)$

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

Algorithm $\Theta\text{CB2}.\mathcal{D}^{\widetilde{E}}(N, A, C, T)$

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

Fig. 8. Algorithms of ΘCB2. For simplicity, $\tau = n$ and $A = \varepsilon$.

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 $|C'| = n$, as well as the attack of Section 4.1. 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 breaks confidentiality (Section 4.3).

7 Applicability to Related Schemes

Other OCB Versions. Our attacks are only applicable to OCB2. For OCB1, the last block is encrypted by XE with a clearly separated mask. For OCB3, the last block is encrypted by XEX when it is n bits and otherwise by XE with a mask separated from those used by XEX.

Other Designs based on OCB. We have not found other AE algorithms based on OCB that could be affected by our attacks. OTR [24] is an inverse-free (for the absence of the blockcipher decryption in the scheme) parallelizable AE having a similar structure as OCB. As it only uses XE for the whole process, it is safe from our attacks. OPP [12] is a permutation-based AE based on OCB. It always uses XEX, or more precisely, a variant of XPX [23], because otherwise an offline permutation inverse query easily breaks the scheme. It is safe because of this consistent use of XPX.

Aoki and Yasuda [2] presented security bounds of OCB when the block cipher has indistinguishability against encryption queries, however only unpredictability for decryption queries (thus is weaker than normal SPRPs). The presented bounds were claimed to cover all versions of OCB including OCB2. Therefore, our attacks invalidated them regarding OCB2.

8 Fixing OCB2

We discuss several ways to prevent our attacks in practice. In principle each of our suggestions would require its own formal security analysis, but we provide one only for the "XEX for the last plaintext block" fix presented in Section 8.1. While also our other proposals intuitively lead to a secure scheme, without conducting further research we cannot fully vouch for their security.

Always using AD. Our forgery attacks from Section 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 PMAC(A) but we are not aware of a way to do so. (Of course, if we could use the Encipher algorithm of Section 5.1 then computing PMAC values is not a challenge; however, Encipher 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 $\mathcal{E}$ in Fig. 1 that PMAC(A) is XOR-ed into the tag only if $A \neq \varepsilon$. We discuss the case that this condition is removed, and PMAC(A) is *always* XOR-ed into the tag, also when $A = \varepsilon$. An initial analysis of the PMAC algorithm (see Fig. 9 in Appendix) shows that the value PMAC(ϵ) 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 $2^m L \oplus \mathtt{len}(0^n)$.[14] While such

[14] We caution that this change might not be sufficient. Our results from Section 4.4 indicate that more plaintexts and ciphertexts have to be rejected: on the encryptor's

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.

USE XEX$^+$. Minematsu and Matsushima [26] 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.

8.1 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 OCB2.$\mathcal{E}_E$ and OCB2.$\mathcal{D}_E$ in Fig. 1 to

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

and

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

respectively. As well as 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 ΘCB2.$\mathcal{E}_{\widetilde{E}}$ and ΘCB2.$\mathcal{D}_{\widetilde{E}}$ in Fig. 8. Then $\Theta\text{CB2f}_{\widetilde{E}}$ is equivalent to OCB2f_E if $\widetilde{E}_K$ is XEX^*_E. To handle a 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 so that $\mathbb{PMAC}_{\text{XEX}^*_E}$ is equivalent to PMAC_E (see Fig. 9 in Appendix). We finally add the following line after line 8 (for ΘCB2.$\mathcal{E}_{\widetilde{E}}$ and ΘCB2.$\mathcal{D}_{\widetilde{E}}$) in Fig. 8

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

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:

---

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 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.

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

$$\mathbf{Adv}^{\mathsf{priv}}_{\mathrm{OCB2f}_{\mathsf{P}}}(\mathcal{A}) = \mathbf{Adv}^{\mathsf{priv}}_{\Theta\mathrm{CB2f}_{\mathrm{XEX}^*_{\widetilde{\mathsf{P}}}}}(\mathcal{A}) \leq \frac{5\sigma^2_{\mathrm{priv}}}{2^n},$$

$$\mathbf{Adv}^{\mathsf{auth}}_{\mathrm{OCB2f}_{\mathsf{P}}}(\mathcal{A}_{\pm}) = \mathbf{Adv}^{\mathsf{auth}}_{\Theta\mathrm{CB2f}_{\mathrm{XEX}^*_{\widetilde{\mathsf{P}}}}}(\mathcal{A}_{\pm}) \leq \frac{5\sigma^2_{\mathrm{auth}}}{2^n} + \frac{4q_v}{2^\tau},$$

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.

Intuitively, the security of OCB2f holds because (1) OCB2f is ΘCB2f using $\widetilde{E}$ instantiated by XEX^*, and (2) ΘCB2f and ΘCB2f′ are indistinguishable (up to collision), and (3) ΘCB2f′ 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 $\Theta\mathrm{CB2}_{\widetilde{\mathsf{P}}}$ with minor changes gives the desired results. A full proof is given in the full version of this article [13].

9 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 OCB3 has a superior software performance than the previous ones, and is clearly recommended by the designers, we think 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 the proofs need to be fixed. Even if we limit our focus to AE, we have many examples, such as NSA's Dual CTR [33,9], EAX-prime [25], GCM [19], and some of the CAESAR submissions [27,8,35] and more. We believe our work emphasizes the need for quality of security proofs, and their active verification.

Acknowledgements

The authors would like to thank Phil Rogaway for his response to our findings, and officials of ISO SC 27 for feedback and suggestions. We also would like to thank the reviewers of CRYPTO 2019 for useful comments.

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). https://doi.org/10.1007/978-3-662-45611-8_6
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). https://doi.org/10.1007/978-3-642-41227-1_12
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). https://doi.org/10.1007/978-3-319-63697-9_1
4. 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). https://doi.org/10.1109/SFCS.1997.646128
5. 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). https://doi.org/10.1007/978-3-540-25937-4_25
6. 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). https://doi.org/10.1007/978-3-642-03317-9_21
7. 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). https://doi.org/10.1007/3-540-46035-7_25
8. 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). https://doi.org/10.1007/978-3-662-53887-6_12
9. 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/`
10. 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/`
11. 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)
12. 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). https://doi.org/10.1007/978-3-662-49890-3_11
13. Inoue, A., Iwata, T., Minematsu, K., Poettering, B.: Cryptanalysis of OCB2: Attacks on authenticity and confidentiality. IACR Cryptology ePrint Archive **2019**, 311 (2019), `https://eprint.iacr.org/2019/311`

14. Inoue, A., Minematsu, K.: Cryptanalysis of OCB2. IACR Cryptology ePrint Archive **2018**, 1040 (2018), https://eprint.iacr.org/2018/1040
15. ISO: Information Technology - Security techniques - Authenticated encryption, ISO/IEC 19772:2009. International Standard ISO/IEC 19772 (2009)
16. 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
17. Iwata, T.: Plaintext Recovery Attack of OCB2. IACR Cryptology ePrint Archive **2018**, 1090 (2018), https://eprint.iacr.org/2018/1090
18. 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). https://doi.org/10.1007/978-3-540-39887-5_11
19. 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). https://doi.org/10.1007/978-3-642-32009-5_3
20. 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). https://doi.org/10.1007/978-3-642-21702-9_18
21. Krovetz, T., Rogaway, P.: The OCB Authenticated-Encryption Algorithm. IRTF RFC 7253 (2014)
22. 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). https://doi.org/10.1007/3-540-45708-9_3
23. 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). https://doi.org/10.1007/978-3-662-53018-4_3
24. 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). https://doi.org/10.1007/978-3-642-55220-5_16
25. 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). https://doi.org/10.1007/978-3-662-43933-3_17
26. Minematsu, K., Matsushima, T.: Generalization and Extension of XEX* Mode. IEICE Transactions **92-A**(2), 517–524 (2009)
27. 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). https://doi.org/10.1007/978-3-662-45611-8_7
28. Poettering, B.: Breaking the confidentiality of OCB2. IACR Cryptology ePrint Archive **2018**, 1087 (2018), https://eprint.iacr.org/2018/1087
29. 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). https://doi.org/10.1145/586110.586125

30. 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). https://doi.org/10.1007/978-3-540-30539-2_2
31. Rogaway, P.: Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. Full version of [30] (2004), available from `http://www.cs.ucdavis.edu/~rogaway/papers/`
32. 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). https://doi.org/10.1007/978-3-540-25937-4_22
33. Rogaway, P.: On the Role Definitions in and Beyond Cryptography. In: ASIAN. LNCS, vol. 3321, pp. 13–32. Springer (2004)
34. 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). https://doi.org/10.1145/501983.502011
35. 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). https://doi.org/10.1007/978-3-319-31301-6_11
36. 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)
37. 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). https://doi.org/10.1007/978-3-319-93387-0_25

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 [14,28,17] that appeared in late 2018 on the IACR ePrint archive. That OCB2 might be flawed was first identified by the authors of [14] when 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 [14]), the author of the second ePrint article [28] announced an IND-CCA vulnerability and first steps towards plaintext recovery, and again three days later, the author of the third ePrint article [17] 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.

Algorithm $\mathrm{PMAC}_E(A)$

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

Algorithm $\mathbb{PMAC}_{\widetilde{E}}(A)$

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

Fig. 9. Left: The algorithm PMAC_E for the use in OCB2. **Right:** A TBC-based PMAC, $\mathbb{PMAC}_{\widetilde{E}}$.

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. 9. 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 $0^{120}10^7$.