Cryptanalysis of OCB2

Akiko Inoue and Kazuhiko Minematsu

NEC Corporation, Japan
a-inoue@cj.jp.nec.com, k-minematsu@ah.jp.nec.com

Abstract. We present practical attacks against OCB2, an ISO-standard authenticated encryption (AE) scheme. OCB2 is a highly-efficient blockcipher mode of operation. It has been extensively studied and widely believed to be secure thanks to the provable security proofs. Our attacks allows the adversary to create forgeries with (almost-known) single encryption query. The source of our attacks is the way OCB2 implements AE using a tweakable blockcipher, called XEX*. We have verified our attacks using a reference code of OCB2. Our attacks do not break the privacy of OCB2, and are not applicable to the others, including OCB1 and OCB3.

Keywords: OCB, Authenticated Encryption, Cryptanalysis, Forgery, XEX

1 Introduction

Authenticated encryption (AE) is a form of symmetric-key encryption that provides both confidentiality and authenticity of the message. Now it is widely accepted that AE is a fundamental security tool for many practical applications, such as TLS.

OCB is a blockcipher mode of operation for AE. It is one of the most celebrated schemes in the cryptography for its beautiful and innovative architecture. OCB is very efficient. In fact, it is as efficient as encryption-only modes, and is parallelizable. There are three versions of OCB. The first version (OCB1) was proposed at ACM CCS 2001 by Rogaway et. al [RBBK01]. The second one (OCB2) was proposed at Asiacrypt 2004 by Rogaway [Rog04a] (hereafter Rog04), and the latest one (OCB3) was proposed at FSE 2011 by Krovetz and Rogaway [KR11]. Each version of OCB has received significant attentions from theory and practice. OCB1 was considered for the security mechanism of Wireless LAN (IEEE 802-11), OCB2 is standardized in ISO/IEC 19772:2009 [ISO09], and OCB3 is in RFC 7253 [RFC14]. Moreover, OCB3 is a finalist of CAESAR competition[1]. Various versions of OCB have been implemented in the cryptographic libraries, such as Botan, BouncyCastle, LibTomCrypt, OpenSSL, SJCL etc.

The security of (all versions of) OCB has been extensively studied. For each version, the author(s) provided the security reductions, that is, a break of OCB

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

implies the break of the underlying blockcipher. If the blockcipher is secure, the security bounds (roughly, the success probability of attacks) are $O(\sigma^2/2^n)$ for σ processed blocks, called "birthday bounds" named after the birthday paradox. From the attacker point of view, Ferguson [Fer02] and Sun et.al [SWZ12] showed collision attacks that need $2^{n/2}$ processed blocks, showing the security bounds are tight. Attacks in the misuse scenarios are proposed by Andreeva et. al [ABL$^+$14] and Ashur et. al [ADL17]. Robustness of the CAESAR candidates including OCB was studied by Damian and Vaudenay [VV18]. From the provable security point of view, Aoki and Yasuda [AY13] showed a relaxation on the assumption of the underlying blockcipher in a modified OCB, and Ritam and Nandi [BN17] showed an improvement on the security bound of OCB3.

Despite aforementioned attacks and studies on provable security, it is widely believed that the security proofs are correct, thus all versions of OCB are provably secure in the standard, nonce-respecting scenario, up to the birthday bounds.

In this paper, we first invalidate this belief by presenting an attack against OCB2. The attack is practical: it needs only one encryption query to create a forgery with success probability one. No heavy computation or large memory is needed, we just take a sum of some blocks to create a forgery. The attack works for any n-bit blockcipher E, and works for any n for which OCB2 can be defined. The forgery can be any long ciphertext depending on the first encryption query, and the encryption query is almost a known-plaintext query. In addition, while the first forgery is existential, the subsequent forgeries have much more freedom, as the adversary learns the mask applied to E from the first forgery. This enables to learn multiple blockcipher inputs and outputs, and it results in an almost-universal forgery attack in the reforgeability setting [BC09, FLLW17] (see Section 4.3).

Our attacks point out a critical flaw of OCB2, and shows that the authenticity bound of OCB2 turns out to be one. We emphasize that this is not a flaw of the global structure of OCB shared by the all versions, but is about the details of OCB2 that has been overlooked for more than a decade.

The root of our attacks is in the tweakable blockcipher (TBC) [LRW02] inside OCB2, called XEX*. Here, an encryption of XEX* is either $C = \Delta \oplus E_K(\Delta \oplus M)$ or $C = E_K(\Delta \oplus M)$, where Δ is derived from the encryption of nonce multiplied by a constant in GF(2^n). The former is called XEX, and the latter is XE, hence XEX* combines XEX and XE.

Original security proof of Rog04 showed that XEX* implements a provably secure TBC and OCB2 can be seen as a mode of XEX*. Rog04 concluded that OCB2 is provably secure since if XEX* was replaced with a perfect tweakable random permutation, the resulting mode is ideally secure. This is an elegant hybrid argument, which significantly reduces the complexity of the proof. In fact, the proof strategy of Rog04 and the idea of using TBC as a mode component was followed by many schemes. In the ongoing CAESAR competition, the most submissions of parallelizable AE based on (tweakable) blockcipher or cryptographic permutation, including OCB3 itself, employed the strategy of Rog04 or OCB in general. However, since XEX* is not a plain TBC (called *tagged* TBC), there are

some access rules that the adversary must follow in the game of distinguishing XEX* from the ideal. We found that, even though OCB2 is a mode of XEX*, the authenticity game for OCB2 forces the adversary to violate the access rule of XEX*. This implies that, even if XEX* is a secure (tagged) TBC, its security can not imply the security of OCB2. When OCB2 is invoked, this violated access occurs around the processing of the last two message blocks, where XEX is used for the last-but-one block and XE is used for the last one. The attack exploits this "violated" access and a relationship between the different masks applied to the blockcipher.

We have confirmed our attacks using the reference code of OCB2 by Krovetz [2]. Appendix A shows an example.

We also show how to fix OCB2, say by using XEX for the last block. Our attack is not applicable to OCB1 and OCB3, and we have not found other schemes based on OCB that could be affected.

2 Preliminaries

2.1 Notations

Let $\{0,1\}^*$ be the set of all binary strings, including the empty string ε. For $X \in \{0,1\}^*$, its bit length is denoted by $|X|$, and the first $c \leq |X|$ bits is denoted by $\mathtt{msb}_c(X)$. Here, $|\varepsilon| = 0$. Let 0^c be the sequence of c zeros, where $0^0 = \varepsilon$. The concatenation of two bit strings X and Y is written $X \,\|\, Y$, or XY when no confusion is possible.

When $0 \leq |X| \leq n$, let $\mathtt{len}(X)$ be the n-bit encoding of $|X|$. We also define zero padding, $X \,\|\, 0^*$, and one-zero padding, $X \,\|\, 10^*$. Both are X when $|X| = n$ and $X \,\|\, 0^* = X \,\|\, 0^{n-|X|}$ and $X \,\|\, 10^* = X \,\|\, 10^{n-|X|-1}$ when $0 \leq |X| < n$.

For $X \in \{0,1\}^*$, we also define the parsing 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 $X[1] \xleftarrow{n} X$, where $X[1] = \varepsilon$.

If X is a random variable distributed uniformly over $\mathcal{X}$, we write $X \xleftarrow{\$} \mathcal{X}$.

(Tweakable) Blockcipher. A tweakable blockcipher (TBC) [LRW02] is a keyed function $\widetilde{E} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \mathcal{M}$ such that for each $(K,T) \in \mathcal{K} \times \mathcal{T}$, $\widetilde{E}(K,T,\cdot)$ is a permutation over $\mathcal{M}$. Here, K is the key and T is a public value called tweak. A conventional block cipher is a TBC with $\mathcal{T}$ being a singleton, and specifically written as $E : \mathcal{K} \times \mathcal{M} \to \mathcal{M}$. The encryption of $X \in \mathcal{M}$ under key $K \in \mathcal{K}$ and tweak $T \in \mathcal{T}$ is $\widetilde{E}(K,T,X)$ and is also written as $\widetilde{E}_K(T,X)$ or $\widetilde{E}_K^T(X)$. For blockcipher we write as $E_K(X)$. The decryption is written as

[2] `http://web.cs.ucdavis.edu/~rogaway/ocb/code-2.0.htm`.

$\widetilde{E}_K^{-1,T}(Y)$ 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 $\widetilde{E}_K^{-1,T}(Y) = X$.

We may omit the subscript K when being keyed with K is obvious. Moreover, for a mode of operation using E_K, we may treat E_K as a key and write as Mode_E. We typically assume $n = 128$.

Galois Field. For $a \in \{0,1\}^n$, we may see it as an element of $\mathrm{GF}(2^n)$ by taking it as a coefficient vector. Following [Rog04a][IK03], by writing $2 \cdot a$ or $2a$, we mean the *doubling* over $\mathrm{GF}(2^n)$, i.e., the multiplication by the generator of the field (the polynomial $\mathtt{x}$) and a.

In a similar fashion, $2^c a$ denotes c-times doublings of a, and $3a$ denotes $2a \oplus a$. Combined expressions such as $2^i \cdot 3^j a$ are similarly defined. For example, $2^2 \cdot 3a = 3(2^2 a) = 2^3 a \oplus 2^2 a$.

For $n = 128$, it is common to define the field $\mathrm{GF}(2^n)$ by the lexicographically-first primitive polynomial, $\mathtt{x}^{128} + \mathtt{x}^7 + \mathtt{x}^2 + \mathtt{x} + 1$. With this, $2a$ is $(a \ll 1)$ if $\mathtt{msb}_1(a) = 0$ and $(a \ll 1) \oplus (0^{120}10000111)$ if $\mathtt{msb}_1(a) = 1$, where $(a \ll 1)$ denotes the left-shift of a by one bit.

2.2 AEAD

In most cases, we say AE to mean Authenticated Encryption with Associated Data (AEAD) [Rog02]. Let Π be an AEAD scheme defined over key space $\mathcal{K}$, nonce space $\mathcal{N}$, associated data (AD) space $\mathcal{A}$, message space $\mathcal{M}$ and tag space $\mathcal{T} = \{0,1\}^\tau$ for some fixed τ. Here, AD is a part of message that should not be encrypted but authenticated. For example AD can be the header information of a network protocol. The encryption of Π is a function $\Pi.\mathcal{E} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{M} \times \mathcal{T}$, and the decryption (or verification) is a function $\Pi.\mathcal{D} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{M} \cup \{\perp\}$, where $\perp$ denotes the verification failure. To encrypt plaintext M with nonce N and AD A and key K, we compute $\Pi.\mathcal{E}_K(N, A, M) = (C, T)$ to produce ciphertext C and tag T. The tuple (N, A, C, T) is sent to the receiver. For decryption, we compute $\Pi.\mathcal{D}_K(N', A', C', T')$ and it returns the plaintext M' if the verification successes, and $\perp$ if the verification fails. We write Π_K to mean the tuple $(\Pi.\mathcal{E}_K, \Pi.\mathcal{D}_K)$.

Security Notions. The security of AEAD can be defined by two notions, privacy and authenticity [BRW04, Rog04c]. The privacy notion captures the indistinguishability from random for a nonce-respecting adversary in a chosen-plaintext-attack setting. Let $\mathcal{A}^{O_1, O_2, \ldots, O_c}$ denote the adversary $\mathcal{A}$ accessing c oracles, $O_1, \ldots, O_c$, in an arbitrarily order. The privacy notion is defined over the game where $\mathcal{A}^{O_1}$ returns a binary decision at the end of the game, and O_1 is either $\Pi.\mathcal{E}_K$ or the random-bits oracle \$. Here, \$ returns $(C, T) \xleftarrow{\$} \{0,1\}^{|M|} \times \mathcal{T}$ for any encryption query (N, A, M). The privacy advantage is defined as

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

where $\mathcal{A}^{O_1} \Rightarrow 1$ denotes the probability that the final decision of $\mathcal{A}$ is 1. We assume that $\mathcal{A}$ in the privacy game is nonce-respecting, that is, $\mathcal{A}$ does not make two queries with the same nonce.

For the authenticity notion, we consider $\mathcal{A}$ accessing $\Pi.\mathcal{E}_K$ and $\Pi.\mathcal{D}_K$, where $\mathcal{A}$ is considered to win when it creates a successful forgery. Formally, the authenticity advantage is defined as

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

where $\mathcal{A}$ forges if it receives any $M' \neq \perp$ from $\Pi.\mathcal{D}_K$. To avoid a trivial win, $\mathcal{A}$ is assumed not to query (N, A, C, T) to $\Pi.\mathcal{D}_K$ when there is a previous encryption query (N, A, M) for some M and the corresponding response (C, T). We also assume $\mathcal{A}$ is nonce-respecting with respect to encryption queries.

2.3 Security of (Tweakable) Blockciphers

Let $\widetilde{E}_K : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \mathcal{M}$ be a TBC. Let $\widetilde{\mathsf{P}}$ be the tweakable uniform random permutation (TURP). This is an information-theoretic TBC with key uniformly distributed over all the tweakable permutations, i.e. the set of $f(T, X) : \mathcal{T} \times \mathcal{M} \to \mathcal{M}$ such that $f(T, *)$ is a permutation over $\mathcal{M}$ for all $T \in \mathcal{T}$. The decryption is denoted by $\widetilde{\mathsf{P}}^{-1}$. We define Tweakable Pseudorandom Permutation (TPRP) advantage and Tweakable Strong PRP (TSPRP) advantage:

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

$$\mathbf{Adv}^{\mathsf{tsprp}}_{\widetilde{E}_K}(\mathcal{A}_{\pm}) \stackrel{\text{def}}{=} \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\widetilde{E}_K, \widetilde{E}_K^{-1}} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\widetilde{\mathsf{P}}, \widetilde{\mathsf{P}}^{-1}} \Rightarrow 1\right]. \quad (2)$$

Here, $\mathcal{A}$ performs Chosen-plaintext attack with chosen tweaks, and $\mathcal{A}_{\pm}$ performs Chosen-ciphertext attack with chosen tweaks (i.e. it can query any (T, M) to encrypt and any (T, C) to decrypt).

For blockcipher $E : \mathcal{K} \times \mathcal{M} \to \mathcal{M}$, we analogously define PRP advantage $\mathbf{Adv}^{\mathsf{prp}}_{E_K}(\mathcal{A})$ and SPRP advantage $\mathbf{Adv}^{\mathsf{sprp}}_{E_K}(\mathcal{A}_{\pm})$, using URP P (which uniformly distributes over all permutations over $\mathcal{M}$) as an ideal primitive.

3 OCB2

As described, OCB2 is a blockcipher mode for AEAD proposed at [Rog04a]. It is parallelizable, and is rate-1 as it needs one blockcipher call to process one message block. The pseudocode of OCB2 is shown in Fig. 1. The basic form of OCB2 presented in Rog04 (Fig. 2) is a plain AE[3], where there is no AD. Section 11 of Rog04 described that the plain OCB2 can be converted into an AEAD, by computing PMAC for AD and taking a sum (XOR) of PMAC output and the tag of plain OCB2 *only if* AD is non-empty. This scheme is referred

[3] It was named OCB1 in the paper.

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

1. $L \leftarrow E(N)$
2. $(M[1], \ldots, M[m]) \xleftarrow{n} M$
3. **for** $i = 1$ **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** $i = 1$ **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** $\perp$

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

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

Fig. 1. Algorithms of OCB2.

to as AEM in Rog04. The specification of PMAC in OCB2 is shown in the full version of Rog04 [Rog04b]. It is different from the initial version [BR02] in that it uses doublings for mask generation and is designed so that it is computationally independent from the plain AE part. We remark that our attack is independent of the specification of PMAC.

4 Attacks

4.1 Minimal Example of Forgery

We describe the minimal example of our forgery attack against OCB2. For simplicity, let $\tau = n$.

1. Encrypt (N, A, M), where A is empty, M is a $2n$-bit message $M = M[1] \,\|\, M[2]$ specified as
$$M[1] = \mathtt{len}(0^n),$$

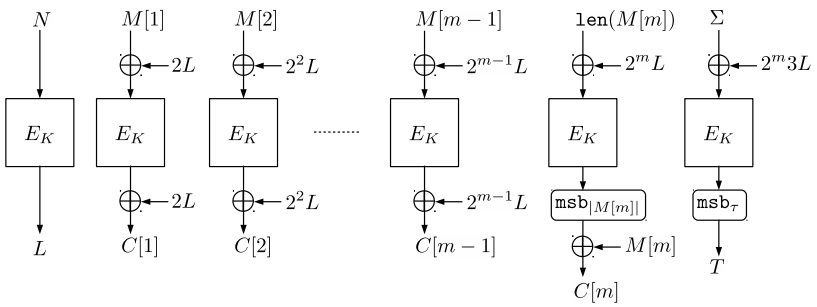

Fig. 2. OCB2 for the case of empty AD.

and $M[2]$ is any n-bit block. Encryption oracle returns the pair of $2n$-bit ciphertext and tag, $C = C[1] \,\|\, C[2]$ and T.

2. Decrypt (N', A', C', T') with $|C'| = n$ such that

$$N' = N, \tag{3}$$
$$A' = \varepsilon, \tag{4}$$
$$C' = C[1] \oplus \mathtt{len}(0^n), \tag{5}$$
$$T' = M[2] \oplus C[2] \tag{6}$$

and T' will be always accepted as valid, while clearly $C' \neq C$ (different lengths), implying a successful forgery.

Why this works? Because

$$C[1] = 2L \oplus E(2L \oplus \mathtt{len}(0^n)) \tag{7}$$
$$C[2] = M[2] \oplus \mathrm{Pad}, \tag{8}$$

where $L = E(N)$ and $\mathrm{Pad} = E(2^2 L \oplus \mathtt{len}(M[2])) = M[2] \oplus C[2]$. Let Pad' and Σ' be the true values for the decryption query. Then C' is decrypted to

$$\begin{aligned}
M' &= C' \oplus \mathrm{Pad}' && (9)\\
&= C' \oplus \underbrace{E(2L \oplus \mathtt{len}(0^n))}_{\mathrm{Pad}'} && (10)\\
&= C[1] \oplus \mathtt{len}(0^n) \oplus E(2L \oplus \mathtt{len}(0^n)) && (11)\\
&= \underbrace{2L \oplus E(2L \oplus \mathtt{len}(0^n))}_{C[1]} \oplus \mathtt{len}(0^n) \oplus E(2L \oplus \mathtt{len}(0^n)) && (12)\\
&= 2L \oplus \mathtt{len}(0^n), && (13)
\end{aligned}$$

and the tag is generated as

$$\begin{aligned} T^* &= E(2 \cdot 3L \oplus \Sigma') && (14)\\ &= E(2 \cdot 3L \oplus \underbrace{C' \oplus \mathrm{Pad}'}_{\Sigma'}) && (15)\\ &= E(2 \cdot 3L \oplus M') && (16)\\ &= E(2 \cdot 3L \oplus \underbrace{2L \oplus \mathtt{len}(0^n)}_{M'}) && (17)\\ &= E(2^2 L \oplus \mathtt{len}(0^n)) && (18)\\ &= \mathrm{Pad}, && (19) \end{aligned}$$

where (18) follows from $2 \cdot 3L = 2^2 L \oplus 2L$. Thus, T^* is determined by the first encryption query. See Fig. 3 for the graphical representation of the attack. As we mentioned, the attack is independent of the AD processing function (PMAC), and of the specification of $\mathtt{len}(*)$.

In the reference code, the input message is assumed to be a byte string and $\mathtt{len}(X)$ is specified as $0^{n-8} \| \ell_X$, where ℓ_X denotes the standard binary encoding of $|X|$. Hence $\mathtt{len}(0^n)$ for $n = 128$ is $0^{120}10^7$.

Extensions. Since we do not need the information on T in the first encryption query, the above attack works if AD is non-empty at the encryption query. The decryption query needs the empty AD. For the same reason, it also works even when the tag is truncated to $\tau < n$ bits; we set $T' = \mathtt{msb}_\tau(M[2] \oplus C[2])$ and it will be accepted with probability one.

4.2 Forgery of Longer Messages

The attack of Section 4.1 can be extended to arbitrarily long messages:

1. Encrypt (N, A, M), where A is arbitrary, M is a message of m blocks $M = M[1] \| \cdots \| M[m-1] \| M[m]$ satisfying

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

   and $M[m]$ is any s-bit string such that $\tau \le s \le n$. Encryption oracle returns the pair of ciphertext and tag, $C = C[1] \| \cdots \| C[m-1] \| C[m]$ and T.
2. Decrypt (N', A', C', T'), such that $N' = N$, $A' = \varepsilon$ and C' has $m-1$ blocks, $C' = C'[1] \| \cdots \| C'[m-2] \| C'[m-1]$, specified as

   $$\begin{aligned} C'[i] &= C[i] \text{ for } 1 \le i \le m-2, && (20)\\ C'[m-1] &= \sum_{i=1}^{m-2} M[i] \oplus C[m-1] \oplus \mathtt{len}(M[m]) && (21)\\ T' &= \mathtt{msb}_\tau(M[m] \oplus C[m]). && (22) \end{aligned}$$

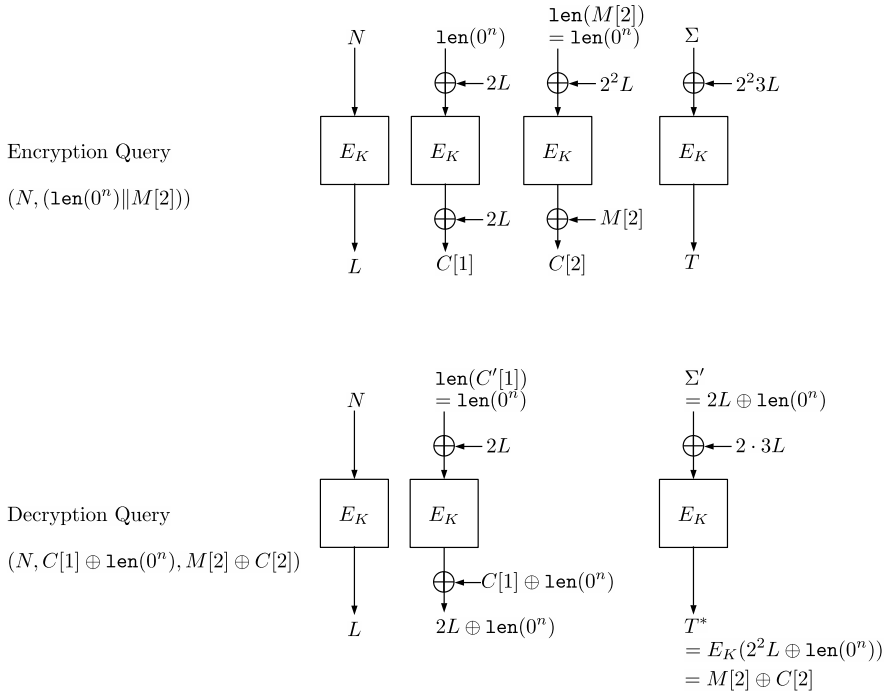

Fig. 3. Attack described at Section 4.1.

This forgery attack again works. Let $\overline{T}'$ be the true string of untruncated tag in the decryption query. Then we have

$$\overline{T}' = E(\Sigma' \oplus 3 \cdot 2^{m-1}L) \tag{23}$$

$$= E\left(\left(\sum_{i=1}^{m-2} M'[i] \oplus C'[m-1] \oplus \mathrm{Pad}'\right) \oplus 3 \cdot 2^{m-1}L\right) \tag{24}$$

$$= E\left(\sum_{i=1}^{m-2} M[i] \oplus C'[m-1] \oplus C[m-1] \oplus 2^{m-1}L \oplus 3 \cdot 2^{m-1}L\right), \tag{25}$$

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 (25) is further

expanded as

$$E\left(\sum_{i=1}^{m-2} M[i] \oplus C'[m-1] \oplus C[m-1] \oplus 2^m L\right) \tag{26}$$

$$= E\left(\sum_{i=1}^{m-2} M[i] \oplus \left(\sum_{i=1}^{m-2} M[i] \oplus C[m-1] \oplus \mathtt{len}(M[m])\right) \oplus C[m-1] \oplus 2^m L\right) \tag{27}$$

$$= E\left(\mathtt{len}(M[m]) \oplus 2^m L\right) \tag{28}$$

$$= \mathrm{Pad} \tag{29}$$

Finally, we have

$$\begin{aligned} T^* &= \mathtt{msb}_\tau(\overline{T}') && (30)\\ &= \mathtt{msb}_\tau(\mathrm{Pad}) && (31)\\ &= \mathtt{msb}_\tau(M[m] \oplus C[m]) \quad (\because \tau \le |M[m]| \le n) && (32)\\ &= T' && (33) \end{aligned}$$

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

4.3 Almost Universal Forgery, Variant 1

We consider what will happen after the attack of Section 4.1 or 4.2, i.e. attacks in the reforgeability scenario. Here, the adversary learns $L = E(N)$ by the first forgery, since

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

is returned by the decryption oracle, for the case of attack of Section 4.2. For $1 \le i \le m-1$, let $X[i]$ and $Y[i]$ be $X[i] = M[i] \oplus 2^i L$ and $Y[i] = C[i] \oplus 2^i L$, respectively. In addition, let $X[m] = \mathtt{len}(M[m]) \oplus 2^m L$ and $Y[m] = \mathrm{Pad}$. With the knowledge of L, the adversary learns the m pairs of E's input and output, $\mathcal{S} = \{(X[i], Y[i]) : i = 1, \ldots, m\}$, which allows the second forgery with more freedom than the first. For example, if the adversary wants to disturb only the j-th and k-th plaintext blocks for some $1 \le j < k \le m-1$, the second decryption query $(N'', A'', C''[1] \,\|\, \cdots \,\|\, C''[m], T'')$ can be set as $N'' = N$, $A'' = \varepsilon$, and

$$\begin{aligned} C''[i] &= C[i] \quad (i \in \{1, \cdots, m\} \setminus \{j, k\}), && (34)\\ C''[j] &= Y[k] \oplus 2^j L, && (35)\\ C''[k] &= Y[j] \oplus 2^k L, && (36)\\ T'' &= T. && (37) \end{aligned}$$

Then, the unverified plaintext will be

$$M''[i] = M[i] \quad (i \in \{1, \cdots, m\} \setminus \{j, k\}) \tag{38}$$
$$M''[j] = X[k] \oplus 2^j L \tag{39}$$
$$M''[k] = X[j] \oplus 2^k L \tag{40}$$

and the valid checksum Σ'' is

$$\Sigma'' = \bigoplus_{i=1}^{m-1} M''[i] \oplus Y[m] \oplus C''[m] \tag{41}$$
$$= \bigoplus_{i \in \{1,\cdots,m-1\} \setminus \{j,k\}} M[i] \oplus M''[j] \oplus M''[k] \oplus \mathrm{Pad} \oplus C[m] \tag{42}$$
$$= \bigoplus_{i \in \{1,\cdots,m-1\} \setminus \{j,k\}} M[i] \oplus M[j] \oplus M[k] \oplus \mathrm{Pad} \oplus C[m] \tag{43}$$
$$= \bigoplus_{i=1}^{m-1} M[i] \oplus \mathrm{Pad} \oplus C[m] \tag{44}$$
$$= \Sigma, \tag{45}$$

sicne $M''[j] \oplus M''[k] = X[k] \oplus 2^j L \oplus X[j] \oplus 2^k L = M[j] \oplus M[k]$. Therefore, the decryption query will be accepted.

Generally, the adversary can arbitrarily choose the i-th plaintext blocks in the second forgery from $\mathcal{I}_i = \{X[j] \oplus 2^i L : j = 1, \ldots, m-1\}$, and by choosing the last ciphertext block, the checksum can be set to (say) the previous checksum value in the encryption query so that the tags are matched. This is a powerful, almost-universal forgery.

However, this attack will not work when the receiver is stateful and detects a replay in nonce, as the first and second forgeries share the same nonce.

4.4 Almost Universal Forgery, Variant 2

To bypass the problem of stateful receiver mentioned above, we consider another variant of attack. This will create an almost universal forgery for a nonce that is never queried with a help of few additional queries.

Suppose the adversary $\mathcal{A}$ performs the forgery attack of Section 4.2 using ℓ blocks. Then she obtains ℓ input-output pairs of E, $\mathcal{S} = \{(X[i], Y[i]) : i = 1, \ldots, \ell\}$, where $E(X[i]) = Y[i]$ for $1 \le i \le \ell$. Let $\mathcal{X} = \{X[1], \ldots, X[\ell]\}$ and $\mathcal{Y} = \{Y[1], \ldots, Y[\ell]\}$. We assume $\mathcal{A}$ wants to create a forgery of the form (N'', A'', C'', T'') with $A'' = \varepsilon$, $|C''|_n = m$, and $N'' = X[k]$ for some $k \in \{1, \ldots, \ell\}$, where N'' has never been used as a nonce for encryption and decryption queries in order to avoid the detectable replay at the receiver. Since she already knows the initial mask $E(N'') = Y[k]$ and $\mathcal{S}$, the most blocks of C'' can be set so that the decrypted plaintext blocks are predictable. The problem is in the last block ($C''[m]$), where we need to make sure that $\mathtt{len}(C''[m]) \oplus 2^m E(N'') \in \mathcal{X}$

so that the corresponding Pad'' is predictable. This is possible by making another encryption query with nonce $N^\$ = \mathtt{len}(C''[m]) \oplus 2^m E(N'')$ and performing the forgery with nonce $N^\$$ to recover $E(N^\$)$. The detailed attack procedure is as follows. For simplicity, we fix $|C''[m]| = n$.

1. Perform the forgery attack of Section 4.2 to obtain $\mathcal{X}$ and $\mathcal{Y}$ of size ℓ.
2. Choose $N'' = X[k] \in \mathcal{X}$ for some k as the target nonce, and set $N^\$ = \mathtt{len}(C''[m]) \oplus 2^m E(N'') = \mathtt{len}(0^n) \oplus 2^m Y[k]$.
3. Perform the forgery of Section 4.1 or 4.2 with nonce $N^\$$. Recover $E(N^\$)$.
4. Arbitrarily choose m elements of $\mathcal{Y}$, $Y[j_1], \ldots, Y[j_{m-1}]$ and $Y[j_{m+1}]$, possibly with duplications. Let $Y[j_m] = E(N^\$)$.
5. Decrypt (N'', A'', C'', T'') such that

$$\begin{aligned} &C''[i] = Y[j_i] \oplus 2^i E(N'') = Y[j_i] \oplus 2^i Y[k], \text{ for } i = 1, \ldots, m-1, \\ &C''[m] = M''[m] \oplus E(N^\$), \\ &T'' = Y[j_{m+1}] \end{aligned}$$

where

$$\begin{aligned} M''[i] &= X[j_i] \oplus 2^i E(N'') = X[j_i] \oplus 2^i Y[k], \\ M''[m] &= X[j_{m+1}] \oplus \bigoplus_{i=1}^{m-1} M''[i] \oplus 3 \cdot 2^m E(N''). \end{aligned}$$

This will be accepted as valid since the true tag is

$$E(\Sigma'' \oplus 3 \cdot 2^m E(N'')) = E\left(\bigoplus_{i=1}^{m-1} M''[i] \oplus M''[m] \oplus 3 \cdot 2^m E(N'')\right) \tag{46}$$

$$= E(X[j_{m+1}]) = Y[j_{m+1}]. \tag{47}$$

Here, Step 3 of our attack can be omitted if $N^\$$ is already in $\mathcal{X}$ by chance.

5 Design Flaw of OCB2

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

$$\mathrm{XEX}_E^{N,i,j}(X) \stackrel{\text{def}}{=} E(2^i L \oplus X) \oplus 2^i L, \tag{48}$$

$$\mathrm{XE}_E^{N,i,j}(X) \stackrel{\text{def}}{=} E(2^i 3^j L \oplus X), \tag{49}$$

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,

$$\mathrm{XEX}_E^{*,b,N,i,j}(X) = \begin{cases} \mathrm{XEX}_E^{N,i,j}(X) & \text{if } b = 1; \\ \mathrm{XE}_E^{N,i,j}(X) & \text{if } b = 0. \end{cases} \tag{50}$$

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 [BDJR97] 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[4] 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;*
2. *Once* $\mathrm{XEX}^{*,b,N,i,j}$ *is queried in either encryption or decryption, then it is not allowed to query* $\mathrm{XEX}^{*,1-b,N,i,j}$.

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 by Fig. 4. 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 $\mathcal{A}_{\pm}$, we have

$$\mathbf{Adv}^{\mathsf{priv}}_{\mathrm{OCB2}_{\mathsf{P}}}(\mathcal{A}) = \mathbf{Adv}^{\mathsf{priv}}_{\Theta\mathrm{CB2}_{\mathrm{XEX}^*_{\widetilde{\mathsf{P}}}}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{tprp}}_{\mathrm{XEX}^*_{\widetilde{\mathsf{P}}}}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{priv}}_{\Theta\mathrm{CB2}_{\widetilde{\mathsf{P}}}}(\mathcal{A}) \tag{51}$$

$$\mathbf{Adv}^{\mathsf{auth}}_{\mathrm{OCB2}_{\mathsf{P}}}(\mathcal{A}_{\pm}) = \mathbf{Adv}^{\mathsf{auth}}_{\Theta\mathrm{CB2}_{\mathrm{XEX}^*_{\widetilde{\mathsf{P}}}}}(\mathcal{A}_{\pm}) \leq \mathbf{Adv}^{\mathsf{tsprp}}_{\mathrm{XEX}^*_{\widetilde{\mathsf{P}}}}(\mathcal{B}_{\pm}) + \mathbf{Adv}^{\mathsf{auth}}_{\Theta\mathrm{CB2}_{\widetilde{\mathsf{P}}}}(\mathcal{A}_{\pm}) \tag{52}$$

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

$$\mathbf{Adv}^{\mathsf{tprp}}_{\mathrm{XEX}^*_{\widetilde{\mathsf{P}}}}(\mathcal{B}) \leq \frac{4.5q^2}{2^n}, \text{ and } \mathbf{Adv}^{\mathsf{tsprp}}_{\mathrm{XEX}^*_{\widetilde{\mathsf{P}}}}(\mathcal{B}_{\pm}) \leq \frac{9.5q^2}{2^n} \tag{53}$$

for any $\mathcal{B}$ and $\mathcal{B}_{\pm}$ that are tag-respecting and use at most q queries. The last terms of (51) and (52) are proved to be almost ideally small: zero for privacy and $2^{n-\tau}/(2^n - 1)$ for authenticity with single decryption query.

[4] An equivalent mode for OCB3 is called ΘCB3 [KR11].

Algorithm ΘCB2.$\mathcal{E}_{\widetilde{E}}(N, A, M)$

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 ΘCB2.$\mathcal{D}_{\widetilde{E}}(N, A, C, T)$

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

The privacy bound is obtained from (53) and (51). 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 a single decryption query [5]. 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 attack, i.e. distinguishing the ciphertext from random using only encryption queries, is not possible.

6 Practicality, and Applicability to Others

Almost Known-Plaintext Query. The first encryption query is almost a known-plaintext query in that it works with a known plaintext except the last two blocks being $\texttt{len}(0^n)\|X$ for some known $X \in \{0,1\}^n$, where $\texttt{len}(0^n) = 0^{120}10^7$ in the reference code. This can happen (e.g.) when a certain padding is already applied to the plaintext.

[5] 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 (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 to apply a hybrid argument of (52) using this definition.

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 [Min14] is an inverse-free (for the absense 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 [GJMN16] is a permutation-based AE based on OCB. It always uses XEX, or more precisely, a variant of XPX [Men16], because otherwise an offline permutation inverse query easily breaks the scheme. It is safe because of this consistent use of XPX.

Standards and Libraries. ISO/IEC 19772:2009 specifies OCB2 and thus is affected. SJCL [6] is a well-known Javascript cryptographic library and it implements OCB2, though we have not tested our attacks on it[7].

7 Fixing OCB2

We suggest several ways to fix OCB2.

Use XEX for the last block. The simplest way is to use XEX for the last block. We call it OCB2f. Its pseudocode is obtained by just changing line 5 of OCB2.$\mathcal{E}_E$ and OCB2.$\mathcal{D}_E$ in Fig. 1 to

$$\begin{aligned}&\text{Pad} \leftarrow 2^m L \oplus E(2^m L \oplus \mathtt{len}(M[m])), \text{ and}\\ &\text{Pad} \leftarrow 2^m L \oplus E(2^m L \oplus \mathtt{len}(C[m])).\end{aligned}$$

OCB2f can be regarded as 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 the mode of $\widetilde{E}_K$ obtained by changing line 4 of ΘCB2.$\mathcal{E}_{\widetilde{E}}$ in Fig. 4 to

$$\text{Pad} \leftarrow \widetilde{E}^{*,1,N,m,0}(\mathtt{len}(C[m])).$$

ΘCB2f.$\mathcal{D}_{\widetilde{E}}$ is defined accordingly. Then $\Theta\text{CB2f}_{\widetilde{E}}$ is equivalent to OCB2f_E if $\widetilde{E}_K$ is XEX^*_E.

The security bounds of OCB2f are the same as those claimed for OCB2:

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

[7] We need to natively use SJCL for attacks: the demo page for SJCL (`https://bitwiseshiftleft.github.io/sjcl/demo/`) cannot be used since (to our understanding) it only accepts ASCII strings for plaintext, while our attacks need a plaintext that cannot be encoded to a valid ASCII string.

Theorem 1. *For adversary $\mathcal{A}$ using q encryption queries and $\mathcal{A}_{\pm}$ using q encryption queries and single decryption query, we have*

$$\begin{aligned}\mathbf{Adv}^{\mathsf{priv}}_{\mathrm{OCB2f}_{\mathsf{P}}}(\mathcal{A}) = \mathbf{Adv}^{\mathsf{priv}}_{\mathrm{OCB2f}_{\mathrm{XEX}^*_{\mathsf{P}}}}(\mathcal{A}) &\le \mathbf{Adv}^{\mathsf{tprp}}_{\mathrm{XEX}^*_{\mathsf{P}}}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{priv}}_{\Theta\mathrm{CB2f}_{\widetilde{\mathsf{P}}}}(\mathcal{A}) \\ &\le \frac{4.5q^2}{2^n} \end{aligned} \tag{54}$$

$$\begin{aligned}\mathbf{Adv}^{\mathsf{auth}}_{\mathrm{OCB2f}_{\mathsf{P}}}(\mathcal{A}_{\pm}) = \mathbf{Adv}^{\mathsf{auth}}_{\mathrm{OCB2f}_{\mathrm{XEX}^*_{\mathsf{P}}}}(\mathcal{A}_{\pm}) &\le \mathbf{Adv}^{\mathsf{tsprp}}_{\mathrm{XEX}^*_{\mathsf{P}}}(\mathcal{B}_{\pm}) + \mathbf{Adv}^{\mathsf{auth}}_{\Theta\mathrm{CB2f}_{\widetilde{\mathsf{P}}}}(\mathcal{A}_{\pm}) \\ &\le \frac{9.5q^2}{2^n} + \frac{2^{n-\tau}}{2^n - 1} \end{aligned} \tag{55}$$

Intuitively, the security of OCB2f is proved because OCB2f is ΘCB2f using $\widetilde{E}$ instantiated by XEX*, and ΘCB2f in the privacy and authenticity games do not force the adversary to violate the access rules (Definition 1). Combining (53) and the proofs of $\Theta\mathrm{CB2}_{\widetilde{\mathsf{P}}}$ from Rog04 with minor changes gives the desired results. In slightly more detail, the proof for privacy is trivially obtained in the same manner to that of Rog04. For the authenticity bound, since this is again almost the same as the proof of OCB2, we provide a proof sketch that focuses on the part that makes an important difference from the proof of OCB2. For simplicity, we ignore AD (assuming it is empty for all queries) and $\tau = n$. A full proof will appear in the full version.

Proof. (Sketch) Suppose $\mathcal{A}_{\pm}$ queries (N_1, M_1), ..., (N_q, M_q) to the encryption oracle and (N', C', T') to the decryption oracle. For $1 \le i \le q$, let (C_i, T_i) be the return value for the query (N_i, M_i). Without loss of generality, we assume $\mathcal{A}_{\pm}$ tries to forge after all encryption queries. We need to consider the following cases.

Case 1: $N' \notin \{N_1, \ldots, N_q\}$
Case 2: $N' = N_i$, $|C'|_n = |C_i|_n$

In these two cases, the proof is almost the same as $\Theta\mathrm{CB2}_{\widetilde{\mathsf{P}}}$ and the adversary can guess the valid tag, T^*, with probability at most $2^{n-\tau}/(2^n - 1)$.

Case 3: $N' = N_i$, $|C'|_n < |C_i|_n$

Let $|C'|_n = c'$, $|C_i|_n = c$. In this case, unlike $\Theta\mathrm{CB2}_{\widetilde{\mathsf{P}}}$, $\widetilde{E}^{*,1,N',c',0}$ is invoked to decrypt $C'[c']$, while $\widetilde{E}^{*,1,N',c',0}$ (where the underlying blockcipher is P) has also been used in an encryption query. Nevertheless, the adversary has to guess T^* without access to $\widetilde{E}^{*,0,N',c',1}$ since $c' \neq c$. Since $\widetilde{E}^{*,0,N',c',1}$ has never queried before, the probability of $T^* = T'$ is $1/2^{\tau}$.

Case 4: $N' = N_i$, $|C'|_n > |C_i|_n$

In the same manner to the above case, the forging probability is $1/2^{\tau}$.

Use XEX$^+$. Minematsu and Matsushima [MM09] proposed an extension of XEX* called XEX$^+$. This allows to use plain blockcipher calls in combination with XEX and XE. MM09 suggested 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 Conclusions

We have presented practical forgery 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 about the interpretation of OCB2 as a mode of TBC which combines XEX and XE. While the latest OCB3 has a superior software performance from the previous ones, and is clearly recommended by the designer, we think OCB2 is still quite influential for its simple description and the introduction of the modular design based on 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 the errors in the proofs the provably-secure schemes can be broken, or the schemes still 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 [Rog04d], [DGW01], EAX-prime [MLMI13], GCM [IOM12], and some of the CAESAR submissions [Nan14], [BS16], [SMAP15] and more. We believe our work to emphasize the quality of security proofs and their active verifications.

Acknowledgements

The authors would like to thank Tetsu Iwata for useful comments.

References

[ABL+14] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to Securely Release Unverified Plaintext in Authenticated Encryption. In *ASIACRYPT (1)*, volume 8873 of *Lecture Notes in Computer Science*, pages 105–125. Springer, 2014.

[ADL17] Tomer Ashur, Orr Dunkelman, and Atul Luykx. Boosting Authenticated Encryption Robustness with Minimal Modifications. In *CRYPTO (3)*, volume 10403 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2017.

[AY13] Kazumaro Aoki and Kan Yasuda. The Security of the OCB Mode of Operation without the SPRP Assumption. In *ProvSec*, volume 8209 of *Lecture Notes in Computer Science*, pages 202–220. Springer, 2013.

[BC09] John Black and Martin Cochran. MAC Reforgeability. In *FSE*, volume 5665 of *Lecture Notes in Computer Science*, pages 345–362. Springer, 2009.

[BDJR97] Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. A Concrete Security Treatment of Symmetric Encryption. In *FOCS '97*, pages 394–403. IEEE Computer Society, 1997.

[BN17] Ritam Bhaumik and Mridul Nandi. Improved Security for OCB3. In *ASIACRYPT (2)*, volume 10625 of *Lecture Notes in Computer Science*, pages 638–666. Springer, 2017.

[BR02] John Black and Phillip Rogaway. A Block-Cipher Mode of Operation for Parallelizable Message Authentication. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 384–397. Springer, 2002.

[BRW04] Mihir Bellare, Phillip Rogaway, and David A. Wagner. The EAX Mode of Operation. In *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 389–407. Springer, 2004.

[BS16] Raphael Bost and Olivier Sanders. Trick or Tweak: On the (In)security of OTR's Tweaks. In *ASIACRYPT (1)*, volume 10031 of *Lecture Notes in Computer Science*, pages 333–353, 2016.

[DGW01] Pompiliu Donescu, Virgil D. Gligor, and David Wagner. A Note on NSA's Dual Counter Mode of Encryption, 2001. `http://www.cs.berkeley.edu/~daw/papers/dcm-prelim.ps/`.

[Fer02] Niels Ferguson. 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/`.

[FLLW17] Christian Forler, Eik List, Stefan Lucks, and Jakob Wenzel. Reforgeability of Authenticated Encryption Schemes. In *ACISP (2)*, volume 10343 of *Lecture Notes in Computer Science*, pages 19–37. Springer, 2017.

[GJMN16] Robert Granger, Philipp Jovanovic, Bart Mennink, and Samuel Neves. Improved Masking for Tweakable Blockciphers with Applications to Authenticated Encryption. In *EUROCRYPT (1)*, volume 9665 of *Lecture Notes in Computer Science*, pages 263–293. Springer, 2016.

[IK03] Tetsu Iwata and Kaoru Kurosawa. OMAC: One-Key CBC MAC. In Thomas Johansson, editor, *FSE*, volume 2887 of *Lecture Notes in Computer Science*, pages 129–153. Springer, 2003.

[IOM12] Tetsu Iwata, Keisuke Ohashi, and Kazuhiko Minematsu. Breaking and Repairing GCM Security Proofs. In *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 31–49. Springer, 2012.

[ISO09] Information Technology - Security techniques - Authenticated encryption, ISO/IEC 19772:2009. International Standard ISO/IEC 19772, 2009.

[KR11] Ted Krovetz and Phillip Rogaway. The Software Performance of Authenticated-Encryption Modes. In *FSE*, volume 6733 of *Lecture Notes in Computer Science*, pages 306–327. Springer, 2011.

[LRW02] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2002.

[Men16] Bart Mennink. XPX: Generalized Tweakable Even-Mansour with Improved Security Guarantees. In *CRYPTO (1)*, volume 9814 of *Lecture Notes in Computer Science*, pages 64–94. Springer, 2016.

[Min14] Kazuhiko Minematsu. Parallelizable Rate-1 Authenticated Encryption from Pseudorandom Functions. In *Eurocrypt*, 2014. to appear.

[MLMI13] Kazuhiko Minematsu, Stefan Lucks, Hiraku Morita, and Tetsu Iwata. Attacks and Security Proofs of EAX-Prime. In *FSE*, volume 8424 of *Lecture Notes in Computer Science*, pages 327–347. Springer, 2013.

[MM09] Kazuhiko Minematsu and Toshiyasu Matsushima. Generalization and Extension of XEX* Mode. *IEICE Transactions*, 92-A(2):517–524, 2009.

[Nan14] Mridul Nandi. Forging Attacks on Two Authenticated Encryption Schemes COBRA and POET. In *ASIACRYPT (1)*, volume 8873 of *Lecture Notes in Computer Science*, pages 126–140. Springer, 2014.

[RBBK01] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: a block-cipher mode of operation for efficient authenticated encryption. In *ACM Conference on Computer and Communications Security*, pages 196–205. ACM, 2001.

[RFC14] The OCB Authenticated-Encryption Algorithm. IRTF RFC 7253, 2014.

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *ACM Conference on Computer and Communications Security*, pages 98–107. ACM, 2002.

[Rog04a] Phillip Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In *ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2004.

[Rog04b] Phillip Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. Full version of [Rog04a], 2004. available from `http://www.cs.ucdavis.edu/~rogaway/papers/`.

[Rog04c] Phillip Rogaway. Nonce-Based Symmetric Encryption. In *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 2004.

[Rog04d] Phillip Rogaway. On the Role Definitions in and Beyond Cryptography. In *ASIAN*, volume 3321 of *Lecture Notes in Computer Science*, pages 13–32. Springer, 2004.

[SMAP15] Willem Schroé, Bart Mennink, Elena Andreeva, and Bart Preneel. Forgery and Subkey Recovery on CAESAR Candidate iFeed. In *SAC*, volume 9566 of *Lecture Notes in Computer Science*, pages 197–204. Springer, 2015.

[SWZ12] Zhelei Sun, Peng Wang, and Liting Zhang. Collision Attacks on Variant of OCB Mode and Its Series. In *Inscrypt*, volume 7763 of *Lecture Notes in Computer Science*, pages 216–224. Springer, 2012.

[VV18] Serge Vaudenay and Damian Vizár. Can Caesar Beat Galois? - Robustness of CAESAR Candidates Against Nonce Reusing and High Data Complexity Attacks. In *ACNS*, volume 10892 of *Lecture Notes in Computer Science*, pages 476–494. Springer, 2018.

A Code Example for Minimal 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;
}
```