A preliminary version of this papers appears in *Fast Software Encryption '04*, Lecture Notes in Computer Science, vol. ?? , R. Bimal and W. Meier ed., Springer-Verlag, 2004. This is the full version.

The EAX Mode of Operation

(A Two-Pass Authenticated-Encryption Scheme Optimized for Simplicity and Efficiency)

M. BELLARE[*] P. ROGAWAY[†] D. WAGNER[‡]

February 2, 2004

Abstract

We propose a block-cipher mode of operation, EAX, for solving the problem of authenticated-encryption with associated-data (AEAD). Given a nonce N, a message M, and a header H, our mode protects the privacy of M and the authenticity of both M and H. Strings N, M, and H are *arbitrary* bit strings, and the mode uses $2\lceil|M|/n\rceil + \lceil|H|/n\rceil + \lceil|N|/n\rceil$ block-cipher calls when these strings are nonempty and n is the block length of the underlying block cipher. Among EAX's characteristics are that it is on-line (the length of a message isn't needed to begin processing it) and a fixed header can be pre-processed, effectively removing the per-message cost of binding it to the ciphertext.

EAX is obtained by first creating a generic-composition method, EAX2, and then collapsing its two keys into one. EAX is provably secure under a standard complexity-theoretic assumption. The proof of this fact is novel and involved.

EAX is an alternative to CCM [26], which was created to answer the wish within standards bodies for a fully-specified and patent-free AEAD mode. As such, CCM and EAX are two-pass schemes, with one pass for achieving privacy and one for authenticity. EAX is simpler and more efficient than CCM, avoiding, for example, elaborate padding rules or nonstandard parameters. With EAX we aimed to do as well as possible, within the space of two-pass schemes, with regard to issues of efficiency, simplicity, elegance, ease of correct use, and provable-security guarantees.

Keywords: Authenticated encryption, CCM, EAX, message authentication, CBC MAC, modes of operation, OMAC, provable security.

[*] Department of Computer Science & Engineering, University of California at San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. E-mail: `mihir@cs.ucsd.edu` WWW: `www-cse.ucsd.edu/users/mihir/`

[†] Department of Computer Science, University of California at Davis, Davis, California 95616, USA; and Department of Computer Science, Faculty of Science, Chiang Mai University, Chiang Mai 50200, Thailand. E-mail: `rogaway@cs.ucdavis.edu` WWW: `www.cs.ucdavis.edu/~rogaway/`

[‡] Department of Electrical Engineering and Computer Science, University of California at Berkeley, Berkeley, California 94720, USA. E-mail: `daw@cs.berkeley.edu` WWW: `http://www.cs.berkeley.edu/~daw/`

Contents

1 Introduction **3**

2 Preliminaries **5**

3 The EAX Algorithm **6**

4 Intellectual Property Statement **9**

5 EAX2 Algorithm **9**

6 Definitions **11**

7 Security Results **13**

8 Acknowledgments **15**

A Definition of CCM **17**

B Criticism of CCM **17**
B.1 Efficiency issues 17
B.2 Parameterization 19
B.3 Complexity 20
B.4 Subtleties of variable-length authentication tags 21
B.5 Security claims 22

C Proof of Security of EAX2 **22**

D Proof of Security of $\mathbb{OMAC}$ **28**

E Proof of Security of EAX **36**

F Recommended API **38**

G Test Vectors **43**

1 Introduction

An authenticated encryption (AE) scheme is a symmetric-key mechanism by which a message M is a transformed into a ciphertext CT with the goal that CT protect both the privacy *and* the authenticity of M. The last few years has seen the emergence of AE as a recognized cryptographic goal. With this has come the development of new authenticated-encryption schemes and the analysis of old ones. This paper offers up a new authenticated-encryption scheme, EAX, and provides a thorough analysis of it. To understand why we are defining a new AE scheme, we need to give some background.

FLAVORS OF AUTHENTICATED ENCRYPTION. It useful to distinguish two kinds of AE schemes. In a *two-pass* scheme we make two passes through the data, one aimed at providing privacy and the other, authenticity. One way of making a two-pass AE scheme is by *generic composition*, wherein one pass constitutes a (privacy-only) symmetric-encryption scheme, while the other pass is a message authentication code (MAC). The encryption scheme and the MAC each use their own key. Analyses of some generic composition methods can be found in [5, 6, 20].

In a *one-pass* AE scheme we make a single pass through the data, simultaneously doing what is needed to engender both privacy and authenticity. Typically, the computational cost is about half that of a two-pass scheme. Such schemes emerged only recently. They include IAPM, OCB, and XCBC [12, 17, 25].

Soon after the emergence of one-pass AE schemes it was realized that often not all the data should be privacy-protected. Changes were needed to the basic definitions and mechanisms in order to support the possibility that some information, like a packet header, must *not* be encrypted. Thus was born the notion of *authenticated-encryption with associated-data* (AEAD), first formally defined in [24]. The non-secret data is called the *associated data* or the *header*. Like an AE schemes, an AEAD scheme might make one pass or two.

STANDARDIZING A TWO-PASS AEAD SCHEME. Traditionally, it has been the designers of applications and network protocols who were responsible for combining privacy and authenticity mechanisms in order to make a two-pass AEAD scheme. This has not worked well. It turns out that there are numerous ways to go wrong in trying to make a secure AEAD scheme, and many protocols, products, and standards have done just that. (For example, see [11] for a wrong one-pass scheme, see [5] for weaknesses in the AEAD mechanism of SSH, and [6, 20] for attacks on some methods of popular use.)

Nowadays, some standards bodies (including NIST, IETF, and IEEE 802.11) would like to standardize on an AEAD scheme. Indeed IEEE 802.11 has already done so. This is a good direction. Standardized AEAD might help minimize errors in mis-combining cryptographic mechanisms.

So far, standards bodies have been unwilling to standardize on any of the one-pass schemes due to pending patents covering them. There is, accordingly, an established desire for standardizing on a two-pass AEAD scheme. The two-pass scheme should be *as good as possible* subject to the limitation of falling within the two-pass framework.

Generic-composition would seem to be the obvious answer. But defining a generic-composition AEAD scheme is not an approach that has moved forward within any of the standards bodies. There would seem to be a number of reasons. One reason is a relatively minor inefficiency—the fact that generic composition methods must use two keys. Probably a bigger issue is that the architectural advantage of generic composition brings with it an "excessive" degree of choice—after deciding on a generic composition method, one still needs two lower-level specifications, namely a symmetric encryption scheme and a MAC, for each of which numerous block-cipher based choices exist. Standards bodies want something self-contained, as well as being a patent-avoiding, block-cipher based, single-key mechanism.

So far, there has been exactly one proposal for such a method (though see the "contemporaneous work" section below). It is called CCM [26], and is due to Whiting, Housley, and Ferguson [26]. CCM has enjoyed rapid success, and is now the required mechanism for IEEE 802.11 wireless LANs as well as 802.15.4 wireless personal area networks. NIST has indicated that it plans to put out a "Recommendation" based on CCM.

OUR CONTRIBUTIONS. It is our view that CCM has a good deal of pointless complexity and inefficiency. It is the first contribution of this paper to explain these limitations. It is the second and main contribution of this paper to provide a new AEAD scheme, EAX, that avoids these limitations.

CCM LIMITATIONS. A description of CCM, together with a detailed description of its shortcomings, can be found in Appendix A. Some of the points we make and elaborate on there are the following. CCM is not on-line, meaning one needs to know the lengths of both the plaintext and the associated data before one can proceed with encryption. This may be inconvenient or inefficient. CCM does not allow pre-processing of static associated data. (If, for example, we have an unchanging header attached to every packet being authenticated, we would like that the cost of authenticating this header be paid only once, meaning header authentication should have no significant cost after a single pre-computation. CCM fails to have this property.) CCM's parameterization is more complex than necessary, including, in addition to the block cipher and tag length, a message-length parameter. CCM's nonce length is restricted in such a way that it may not provide adequate security when nonces are chosen randomly. Finally, CCM implementations could suffer performance hits because the algorithm can disrupt word alignment in the associated data.

EAX AND ITS ATTRIBUTES. EAX is a nonce-using AEAD scheme employing no tool beyond the block cipher $E\colon \mathsf{Key} \times \{0,1\}^n \to \{0,1\}^n$ on which it is based. We expect that E will often be instantiated by AES, but we make no restrictions in this direction. (In particular we do not require that $n = 128$.) Nothing is assumed about the nonces except that they are non-repeating. EAX provides both privacy, in the sense of indistinguishability from random bits, and authenticity, in the sense of an adversary's inability to produce a new but valid $\langle$nonce, header, ciphertext$\rangle$ triple. EAX is simple, avoiding complicated length-annotation. It is a conventional two-pass AEAD scheme, making a separate privacy pass and authenticity pass, using no known intellectual property.

EAX is flexible in the functionality it provides. It supports arbitrary-length messages: the message space is $\{0,1\}^*$. The key space for EAX is the key space Key of the underlying block cipher. EAX supports arbitrary nonces, meaning the nonce space is $\{0,1\}^*$. Any tag length $\tau \in [0 \,..\, n]$ is possible, to allow each user to select how much security she wants from the authenticity guarantees. The only user-selectable parameters are the block cipher E and that tag length τ.

EAX has desirable performance attributes. Message expansion is minimal: the length of the ciphertext (which, following the conventions of [25], excludes the nonce) is only τ bits more than the length of the plaintext. Implementations can profitably pre-process static associated data. (If an unchanging header is attached to every packet, authenticating this header has no significant cost after a single pre-computation.) Key-setup is efficient: all block-cipher calls use the same underlying key, so that we do not incur the cost of key scheduling more than once. For both encryption and decryption, EAX uses only the forward direction of the block cipher, so that hardware implementations do not need to implement the decryption functionality of the block cipher. The scheme is on-line for both the plaintext M and the associated data H, which means that one can process streaming data on-the-fly, using constant memory, not knowing when the stream will stop.

PROVABLE SECURITY. We prove that EAX is secure assuming that the block cipher that it uses is a secure pseudorandom permutation (PRP). Security for EAX means indistinguishability from random bits *and* authenticity of ciphertexts. The combination implies other desirable goals, like nonmalleability and indistinguishability under a chosen-ciphertext attack.

The proof of security for EAX is surprisingly complex. The key-collapse of EAX2 destroys a fundamental abstraction boundary. Our security proof relies on a result about the security of a tweakable extension of OMAC (Lemma 4) in which an adversary can obtain not only a tag for a message of its choice, but also an associated key-stream.

PRAGMATICS. The main reason there is any interest in two-pass schemes, as we have already discussed, is that one-pass schemes would seem to be subject to patents. Motivated by this, standardization bodies have expressed the intent of standardizing on a conventional, two-pass scheme, even understanding the factor-of-two performance hit. The merit of this judgment is debatable, but the pragmatic reality is that there has emerged a desire for a conventional scheme, like EAX, that is *as good as possible* subject to the two-pass constraint. Lack of a scheme like EAX will simply lead to an inferior scheme being standardized, which is to the disadvantage of the user community. Accordingly, EAX addresses a real and practical design problem. We took up work on this design problem at the suggestion of the co-Chair of the IRTF (Internet Research Task Force), which supports the standardization efforts of the IETF. We believe that EAX has the potential for widespread adoption and use.

AFTERWARDS. One non-goal of EAX was to be parallelizable. Another recent two-pass design, CWC [19], is parallelizable. It pays for this advantage with a somewhat complex algorithm, based on Carter-Wegman hashing using polynomial evaluation over a prime field. More recent still is GCM [22], a parallelizable, two-pass design based on multiplication in the finite field with 2^{128} elements.

Other recent AEAD mechanisms include Helix [10] and SOBER-128 [13]. These are stream ciphers that aim to provide authenticity. The provable-security methodology does not apply to these objects since they are built directly rather than from lower level primitives.

2 Preliminaries

All strings in this paper are over the binary alphabet $\{0,1\}$. For $\mathcal{L}$ a set of strings and $n \geq 0$ a number, we let $\mathcal{L}^n$ and $\mathcal{L}^*$ have their usual meanings. The concatenation of strings X and Y is denoted $X \,\|\, Y$ or simply $X\,Y$. The string of length 0, called the *empty string*, is denoted ε. If $X \in \{0,1\}^*$ we let $|X|$ denote its length, in bits. If $X \in \{0,1\}^*$ and $\ell \leq |X|$ then the first ℓ bits of X are denoted X [first ℓ bits]. The set $\textsc{Byte} = \{0,1\}^8$ contains all the strings of length 8, and a string $X \in \textsc{Byte}^*$ is called a *byte string* or an *octet string*. If $X \in \textsc{Byte}^*$ we let $\|X\|_8 = |X|/8$ denote its length in bytes. For $\ell \geq 1$ a number, we write $\textsc{Byte}^{<\ell}$ for all byte strings having fewer than ℓ bytes. If $X \in \textsc{Byte}^*$ and $\ell \leq \|X\|_n$ then the first ℓ bytes of X are denoted X [first ℓ bytes]. When $X \in \{0,1\}^n$ is a nonempty string and $t \in \mathbb{N}$ is a number we let $X + t$ be the n-bit string that results from regarding X as a nonnegative number x (binary notation, most-significant-bit first), adding x to t, taking the result modulo 2^n, and converting this number back into an n-bit string. If $t \in [0..2^n - 1]$ we let $[t]_n$ denote the encoding of t into an n-bit binary string (msb first, lsb last). If X and P are strings then we let $X \oplus\!\!\rightarrow P$ (the *xor-at-the-end* operator) denote the string of length $\ell = \max\{|X|, |P|\}$ bits that is obtained by prepending $\big||X| - |P|\big|$ zero-bits to the shorter string and then xoring this with the other string. (In other words, xor the shorter string into the *end* of the longer string.) A *block cipher* is a function $E\colon \mathsf{Key} \times \{0,1\}^n \to \{0,1\}^n$ where Key is a finite, nonempty set and $n \geq 1$ is a number and $E_K(\cdot) = E(K, \cdot)$ is a permutation on $\{0,1\}^n$. The number n is called the *block length.* Throughout this note we fix such a block cipher E.

In Figure 1 we define the algorithms CBC, CTR, pad, OMAC (no superscript), and OMAC$^\bullet$ (with superscript). The algorithms CBC (the CBC MAC) and CTR (counter-mode encryption) are standard. Algorithm pad is used only to define OMAC. Algorithm OMAC [14] is a pseudorandom function (PRF) that is a one-key variant of the algorithm XCBC [9]. Algorithm OMAC$^\bullet$ is like OMAC but takes an extra argument, the integer t. This algorithm is a "tweakable" PRF [21], tweaked in the most simple way possible.

We explain the notation used in the definition of OMAC. The value of iL (line 40: i an integer in $\{2,4\}$ and $L \in \{0,1\}^n$) is the n-bit string that is obtained by multiplying L by the n-bit string that represents the number i. The multiplication is done in the finite field $\mathrm{GF}(2^n)$ using a canonical polynomial to represent field points. The canonical polynomial we select is the lexicographically first polynomial among the irreducible polynomials of degree n that have a minimum number of nonzero coefficients. For $n = 128$ the indicated polynomial is $\mathsf{x}^{128} + \mathsf{x}^7 + \mathsf{x}^2 + \mathsf{x} + 1$. In that case, $2L = L{\ll}1$ if the first bit of L is 0 and $2L = (L{\ll}1) \oplus 0^{120}10000111$ otherwise, where $L{\ll}1$ means the left shift of L by one position (the first bit vanishing and a zero entering into the last bit). The value of $4L$ is simply $2(2L)$. We warn that to avoid side-channel attacks one must implement the doubling operation in a constant-time manner.

Algorithm $\mathrm{CBC}_K\ (M)$	**Algorithm** $\mathrm{CTR}_K^{\mathcal{N}}\ (M)$
10 Let $M_1 \cdots M_m \leftarrow M$ where $\lvert M_i\rvert = n$	20 $m \leftarrow \lceil \lvert M\rvert/n \rceil$
11 $C_0 \leftarrow 0^n$	21 $S \leftarrow E_K(\mathcal{N}) \parallel E_K(\mathcal{N}{+}1) \parallel \cdots \parallel E_K(\mathcal{N}{+}m{-}1)$
12 **for** $i \leftarrow 1$ **to** m **do**	22 $C \leftarrow M \oplus S$ [first $\lvert M\rvert$ bits]
13 $\quad C_i \leftarrow E_K(M_i \oplus C_{i-1})$	23 **return** C
14 **return** C_m	

Algorithm $\mathsf{pad}\,(M;\ B, P)$	**Algorithm** $\mathrm{OMAC}_K\ (M)$
30 **if** $\lvert M\rvert \in \{n, 2n, 3n, \ldots\}$	40 $L \leftarrow E_K(0^n);\ \ B \leftarrow 2L;\ \ P \leftarrow 4L$
31 **then return** $M \oplus\!\!\rightarrow B$,	41 **return** $\mathrm{CBC}_K(\mathsf{pad}\,(M;\ B, P))$
32 **else return** $(M \parallel 10^{n-1-(\lvert M\rvert \bmod n)}) \oplus\!\!\rightarrow P$	**Algorithm** $\mathrm{OMAC}_K^t\ (M)$
	50 **return** $\mathrm{OMAC}_K([t]_n \parallel M)$

Figure 1: Basic building blocks. The block cipher $E\colon \mathsf{Key} \times \{0,1\}^n \to \{0,1\}^n$ is fixed and $K \in \mathsf{Key}$. For CBC, $M \in (\{0,1\}^n)^+$. For CTR, $M \in \{0,1\}^*$ and $\mathcal{N} \in \{0,1\}^n$. For pad, $M \in \{0,1\}^*$ and $B, P \in \{0,1\}^n$ and the operation $\oplus\!\!\rightarrow$ xors the shorter string into the end of longer one. For OMAC, $M \in \{0,1\}^*$ and $t \in [0..2^n - 1]$ and the multiplication of a number by a string L is done in $\mathrm{GF}(2^n)$.

We have made a small modification to the OMAC algorithm as it was originally presented, changing one of its two constants. Specifically, the constant 4 at line 40 was the constant $1/2$ (the multiplicative inverse of 2) in the original definition of OMAC [14]. The OMAC authors indicate that they will promulgate this modification [15], which slightly simplifies implementations.

3 The EAX Algorithm

ALGORITHM. Fix a block cipher $E\colon \mathsf{Key} \times \{0,1\}^n \to \{0,1\}^n$ and a tag length $\tau \in [0..n]$. These parameters should be fixed at the beginning of a particular session that will use EAX mode. Typically, the parameters would be agreed to in an authenticated manner between the sender and the receiver, or they would be fixed for all time for some particular application. Given these parameters, EAX provides a nonce-based AEAD scheme $\mathrm{EAX}[E, \tau]$ whose encryption algorithm has signature $\mathsf{Key} \times \mathsf{Nonce} \times \mathsf{Header} \times \mathsf{Plaintext} \to \mathsf{Ciphertext}$ and whose decryption algorithm has signature $\mathsf{Key} \times \mathsf{Nonce} \times \mathsf{Header} \times \mathsf{Ciphertext} \to \mathsf{Plaintext} \cup \{\textsc{Invalid}\}$ where Nonce, Header, Plaintext, and Ciphertext are all $\{0,1\}^*$. The EAX algorithm is specified in Figure 2 and a picture illustrating EAX encryption is given in Figure 3. We now discuss various features of our algorithm and choices underlying the design.

NO ENCODINGS. We have avoided any nontrivial encoding of multiple strings into a single one.[1] Some other approaches that we considered required a PRF to be applied to what was logically a tuple, like (N, H, C). Doing this raises encoding issues we did not want to deal with because, ultimately, there would seem to be no simple, efficient, compelling, on-line way to encode multiple strings into a single one. Alternatively, one could avoid encodings and consider a new kind of primitive, a multi-argument PRF. But this would be a non-standard tool and we didn't want to use any non-standard tools. All in all, it seemed best to find a way to sidestep the need to do encodings.

[1] One could view the prefixing of $[t]_n$ to M in the definition of $\mathrm{OMAC}_K^t(M)$ as an encoding, but $[t]_n$ is a constant, fixed-length string, and the aim here is just to "tweak" the PRF. This is very different from needing to encode arbitrary-length strings into a single string.

Algorithm $\text{EAX.Encrypt}_K^{N\,H}(M)$

$\mathcal{N} \leftarrow \text{OMAC}_K^0(N)$
$\mathcal{H} \leftarrow \text{OMAC}_K^1(H)$
$C \leftarrow \text{CTR}_K^{\mathcal{N}}(M)$
$\mathcal{C} \leftarrow \text{OMAC}_K^2(C)$
$\mathit{Tag} \leftarrow \mathcal{N} \oplus \mathcal{C} \oplus \mathcal{H}$
$T \leftarrow \mathit{Tag}$ [first τ bits]
return $\mathit{CT} \leftarrow C \,\|\, T$

Algorithm $\text{EAX.Decrypt}_K^{N\,H}(\mathit{CT})$

if $|\mathit{CT}| < \tau$ **then return** INVALID
Let $C \,\|\, T \leftarrow \mathit{CT}$ where $|T| = \tau$
$\mathcal{N} \leftarrow \text{OMAC}_K^0(N)$
$\mathcal{H} \leftarrow \text{OMAC}_K^1(H)$
$\mathcal{C} \leftarrow \text{OMAC}_K^2(C)$
$\mathit{Tag}' \leftarrow \mathcal{N} \oplus \mathcal{C} \oplus \mathcal{H}$
$T' \leftarrow \mathit{Tag}'$ [first τ bits]
if $T \neq T'$ **then return** INVALID
$M \leftarrow \text{CTR}_K^{\mathcal{N}}(C)$
return M

Figure 2: Encryption and decryption under EAX mode. The plaintext is M, the ciphertext is CT, the key is K, the nonce is N, and the header is H. The mode depends on a block cipher E (that CTR and OMAC implicitly use) and a tag length τ.

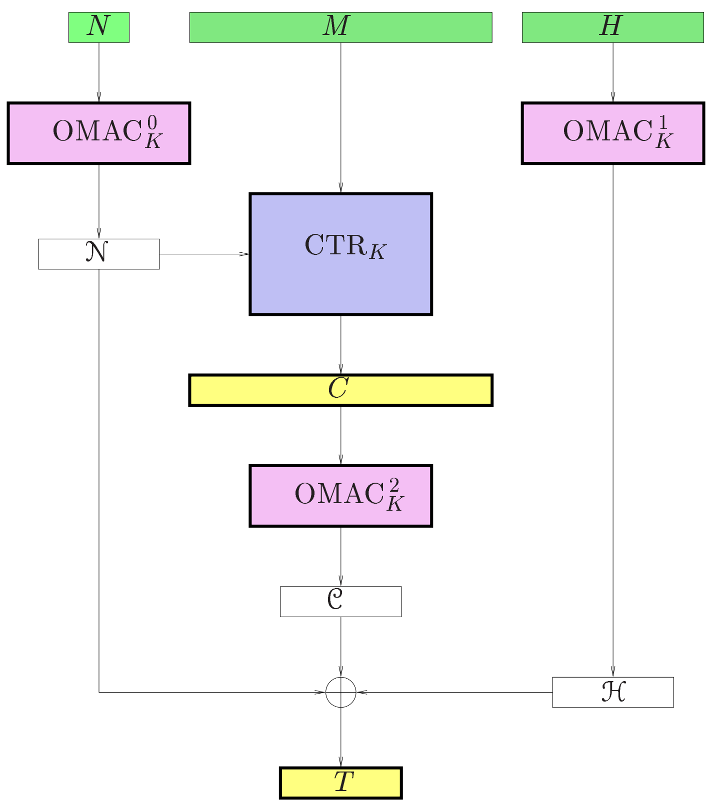

Figure 3: Encryption under EAX. The message is M, the key is K, and the header is H. The ciphertext is $\mathit{CT} = C \,\|\, T$.

WHY NOT GENERIC COMPOSITION? Why have we specified a block-cipher based (BC-based) AEAD scheme instead of following the generic-composition approach of combining a (privacy-only) encryption method and

	CCM	EAX
Functionality	AE with AD	AE with AD
Built from	Block cipher E with 128-bit blocksize	Block cipher E with n-bit blocksize
Parameters	Block cipher E Tag length $\boldsymbol{\tau} \in \{4, 6, 8, 10, 12, 14, 16\}$ Length of message length field $\boldsymbol{\lambda} \in [2..8]$	Block cipher E Tag length $\tau \in [0..n]$
Message space	Parameterized: 7 choices: $\boldsymbol{\lambda} \in [2..8]$. Each possible message space a subset of BYTE^*, from $\text{BYTE}^{2^{16}-1}$ to $\text{BYTE}^{<2^{64}-1}$	$\{0,1\}^*$
Nonce space	Parameterized, with a value of $15 - \boldsymbol{\lambda}$ bytes. From 56 bits to 104 bits	$\{0,1\}^*$
Key space	One block-cipher key	One block-cipher key
Ciphertext expansion	$\boldsymbol{\tau}$ bytes	τ bits
Block-cipher calls	$2\left\lceil\frac{\lvert M\rvert}{128}\right\rceil + \left\lceil\frac{\lvert H\rvert}{128}\right\rceil + 2 + \delta$, for $\delta \in \{0,1\}$	$2\left\lceil\frac{\lvert M\rvert}{n}\right\rceil + \left\lceil\frac{\lvert H\rvert}{n}\right\rceil + \left\lceil\frac{\lvert N\rvert}{n}\right\rceil$
Block-cipher calls with static header	$2\left\lceil\frac{\lvert M\rvert}{128}\right\rceil + \left\lceil\frac{\lvert H\rvert}{128}\right\rceil + 2 + \delta$, for $\delta \in \{0,1\}$	$2\left\lceil\frac{\lvert M\rvert}{n}\right\rceil + \left\lceil\frac{\lvert N\rvert}{n}\right\rceil$
Key setup	Block cipher subkeys	Block cipher subkeys 3 block-cipher calls
IV requirements	Non-repeating nonce	Non-repeating nonce
Parallelizable?	No	No
On-line?	No	Yes
Preprocessing (/msg)	Limited (key stream)	Limited (key stream, header)
Memory rqmts	Small constant	Small constant
Provable security?	Yes (if E is a good PRP) Bound of $\Theta(\sigma^2/2^{128})$	Yes (if E is a good PRP) Bound of $\Theta(\sigma^2/2^n)$
Patent-encumbered?	No	No

Figure 4: A comparison of basic characteristics of CCM and EAX. The count on block-cipher calls for EAX ignores key-setup costs. We denote by τ the length of the EAX tag in bits, and by $\boldsymbol{\tau}$ (boldface) the length of the CCM tag in bytes.

a message authentication code? In fact, there are reasonable arguments in favor of generic composition, based on aesthetic or architectural sensibilities. One can argue that generic composition better separates conceptually independent elements (privacy and authenticity) and, correspondingly, allows greater implementation flexibility [6, 20]. Correctness becomes much simpler and clearer as well. All the same, BC-based AEAD modes have some important advantages of their own. They make it easier for implementors to use a scheme without knowing a lot of cryptography, presenting a simpler abstraction boundary. They make it easier to obtain interoperably. They reduce the risk that implementors will choose insecure parameters. They can save on key bits and key-setup time, as generic-composition methods invariably require a pair of separate keys.

EAX can be viewed as having been derived from a generic-composition scheme we call EAX2, described in Section 5. Specifically, one instantiates EAX2 using CTR mode (counter mode) and OMAC, and then collapses the two keys into one. If one favors generic composition, EAX2 is a nice algorithm for it.

ON-LINE. We say that an algorithm is *on-line* if it is able to process a stream of data as it arrives, with constant memory, not knowing in advance when the stream will end. Observe then that on-line methods should not require knowledge of the length of a message until the message is finished. A failure to be on-line has been regarded as a significant defect for an encryption scheme or a MAC. EAX is on-line.

Now it is true that in many contexts where one would be encrypting a string one *does* know the length of the string in advance. For example, many protocols will already have "packaged up" the string length at a lower level. In effect, such strings have been represented in the computing system as sequence of bytes and a count of those bytes. But there are also contexts where one does *not* know the length of a message in advance of getting an indication that it is over. For examples, a printable string is often represented in computer systems as a sequence of non-zero bytes followed by a terminal zero-byte. Certainly one should be able to efficiently encrypt a string which has been represented in this way.

ABILITY TO PROCESS STATIC AD. In many scenarios the associated data H will be static over the course of a communications session. For example, the associated data may include information such as the IP address of the sender, the receiver, and fixed cryptographic parameters associated to this session. In such a case one would like that the amount of time to compute Encrypt$_K^{N\ H}(M)$ and Decrypt$_K^{N\ H}(C)$ should be independent of $|H|$, disregarding the work done in a preprocessing step. The significance of this goal was already explained in [24]. EAX achieves this goal.

ADDITIONAL FEATURES. Invalid messages can be rejected at half the cost of decryption. This is one of the benefits of following what is basically an encrypt-then-authenticate approach as opposed to an authenticate-then-encrypt approach.

To obtain a MAC as efficient as the PRF underlying EAX define $\mathrm{MAC}_K(H) =$ Encrypt$_K^{0^n\ H}(\varepsilon)$.

COMPARISON WITH CCM. Appendix A provides a description of CCM together with a discussion of its limitations. Figure 4 summarizes some of that discussion by comparing EAX and CCM along a few relevant dimensions.

4 Intellectual Property Statement

The authors neither have, nor are of aware of, any patents or pending patents relevant to EAX. We do not intend to apply for any patents covering this technology. Our work for this note is hereby placed in the public domain. As far as we know, EAX is free and unencumbered for all uses.

5 EAX2 Algorithm

To understand the the proof of security of EAX and the approach taken for its design, we introduce EAX2, a generic composition method. EAX is EAX2 for the particular case of CTR encryption and OMAC authentication, but then collapsed to a single key.

EAX2 COMPOSITION. Let $F\colon \mathsf{Key1} \times \{0,1\}^* \to \{0,1\}^n$ be a PRF, where $n \geq 2$. Let $\Pi = (\mathcal{E}, \mathcal{D})$ be an IV-based encryption scheme having key space $\mathsf{Key2}$ and IV space $\{0,1\}^n$. This means that $\mathcal{E}\colon \mathsf{Key2} \times \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^*$ and $\mathcal{D}\colon \mathsf{Key2} \times \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^*$ and $\mathsf{Key2}$ is a set of keys and for every $K \in \mathsf{Key2}$ and $\mathcal{N} \in \{0,1\}^n$ and $M \in \{0,1\}^*$, if $C = \mathcal{E}_K^{\mathcal{N}}(M)$ then $\mathcal{D}_K^{\mathcal{N}}(C) = M$. Let $\tau \leq n$ be a number. Now given F and Π and τ we define an AEAD scheme $\mathrm{EAX2}[\Pi, F, \tau] = (\mathrm{EAX2.Encrypt}, \mathrm{EAX2.Decrypt})$ as follows. Set $F_K^t(M) = F_K([t]_n \parallel M)$. Set $\mathsf{Key} = \mathsf{Key1} \times \mathsf{Key2}$. Then the encryption algorithm EAX2.Encrypt: $\mathsf{Key} \times \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ and the decryption algorithm EAX2.Decrypt: $\mathsf{Key} \times \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \cup \{\textsc{Invalid}\}$ are defined in Figure 5. Scheme $\mathrm{EAX2}[\Pi, F, \tau]$ is provably secure under natural assumptions about Π and F. See Section 7.

EAX1 COMPOSITION. Let EAX1 be the single-key variant of EAX2 where one insists that $\mathsf{Key} = \mathsf{Key1} = \mathsf{Key2}$ and where one keys F, $\mathcal{E}$, and $\mathcal{D}$ with a single key $K \in \mathsf{Key}$. One associates to F and Π the scheme

Algorithm EAX2.Encrypt$^{N\,H}_{K1,K2}\,(M)$

$\mathcal{N} \leftarrow F^0_{K1}(N)$
$\mathcal{H} \leftarrow F^1_{K1}(H)$
$C \leftarrow \mathcal{E}^{\mathcal{N}}_{K2}(M)$
$\mathcal{C} \leftarrow F^2_{K1}(C)$
$\mathit{Tag} \leftarrow \mathcal{N} \oplus \mathcal{C} \oplus \mathcal{H}$
$T \leftarrow \mathit{Tag}$ [first τ bits]
return $\mathit{CT} \leftarrow C \parallel T$

Algorithm EAX2.Decrypt$^{N\,H}_{K1,K2}\,(\mathit{CT})$

if $|\mathit{CT}| < \tau$ **then return** INVALID
Let $C \parallel T \leftarrow \mathit{CT}$ where $|T| = \tau$
$\mathcal{N} \leftarrow F^0_{K1}(N)$
$\mathcal{H} \leftarrow F^1_{K1}(H)$
$\mathcal{C} \leftarrow F^2_{K1}(C)$
$\mathit{Tag}' \leftarrow \mathcal{N} \oplus \mathcal{C} \oplus \mathcal{H}$
$T' \leftarrow \mathit{Tag}'$ [first τ bits]
if $T \neq T'$ **then return** INVALID
$M \leftarrow \mathcal{D}^{\mathcal{N}}_{K2}(C)$
return M

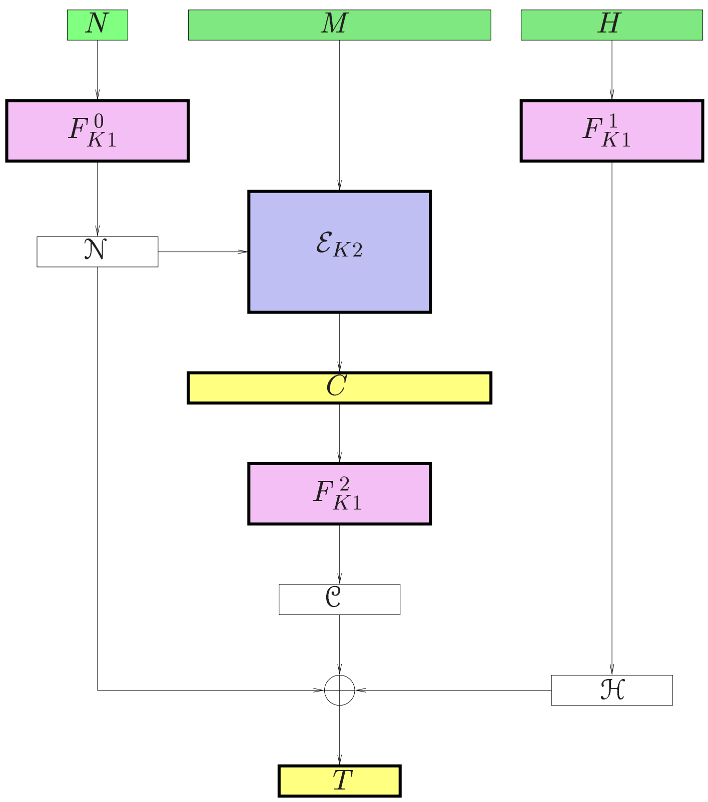

Figure 5: Encryption and decryption under EAX2. The mode is built from a PRF $F\colon \text{Key1} \times \{0,1\}^* \to \{0,1\}^n$ and an IV-based encryption scheme $\Pi = (\mathcal{E}, \mathcal{D})$ having key space Key2 and message space $\{0,1\}^*$. The plaintext is M and the key is $(K1, K2)$ and the header is H. By F^i_K we mean the function where $F^i_K(M) = F_K([i]_n \parallel M)$.

EAX1$[\Pi, F, \tau]$ that is defined as with EAX2 but where the one key K keys everything. Notice that $\text{EAX}[E,\tau] = \text{EAX1}[\text{CTR}[E], \text{OMAC}[E], \tau]$. This is a useful way to look at EAX.

6 Definitions

AEAD SCHEMES. A *set of keys* is a nonempty set having a distribution (the uniform distribution when the set is finite). A (nonce-based) *authenticated-encryption with associated-data* (AEAD) scheme is a pair of algorithms $\mathbf{\Pi} = (\mathbf{E}, \mathbf{D})$ where $\mathbf{E}$ is a deterministic *encryption* algorithm $\mathbf{E}\colon \mathsf{Key} \times \mathsf{Nonce} \times \mathsf{Header} \times \mathsf{Plaintext} \to \mathsf{Ciphertext}$ and a $\mathbf{D}$ is a deterministic *decryption* algorithm $\mathbf{D}\colon \mathsf{Key} \times \mathsf{Nonce} \times \mathsf{Header} \times \mathsf{Ciphertext} \to \mathsf{Plaintext} \cup \{\textsc{Invalid}\}$. The *key space* Key is a set of keys while the *nonce space* Nonce and the *header space* Header (also called the space of *associated data*) are nonempty sets of strings. We write $\mathbf{E}_K^{N\,H}(M)$ for $\mathbf{E}(K, N, H, M)$ and $\mathbf{D}_K^{N\,H}(CT)$ for $\mathbf{D}(K, N, H, CT)$. We require that $\mathbf{D}_K^{N\,H}(\mathbf{E}_K^{N\,H}(M)) = M$ for all $K \in \mathsf{Key}$ and $N \in \mathsf{Nonce}$ and $H \in \mathsf{Header}$ and $M \in \mathsf{Plaintext}$. In this note we assume, for notational simplicity, that Nonce, Header, Plaintext, and Ciphertext are all $\{0,1\}^*$ and that $|\mathbf{E}_K^{N\,H}(M)| = |M|$. An adversary is a program with access to one or more oracles.

NONCE-RESPECTING. Suppose A is an adversary with access to an *encryption oracle* $\mathbf{E}_K^{\cdot\,\cdot}(\cdot)$. This oracle, on input (N, H, M), returns $\mathbf{E}_K^{N\,H}(M)$. Let $(N_1, H_1, M_1), \ldots, (N_q, H_q, M_q)$ denote its oracle queries. The adversary is said to be *nonce-respecting* if $N_1, \ldots, N_q$ are always distinct, regardless of oracle responses and regardless of A's internal coins.

PRIVACY OF AEAD SCHEMES. We consider adversaries with access to an encryption oracle $\mathbf{E}_K^{\cdot\,\cdot}(\cdot)$. We assume that any privacy-attacking adversary is nonce-respecting. The advantage of such an adversary A in violating the privacy of AEAD scheme $\mathbf{\Pi} = (\mathbf{E}, \mathbf{D})$ having key space Key is

$$\mathbf{Adv}_{\mathbf{\Pi}}^{\mathbf{priv}}(A) \;=\; \Pr\left[K \xleftarrow{\$} \mathsf{Key}:\; A^{\mathbf{E}_K^{\cdot\,\cdot}(\cdot)} = 1\right] - \Pr\left[K \xleftarrow{\$} \mathsf{Key}:\; A^{\$^{\cdot\,\cdot}(\cdot)} = 1\right]$$

where $\$^{\cdot\,\cdot}(\cdot)$ denotes the oracle that on input (N, H, M) returns a random string of length $|M|$.

AUTHENTICITY OF AEAD SCHEMES. This time we provide the adversary with two oracles, an encryption oracle $\mathbf{E}_K^{\cdot\,\cdot}(\cdot)$ as above and also a *verification oracle* $\widehat{\mathbf{D}}_K^{\cdot\,\cdot}(\cdot)$. The latter oracle takes input (N, H, CT) and returns 1 if $\mathbf{D}_K^{N\,H}(CT) \in \mathsf{Plaintext}$ and returns 0 if $\mathbf{D}_K^{N\,H}(CT) = \textsc{Invalid}$. The adversary is assumed to satisfy three conditions, and these must hold regardless of the responses to its oracle queries and regardless of A's internal coins:

- Adversary A must be nonce-respecting. (The condition is understood to apply only to the adversary's encryption oracle. Thus a nonce used in an encryption-oracle query may be used in a verification-oracle query.)
- Adversary A may never make a verification-oracle query (N, H, CT) such that the encryption oracle previously returned CT in response to a query (N, H, M).
- Adversary A must call its verification-oracle exactly once, and may not subsequently call its encryption oracle. (That is, it makes a sequence of encryption-oracle queries, then a verification-oracle query, and then halts.)

We say that such an adversary *forges* if its verification oracle returns 1 in response to the single query made to it. The advantage of such an adversary A in violating the authenticity of AEAD scheme $\mathbf{\Pi} = (\mathbf{E}, \mathbf{D})$ having key space Key is

$$\mathbf{Adv}_{\mathbf{\Pi}}^{\mathbf{auth}}(A) \;=\; \Pr\left[K \xleftarrow{\$} \mathsf{Key}:\; A^{\mathbf{E}_K^{\cdot\,\cdot}(\cdot),\, \widehat{\mathbf{D}}_K^{\cdot\,\cdot}(\cdot)} \text{ forges}\right] .$$

IV-BASED ENCRYPTION. An *IV-based encryption scheme* (an IVE scheme) is a pair of algorithms $\Pi = (\mathcal{E}, \mathcal{D})$ where $\mathcal{E}\colon \mathsf{Key} \times \mathsf{IV} \times \mathsf{Plaintext} \to \mathsf{Ciphertext}$ is a deterministic *encryption* algorithm and $\mathcal{D}\colon \mathsf{Key} \times \mathsf{IV} \times \mathsf{Ciphertext} \to \mathsf{Plaintext} \cup \{\textsc{Invalid}\}$ is a deterministic *decryption* algorithm. The *key space* Key is a set of keys and the *plaintext space* Plaintext and *ciphertext space* Ciphertext and *IV space* IV are all nonempty sets of strings. We write $\mathcal{E}_K^R(M)$ for $\mathcal{E}(K, R, M)$ and $\mathcal{D}_K^R(C)$ for $\mathcal{D}(K, R, C)$. We require that $\mathcal{D}_K^R(\mathcal{E}_K^R(M)) = M$ for all $K \in \mathsf{Key}$ and $R \in \mathsf{IV}$ and $M \in \mathsf{Plaintext}$. We assume, as before, that $\mathsf{Plaintext} = \mathsf{Ciphertext} = \{0,1\}^*$

and that $|\mathcal{E}_K^R(M)| = |M|$. We also assume that $\mathsf{IV} = \{0,1\}^n$ for some $n \ge 1$ called the *IV length*.

PRIVACY OF IVE SCHEMES WITH RANDOM IVS. Let $\Pi = (\mathcal{E}, \mathcal{D})$ be an IVE scheme with key space Key and IV space $\mathsf{IV} = \{0,1\}^n$. Let $\mathcal{E}^\$$ be the probabilistic algorithm defined from $\mathcal{E}$ that, on input K and M, chooses an IV R at random from $\{0,1\}^n$, computes $C \leftarrow \mathcal{E}_K^R(M)$, and then returns C along with the chosen IV:

Algorithm $\mathcal{E}_K^\$(M)$ // The probabilistic encryption scheme built from IVE scheme $\mathcal{E}$
$R \xleftarrow{\$} \{0,1\}^n$; $C \leftarrow \mathcal{E}_K^R(M)$; **return** $R \,\|\, C$

Then we define the advantage of an adversary A in violating the privacy of Π (as an encryption scheme using random IV) by

$$\mathbf{Adv}_\Pi^{\mathrm{priv}}(A) \;=\; \Pr\left[K \xleftarrow{\$} \mathsf{Key} :\; A^{\mathcal{E}_K^\$(\cdot)} = 1\right] - \Pr\left[K \xleftarrow{\$} \mathsf{Key} :\; A^{\$(\cdot)} = 1\right]$$

where $\$(\cdot)$ denotes the oracle that on input M returns a random string of length $n + |M|$. This is just the ind\$-privacy of the randomized symmetric encryption scheme associated to Π. We comment that we have used a superscript of "priv" for an IVE scheme and "$\mathbf{priv}$" (bold font) for an AEAD scheme.

PSEUDORANDOM FUNCTIONS. A *family of functions*, or a *pseudorandom function* (PRF), is a map $F\colon \mathsf{Key} \times D \to \{0,1\}^n$ where Key is a set of keys and D is a nonempty set of strings. We call n the *output length* of F. We write F_K for the function $F(K, \cdot)$ and we write $f \xleftarrow{\$} F$ to mean $K \xleftarrow{\$} \mathsf{Key}$; $f \leftarrow F_K$. We denote by $\mathcal{R}_n^*$ the set of all functions with domain $\{0,1\}^*$ and range $\{0,1\}^n$; by $\mathcal{R}_n^n$ the set of all functions with domain $\{0,1\}^n$ and range $\{0,1\}^n$; and by $\mathcal{R}_n^I$ the set of all functions with domain I and range $\{0,1\}^n$. We identify a function with its key, making $\mathcal{R}_n^n$, $\mathcal{R}_n^*$ and $\mathcal{R}_n^I$ pseudorandom functions. The advantage of adversary A in violating the pseudorandomness of the family of functions $F\colon \mathsf{Key} \times \{0,1\}^* \to \{0,1\}^n$ is

$$\mathbf{Adv}_F^{\mathrm{prf}}(A) \;=\; \Pr\left[K \xleftarrow{\$} \mathsf{Key} :\; A^{F_K(\cdot)} = 1\right] - \Pr\left[\rho \xleftarrow{\$} \mathcal{R}_n^* :\; A^{\rho(\cdot)} = 1\right]$$

A family of functions $E\colon \mathsf{Key} \times D \to \{0,1\}^n$ is a *block cipher* if $D = \{0,1\}^n$ and each E_K is a permutation. We let $\mathcal{P}_n$ denote all the permutations on $\{0,1\}^n$ and define

$$\mathbf{Adv}_E^{\mathrm{prp}}(A) \;=\; \Pr\left[K \xleftarrow{\$} \mathsf{Key} :\; A^{E_K(\cdot)} = 1\right] - \Pr\left[\pi \xleftarrow{\$} \mathcal{P}_n :\; A^{\pi(\cdot)} = 1\right]$$

RESOURCES. If xxx is an advantage notion for which $\mathbf{Adv}_\Pi^{\mathrm{xxx}}(A)$ has been defined we write $\mathbf{Adv}_\Pi^{\mathrm{xxx}}(R)$ for the maximal value of $\mathbf{Adv}_\Pi^{\mathrm{xxx}}(A)$ over all adversaries A that use resources at most R. When counting the resource usage of an adversary, one maximizes over all possible oracle responses, including those that could not be returned by any experiment we have specified for adversarial advantage. Resources of interest are: t—the running time; q—the total number of oracle queries; q_{e}—the number of oracle queries to the adversary's first oracle; q_{v}—the number of oracle queries to the adversary's second oracle; and σ—the data complexity. The running time t of an algorithm is its actual running time (relative to some fixed RAM model of computation) plus its description size (relative to some standard encoding of algorithms). The data complexity σ is defined as the sum of the lengths of all strings encoded in the adversary's oracle queries, plus the total number of all of these strings.[2] In this paper the length of strings is measured in n-bit blocks, for some understood value n. The number of blocks in a string M is defined as $\|M\|_n = \max\{1, \lceil |M|/n \rceil\}$, so that the empty string counts as one block. As an example, an adversary that asks queries $(N_1, H_1, M_1), (N_2, H_2, M_2)$ to its first oracle and query (N, H, M) to its second oracle has data complexity $\|N_1\|_n + \|H_1\|_n + \|M_1\|_n + \|N_2\|_n + \|H_2\|_n + \|M_2\|_n + \|N\|_n + \|H\|_n + \|M\|_n + 9$. The name of a resource measure (t, t', q, etc.) will be enough to make clear what resource it refers to.

When we use big-O notation it is understood that the constant hidden inside the notation may depend on n. We write $\widetilde{O}(f(x))$ for $O(f(x)\lg(f(x)))$. When F is a function we write $\mathrm{Time}_F(\sigma))$ for the maximal amount of time to compute the function F over inputs of total length σ. When $\Pi = (\mathcal{E}, \mathcal{D})$ is an AEAD scheme or an

[2] There is a certain amount of arbitrariness in this convention, but it is reasonable and simplifies subsequent accounting.

IVE scheme with key space Key we write $\mathrm{Time}_{\mathcal{E}}(\sigma)$ for the time to compute a random element $K \stackrel{\$}{\leftarrow}$ Key plus the maximal amount of time to compute the function $\mathcal{E}_K$ on arguments of total length σ.

7 Security Results

We first obtain results about the security of EAX2 and then prove a result about the security of a tweakable-OMAC extension. These results are applied to derive results about the security of EAX. The notation and security measures referred to below are defined in Section 6.

SECURITY OF EAX2. We begin by considering the EAX2$[\Pi, F, \tau]$ scheme with F being equal to $\mathcal{R}_n^n$, the set of all functions with domain $\{0,1\}^n$ and range $\{0,1\}^n$. In other words, we are considering the case where F_{K1} is a random function with domain $\{0,1\}^n$ and range $\{0,1\}^n$. First we show that EAX2$[\Pi, \mathcal{R}_n^n, \tau]$ inherits the privacy of the underlying IVE scheme Π. The proof of the following is in Appendix C.

Lemma 1 [Privacy of EAX2 with a random PRF] Let Π be an IVE scheme with IV space $\{0,1\}^n$ and let $\tau \in [0..n]$. Then

$$\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{EAX2}[\Pi,\mathcal{R}_n^n,\tau]}(t,q,\sigma) \quad \le \quad \mathbf{Adv}^{\mathrm{priv}}_{\Pi}(t',q,\sigma)$$

where $t' = t + \widetilde{O}(\sigma)$. □

We now turn to authenticity. The following shows that EAX2$[\Pi, \mathcal{R}_n^n, \tau]$ provides authenticity under the assumption that the underlying IVE scheme Π provides privacy. The proof is in Appendix C.

Lemma 2 [Authenticity of EAX2 with a random PRF] Let Π be an IVE scheme with IV space $\{0,1\}^n$ and let $\tau \in [0..n]$. Then

$$\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{EAX2}[\Pi,\mathcal{R}_n^n,\tau]}(t,q,\sigma) \quad \le \quad \mathbf{Adv}^{\mathrm{priv}}_{\Pi}(t',q,\sigma) + 2^{-\tau}$$

where $t' = t + \widetilde{O}(\sigma)$. □

Our definition of authenticity allows the adversary only one query to its verification oracle, meaning only one forgery attempt. A standard argument says that the advantage of an adversary making q_{v} verification queries can grow by a factor of at most q_{v}. As per the above this means it is at most $q_{\mathrm{v}} \cdot [2^{-\tau} + \mathbf{Adv}^{\mathrm{priv}}_{\Pi}(t',q,\sigma)]$. We believe that in fact the bound is better than this, namely that it is $q_{\mathrm{v}} 2^{-\tau} + \mathbf{Adv}^{\mathrm{priv}}_{\Pi}(t',q,\sigma)$. However, we do not have a proof of this stronger bound. Such a proof would require an extension of Lemma 7 that we have not been able to prove to date.

The above allows us to obtain results about the security of the general EAX2$[\Pi, F, \tau]$ scheme based on assumptions about the security of the component schemes. The proof of the following is in Appendix C.

Theorem 3 [Security of EAX2] Let $F\colon$ Key1 $\times \{0,1\}^* \to \{0,1\}^n$ be a family of functions, let $\Pi = (\mathcal{E}, \mathcal{D})$ be an IVE scheme with IV space $\{0,1\}^n$ and let $\tau \in [0..n]$. Then

$$\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{EAX2}[\Pi,F,\tau]}(t,q,\sigma) \quad \le \quad \mathbf{Adv}^{\mathrm{priv}}_{\Pi}(t_2,q,\sigma) + \mathbf{Adv}^{\mathrm{prf}}_{F}(t_1, 3q+3, \sigma) + 2^{-\tau} \tag{1}$$

$$\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{EAX2}[\Pi,F,\tau]}(t,q,\sigma) \quad \le \quad \mathbf{Adv}^{\mathrm{priv}}_{\Pi}(t_2,q,\sigma) + \mathbf{Adv}^{\mathrm{prf}}_{F}(t_3, 3q, \sigma) \tag{2}$$

where $t_1 = t + \mathrm{Time}_{\mathcal{E}}(\sigma) + \widetilde{O}(\sigma)$ and $t_2 = t + \widetilde{O}(\sigma + nq)$ and $t_3 = t + \mathrm{Time}_{\mathcal{E}}(\sigma) + \widetilde{O}(\sigma)$. □

We remark that although "birthday" terms of the form $\sigma^2/2^n$ or $q^2/2^n$ do not appear explicitly in the bounds above, they may appear when we bound the $\mathbf{Adv}_{\Pi}^{\text{priv}}(\cdot,\cdot,\cdot)$ and $\mathbf{Adv}_{F}^{\text{prf}}(\cdot,\cdot,\cdot)$ in terms of their arguments.

Security of a Tweakable-OMAC Extension. This section develops the core result underlying why key-reuse "works" across OMAC and CTR modes. To do this, we consider the following extension of the tweakable-OMAC construction. Fix $n \geq 1$ and let $t \in \{0,1,2\}$ and $\rho \in \mathcal{R}_n^n$ and $M \in \{0,1\}^*$ and $s \in \mathbb{N}$. Then define

Algorithm $\mathbb{OMAC}_\rho(t, M, s)$

$R \leftarrow \mathrm{OMAC}_\rho^t(M)$

for $j \leftarrow 0$ **to** $s-1$ **do** $S_j \leftarrow \rho(R+j)$

return $R\, S_0 S_1 \cdots S_{s-1}$

Thus an $\mathbb{OMAC}_\rho$ oracle, when asked (t, M, s), returns not only $R = \mathrm{OMAC}_\rho^t(M)$ but also a key stream $S_0 S_1 \ldots S_s$ formed using CTR-mode and start-index R. We emphasize that the key stream is formed using the *same* function ρ (that is, the same key) that underlies the OMAC computation. Note too that we have limited the tweak t to a small set, $\{0,1,2\}$.

We imagine providing an adversary A with one of two kinds of oracles. The first is an oracle $\mathbb{OMAC}_\rho(\cdot,\cdot,\cdot)$ for a randomly chosen $\rho \in \mathcal{R}_n^n$. The second is an oracle $\$_n(\cdot,\cdot,\cdot)$ that, on input (t, M, s), returns $n(s+1)$ random bits. Either way, we assume that the adversary is *length-committing*: if the adversary asks a query (t, M, s) it does not ask any subsequent query (t, M, s'). As the adversary runs, it asks some sequence of queries $(t_1, M_1, s_1), \ldots, (t_q, M_q, s_q)$. The resources of interest to us are the sum of the block lengths of the messages being MACed, $\sigma_1 = \sum \|M_i\|_n$, and the total number $\sigma_2 = \sum s_i$ of key-stream blocks that the adversary requests. We claim that a reasonable adversary will have little advantage in telling apart the two oracles, and we bound its distinguishing probability in terms of the resources σ_1 and σ_2 that it expends. Recall that for oracles X and Y and an adversary A we measure A's ability to distinguish between oracles X and Y by the number $\mathbf{Adv}_{X,Y}^{\text{dist}}(A) = \Pr[A^X = 1] - \Pr[A^Y = 1]$. The proof of the following is in Appendix D.

Lemma 4 [Pseudorandomness of $\mathbb{OMAC}$] Fix $n \geq 2$. Then, for length-committing adversaries,

$$\mathbf{Adv}_{\mathbb{OMAC}[\mathcal{R}_n^n],\$_n}^{\text{dist}}(\sigma_1, \sigma_2) \quad \leq \quad \frac{(\sigma_1 + \sigma_2 + 3)^2}{2^n} \qquad \square$$

Security of EAX. We are now ready to consider the security of EAX. The proof of the following is in Appendix E.

Theorem 5 [Security of EAX] Let $n \geq 2$ and $\tau \in [0..n]$. Then

$$\mathbf{Adv}_{\mathrm{EAX}[\mathcal{R}_n^n,\tau]}^{\mathbf{priv}}(\sigma) \quad \leq \quad \frac{9\,\sigma^2}{2^n}$$

$$\mathbf{Adv}_{\mathrm{EAX}[\mathcal{R}_n^n,\tau]}^{\mathbf{auth}}(\sigma) \quad \leq \quad \frac{10.5\,\sigma^2}{2^n} + \frac{1}{2^\tau} \qquad \square$$

Finally, we may, in the customary way, pass to the corresponding complexity-theoretic result where we start with an arbitrary block cipher E.

Corollary 6 [Security of EAX] Let $n \geq 2$ and $E\colon \mathsf{Key} \times \{0,1\}^n \times \{0,1\}^n$ be a block cipher and let $\tau \in [0..n]$. Then

$$\begin{aligned}
\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{EAX}[E,\tau]}(t,\sigma) &\leq \frac{9.5\,\sigma^2}{2^n} + \mathbf{Adv}^{\mathrm{prp}}_{E}(t',\sigma) \\
\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{EAX}[E,\tau]}(t,\sigma) &\leq \frac{11\,\sigma^2}{2^n} + \frac{1}{2^\tau} + \mathbf{Adv}^{\mathrm{prp}}_{E}(t',\sigma)
\end{aligned}$$

where $t' = t + O(\sigma)$. $\square$

We omit the proof, which is completely standard.

8 Acknowledgments

We received useful comments and information from Paulo Barreto, Niels Ferguson, Brian Gladman, Jack Lloyd, David McGrew, Jesse Walker, and Doug Whiting. An anonymous referee provided an unusually thoughtful and thorough reading.

Mihir Bellare's work was funded by NSF grants CCR-0098123, ANR-0129617, CCR-0208842, and by an IBM Faculty Partnership Development Award. Phil Rogaway's work was funded by NSF CCR-0208842 and a gift from CISCO Systems. David Wagner's work was funded by NSF CCR-0113941.

References

[1] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption: Analysis of the DES modes of operation. *Proceedings of the 38th Symposium on Foundations of Computer Science*, IEEE, 1997.

[2] M. Bellare, R. Guérin, and P. Rogaway. XOR MACs: New methods for message authentication using finite pseudorandom functions. *Advances in Cryptology – CRYPTO '95*, Lecture Notes in Computer Science, vol. 963, D. Coppersmith ed., Springer-Verlag, 1995.

[3] M. Bellare, O. Goldreich, and H. Krawczyk. Stateless evaluation of pseudorandom functions: Security beyond the birthday barrier. *Advances in Cryptology – CRYPTO '96*, Lecture Notes in Computer Science, vol. 1109, N. Koblitz ed., Springer-Verlag, 1996.

[4] M. Bellare, J. Kilian, and P. Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences* (JCSS), vol. 61, no. 3, pp. 362–399, Dec 2000.

[5] M. Bellare, T. Kohno, and C. Namprempre. Authenticated encryption in SSH: provably fixing the SSH binary packet protocol. *Proceedings of the 9th Annual Conference on Computer and Communications Security*, ACM, 2002.

[6] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Advances in Cryptology – ASIACRYPT '00*, Lecture Notes in Computer Science, vol. 1976, T. Okamoto ed., Springer-Verlag, 2000.

[7] M. Bellare and P. Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient encryption. *Advances in Cryptology – ASIACRYPT '00*, Lecture Notes in Computer Science, vol. 1976, T. Okamoto ed., Springer-Verlag, 2000.

[8] M. Bellare, P. Rogaway and D. Wagner. The EAX mode of operation (A two-Pass authenticated-encryption scheme optimized for simplicity and efficiency). Extended abstract of this paper, in *Fast Software Encryption '04*, Lecture Notes in Computer Science, vol. ?? , R. Bimal and W. Meier ed., Springer-Verlag, 2004.

[9] J. Black and P. Rogaway. CBC MACs for arbitrary-length messages: The three-key constructions. *Advances in Cryptology – CRYPTO '00*, Lecture Notes in Computer Science, vol. 1880, M. Bellare ed., Springer-Verlag, 2000.

[10] N. Ferguson, D. Whiting, B. Schneier, J. Kelsey, S. Lucks, and T. Kohno. Helix: Fast encryption and authentication in a single cryptographic primitive. *Fast Software Encryption* (FSE 2003), Lecture Notes in Computer Science, vol. 2887, Springer-Verlag, pp. 330–346, 2003.

[11] V. Gligor and P. Donescu. Integrity-aware PCBC encryption. Security Protocols, 7th International Workshop. Lecture Notes in Computer Science, vol. 1796, Springer-Verlag, pp. 153–171, 1999.

[12] V. Gligor and P. Donescu. Fast encryption and authentication: XCBC encryption and XECB authentication modes. Presented at the 2nd NIST Workshop on AES Modes of Operation, Santa Barbara, CA, August 24, 2001.

[13] P. Hawkes and G. Rose. Primitive specification for SOBER-128. Cryptology ePrint Archive Report 2003/48. April 2003.

[14] T. Iwata and K. Kurosawa. OMAC: One-key CBC MAC. *Fast Software Encryption* (FSE 2003), Lecture Notes in Computer Science, vol. 2887, Springer-Verlag, pp. 129–153, 2003.

[15] T. Iwata and K. Kurosawa. Personal communications, January 2002.

[16] J. Jonsson. On the security of CTR + CBC-MAC. *Proceedings of Selected Areas of Cryptography (SAC)*, 2002.

[17] C. Jutla. Encryption modes with almost free message integrity. *Advances in Cryptology – EUROCRYPT '01*, Lecture Notes in Computer Science, vol. 2045 , B. Pfitzmann ed., Springer-Verlag, 2001.

[18] J. Katz and M. Yung. Unforgeable encryption and adaptively secure modes of operation. *Fast Software Encryption '00*, Lecture Notes in Computer Science, vol. 1978, B. Schneier ed., Springer-Verlag, 2000.

[19] T. Kohno, J. Viega, and D. Whiting. A high-performance conventional authenticated encryption mode. *Fast Software Encryption* (FSE 2004), Lecture Notes in Computer Science, Springer-Verlag, 2004.

[20] H. Krawczyk. The order of encryption and authentication for protecting communications (or: how Secure is SSL?). *Advances in Cryptology – CRYPTO '01*, Lecture Notes in Computer Science, vol. 2139, J. Kilian ed., Springer-Verlag, 2001.

[21] M. Liskov, R. Rivest, and D. Wagner. *Advances in Cryptology – CRYPTO '02*, Lecture Notes in Computer Science, vol. 2442, pp. 31–46, Springer-Verlag, 2002.

[22] D. McGrew and J. Viega. Flexible and efficient message authentication in hardware and software. Manuscript, 2003. Available from http://www.zork.org/

[23] E. Petrank and C. Rackoff. CBC MAC for real-time data sources. *Journal of Cryptology*, vol. 13, no. 3 pp. 315–338, 2000.

[24] P. Rogaway. Authenticated-encryption with associated-data. *Proceedings of the 9th Annual Conference on Computer and Communications Security* (CCS-9), pp. 98–107, ACM, 2002.

[25] P. Rogaway, M. Bellare, and J. Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Transactions on Information and System Security* (TISSEC), vol. 6, no. 3, pp. 365–403, Aug. 2003.

[26] D. Whiting, R. Housley, and N. Ferguson. Counter with CBC-MAC (CCM). June 2002. Available at http://csrc.nist.gov/encryption/modes/proposedmodes/

A Definition of CCM

Since CCM [26] was a major motivation for our work, we recall its definition, writing it in a new form. First some notation. Write string constants in hexadecimal, as in 0xFFFE. When $X \in \{0,1\}^\ell$ is a nonempty string and $i \in \mathbb{N}$ is a number we let $X + i$ be the ℓ-bit string that results from regarding X as a nonnegative number x (binary notation, msb first), adding x to i, taking the result modulo 2^n, and converting this number back into an ℓ-bit string. Now CCM depends on three parameters:

- E — the *block cipher* — where $E\colon \mathsf{Key} \times \{0,1\}^{128} \to \{0,1\}^{128}$
- $\boldsymbol{\tau}$ — the *tag length* — where $\boldsymbol{\tau} \in \{4,6,8,10,12,14,16\}$
- λ — the *length-of-the-message-length-field* — where $\lambda \in \{2,3,4,5,6,7,8\}$

Once parameters $(E, \boldsymbol{\tau}, \lambda)$ have been fixed, where $E\colon \mathsf{Key} \times \{0,1\}^{128} \to \{0,1\}^{128}$ is a block cipher, CCM is the AE scheme specified in Figure 6. The nonce space is $\mathsf{Nonce} = \textsc{Byte}^{15-\lambda}$ and the header space is $\mathsf{Header} = \textsc{Byte}^{<2^{64}}$ and the message space is $\mathsf{Plaintext} = \textsc{Byte}^{<2^{8\lambda}}$. There is a tradeoff between the length of nonces, $\eta = |N| = 15 - \lambda$ bytes, and the longest permitted message, $256^\lambda - 1$ bytes.

B Criticism of CCM

We partition our criticism into five categories: efficiency, parameterization, complexity, variable-tag-length subtleties, and some wrong security claims.

B.1 Efficiency issues

We discuss three efficiency problems with CCM: (a) CCM is not on-line, (b) CCM disrupts word-alignment, and (c) CCM can't pre-process static associated data.

NOT ON-LINE. Here, an algorithm being *on-line* refers to its being able to process a stream of data as it arrives, with constant memory, not knowing in advance when the stream will end. Observe then that on-line methods should not require knowledge of the length of a message until the message is finished.

CCM fails to be on-line in both the plaintext and the associated data: one needs to know the length of both of these before one can proceed with encryption.

For message authentication codes, the significance of being on-line was brought out by the work of Petrank and Rackoff [23], whose work was motivated by the observation that the length-prepend CBC MAC (and other suggestions appearing in [4]) were not on-line. Since their paper, a failure to be on-line has been regarded as a significant defect for an encryption scheme or a MAC.

Now it is true that in many contexts where one would be encrypting a string one *does* know the length of the string in advance. For example, many protocols will already have "packaged up" the string length at a lower level. In effect, such strings have been represented in the computing system as sequence of bytes and a count of those bytes. But there are also contexts where one does *not* know the length of a message in advance of getting an indication that it is over. For examples, a printable string is often represented in computer systems as a sequence of non-zero bytes followed by a terminal zero-byte. Certainly one should be able to efficiently encrypt a string which has been represented in this way.

DISRUPTS WORD-ALIGNMENT. Length-prepend annotation causes an additional problem for the associated data (also called the header)—namely, CCM disrupts its word-alignment. This is a problem when the associated data, H, is long. To understand this issue, remember that most modern machines perform operations much more efficiently when pointers into memory fall along word-boundaries (which typically occur every 4 or 8 bytes). A typical software implementation of a CBC MAC, for example, will exhibit much worse performance if it is called on an argument which is not word-aligned. By prepending length-annotation to the associated data H,

Algorithm CCM.Encrypt$_K^{N\,H}(M)$ // $N \in \textsc{Byte}^{15-\lambda}$ and $H \in \textsc{Byte}^{<2^{64}}$ and $M \in \textsc{Byte}^{<2^{\lambda}}$

$B \leftarrow 0 \;\|\;$ **if** $H = \varepsilon$ **then** 0 **else** 1 **endif** $\;\|\; [\boldsymbol{\tau}/2 - 1]_3 \;\|\; [\lambda - 1]_3 \;\|$
  $N \;\|\; [\|M\|_n]_{8\lambda} \;\|$
  if $H = \varepsilon$ **then** ε **elseif** $\|H\|_n < 62580$ **then** $[\|H\|_n]_{16}$ **elseif** $\|H\|_n < 2^{32}$
  then `0xFFFE` $\|\; [\|H\|_n]_{32}$ **else** `0xFFFF` $\|\; [\|H\|_n]_{64}$ **endif** $\;\|$
  $H \;\|$
  if $H = \varepsilon$ **then** ε **elseif** $\|H\|_n < 62580$ **then** $[0]_n{}^{(14-\|H\|_n) \bmod 16}$
  elseif $\|H\|_n < 2^{32}$ **then** $[0]_n{}^{(10-\|H\|_n) \bmod 16}$ **else** $[0]_n{}^{(6-\|H\|_n) \bmod 16}$ **endif**
  $\| \;\; M \;\; \|$
  $[0]_n{}^{(-\|M\|_n) \bmod 16}$
$U \leftarrow \mathrm{CBC}_K(B)$
$A_0 \leftarrow [\lambda - 1]_8 \;\|\; N \;\|\; [0]_n{}^{15-\lambda}$
$V \,\|\, C \leftarrow \mathrm{CTR}_K^{A_0}\,(U \,\|\, M)$ where $|V| = 128$
$T \leftarrow V$ [first $\boldsymbol{\tau}$ bytes]
return $CT \leftarrow C \,\|\, T$

Algorithm CCM.Decrypt$_K^{N\,H}(CT)$ // $N \in \textsc{Byte}^{15-\lambda}$ and $H \in \textsc{Byte}^{<2^{64}}$ and $CT \in \textsc{Byte}^{*}$

if $\|CT\|_n < \boldsymbol{\tau}$ **then return** Invalid
Partition CT into $C \,\|\, T$ where $\|T\|_n = \boldsymbol{\tau}$
if $\|C\|_n > 2^{\lambda} - 1$ **then return** Invalid

$A_0 \leftarrow [\lambda - 1]_8 \;\|\; N \;\|\; [0]_n{}^{15-\lambda}$
$M \leftarrow \mathrm{CTR}_K^{A_0+1}\,(C)$

$B \leftarrow 0 \;\|\;$ **if** $H = \varepsilon$ **then** 0 **else** 1 **endif** $\;\|\; [\boldsymbol{\tau}/2 - 1]_3 \;\|\; [\lambda - 1]_3 \;\|$
  $N \;\|\; [\|M\|_n]_{8\lambda} \;\|$
  if $H = \varepsilon$ **then** ε **elseif** $\|H\|_n < 62580$ **then** $[\|H\|_n]_{16}$ **elseif** $\|H\|_n < 2^{32}$
  then `0xFFFE` $\|\; [\|H\|_n]_{32}$ **else** `0xFFFF` $\|\; [\|H\|_n]_{64}$ **endif**
  $\| \;\; H \;\; \|$
  if $H = \varepsilon$ **then** ε **elseif** $\|H\|_n < 62580$ **then** $[0]_n{}^{(14-\|H\|_n) \bmod 16}$
  elseif $\|H\|_n < 2^{32}$ **then** $[0]_n{}^{(10-\|H\|_n) \bmod 16}$ **else** $[0]_n{}^{(6-\|H\|_n) \bmod 16}$ **endif**
  $\| \;\; M \;\; \|$
  $[0]_n{}^{(-\|M\|_n) \bmod 16}$
$U \leftarrow \mathrm{CBC}_K(B)$
$V \leftarrow E_K(A_0) \oplus U$
$T' \leftarrow V$ [first $\boldsymbol{\tau}$ bytes]
if $T \neq T'$ **then return** Invalid
return M

Figure 6: Encryption and decryption under $\mathrm{CCM}[E, \boldsymbol{\tau}, \lambda]$. The block cipher is $E\colon$ Key $\times \{0,1\}^{128} \to \{0,1\}^{128}$ and the tag length (in bytes) is $\boldsymbol{\tau} \in \{4, 6, 8, 10, 12, 14, 16\}$ and the length-of-the-message-length-field (in bytes) is $\lambda \in \{2, 3, 4, 5, 6, 7, 8\}$ and the nonce length (in bytes) is $\eta = 15 - \lambda$.

λ	2	3	4	5	6	7	8
msgs shorter than	64 KBytes	16 MBytes	8 GBytes	2^{40} bytes	2^{48} bytes	2^{56} bytes	2^{64} bytes
$\eta = 15 - \lambda$	13	12	11	10	9	8	7
8η (nonce length)	104 bits	96bits	88 bits	80 bits	72 bits	64 bits	56 bits

Figure 7: CCM-allowed values of the length-of-the-message-length-field, λ, the corresponding bound on message lengths, and the resulting length of nonces $\eta = \|N\|_n$ that one must use, measured in bytes and then in bits.

this length-annotation *not* being a multiple of 16 or even 4 bytes, one can expect that typical implementations will suffer a big performance hit.

The disruption of word-alignment is not a big concern if the associated data is just a few bytes, as we expect that it often will be. But, again, NIST and others are considering CCM for use as a general-purpose AEAD algorithm. We have no idea how long will be the associated data. For all we know, the user is primarily interested in authenticating traffic, is doing this to a large volume of traffic, and is encrypting nothing or almost nothing. We don't want the authentication side of an AEAD scheme to be significantly more costly than using a dedicated CBC MAC.

CAN'T PRE-PROCESS STATIC AD. In many scenarios the associated data H will be static over the course of a communications session. For example, the associated data may including information such as the IP address of the sender, the receiver, and fixed cryptographic parameters associated to this session. In such a case one would like that the amount of time to compute $\text{Encrypt}_K^{N\,H}(M)$ and $\text{Decrypt}_K^{N\,H}(C)$ should be independent of $|H|$ (disregarding the work done in a preprocessing step). The significance of this goal was already explained in [24], and a simple approach for achieving this goal was given there. Basically, the reason that CCM fails to allow pre-processing of associated data H is that the algorithm encodes the nonce N and the message length $\|M\|_n$ before H rather than after it.

B.2 Parameterization

We criticize a few aspects of CCM related to its parameterization: (a) the requirement for the user to specify a length-of-the-message-length-field parameter; (b) the fact that this choice involves a trade-off against a conceptually unrelated quantity, the nonce length; (c) the marginal utility of the mode for random nonces; (d) the mode's strict byte orientation.

AN INAPPROPRIATE PARAMETER. The user of CCM is presented with a parameter, the-length-of-the-message-length-field, that she really has no business seeing. This parameter can be regarded as a surrogate for the maximum message length. While it is reasonable to fix a suitably large maximum message length, such as $2^{64} - 1$ bytes, it seems undesirable to force the user to think about choosing smaller message spaces. The very name of this parameter makes clear that this was always conceived of as an implementation-oriented parameter and not a fundamental characteristic of an AEAD scheme.

TRADEOFF BETWEEN NONCE LENGTH AND MAXIMUM MESSAGE LENGTH. Worse than the fact that the user must choose a parameter value that she shouldn't have to think about is that the definition of CCM involves a tradeoff between two things that are conceptually unrelated: the maximal message length and the length of the scheme's nonce. The tradeoff is summarized in Figure 7, which shows the seven allowed values of the-length-of-the-message-length-field parameter, the message space one gets as a result (restricted to octet string), and the corresponding value that is mandated for the the nonce length.

There seem to us several reasonable choices for what should be the nonce length of an AEAD scheme that is based on a 128-bit block cipher. (i) One reasonable choice is 64 bits. This value is large enough to handle a counter, as no real application will encrypt as many as 2^{64} messages. (ii) Another reasonable choice is 128 bits. This choice is natural for matching the block length and being the right length for handling a random value for the nonce. (iii) A third natural choice is any string (or any byte string). This is conceptually clean, and it might be convenient, for example, to allow nonces that are initially 1 byte, and then grow to 2 bytes, and so forth.

One thing that does not make sense to us is to say that the nonce length is a number of bytes that is 15 minus the log base two of one more than the size of longest permissible message length. The message space and the nonce space have nothing to do with each other.

MARGINAL UTILITY WITH RANDOM NONCES. The fact that CCM nonce lengths are allowed to exceed 64 bits suggests that its inventors are thinking of random values as possible choices for the nonce. Though the maximal nonce length (104 bits) may be acceptable for a random nonce, the minimal nonce length (56 bits) probably is not. In general, the suitability of CCM for random nonces is linked to the length of the longest message one wishes to be able to handle, an unnecessary and undesirable connection.

BYTE ORIENTATION. Though some would view this criticism as strictly a matter of taste, we ourselves do not like that CCM is only defined on octet strings. It is not that one is all that likely to need to use an AEAD scheme on strings that are not octet strings. It is more that cryptographic algorithms reach beyond technological conventions like the prevalence byte-orientation in computing systems. To put things in perspective: most cryptographers would have viewed it as a poor choice if MD4, MD5, and SHA1 had only been defined on octet strings. It is no less a defect if a general-purpose AEAD scheme is only defined on octet strings.[3]

B.3 Complexity

While human-perceived complexity is inherently subjective, we ourselves find CCM to be complex. The authors of the current document can not even remember the definition of CCM without consulting its defining document. To us, a mode of operation with so many details that one cannot easily remember it (even after working with it for a few days) is off to a bad start.

BIT MANIPULATIONS. We see two underlying causes for this complexity. The first is all the bit twiddling that CCM does. We believe that it is preferable for a mode of operation to avoid bit manipulations beyond standard padding or length annotation. The central concern for a mode of operation is the correct and efficient use of the block cipher. That purpose has never been shown to need tricky ways to package up arguments or encode string lengths.

MISSING ABSTRACTION BOUNDARY. More fundamentally, we see the complexity of CCM as stemming from the fact that it is designed directly on top of a block cipher—in particular, it was not designed to use any particular message authentication code. In our own exposition of CCM we have done our best to "push upwards" the abstraction boundary to which CCM writes, so that we could "call out" to the raw CBC MAC and CTR encryption. Doing this is at odds with the defining document [26], but CCM's authors have expressed the viewpoint that they are combining such primitives, and so one rather expects a description of CCM in those terms. But the raw CBC MAC is not a secure MAC and the authentication tag that CCM computes can only be seen as something computed by a process integral to the entire mode. There's no sense in which one can say "do CCM using this other message authentication code." CCM employs no autonomous MAC.

It is our view that an AEAD scheme should be designed on top of higher-level primitives than a block cipher. Even if the higher-level primitives are to be implemented using a block cipher, the abstraction boundary helps the scheme to be conceptually clean and support a convincing security analysis. The complexity that one is concerned with isn't the number of lines to implement the mode (which is certainly small) or write it down

[3] Nothing in this paragraph should be understood to suggest that an implementation is under any compulsion to implement an AEAD scheme that operates on arbitrary bit strings (no more than people implement SHA1 to operate on arbitrary bit strings). We are simply saying that it should be well-defined.

in pseudocode. It is the conceptual complexity of an algorithm as induced by the distance between it and what it sits on top of.

ILLUSTRATION 1. A glimpse of CCM's complexity can be seen from the fact that correctness crucially depends on the encoding convention: namely, the authors have excluded the possibility of $\boldsymbol{\tau} = 2$ (i.e., two-byte tags), which means that at least one of bits 3, 4, or 5 of the first block of B, which holds $[\boldsymbol{\tau}/2 - 1]_3$ is non-zero, while these bits are always zero in the initial counter value A_0. If one allowed a tag of $\boldsymbol{\tau} = 2$ bytes[4] or if one had encoded $\boldsymbol{\tau}$ as $[\boldsymbol{\tau}/2 - 2]_3$ instead of $[\boldsymbol{\tau}/2 - 1]_3$ the CCM method would be wrong.[5]

ILLUSTRATION 2. A final way to get at the complexity of CCM is to precisely answer the question, *how many block cipher calls does CCM use?* The answer is given by the following expression:

$$\textit{NumCallsCCM}(M, H) \quad = \quad 2\left\lceil \frac{|M|}{128} \right\rceil + \left\lceil \frac{|H|}{128} \right\rceil + 2 + \delta(|H|)$$

where $\delta(i) \in \{0, 1\}$ is defined as follows: letting

$$\lambda(i) \quad = \quad \begin{cases} 0 & \text{if } i = 0 \\ 16 & \text{if } i \in [8, 8 \cdot 62572] \\ 48 & \text{if } i \in [8 \cdot 62580, 2^{35} - 8] \\ 80 & \text{if } i \in [2^{35}, 2^{64} - 8] \end{cases}$$

we set $\delta(i) = 0$ if $(i \bmod 128) + \lambda(i) \leq 128$, and $\delta(i) = 1$ otherwise. It appears that much of this complexity is so that $\delta(|H|)$ will be more often 0 than 1.

B.4 Subtleties of variable-length authentication tags

In CCM, the authentication tag $\boldsymbol{\tau}$ is of variable length: it is permitted to be 4, 6, 8, 10, 12, 14, or 16 bytes long. Variable-length tags come with some security risks, if the schemes are not implemented carefully or what they achieve is not stated clearly.

A SCENARIO AND AN ATTACK. In the design of Internet protocols, a common slogan is "be conservative in what you send, and liberal in what you accept." Imagine a CCM implementation takes this literally: the sender always creates messages with a 16-byte tag, but the receiver accepts messages if they have a valid tag of any permitted length.

How secure would such an implementation be? Of course, since the attacker is free to choose a tag of any length, a smart attacker will choose the tag length that is most convenient for him: presumably, 4-byte tags. Clearly such an attacker can generate a valid ciphertext within 2^{32} tries. This vulnerability is unavoidable in any scheme with authentication tags that are only four bytes long. However, this attack might be of limited value to the attacker, because it is a *blind* forgery: the attacker cannot control what message will be accepted by the recipient.

We point out that a worse attack on CCM is possible in the envisaged scenario: a more clever attacker can fully control *what* message the recipient will be tricked into accepting. The reason for this is that, in CCM, the transmitted ciphertext has the form $C \,\|\, T$ where T is an authentication tag and where the received message M is computed as a function of C but not $\boldsymbol{\tau}$. The *directed* forgery attack on CCM is as follows. Suppose the attacker intercepts a single ciphertext $C \,\|\, T_{16}$ that is the encryption of some message M formed by the legitimate sender.

[4] We comment that allowing tags of one byte, or even one bit, is a reasonable thing to do, as there are contexts, like authenticating a video frame, where one has to forge many messages to have a detrimental effect.

[5] Criticism that an algorithm "would be wrong if the following change was made" is never-ending and inherently unconvincing— "but the algorithm *isn't* that way" is a quite sufficient response. Here the criticism is simply being used to emphasize that CCM's correctness is integrally wrapped up in its encoding-scheme details.

Imagine that the attacker has a difference Δ that it would like to XOR into the message; for instance, the attacker might want to flip certain bit positions in M. Then the attacker can generate the 2^{32} ciphertexts of the form $(C \oplus \Delta) \parallel T_4$ where T_4 varies over all 4-byte values. Most of these will be rejected by the receiver as having invalid authentication tags, but one will be accepted as a valid encryption of the modified message $M \oplus \Delta$. Thus an adversary can forge any message it likes with 2^{32} tries, given a single ciphertext (with known plaintext) that was authenticated with a 128-bit tag. One can reasonably maintain that authenticating a message with a 128-bit tag should not have had this consequence.

INTERPRETATION. Does the attack above mean that the tag length $\boldsymbol{\tau}$ should have been used for computing the ciphertext core C? In our opinion, the answer is: "not necessarily." Rather, the attack highlights that the specification [26] has made it unclear what the goal is with respect to handling a multiplicity of tag lengths.

We suggest that the tag length parameter $\boldsymbol{\tau}$ should be fixed at key-negotiation time, bound securely to the key, and negotiated authentically between both parties. Once a session has begun, there should be only a single value $\boldsymbol{\tau}$ that will be accepted by the receiver, and this should remain unchanged throughout the lifetime of the session. The receiver shouldn't accept a new $\boldsymbol{\tau}$ in the middle of a session any more than it would accept a new block cipher E. All parameters should have the same status. Under this interpretation the inclusion of $\boldsymbol{\tau}$ within the string B in CCM was not necessary to achieve the security goals. This is not a flaw in CCM, but it underscores the need to think carefully about the desired security goals.

B.5 Security claims

Jonsson has done an admirable job of finding an abstraction of CCM that permits a security proof, and going ahead and giving such a proof [16]. Though none of us have studied the proof in detail, it seems credible and well-conceived. This is fortunate, because many of the security comments in the WHF writeup itself [26] can only be described as uninformed. For example, in Section 1.8 the authors claim that CCM "is secure against attackers limited to 2^{128} steps of operation if the key K is 256 bits or longer." Such a claim is unsupported by any known results and would seem to be wrong under any reasonable interpretation, as privacy itself vanishes by the time that 2^{64} blocks have been enciphered. Later, in Section 1.10 [26], we hear that "[by enciphering the CBC MAC] we avoid CBC-MAC collision attacks. If the block cipher behaves as a pseudo-random permutation, then the key stream is indistinguishable from a random string. Thus the attacker gets no information about the CBC-MAC result. The only avenue of attack that is left is differential-style attack, which has no significant chance of success if the block cipher is a pseudo-random permutation." This paragraph is so far from saying something technically accurate that we wouldn't know where to begin. Of course wrong or unscientific security claims are not an indictment of the method they speak about; our only point is that one needs to ignore the security statements of the writeup, regard it only as an algorithm specification, and turn to Jonsson [16] for more scientific assertions.

C Proof of Security of EAX2

Proof of Lemma 1: Let $\mathsf{Key2}$ be the key space of the IVE scheme $\Pi = (\mathcal{E}, \mathcal{D})$. Let A be an adversary attacking the privacy of the AEAD scheme $\mathrm{EAX2}[\Pi, \mathcal{R}_n^n, \tau] = (\mathbf{E}, \mathbf{D})$. Assume that A makes at most q oracle queries, has data complexity at most σ, and running time at most t. Using A, we construct an adversary P, defined in Figure 8, for attacking the privacy of Π. Observe that P makes at most q oracle queries, has data complexity σ, and running time at most t'. Regarding the design of P, our first claim is that its assignment of a value to $f([0]_n \parallel N)$, made in answering an oracle query of A, is legitimate because $f([0]_n \parallel N)$ was not

Adversary $P^{e(\cdot)}$

Initially, f is everywhere undefined

Run A

When A makes oracle query (N, H, M) answer the query as follows:

$\mathcal{N} \,\|\, C \xleftarrow{\$} e(M)$ // where $|\mathcal{N}| = n$

$f([0]_n \,\|\, N) \leftarrow \mathcal{N}$

if $f([1]_n \,\|\, H)$ is undefined **then** $f([1]_n \,\|\, H) \xleftarrow{\$} \{0,1\}^n$

$\mathcal{H} \leftarrow f([1]_n \,\|\, H)$

if $f([2]_n \,\|\, C)$ is undefined **then** $f([2]_n \,\|\, C) \xleftarrow{\$} \{0,1\}^n$

$\mathcal{C} \leftarrow f([2]_n \,\|\, C)$

Let T be the first τ bits of $\mathcal{N} \oplus \mathcal{H} \oplus \mathcal{C}$

Return $CT \leftarrow C \,\|\, T$ as the oracle response

When A outputs a bit, d, **return** d

Figure 8: Adversary P attacking the privacy of IVE scheme Π using as subroutine adversary A attacking the privacy of $\mathbf{\Pi} = \mathrm{EAX2}[\Pi, \mathcal{R}_n^n, \tau]$.

previously defined. This is true because A is nonce-respecting. Now we claim that

$$\Pr\left[K2 \xleftarrow{\$} \mathsf{Key2} :\ P^{\mathcal{E}^{\$}_{K2}(\cdot)} = 1\right] \;=\; \Pr\left[f \xleftarrow{\$} \mathcal{R}_n^n ;\ K2 \xleftarrow{\$} \mathsf{Key2} :\ A^{\mathbf{E}^{\cdot,\cdot}_{f,K2}(\cdot)} = 1\right] \tag{3}$$

$$\Pr\left[K2 \xleftarrow{\$} \mathsf{Key2} :\ P^{\$(\cdot)} = 1\right] \;=\; \Pr\left[f \xleftarrow{\$} \mathcal{R}_n^n ;\ K2 \xleftarrow{\$} \mathsf{Key2} :\ A^{\$^{\cdot,\cdot}(\cdot)} = 1\right] . \tag{4}$$

Subtracting, we get

$$\mathbf{Adv}^{\mathrm{priv}}_{\Pi}(P) \;=\; \mathbf{Adv}^{\mathbf{priv}}_{\mathrm{EAX2}[\Pi, \mathcal{R}_n^n, \tau]}(A) \ ,$$

which concludes the proof. We now justify the two equations above. The first is clear from the definitions. With regarding Equation (4), we need to check that when P's oracle is $\$(\cdot)$, the oracle-responses returned to A are uniformly and independently distributed. Such a response has the form $C \,\|\, T$. We know that C is random because it is chosen by P's oracle. The reason T is also random is that it is the xor of some quantities with $\mathcal{N}$ and the latter, being returned by P's oracle, is random. ▮

Towards the proof of Lemma 2 we consider a new game and a lemma about it. The game is parameterized by integers $m, \tau \geq 1$. Let I denote the set of all strings of length at most m and let $f\colon\ I \to \{0,1\}^{\tau}$. We consider an adversary with access to two oracles, $\mathrm{XTag}^f(\cdot)$ and $\mathrm{XVf}^f(\cdot,\cdot)$. The *xor-tag* oracle $\mathrm{XTag}^f(\cdot)$ takes input a set $S \subseteq I$ and returns $\sum_{x \in S} f(x)$, the sum here being modulo two, ie. XOR. The *xor-verify* oracle $\mathrm{XVf}^f(\cdot,\cdot)$ takes input a set $S \subseteq I$ and a string T. It returns 1 if $T = \sum_{x \in S} f(x)$ and 0 otherwise. We require that A make exactly one query to its xor-verify oracle and that this be its last oracle query. (That is, it makes a sequence of queries to its xor-tag oracle, then a query to its xor-verify oracle, and then halts.) We say that A *forges* if its query to its xor-verify oracle results in the oracle returning 1. We let

$$\mathbf{Adv}^{\mathrm{xtag}}_{m,\tau}(A) \;=\; \Pr\left[f \xleftarrow{\$} \mathcal{R}^I_\tau :\ A^{\mathrm{XTag}^f(\cdot),\, \mathrm{XVf}^f(\cdot,\cdot)} \text{ forges}\right] \ .$$

Towards stating the lemma we need about this advantage, we need some notation. Let $c = |I|$ and let $x_1, \ldots, x_c$ denote a lexicographic ordering of I. If $S \subseteq I$ we let $\mathrm{ChV}(S)$ denote its c-bit characteristic vector, meaning $\mathrm{ChV}(S)[j] = 1$ if $x_j \in S$ and 0 otherwise ($1 \leq j \leq c$). Suppose adversary A makes xor-tag queries $S_1, \ldots, S_q$ and finally a xor-verify query (S, T). We say that A is *rank respecting* if $\mathrm{ChV}(S)$ is not a linear combination

Adversary $B^{\mathrm{XTag}^f(\cdot),\, \mathrm{XVf}^f(\cdot,\cdot)}$

    $i \leftarrow 0$;

    Run A

        When A makes an xor-tag query S

            if $\mathrm{ChV}(S)$ is linearly dependent on $\mathrm{ChV}(S_1), \ldots, \mathrm{ChV}(S_i)$

                then Let $L \subseteq \{1, \ldots, i\}$ be such that $\mathrm{ChV}(S) = \sum_{l \in L} \mathrm{ChV}(S_l)$; $A_i \leftarrow \sum_{l \in L} A_l$

                else $i \leftarrow i + 1$; $S_i \leftarrow S$; $A_i \leftarrow \mathrm{XTag}^f(S_i)$

            Return A_i to A as the oracle response

        When A makes an xor-verify query (S, T)

            for $j = i + 1, \ldots, c - 1$ **do**

                Pick some $S_j \subseteq I$ such that $\mathrm{ChV}(S), \mathrm{ChV}(S_1), \ldots, \mathrm{ChV}(S_j)$ are linearly independent

                $A_j \leftarrow \mathrm{XTag}^f(S_j)$

            Return $\mathrm{XVf}^f(S, T)$ to A as the oracle response

Figure 9: Adversary for the proof of Lemma 7.

of $\mathrm{ChV}(S_1), \ldots, \mathrm{ChV}(S_q)$. (This must be true regardless of oracle responses and regardless of A's internal coins.) In considering linear combinations we are working over the field of two elements.

Lemma 7 Let $m, \tau \geq 1$ be integers and let A be a rank-respecting adversary. Then

$$\mathbf{Adv}^{\mathrm{xtag}}_{m,\tau}(A) \quad \leq \quad 2^{-\tau} \ . \qquad \Box$$

Proof of Lemma 7: This lemma is pretty much implicit in [2, 3], but for completeness we provide a proof here. First, some notation. Let I be the set of all strings of length at most m and let $c = |I|$. When we write a sum of vectors, we mean the vectors are being added componentwise modulo 2. When we write a sum of τ-bit strings, we mean the bitwise XOR.

We begin by considering the adversary B depicted in Figure 9. It has the following features:

- $\mathbf{Adv}^{\mathrm{xtag}}_{m,\tau}(B) = \mathbf{Adv}^{\mathrm{xtag}}_{m,\tau}(A)$.
- B makes exactly $c - 1$ xor-tag oracle queries.
- B makes exactly one xor-verify query and this is the last oracle query it makes.
- Let $S_1, \ldots, S_{c-1}$ be the xor-tag oracle queries made by B, and let S_c be the first component of the pair that constitutes the xor-verify oracle query made by B. Then $\mathrm{ChV}(S_1), \ldots, \mathrm{ChV}(S_c)$ are linearly independent.

To complete the proof we will show that $\mathbf{Adv}^{\mathrm{xtag}}_{m,\tau}(B) \leq 2^{-\tau}$.

Let $f : I \to \{0,1\}^\tau$ denote the function chosen at random in the game. Let $\mathbf{S}_i$ be the random variable taking value the i-th xor-tag oracle query made by B ($1 \leq i \leq c - 1$), and let $\mathbf{S}_c$ denote the random variable taking value the first component of the pair that constitutes the xor-verify oracle query made by B. For $1 \leq i \leq c$ let $\mathbf{A}_i$ be the random variable taking value the response returned by the game to xor-tag oracle query $\mathbf{S}_i$. (Query $\mathbf{S_c}$ is not made to the xor-tag oracle by B, but we define the random variable whose value is its response anyway). That is:

$$\mathbf{A}_i \quad = \quad \sum_{x \in \mathbf{S}_i} f(x) \qquad (1 \leq i \leq c) \ .$$

Let $S_1, \ldots, S_{c-1}$ be any sequence of xor-tag queries made by B, and let $A_1, \ldots, A_{c-1}$ be responses returned to them. Let S_c be the first component of the pair constituting a following xor-verify query made by B. Let A_c be *any* τ-bit strings. We claim that

$$\Pr\left[\, \mathbf{A}_c = A_c \mid (\mathbf{S}_1, \ldots, \mathbf{S}_c, \mathbf{A}_1, \ldots, \mathbf{A}_{c-1}) = (S_1, \ldots, S_c, A_1, \ldots, A_{c-1}) \,\right] \;=\; 2^{-\tau} \,, \tag{5}$$

the probability being over the choice of the function f alone. This implies that $\mathbf{Adv}^{\mathrm{xtag}}_{m,\tau}(B) = 2^{-\tau}$, which completes the proof. It remains to justify Equation (5).

Let M be the $c-1$ by c matrix whose i-th row is $\mathrm{ChV}(S_i)$ $(1 \le i \le c-1)$ and let $\overline{M}$ be the c by c matrix whose i-th row is $\mathrm{ChV}(S_i)$ $(1 \le i \le c)$. Since $\mathrm{ChV}(S_1), \ldots, \mathrm{ChV}(S_c)$ are linearly independent, $\overline{M}$ is non-singular. Let $x_1, \ldots, x_c$ denote a lexicographic ordering of I. We identify f with the (column) vector $f = (f(x_1), \ldots, f(x_c))$. Below we use "$\cdot$" to denote matrix-vector multiplication. Then we have

$$\begin{aligned}
&\Pr\left[\, \mathbf{A}_c = A_c \mid (\mathbf{S}_1, \ldots, \mathbf{S}_c, \mathbf{A}_1, \ldots, \mathbf{A}_{c-1}) = (S_1, \ldots, S_c, A_1, \ldots, A_{c-1}) \,\right] \\
&= \frac{|\{\, f \in \mathcal{R}^I_\tau \;:\; \overline{M} \cdot f = (A_1, \ldots, A_c) \,\}|}{|\{\, f \in \mathcal{R}^I_\tau \;:\; M \cdot f = (A_1, \ldots, A_{c-1}) \,\}|} \\
&= \frac{|\{\, f \in \mathcal{R}^I_\tau \;:\; \overline{M} \cdot f = (A_1, \ldots, A_c) \,\}|}{\sum_{A \in \{0,1\}^\tau} |\{\, f \in \mathcal{R}^I_\tau \;:\; \overline{M} \cdot f = (A_1, \ldots, A_{c-1}, A) \,\}|} \\
&= \frac{1}{\sum_{A \in \{0,1\}^\tau} 1} \qquad\qquad (6) \\
&= \frac{1}{2^\tau} \,.
\end{aligned}$$

Above, Equation (6) is true because $\overline{M}$ is non-singular. ▮

We will now use Lemma 7 to prove Lemma 2.

Proof of Lemma 2: Let B be an adversary attacking the authenticity of $\mathrm{EAX2}[\Pi, \mathcal{R}^n_n, \tau]$. Assume it makes at most q_{e} encryption oracle queries, has data complexity at most σ, and running time at most t. Let m be large enough that no string in an oracle query of B has length exceeding m, regardless of oracle responses and regardless of B's internal coins. Let I be the set of all strings of length at most m. For any $f\colon I \to \{0,1\}^n$ we define:

Algorithm $\mathbf{EE}^{N\,H}_f\,(M)$

$\mathcal{N} \leftarrow f([0]_n \,\|\, N)$
$\mathcal{H} \leftarrow f([1]_n \,\|\, H)$
$C \xleftarrow{\$} \{0,1\}^{|M|}$
$\mathcal{C} \leftarrow f([2]_n \,\|\, C)$
$\mathit{Tag} \leftarrow \mathcal{N} \oplus \mathcal{C} \oplus \mathcal{H}$
$T \leftarrow \mathit{Tag}$ [first τ bits]
return $\mathit{CT} \leftarrow C \,\|\, T$

Algorithm $\widehat{\mathbf{DD}}^{N\,H}_f\,(\mathit{CT})$

if $|\mathit{CT}| < \tau$ **then return** INVALID
Let $C \,\|\, T \leftarrow \mathit{CT}$ where $|T| = \tau$
$\mathcal{N} \leftarrow f([0]_n \,\|\, N)$
$\mathcal{H} \leftarrow f([1]_n \,\|\, H)$
$\mathcal{C} \leftarrow f([2]_n \,\|\, C)$
$\mathit{Tag}' \leftarrow \mathcal{N} \oplus \mathcal{C} \oplus \mathcal{H}$
$T' \leftarrow \mathit{Tag}'$ [first τ bits]
if $T \ne T'$ **then return** INVALID **else return** 1

We let

$$\mathbf{Adv}^{\mathrm{rauth}}(B) \;=\; \Pr\left[f \xleftarrow{\$} \mathcal{R}^I_n \,:\; B^{\mathbf{EE}^{\cdot\,\cdot}_f(\cdot),\, \widehat{\mathbf{DD}}^{\cdot\,\cdot}_f(\cdot)} \text{ forges} \right] \,.$$

We will construct a rank-respecting adversary A such that

$$\mathbf{Adv}^{\mathrm{rauth}}(B) \;\le\; \mathbf{Adv}^{\mathrm{xtag}}_{m,\tau}(A) \,. \tag{7}$$

We will also construct an adversary P, using resources t', q, σ and attacking the privacy of Π, such that

$$\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{EAX2}[\Pi,\mathcal{R}^n_n,\tau]}(B) - \mathbf{Adv}^{\mathrm{rauth}}(B) \quad \leq \quad \mathbf{Adv}^{\mathrm{priv}}_{\Pi}(P)\ . \tag{8}$$

Thus we have

$$\begin{aligned} \mathbf{Adv}^{\mathbf{auth}}_{\mathrm{EAX2}[\Pi,\mathcal{R}^n_n,\tau]}(B) &= \mathbf{Adv}^{\mathrm{rauth}}(B) + \Big(\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{EAX2}[\Pi,\mathcal{R}^n_n,\tau]}(B) - \mathbf{Adv}^{\mathrm{rauth}}(B)\Big) \\ &\leq \mathbf{Adv}^{\mathrm{xtag}}_{m,\tau}(A) + \mathbf{Adv}^{\mathrm{priv}}_{\Pi}(P) \\ &\leq 2^{-\tau} + \mathbf{Adv}^{\mathrm{priv}}_{\Pi}(t', q, \sigma)\ , \end{aligned}$$

where the last inequality uses Lemma 7. This completes the proof of the lemma. It remains to construct the adversaries A and P indicated above.

Adversary $A^{\mathrm{XTag}^f(\cdot),\, \mathrm{XVf}^f(\cdot,\cdot)}$ defines the following subroutines:

Subroutine $\mathrm{SimE}^{N\,H}\ (M)$

> $C \xleftarrow{\$} \{0,1\}^{|M|}$
> $S \leftarrow \{\, [0]_n \parallel N,\ [1]_n \parallel H,\ [2]_n \parallel C \,\}$
> $T \leftarrow \mathrm{XTag}^f(S)$
> **return** $CT \leftarrow C \parallel T$

Subroutine $\mathrm{SimD}^{N\,H}\ (CT)$

> **if** $|CT| < \tau$ **then return** INVALID
> Let $C \parallel T \leftarrow CT$ where $|T| = \tau$
> $S \leftarrow \{\, [0]_n \parallel N,\ [1]_n \parallel H,\ [2]_n \parallel C \,\}$
> **if** $\mathrm{XVf}^f(S, T) = 0$ **then return** INVALID
> **else return** 1

Adversary A then runs $B^{\mathrm{SimE}^{\cdot\cdot}(\cdot),\, \mathrm{SimD}^{\cdot\cdot}(\cdot)}$. Equation (7) is true because for any choice of the underlying function f we have $\mathrm{SimE}^{\cdot\cdot}(\cdot) = \mathbf{EE}^{\cdot\cdot}_f(\cdot)$ and $\mathrm{SimD}^{\cdot\cdot}(\cdot) = \widehat{\mathbf{DD}}^{\cdot\cdot}_f(\cdot)$. It remains to show that A is rank-respecting. Let $c = |I|$ and let $x_1, \ldots, x_c$ denote a lexicographic ordering of I. For $1 \leq i \leq q$ let (N_i, H_i, M_i) be the i-th encryption-oracle query made by B, leading to A making xor-tag query S_i, and let (N, H, CT) denote the verification query made by B, leading to A making xor-verify query (S, T). Let $CT = C \parallel T$ where $|T| = \tau$. Imagine a matrix whose i-th row is $\mathrm{ChV}(S_i)$ ($1 \leq i \leq q$) and whose $(q+1)$-th row is $\mathrm{ChV}(S)$. Column j is called an l-column if x_j is prefixed by $[l]_n$ ($0 \leq l \leq 2$ and $1 \leq j \leq c$). Since A is nonce-respecting there exists a set D of q 0-columns such that the submatrix formed by the first q rows of the matrix and the columns in D is a q by q identity matrix. Since $\mathrm{ChV}(S)$ has exactly one 1 in a 0-column, the only way that $\mathrm{ChV}(S)$ could be a linear combination of $\mathrm{ChV}(S_1), \ldots, \mathrm{ChV}(S_q)$ is that it equals $\mathrm{ChV}(S_i)$ for some i ($1 \leq i \leq q$). This means that $N = N_i$, $H = H_i$ and the response to B's i-th oracle query was CT. But this contradicts the condition we imposed on B that disallowed a verification-oracle query (N, H, CT) such that CT had been obtained in response to an encryption-oracle query (N, H, M). (It is important here that we required the condition to hold regardless of the responses to oracle queries and the coin tosses of B.) So $\mathrm{ChV}(S)$ cannot equal $\mathrm{ChV}(S_i)$. This completes the proof that A is rank-respecting.

We now turn to the design of adversary P. It is depicted in Figure 10. It is an extension of the adversary constructed in the proof of Lemma 1 that also handles verification-oracle queries. A crucial feature of EAX2 we have exploited in order to be able to respond to verification-oracle queries is that the validity of a ciphertext can be verified without decrypting under the IVE scheme. Regarding the design of P, our first claim is that its assignment of a value to $f([0]_n \parallel N)$, made in answering an encryption-oracle query of B, is legitimate because $f([0]_n \parallel N)$ was not previously defined. This is true for two reasons. The first is that B is nonce-respecting. The second is that B does not make any encryption-oracle queries after it has made its verification-oracle query. (The verification-oracle query might define $f([0]_n \parallel N)$, but since no encryption-oracle queries follow we do not have to be concerned about $f([0]_n \parallel N)$ being defined at the time of answering one of them.) Now we turn

Adversary $P^{e(\cdot)}$

Initially, f is everywhere undefined

Run B

When B makes encryption-oracle query (N, H, M):

$\mathcal{N} \parallel C \xleftarrow{\$} e(M)$ // where $|\mathcal{N}| = n$

$f([0]_n \parallel N) \leftarrow \mathcal{N}$

if $f([1]_n \parallel H)$ is undefined **then** $\mathcal{H} \leftarrow f([1]_n \parallel H) \xleftarrow{\$} \{0,1\}^n$

if $f([2]_n \parallel C)$ is undefined **then** $\mathcal{C} \leftarrow f([2]_n \parallel C) \xleftarrow{\$} \{0,1\}^n$

Let T be the first τ bits of $\mathcal{N} \oplus \mathcal{H} \oplus \mathcal{C}$

Return $CT \leftarrow C \parallel T$ to B as the oracle response

When B makes verification-oracle query (N, H, CT):

if $|CT| < \tau$ **then return** INVALID to B as the oracle response

Let $C \parallel T \leftarrow CT$ where $|T| = \tau$

if $f([0]_n \parallel N)$ is undefined **then** $f([0]_n \parallel N) \xleftarrow{\$} \{0,1\}^n$

$\mathcal{N} \leftarrow f([0]_n \parallel N)$

if $f([1]_n \parallel H)$ is undefined **then** $f([1]_n \parallel H) \xleftarrow{\$} \{0,1\}^n$

$\mathcal{H} \leftarrow f([1]_n \parallel H)$

if $f([2]_n \parallel C)$ is undefined **then** $f([2]_n \parallel C) \xleftarrow{\$} \{0,1\}^n$

$\mathcal{C} \leftarrow f([2]_n \parallel C)$

Let T' be the first τ bits of $\mathcal{N} \oplus \mathcal{H} \oplus \mathcal{C}$

if $T = T'$ **then** $d \leftarrow 1$ **else** $d \leftarrow 0$

if $d = 0$

then return INVALID to B as the oracle response

else return 1 to B as the oracle response

return d

Figure 10: Adversary P attacking the privacy of IVE scheme Π in the proof of Lemma 2.

to the analysis. Let Key2 be the key-space of Π. It is easy to see that

$$\Pr\left[K2 \xleftarrow{\$} \mathsf{Key2} : \ P^{\mathcal{E}^{\$}_{K2}(\cdot)} = 1\right] = \mathbf{Adv}^{\mathbf{auth}}_{\mathrm{EAX2}[\Pi, \mathcal{R}^n_n, \tau]}(B) \tag{9}$$

$$\Pr\left[K2 \xleftarrow{\$} \mathsf{Key2} : \ P^{\$(\cdot)} = 1\right] = \mathbf{Adv}^{\mathrm{rauth}}(B) \ . \tag{10}$$

Subtracting, we get Equation (8), and this concludes the proof. ∎

Proof of Theorem 3: Let A be an adversary using resources at most (t, q, σ) that attacks the authenticity of $\mathbf{\Pi} = (\mathbf{E}, \mathbf{D}) = \mathrm{EAX2}[\Pi, F, \tau]$. Using A, we construct an adversary B for distinguishing $f \xleftarrow{\$} F$ from $f \xleftarrow{\$} \mathcal{R}^n_n$. Adversary B, which has oracle f, works as follows. At the beginning of B's execution it chooses $K2 \xleftarrow{\$} \mathsf{Key2}$ where $\mathsf{Key2}$ is the key space of Π. Then B runs A. When A makes an oracle query (N_i, H_i, M_i) adversary B computes $\mathcal{N}_i \leftarrow f([0]_n \parallel N_i)$ and $C_i \leftarrow \mathcal{E}^{\mathcal{N}_i}_{K2}(M_i)$ and $\mathcal{H}_i \leftarrow f([1]_n \parallel H_i)$ and $\mathcal{C}_i \leftarrow f([2]_n \parallel C_i)$ and $T_i \leftarrow (\mathcal{N}_i \oplus \mathcal{C}_i \oplus \mathcal{H}_i)$ [first τ bits] and $CT_i \leftarrow C_i \parallel T_i$ and then B returns to A the string CT_i. When A

halts, outputting an attempted forgery $(N, H, C \,\|\, T)$, adversary B checks if this is a valid forgery: (1) it checks if $(N, H, C\|T)$ is distinct from every $(N_i, H_i, C_i\|T_i)$ that has been computed; (2) it computes $\mathcal{N} \leftarrow f([0]_n \| N)$ and $\mathcal{H} \leftarrow f([1]_n \,\|\, H)$ and $\mathcal{C} \leftarrow f([2]_n \,\|\, C)$ and sees if $T = (\mathcal{N} \oplus \mathcal{H} \oplus \mathcal{C})$ [first τ bits]. If both conditions hold then B returns the bit 1 (it guesses that $f = F_{K1}$ for a random $K1$) and otherwise it outputs the bit 0 (it guesses that f is a random function from $\mathcal{R}_n^n$).

Note that B makes a total of $3q + 3$ oracle calls. The total length of those queries is σ. (Recall our convention that we include in σ the output length and the number of components in each vector that is queried.) The running time of B is $t_1 = t + \mathrm{Time}_{\mathcal{E}}(\sigma) + \widetilde{O}(\sigma)$. Finally, adversary B provides A a perfect simulation of EAX2$[\Pi, F, \tau]$ if f is selected by $f \xleftarrow{\$} F$ while B provides A a perfect simulation of EAX2$[\Pi, \mathcal{R}_n^n, \tau]$ if $f \xleftarrow{\$} \mathcal{R}_n^n$. Thus using Lemma 2 we have that

$$\begin{aligned}
&\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{EAX2}[\Pi,F,\tau]}(A)\\
&= \Big(\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{EAX2}[\Pi,F,\tau]}(A) - \mathbf{Adv}^{\mathbf{auth}}_{\mathrm{EAX2}[\Pi,\mathcal{R}_n^n,\tau]}(A)\Big) + \mathbf{Adv}^{\mathbf{auth}}_{\mathrm{EAX2}[\Pi,\mathcal{R}_n^n,\tau]}(A)\\
&\leq \Big(\Pr[f \xleftarrow{\$} F :\ B^f = 1] - \Pr[f \xleftarrow{\$} \mathcal{R}_n^n :\ B^f = 1]\Big) + 2^{-\tau} + \mathbf{Adv}^{\mathrm{priv}}_{\Pi}(t_2, q, \sigma)\\
&= \mathbf{Adv}^{\mathrm{prf}}_{F}(B) + \mathbf{Adv}^{\mathrm{priv}}_{\Pi}(t_2, q, \sigma) + 2^{-\tau}\\
&\leq \mathbf{Adv}^{\mathrm{prf}}_{F}(t_1,\ 3q+3,\ \sigma) + \mathbf{Adv}^{\mathrm{priv}}_{\Pi}(t_2, q, \sigma) + 2^{-\tau}\ .
\end{aligned}$$

This completes the proof of Equation (1).

Reusing the name, let A be an adversary that attacks the privacy of $\mathbf{\Pi} = (\mathbf{E}, \mathbf{D})$ and uses resources at most (t, q, σ). Reusing the name, we construct an adversary B for attacking the pseudorandomness of F.

Adversary B, which has an oracle for f, is constructed as follows. At the beginning of B's execution it chooses $K2 \xleftarrow{\$} \mathsf{Key2}$ where $\mathsf{Key2}$ is the key space of Π. Then B runs A. When A makes an oracle call (N_i, H_i, M_i) adversary B computes $\mathcal{N}_i \leftarrow f([0]_n \,\|\, N_i)$ and $C_i \leftarrow \mathcal{E}^{\mathcal{N}_i}_{K2}(M_i)$ and $\mathcal{H}_i \leftarrow f([1]_n \,\|\, H_i)$ and $\mathcal{C}_i \leftarrow f([2]_n \,\|\, C_i)$ and $T_i \leftarrow (\mathcal{N}_i \oplus \mathcal{C}_i \oplus \mathcal{H}_i)$ [first τ bits] and $CT_i \leftarrow C_i \,\|\, T_i$ and then B returns to A the string CT_i. When A halts, outputting a bit b, adversary B outputs the same bit b.

The total number of oracle queries made by B is $3q$. The total length of these queries is at most σ. The running time of B is $t_3 = t + \mathrm{Time}_{\mathcal{E}}(\sigma) + \widetilde{O}(\sigma)$. Finally, adversary B provides A a perfect simulation of EAX2$[\Pi, F, \tau]$ if f is selected by $f \xleftarrow{\$} F$ while B provides A a perfect simulation of EAX2$[\Pi, \mathcal{R}_n^n, \tau]$ if $f \xleftarrow{\$} \mathcal{R}_n^n$. Now using Lemma 1 we have that

$$\begin{aligned}
\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{EAX2}[\Pi,F,\tau]}(A) &= \Big(\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{EAX2}[\Pi,F,\tau]}(A) - \mathbf{Adv}^{\mathbf{priv}}_{\mathrm{EAX2}[\Pi,\mathcal{R}_n^n,\tau]}(A)\Big) + \mathbf{Adv}^{\mathbf{priv}}_{\mathrm{EAX2}[\Pi,\mathcal{R}_n^n,\tau]}(A)\\
&\leq \Big(\Pr[f \xleftarrow{\$} F :\ B^f = 1] - \Pr[f \xleftarrow{\$} \mathcal{R}_n^n :\ B^f = 1]\Big) + \mathbf{Adv}^{\mathrm{priv}}_{\Pi}(t_2, q, \sigma)\\
&= \mathbf{Adv}^{\mathrm{prf}}_{\Pi}(B) + \mathbf{Adv}^{\mathrm{priv}}_{\Pi}(t_2, q, \sigma)\\
&\leq \mathbf{Adv}^{\mathrm{prf}}_{\Pi}(t_3, 3q, \sigma) + \mathbf{Adv}^{\mathrm{priv}}_{\Pi}(t_2, q, \sigma)
\end{aligned}$$

This completes the proof of Equation (2). ▮

D Proof of Security of $\mathbb{OMAC}$

Proof of Lemma 4: Let A be a length-committing adversary for distinguishing $\mathbb{OMAC}[\mathcal{R}_n^n]$ from a random function. Assume that A uses resources (σ_1, σ_2). Without loss of generality we assume that A is deterministic and makes no repeated queries. We simulate the behavior of an $\mathbb{OMAC}[\mathcal{R}_n^n]$ oracle as show in Figure 11. That

Initialization

00 $\rho \xleftarrow{\$} \mathcal{R}_n^n$
01 $L_0 \leftarrow \rho([0]_n)$; $L_1 \leftarrow \rho([1]_n)$; $L_2 \leftarrow \rho([2]_n)$

On query $(t,\, M,\, s)$**, where** $t \in \{0,1,2\}$ **and** $M = M_1 \cdots M_m$ **and** $s \in \mathbb{N}$

10 **if** $|M| = n$ **then** $T \leftarrow \rho(M_1 \oplus L_t \oplus 2L_0)$
20 **elseif** $|M| < n$ **then** $T \leftarrow \rho(\widetilde{M_1} \oplus L_t \oplus 4L_0)$
30 **else** **if** $Y[t,\, M_1]$ is undefined **then** $Y[t,\, M_1] \leftarrow \rho(M_1 \oplus L_t)$
31 $u \leftarrow$ the largest number in $[1 \mathinner{..} m-1]$ s.t. $Y[t,\, M_{1..u}]$ is defined
32 **for** $j \leftarrow u+1$ **to** $m-1$ **do** $Y[t,\, M_{1..j}] \leftarrow \rho(Y[t,\, M_{1..j-1}] \oplus M_j)$
33 **if** $|M_m| = n$ **then** $T \leftarrow \rho(Y[t,\, M_{1..m-1}] \oplus M_m \oplus 2L_0)$
34 **if** $|M_m| < n$ **then** $T \leftarrow \rho(Y[t,\, M_{1..m-1}] \oplus \widetilde{M_m} \oplus 4L_0)$
40 **for** $j \leftarrow 0$ **to** $s-1$ **do** $S_j \leftarrow \rho(T{+}j{-}1)$
50 **return** $T \parallel S_0 \cdots S_{s-1}$

Figure 11: Game $\mathbf{Q1}$, which perfectly simulates an OMAC_ρ^t oracle for $t \in \{0,1,2\}$ and ρ a random function from $\mathcal{R}_n^n$.

figure depicts a mechanism, game $\mathbf{Q1}$, that coincides with the definition of $\mathbb{OMAC}[\mathcal{R}_n^n]$. As before, we use the notation $\widetilde{X}$ for the padded version of the string X, namely $\widetilde{X} = X10^{n-|X|-1}$.

Game $\mathbf{Q1}$ is not the most obvious simulation of an $\mathbb{OMAC}[\mathcal{R}_n^n]$ oracle. In particular, the game distinguishes among the following cases: one-block messages that are a full n bits (line 10); one-block messages that fall short of n bits (line 20); messages with two or more blocks where the final block is a full block (line 33); and messages with two or more blocks where the final block is a short block (line 34). In addition to breaking into these cases, we implement memoization by way of the array Y. In particular, when a query $M_1 \ldots M_m$ is asked we record (memoize) the intermediate values that we get as we CBC our way down $M_1 \cdots M_{m-1}$. If any of prefixes $M_1, M_{1..2}, \ldots, M_{1..m-1}$ should arise again with the same tweak we will not re-compute the values needed as we chain down the message, looking up the answer in the array Y instead. Notice that memoization stops one block short of the final block M_m and that the memoization is tweak-dependent.

To help us understand the behavior of game $\mathbf{Q1}$ we make some changes to it, yielding game $\mathbf{Q2}$, defined in Figure 12. As is standard, game $\mathbf{Q2}$ avoids choosing $\rho \xleftarrow{\$} \mathcal{R}_n^n$ at the beginning and instead fills in values incrementally. We write $\mathrm{Domain}(\rho)$ for the set of all $X \in \{0,1\}^n$ such that $\rho(X)$ has been assigned some value. Any time we need a $\rho(X)$ value, if it is not yet defined then we choose a value at random from $\{0,1\}^n$ and make this to be $\rho(X)$. Any time we need a value for $\rho(X)$ that has been defined already, we use that old value. In the latter case we also set a flag *bad*. The flag *bad* effects nothing visible to the adversary, but it is central to our subsequent analysis. It is easy to verify that games $\mathbf{Q1}$ and $\mathbf{Q2}$ provide identical views to any adversary that interacts with them, so $\Pr[A^{\mathbf{Q1}} = 1] = \Pr[A^{\mathbf{Q2}} = 1]$.

Also defined in Figure 12 is game $\mathbf{R1}$. This game is obtained by dropping the highlighted statements from game $\mathbf{Q1}$. We only omit statements that immediately follow the setting of the flag *bad*. The game $\mathbf{R1}$ is easily seen to return $n(s-1)$ random bits in response to any query (t, M, s). Thus $\mathbf{Adv}^{\mathrm{dist}}_{\mathbf{Q2},\mathbf{R1}}(A) = |\Pr[A^{\mathbf{Q2}} = 1] - |\Pr[A^{\mathbf{R1}} = 1]| \le \Pr[A^{\mathbf{R1}}$ sets *bad*]. This is the standard setup for analyses within the game-playing paradigm.

To more easily understand game $\mathbf{R1}$ we rewrite it a bit, resulting in the game $\mathbf{R2}$ shown in Figure 13. To understand the change from game $\mathbf{R1}$ to game $\mathbf{R2}$ notice that, having eliminated the seven highlighted statements of game $\mathbf{Q2}$, the $\rho(X)$-values are no longer actually used in game $\mathbf{R1}$: all that one needs to keep track

Initialization

$L_0 \xleftarrow{\$} \{0,1\}^n$; $L_1 \xleftarrow{\$} \{0,1\}^n$; $L_2 \xleftarrow{\$} \{0,1\}^n$
$\rho([0]_n) \leftarrow L_0$; $\rho([1]_n) \leftarrow L_1$; $\rho([2]_n) \leftarrow L_2$
bad $\leftarrow$ `false`

On query $(t,\ M,\ s)$**, where** $t \in \{0,1,2\}$ **and** $M = M_1 \cdots M_m$ **and** $s \in \mathbb{N}$

$T \xleftarrow{\$} \{0,1\}^n$
if $|M| = n$ **then**
  $X_1 \leftarrow M_1 \oplus L_t \oplus 2L_0$; **if** $X_1 \in \mathrm{Domain}(\rho)$ **then** *bad* $\leftarrow$ `true` , $T \leftarrow \rho(X_1)$
  $\rho(X_1) \leftarrow T$
elseif $|M| < n$ **then**
  $X_1 \leftarrow \widetilde{M_1} \oplus L_t \oplus 4L_0$; **if** $X_1 \in \mathrm{Domain}(\rho)$ **then** *bad* $\leftarrow$ `true` , $T \leftarrow \rho(X_1)$
  $\rho(X_1) \leftarrow T$
else **if** $Y[t,\ M_1]$ is undefined **then**
    $X_1 \leftarrow M_1 \oplus L_t$; $Y[t,\ M_1] \xleftarrow{\$} \{0,1\}^n$
    if $X_1 \in \mathrm{Domain}(\rho)$ **then** *bad* $\leftarrow$ `true` , $Y[t,\ M_1] \leftarrow \rho(X_1)$
    $\rho(X_1) \leftarrow Y[t,\ M_1]$
  $u \leftarrow$ the largest number in $[1\ ..\ m-1]$ s.t. $Y[t,\ M_{1..u}]$ is defined
  for $j \leftarrow u+1$ **to** $m-1$ **do**
    $X_j \leftarrow Y[t,\ M_{1..j-1}] \oplus M_j$; $Y[t,\ M_{1..j}] \xleftarrow{\$} \{0,1\}^n$
    if $\rho(X_j)$ is defined **then** *bad* $\leftarrow$ `true` , $Y[t,\ M_{1..j}] \leftarrow \rho(X_j)$
    $\rho(X_m) \leftarrow Y[t,\ M_{1..j}]$
  if $|M_m| = n$ **then**
    $X_m \leftarrow Y[t,\ M_{1..m-1}] \oplus M_m \oplus 2L_0$
    if $X_m \in \mathrm{Domain}(\rho)$ **then** *bad* $\leftarrow$ `true` , $T \leftarrow \rho(X_m)$
    $\rho(X_m) \leftarrow T$
  if $|M_m| < n$ **then**
    $X_m \leftarrow Y[t,\ M_{1..m-1}] \oplus \widetilde{M_m} \oplus 4L_0$
    if $X_m \in \mathrm{Domain}(\rho)$ **then** *bad* $\leftarrow$ `true` , $T \leftarrow \rho(X_m)$
    $\rho(X_m) \leftarrow T$
for $j \leftarrow 0$ **to** $s-1$ **do**
  $S_j \xleftarrow{\$} \{0,1\}^n$
  if $T{+}j{-}1 \in \mathrm{Domain}(\rho)$ **then** *bad* $\leftarrow$ `true` , $S_j \leftarrow \rho(T{+}j{-}1)$
  $\rho(T{+}j{-}1) \leftarrow S_j$
return $T \parallel S_0 \cdots S_{s-1}$

Figure 12: Game $\mathbf{Q2}$, which is equivalent to game $\mathbf{Q1}$. Game $\mathbf{R1}$ is obtained by omitting the highlighted statements.

of is whether or not a point X has already been placed into the domain of ρ. Thus game $\mathbf{R2}$ ceases to keep track of ρ-values; instead, we record in the variable $\mathcal{R}$ what would be the domain of ρ at this point in time. So $\mathcal{R}$ starts off as $\{[0]_n, [1]_n, [2]_n\}$ (corresponding to the fact that $\rho([0]_n)$, $\rho([1]_n)$, and $\rho([2]_n)$ are defined in game $\mathbf{R1}$) and then, whenever a point X would have been placed into the domain of ρ, with some value being assigned to $\rho(X)$, we simply add X to the set $\mathcal{R}$, not bothering with anything else. Instead of testing if a given point is in the domain of ρ we test if it is in $\mathcal{R}$. At this point we notice that the random value T is not used until the final lines of the game, so, for added clarity, we move down in the program the choosing of the random value T. We have that $\Pr[A^{\mathbf{R1}}$ sets *bad*$] = \Pr[A^{\mathbf{R2}}$ sets *bad*$]$. Given what we have said so far, $\mathbf{Adv}^{\mathrm{prf}}_{\mathbb{OMAC}[\mathcal{R}^n_n]}(A) = \mathbf{Adv}^{\mathrm{dist}}_{\mathbf{Q1},\mathbf{R2}}(A) \le \Pr[A^{\mathbf{R2}}$ sets *bad*$]$. Our job has been reduced to understanding the adversary's chance of setting *bad* in game $\mathbf{R2}$.

Let us dispense right away with the chance that *bad* is set at line 73. The value T is chosen at random at line 70 and then we see if any of the s points $T, T+1, \ldots, T+s-1$ are in the set $\mathcal{R}$. Now $|\mathcal{R}| \le \sigma_1 + \sigma_2 + 3$ throughout the execution of game $\mathbf{R2}$ and we are testing for the presence of at most σ_2 points in $\mathcal{R}$, and so

$$\Pr[\textit{bad} \text{ gets set at line 73 of game } \mathbf{R2}] \quad \le \quad \frac{\sigma_2(\sigma_1+\sigma_2+3)}{2^n} \tag{11}$$

Let game $\mathbf{R3}$ coincide with game $\mathbf{R2}$ except for eliminating line 73. The probability that *bad* gets set in game $\mathbf{R2}$ is at most the probability that *bad* gets set in line 73 of game $\mathbf{R2}$ plus the probability that *bad* gets set in a line other than line 73 of game $\mathbf{R2}$. So by equation (11) we have that

$$\Pr[\textit{bad} \text{ gets set in game } \mathbf{R2}] \quad \le \quad \Pr[\textit{bad} \text{ gets set in game } \mathbf{R3}] + \frac{\sigma_2(\sigma_1+\sigma_2+3)}{2^n} \tag{12}$$

We proceed with the analysis of game $\mathbf{R3}$.

The values S_j returned to the adversary in game $\mathbf{R3}$ have no impact on any internal variable maintained by the game (these values are chosen, returned to the adversary, and never used again). The only significance of the q T-values returned to the adversary is to define some σ_2 $\mathcal{R}$-values—values that the adversary does not control. Thus we will only be giving the adversary *more* power if we allow it to select an initial set of $\sigma_2 + 3$ values for $\mathcal{R}$ (the "$+3$" reflecting the three values that were assigned to $\mathcal{R}$ at line 01) and have it interact no further with the game, since everything is at that point determined. In other words, the game is made noninteractive, but we maximize over all possible choices $\{\mathsf{R}_1, \ldots, \mathsf{R}_{\sigma_2+3}\}$ of initial values for $\mathcal{R}$.The adversary's corresponding queries are now fixed.[6] The modified game is shown in Figure 14. We regard all of $\mathtt{q}$ and $\mathtt{t}^1, \ldots, \mathtt{t}^{\mathtt{q}} \in \{0,1,2\}$ and strings $\mathtt{M}^1, \ldots, \mathtt{M}^{\mathtt{q}}$ having block lengths $\mathtt{m}^1, \ldots, \mathtt{m}^{\mathtt{q}}$ and strings $\mathsf{R}_1, \cdots, \mathsf{R}_{\sigma_2+3} \in \{0,1\}^n$ as fixed constants associated to the game. We must bound the probability that *bad* gets set to `true` in game $\mathbf{S}$. We do that with a case analysis.

If the flag *bad* gets set in game $\mathbf{S}$ it is because a point X^s_j gets computed in one of lines 11, 21, 31, 41, 51, 61 (six possibilities), but that point was *already* placed in $\mathcal{R}$ by an earlier execution of one of lines 01, 12, 22, 33, 43, 53, 63 (seven possibilities). This gives a total of $6 \times 7 = 42$ cases. We refer to the "current" point as X^s_j and the "earlier" point as X^r_i. The current point must follow the earlier point under the natural ordering. The current point X^s_j and the earlier point X^r_i can be set as any of the following:

[6] Note that, strictly speaking, the attacker can also control s for each invocation. However, it is still true that we are only giving the adversary more power if we allow the adversary to select $\sigma_2 + 3$ values in advance rather than observing the S- and T-values as we go.

Initialization

$L_0 \xleftarrow{\$} \{0,1\}^n$; $L_1 \xleftarrow{\$} \{0,1\}^n$; $L_2 \xleftarrow{\$} \{0,1\}^n$
$\mathcal{R} \leftarrow \{[0]_n, [1]_n, [2]_n\}$
bad $\leftarrow$ `false`

On query $(t,\ M,\ s)$**, where** $t \in \{0,1,2\}$ **and** $M = M_1 \cdots M_m$ **and** $s \in \mathbb{N}$

if $|M| = n$ **then**
    $X_1 \leftarrow M_1 \oplus L_t \oplus 2L_0$; **if** $X_1 \in \mathcal{R}$ **then** *bad* $\leftarrow$ `true`
    $\mathcal{R} \leftarrow \mathcal{R} \cup \{X_1\}$
elseif $|M| < n$ **then**
    $X_1 \leftarrow \widetilde{M_1} \oplus L_t \oplus 4L_0$; **if** $X_1 \in \mathcal{R}$ **then** *bad* $\leftarrow$ `true`
    $\mathcal{R} \leftarrow \mathcal{R} \cup \{X_1\}$
else **if** $Y[t,\ M_1]$ is undefined **then**
        $X_1 \leftarrow M_1 \oplus L_t$; $Y[t,\ M_1] \xleftarrow{\$} \{0,1\}^n$
        if $X_1 \in \mathcal{R}$ **then** *bad* $\leftarrow$ `true`
        $\mathcal{R} \leftarrow \mathcal{R} \cup \{X_1\}$
    $u \leftarrow$ the largest number in $[1 \mathinner{..} m-1]$ s.t. $Y[t,\ M_{1..u}]$ is defined
    for $j \leftarrow u+1$ **to** $m-1$ **do**
        $X_j \leftarrow Y[t,\ M_{1..j-1}] \oplus M_j$; $Y[t,\ M_{1..j}] \xleftarrow{\$} \{0,1\}^n$
        if $X_j \in \mathcal{R}$ **then** *bad* $\leftarrow$ `true`
        $\mathcal{R} \leftarrow \mathcal{R} \cup \{X_j\}$
    if $|M_m| = n$ **then**
        $X_m \leftarrow Y[t,\ M_{1..m-1}] \oplus M_m \oplus 2L_0$
        if $X_m \in \mathcal{R}$ **then** *bad* $\leftarrow$ `true`
        $\mathcal{R} \leftarrow \mathcal{R} \cup \{X_m\}$
    if $|M_m| < n$ **then**
        $X_m \leftarrow Y[t,\ M_{1..m-1}] \oplus \widetilde{M_m} \oplus 4L_0$
        if $X_m \in \mathcal{R}$ **then** *bad* $\leftarrow$ `true`
        $\mathcal{R} \leftarrow \mathcal{R} \cup \{X_m\}$

$T \xleftarrow{\$} \{0,1\}^n$
for $j \leftarrow 0$ **to** $s-1$ **do**
    $S_j \xleftarrow{\$} \{0,1\}^n$
    if $T+j-1 \in \mathcal{R}$ **then** *bad* $\leftarrow$ `true`
$\mathcal{R} \leftarrow \mathcal{R} \cup \{T,\ T+1,\ \ldots,\ T+s-1\}$
return $T \parallel S_0 \cdots S_{s-1}$

Figure 13: Game $\mathbf{R2}$, which is equivalent to $\mathbf{R1}$ but no longer maintains the function ρ.

$L_0 \xleftarrow{\$} \{0,1\}^n$; $L_1 \xleftarrow{\$} \{0,1\}^n$; $L_2 \xleftarrow{\$} \{0,1\}^n$
$\mathcal{R} \leftarrow \{\mathsf{R}_1, \ldots, \mathsf{R}_{\sigma_2+3}\}$
bad $\leftarrow$ `false`

for $s \leftarrow 1$ **to** $\mathtt{q}$ **do**
    if $|\mathtt{M}^s| = n$ **then**
        $X_1^s \leftarrow \mathtt{M}_1^s \oplus L_{\mathtt{t}^s} \oplus 2L_0$; **if** $X_1^s \in \mathcal{R}$ **then** *bad* $\leftarrow$ `true`
        $\mathcal{R} \leftarrow \mathcal{R} \cup \{X_1^s\}$
    elseif $|\mathtt{M}^s| < n$ **then**
        $X_1^s \leftarrow \widetilde{\mathtt{M}_1^s} \oplus L_{\mathtt{t}^s} \oplus 4L_0$; **if** $X_1^s \in \mathcal{R}$ **then** *bad* $\leftarrow$ `true`
        $\mathcal{R} \leftarrow \mathcal{R} \cup \{X_1^s\}$
    else **if** $Y[\mathtt{t}^s, \mathtt{M}_1^s]$ is undefined **then**
            $X_1^s \leftarrow \mathtt{M}_1^s \oplus L_{\mathtt{t}^s}$; $Y[\mathtt{t}^s, \mathtt{M}_1^s] \xleftarrow{\$} \{0,1\}^n$
            if $X_1^s \in \mathcal{R}$ **then** *bad* $\leftarrow$ `true`
            $\mathcal{R} \leftarrow \mathcal{R} \cup \{X_1^s\}$
        $u \leftarrow$ the largest number in $[1 \,..\, \mathtt{m}^s - 1]$ s.t. $Y[\mathtt{t}^s, \mathtt{M}_{1..u}^s]$ is defined
        for $j \leftarrow u+1$ **to** $\mathtt{m}^s - 1$ **do**
            $X_j^s \leftarrow Y[\mathtt{t}^s, \mathtt{M}_{1..j-1}^s] \oplus \mathtt{M}_j^s$; $Y[\mathtt{t}^s, \mathtt{M}_{1..j}^s] \xleftarrow{\$} \{0,1\}^n$
            if $X_j^s \in \mathcal{R}$ **then** *bad* $\leftarrow$ `true`
            $\mathcal{R} \leftarrow \mathcal{R} \cup \{X_j^s\}$
        if $|\mathtt{M}_{\mathtt{m}^s}^s| = n$ **then**
            $X_{\mathtt{m}^s}^s \leftarrow Y[\mathtt{t}^s, \mathtt{M}_{1..\mathtt{m}^s-1}^s] \oplus \mathtt{M}_{\mathtt{m}^s}^s \oplus 2L_0$
            if $X_{\mathtt{m}^s}^s \in \mathcal{R}$ **then** *bad* $\leftarrow$ `true`
            $\mathcal{R} \leftarrow \mathcal{R} \cup \{X_{\mathtt{m}^s}^s\}$
        if $|\mathtt{M}_{\mathtt{m}^s}^s| < n$ **then**
            $X_{\mathtt{m}^s}^s \leftarrow Y[\mathtt{t}^s, \mathtt{M}_{1..\mathtt{m}^s-1}^s] \oplus \widetilde{\mathtt{M}_{\mathtt{m}^s}^s} \oplus 4L_0$
            if $X_{\mathtt{m}^s}^s \in \mathcal{R}$ **then** *bad* $\leftarrow$ `true`
            $\mathcal{R} \leftarrow \mathcal{R} \cup \{X_{\mathtt{m}^s}^s\}$

Figure 14: Game S, the noninteractive game that the analysis focuses on.

case	earlier point X^r_i	current point X^s_j	$\Pr[X^r_i = X^s_j]$	explanation
1	R_ℓ	$\mathtt{M}^s_1 \oplus L_{\mathsf{t}^s} \oplus 2L_0$	2^{-n}	randomness of L_0
2	$\mathtt{M}^r_1 \oplus L_{\mathsf{t}^r} \oplus 2L_0$	$\mathtt{M}^s_1 \oplus L_{\mathsf{t}^s} \oplus 2L_0$	0 or 2^{-n}	no repeated queries / randomness of L_{t^s}
3	$\widetilde{\mathtt{M}^r_1} \oplus L_{\mathsf{t}^r} \oplus 4L_0$	$\mathtt{M}^s_1 \oplus L_{\mathsf{t}^s} \oplus 2L_0$	2^{-n}	randomness of L_0
4	$\mathtt{M}^r_1 \oplus L_{\mathsf{t}^r}$	$\mathtt{M}^s_1 \oplus L_{\mathsf{t}^s} \oplus 2L_0$	2^{-n}	randomness of L_0
5	$Y[\mathsf{t}^r, \mathtt{M}^r_{1..i-1}] \oplus \mathtt{M}^r_i$	$\mathtt{M}^s_1 \oplus L_{\mathsf{t}^s} \oplus 2L_0$	2^{-n}	randomness of L_0
6	$Y[\mathsf{t}^r, \mathtt{M}^r_{1..\mathtt{m}^r-1}] \oplus \mathtt{M}^r_i \oplus 2L_0$	$\mathtt{M}^s_1 \oplus L_{\mathsf{t}^s} \oplus 2L_0$	2^{-n}	randomness of $Y[\mathsf{t}^r, \mathtt{M}^r_{1..\mathtt{m}^r-1}]$
7	$Y[\mathsf{t}^r, \mathtt{M}^s_{1..\mathtt{m}^r-1}] \oplus \widetilde{\mathtt{M}^r_i} \oplus 4L_0$	$\mathtt{M}^s_1 \oplus L_{\mathsf{t}^s} \oplus 2L_0$	2^{-n}	randomness of $Y[\mathsf{t}^r, \mathtt{M}^r_{1..\mathtt{m}^r-1}]$
8	R_ℓ	$\widetilde{\mathtt{M}^s_1} \oplus L_{\mathsf{t}^s} \oplus 4L_0$	2^{-n}	randomness of L_0
9	$\mathtt{M}^r_1 \oplus L_{\mathsf{t}^r} \oplus 2L_0$	$\widetilde{\mathtt{M}^s_1} \oplus L_{\mathsf{t}^s} \oplus 4L_0$	2^{-n}	randomness of L_0
10	$\widetilde{\mathtt{M}^r_1} \oplus L_{\mathsf{t}^r} \oplus 4L_0$	$\widetilde{\mathtt{M}^s_1} \oplus L_{\mathsf{t}^s} \oplus 4L_0$	0 or 2^{-n}	no repeated queries / randomness of L_{t^s}
11	$\mathtt{M}^r_1 \oplus L_{\mathsf{t}^r}$	$\widetilde{\mathtt{M}^s_1} \oplus L_{\mathsf{t}^s} \oplus 4L_0$	2^{-n}	randomness of L_0
12	$Y[\mathsf{t}^r, \mathtt{M}^r_{1..i-1}] \oplus \mathtt{M}^r_i$	$\widetilde{\mathtt{M}^s_1} \oplus L_{\mathsf{t}^s} \oplus 4L_0$	2^{-n}	randomness of L_0
13	$Y[\mathsf{t}^r, \mathtt{M}^r_{1..\mathtt{m}^r-1}] \oplus \mathtt{M}^r_i \oplus 2L_0$	$\widetilde{\mathtt{M}^s_1} \oplus L_{\mathsf{t}^s} \oplus 4L_0$	2^{-n}	randomness of $Y[\mathsf{t}^r, \mathtt{M}^r_{1..\mathtt{m}^r-1}]$
14	$Y[\mathsf{t}^r, \mathtt{M}^s_{1..\mathtt{m}^r-1}] \oplus \widetilde{\mathtt{M}^r_i} \oplus 4L_0$	$\widetilde{\mathtt{M}^s_1} \oplus L_{\mathsf{t}^s} \oplus 4L_0$	2^{-n}	randomness of $Y[\mathsf{t}^r, \mathtt{M}^r_{1..\mathtt{m}^r-1}]$
15	R_ℓ	$\mathtt{M}^s_1 \oplus L_{\mathsf{t}^s}$	2^{-n}	randomness of L_{t^s}
16	$\mathtt{M}^r_1 \oplus L_{\mathsf{t}^r} \oplus 2L_0$	$\mathtt{M}^s_1 \oplus L_{\mathsf{t}^s}$	2^{-n}	randomness of L_0
17	$\widetilde{\mathtt{M}^r_1} \oplus L_{\mathsf{t}^r} \oplus 4L_0$	$\mathtt{M}^s_1 \oplus L_{\mathsf{t}^s}$	2^{-n}	randomness of L_0
18	$\mathtt{M}^r_1 \oplus L_{\mathsf{t}^r}$	$\mathtt{M}^s_1 \oplus L_{\mathsf{t}^s}$	0 or 2^{-n}	memoization / randomness of L_{t^s}
19	$Y[\mathsf{t}^r, \mathtt{M}^r_{1..i-1}] \oplus \mathtt{M}^r_i$	$\mathtt{M}^s_1 \oplus L_{\mathsf{t}^s}$	2^{-n}	randomness of L_{t^s}
20	$Y[\mathsf{t}^r, \mathtt{M}^r_{1..\mathtt{m}^r-1}] \oplus \mathtt{M}^r_i \oplus 2L_0$	$\mathtt{M}^s_1 \oplus L_{\mathsf{t}^s}$	2^{-n}	randomness of $Y[\mathsf{t}^r, \mathtt{M}^r_{1..\mathtt{m}^r-1}]$
21	$Y[\mathsf{t}^r, \mathtt{M}^s_{1..\mathtt{m}^r-1}] \oplus \widetilde{\mathtt{M}^r_i} \oplus 4L_0$	$\mathtt{M}^s_1 \oplus L_{\mathsf{t}^s}$	2^{-n}	randomness of $Y[\mathsf{t}^r, \mathtt{M}^r_{1..\mathtt{m}^r-1}]$
22	R_ℓ	$Y[\mathsf{t}^s, \mathtt{M}^s_{1..j-1}] \oplus \mathtt{M}^s_j$	2^{-n}	randomness of $Y[\mathsf{t}^s, \mathtt{M}^s_{1..j-1}]$
23	$\mathtt{M}^r_1 \oplus L_{\mathsf{t}^r} \oplus 2L_0$	$Y[\mathsf{t}^s, \mathtt{M}^s_{1..j-1}] \oplus \mathtt{M}^s_j$	2^{-n}	randomness of $Y[\mathsf{t}^s, \mathtt{M}^s_{1..j-1}]$
24	$\widetilde{\mathtt{M}^r_1} \oplus L_{\mathsf{t}^r} \oplus 4L_0$	$Y[\mathsf{t}^s, \mathtt{M}^s_{1..j-1}] \oplus \mathtt{M}^s_j$	2^{-n}	randomness of $Y[\mathsf{t}^s, \mathtt{M}^s_{1..j-1}]$
25	$\mathtt{M}^r_1 \oplus L_{\mathsf{t}^r}$	$Y[\mathsf{t}^s, \mathtt{M}^s_{1..j-1}] \oplus \mathtt{M}^s_j$	2^{-n}	randomness of $Y[\mathsf{t}^s, \mathtt{M}^s_{1..j-1}]$
26	$Y[\mathsf{t}^r, \mathtt{M}^r_{1..i-1}] \oplus \mathtt{M}^r_i$	$Y[\mathsf{t}^s, \mathtt{M}^s_{1..j-1}] \oplus \mathtt{M}^s_j$	0 or 2^{-n}	memoization / randomness of $Y[\mathsf{t}^r, \mathtt{M}^r_{1..i-1}]$
27	$Y[\mathsf{t}^r, \mathtt{M}^r_{1..\mathtt{m}^r-1}] \oplus \mathtt{M}^r_i \oplus 2L_0$	$Y[\mathsf{t}^s, \mathtt{M}^s_{1..j-1}] \oplus \mathtt{M}^s_j$	2^{-n}	randomness of L_0
28	$Y[\mathsf{t}^r, \mathtt{M}^s_{1..\mathtt{m}^r-1}] \oplus \widetilde{\mathtt{M}^r_i} \oplus 4L_0$	$Y[\mathsf{t}^s, \mathtt{M}^s_{1..j-1}] \oplus \mathtt{M}^s_j$	2^{-n}	randomness of L_0
29	R_ℓ	$Y[\mathsf{t}^s, \mathtt{M}^s_{1..\mathtt{m}^s-1}] \oplus \mathtt{M}^s_j \oplus 2L_0$	2^{-n}	randomness of $Y[\mathsf{t}^s, \mathtt{M}^s_{1..\mathtt{m}^s-1}]$
30	$\mathtt{M}^r_1 \oplus L_{\mathsf{t}^r} \oplus 2L_0$	$Y[\mathsf{t}^s, \mathtt{M}^s_{1..\mathtt{m}^s-1}] \oplus \mathtt{M}^s_j \oplus 2L_0$	2^{-n}	randomness of $Y[\mathsf{t}^s, \mathtt{M}^s_{1..\mathtt{m}^s-1}]$
31	$\widetilde{\mathtt{M}^r_1} \oplus L_{\mathsf{t}^r} \oplus 4L_0$	$Y[\mathsf{t}^s, \mathtt{M}^s_{1..\mathtt{m}^s-1}] \oplus \mathtt{M}^s_j \oplus 2L_0$	2^{-n}	randomness of $Y[\mathsf{t}^s, \mathtt{M}^s_{1..\mathtt{m}^s-1}]$
32	$\mathtt{M}^r_1 \oplus L_{\mathsf{t}^r}$	$Y[\mathsf{t}^s, \mathtt{M}^s_{1..\mathtt{m}^s-1}] \oplus \mathtt{M}^s_j \oplus 2L_0$	2^{-n}	randomness of $Y[\mathsf{t}^s, \mathtt{M}^s_{1..\mathtt{m}^s-1}]$
33	$Y[\mathsf{t}^r, \mathtt{M}^r_{1..i-1}] \oplus \mathtt{M}^r_i$	$Y[\mathsf{t}^s, \mathtt{M}^s_{1..\mathtt{m}^s-1}] \oplus \mathtt{M}^s_j \oplus 2L_0$	2^{-n}	randomness of L_0
34	$Y[\mathsf{t}^r, \mathtt{M}^r_{1..\mathtt{m}^r-1}] \oplus \mathtt{M}^r_i \oplus 2L_0$	$Y[\mathsf{t}^s, \mathtt{M}^s_{1..\mathtt{m}^s-1}] \oplus \mathtt{M}^s_j \oplus 2L_0$	0 or 2^{-n}	no repeated queries / randomness of $Y[\mathsf{t}^s, \mathtt{M}^s_{1..\mathtt{m}^s-1}]$
35	$Y[\mathsf{t}^r, \mathtt{M}^s_{1..\mathtt{m}^r-1}] \oplus \widetilde{\mathtt{M}^r_i} \oplus 4L_0$	$Y[\mathsf{t}^s, \mathtt{M}^s_{1..\mathtt{m}^s-1}] \oplus \mathtt{M}^s_j \oplus 2L_0$	2^{-n}	randomness of L_0
36	R_ℓ	$Y[\mathsf{t}^s, \mathtt{M}^s_{1..\mathtt{m}^s-1}] \oplus \widetilde{\mathtt{M}^s_j} \oplus 4L_0$	2^{-n}	randomness of $Y[\mathsf{t}^s, \mathtt{M}^s_{1..\mathtt{m}^s-1}]$
37	$\mathtt{M}^r_1 \oplus L_{\mathsf{t}^r} \oplus 2L_0$	$Y[\mathsf{t}^s, \mathtt{M}^s_{1..\mathtt{m}^s-1}] \oplus \widetilde{\mathtt{M}^s_j} \oplus 4L_0$	2^{-n}	randomness of $Y[\mathsf{t}^s, \mathtt{M}^s_{1..\mathtt{m}^s-1}]$
38	$\widetilde{\mathtt{M}^r_1} \oplus L_{\mathsf{t}^r} \oplus 4L_0$	$Y[\mathsf{t}^s, \mathtt{M}^s_{1..\mathtt{m}^s-1}] \oplus \widetilde{\mathtt{M}^s_j} \oplus 4L_0$	2^{-n}	randomness of $Y[\mathsf{t}^s, \mathtt{M}^s_{1..\mathtt{m}^s-1}]$
39	$\mathtt{M}^r_1 \oplus L_{\mathsf{t}^r}$	$Y[\mathsf{t}^s, \mathtt{M}^s_{1..\mathtt{m}^s-1}] \oplus \widetilde{\mathtt{M}^s_j} \oplus 4L_0$	2^{-n}	randomness of $Y[\mathsf{t}^s, \mathtt{M}^s_{1..\mathtt{m}^s-1}]$
40	$Y[\mathsf{t}^r, \mathtt{M}^r_{1..i-1}] \oplus \mathtt{M}^r_i$	$Y[\mathsf{t}^s, \mathtt{M}^s_{1..\mathtt{m}^s-1}] \oplus \widetilde{\mathtt{M}^s_j} \oplus 4L_0$	2^{-n}	randomness of L_0
41	$Y[\mathsf{t}^r, \mathtt{M}^r_{1..\mathtt{m}^r-1}] \oplus \mathtt{M}^r_i \oplus 2L_0$	$Y[\mathsf{t}^s, \mathtt{M}^s_{1..\mathtt{m}^s-1}] \oplus \widetilde{\mathtt{M}^s_j} \oplus 4L_0$	2^{-n}	randomness of L_0
42	$Y[\mathsf{t}^r, \mathtt{M}^s_{1..\mathtt{m}^r-1}] \oplus \widetilde{\mathtt{M}^r_i} \oplus 4L_0$	$Y[\mathsf{t}^s, \mathtt{M}^s_{1..\mathtt{m}^s-1}] \oplus \widetilde{\mathtt{M}^s_j} \oplus 4L_0$	0 or 2^{-n}	no repeated queries / randomness of $Y[\mathsf{t}^s, \mathtt{M}^s_{1..\mathtt{m}^s-1}]$

Figure 15: Case analysis for the proof of $\mathbb{OMAC}$

line 01	X_ℓ^0	R_ℓ
line 12	X_1^r	$\mathtt{M}_1^r \oplus L_{\mathtt{t}^r} \oplus 2L_0$
line 22	X_1^r	$\widetilde{\mathtt{M}_1^r} \oplus L_{\mathtt{t}^r} \oplus 4L_0$
line 33	X_1^r	$\mathtt{M}_1^r \oplus L_{\mathtt{t}^r}$
line 44	X_i^r	$Y[\mathtt{t}^r,\, \mathtt{M}_{1..i-1}^r] \oplus \mathtt{M}_i^r$
line 53	$X_{m^r}^r$	$Y[\mathtt{t}^r,\, \mathtt{M}_{1..\mathtt{m}^r-1}^r] \oplus \mathtt{M}_i^r \oplus 2L_0$
line 63	$X_{m^r}^r$	$Y[\mathtt{t}^r,\, \mathtt{M}_{1..\mathtt{m}^r-1}^r] \oplus \widetilde{\mathtt{M}_i^r} \oplus 4L_0$

line 11	X_1^s	$\mathtt{M}_1^s \oplus L_{\mathtt{t}^s} \oplus 2L_0$
line 21	X_1^s	$\widetilde{\mathtt{M}_1^s} \oplus L_{\mathtt{t}^s} \oplus 4L_0$
line 31	X_1^s	$\mathtt{M}_1^s \oplus L_{\mathtt{t}^s}$
line 41	X_j^s	$Y[\mathtt{t}^s,\, \mathtt{M}_{1..j-1}^s] \oplus \mathtt{M}_j^s$
line 51	$X_{m^s}^s$	$Y[\mathtt{t}^s,\, \mathtt{M}_{1..\mathtt{m}^s-1}^s] \oplus \mathtt{M}_j^s \oplus 2L_0$
line 61	$X_{m^s}^s$	$Y[\mathtt{t}^s,\, \mathtt{M}_{1..\mathtt{m}^s-1}^s] \oplus \widetilde{\mathtt{M}_j^s} \oplus 4L_0$

Each current point X_j^s that gets considered during game **S** has a "type" which is one of the six possibilities above. Each earlier point X_i^r likewise has a "type" which is one of the seven possibilities above. The type of a point does not depend on random choices made during the execution of the game **S**; the type of a point is determined once the constants associated to the game are fixed. If we look at a pair of earlier/current points (X_i^r, X_j^s) each point will have some particular type—there are 42 pairs of types in all.

We now claim that for any current point X_j^s and any earlier point X_i^r, the probability that the values assigned to these two points are the same is at most 2^{-n}. This is verified by a case analysis, going over all 42 possibilities for the type of X_i^r and X_i^s. The case analysis is outlined in Figure 15. We add justification to three representative examples:

– *Case 2.* We are trying to bound $\Pr[\mathtt{M}_1^r \oplus L_{\mathtt{t}^r} \oplus 2L_0 = \mathtt{M}_1^s \oplus L_{\mathtt{t}^s} \oplus 2L_0] = \Pr[\mathtt{M}_1^r \oplus \mathtt{M}_1^s = L_{\mathtt{t}^r} \oplus L_{\mathtt{t}^s}]$ where $r < s$. Here $|M^r| < n$ and $|M^s| < n$. Subcase 2A: if $\mathtt{t}^r = \mathtt{t}^s$ then $M^r \neq M^s$ because of the constraint that adversary A was allowed to make no (t, M, s) query following an earlier (t, M, r) query, and so the indicated probability is 0. Subcase 2B: if $\mathtt{t}^r \neq \mathtt{t}^s$ then $L_{\mathtt{t}^r}$ and $L_{\mathtt{t}^s}$ are random and independent, and so $\Pr[\mathtt{M}_1^r \oplus \mathtt{M}_1^s = L_{\mathtt{t}^r} \oplus L_{\mathtt{t}^s}] = 2^{-n}$.

– *Case 9.* We are bounding $\Pr[\mathtt{M}_1^r \oplus L_{\mathtt{t}^r} \oplus 2L_0 = \widetilde{\mathtt{M}_1^s} \oplus L_{\mathtt{t}^s} \oplus 4L_0] = \Pr[\mathtt{M}_1^r \oplus \widetilde{\mathtt{M}_1^s} = L_{\mathtt{t}^r} \oplus L_{\mathtt{t}^s} \oplus 6L_0]$. If $\mathtt{t}^r = \mathtt{t}^s$ then this is $\Pr[\mathtt{M}_1^r \oplus \widetilde{\mathtt{M}_1^s} = 6L_0] = 2^{-n}$ because L_0 is random and independent of the left-hand side. If $\mathtt{t}^r = 0$ and $\mathtt{t}^s \neq 0$ then this is $\Pr[\mathtt{M}_1^r \oplus \widetilde{\mathtt{M}_1^s} \oplus L_{\mathtt{t}^s} = 7L_0] = 2^{-n}$ because L_0 is random and independent of the left-hand side. The case for $\mathtt{t}^r \neq 0$ and $\mathtt{t}^s = 0$ is the same way, as is the case for $\mathtt{t}^r \neq 0$ and $\mathtt{t}^s \neq 0$ and $\mathtt{t}^r \neq \mathtt{t}^s$.

– *Case 34.* This case arises for messages $\mathtt{M}^r$ and $\mathtt{M}^s$ having two or more blocks and both messages having a full final block. We want to bound $\Pr[Y[\mathtt{t}^r,\, \mathtt{M}_{1..\mathtt{m}^r-1}^r] \oplus \mathtt{M}_i^r \oplus 2L_0 = Y[\mathtt{t}^s,\, \mathtt{M}_{1..\mathtt{m}^s-1}^s] \oplus \mathtt{M}_j^s \oplus 2L_0]$ which is $\Pr[\mathtt{M}_i^r \oplus \mathtt{M}_j^s = Y[\mathtt{t}^r,\, \mathtt{M}_{1..\mathtt{m}^r-1}^r] \oplus Y[\mathtt{t}^s,\, \mathtt{M}_{1..\mathtt{m}^s-1}^s]]$. Observe that $Y[\mathtt{t}^r,\, \mathtt{M}_{1..\mathtt{m}^r-1}^r]$and $Y[\mathtt{t}^s,\, \mathtt{M}_{1..\mathtt{m}^s-1}^s]$ are random from $\{0,1\}^n$, being chosen from this set in an earlier execution of line 41 or line 31. If they are the identical random variable, that is, $\mathtt{t}^r = \mathtt{t}^s$ and $\mathtt{M}^r$ less its final block is identical to $\mathtt{M}^s$ less its final block, then $\Pr[\mathtt{M}_i^r \oplus \mathtt{M}_j^s = 0] = 0$ because there are no repeated queries. If they are different random variables then they are independent and $\Pr[\mathtt{M}_i^r \oplus \mathtt{M}_j^s = Y[\mathtt{t}^r,\, \mathtt{M}_{1..\mathtt{m}^r-1}^r] \oplus Y[\mathtt{t}^s,\, \mathtt{M}_{1..\mathtt{m}^s-1}^s]] = 2^{-n}$.

The justifications for the remaining 39 cases are analogous. We leave the reader to check the table, which is the technical heart of the proof.

We are now ready to conclude the proof. As the σ_1 current points X_j^s are considered the probability that the kth current point X_j^s collides with a given earlier one of the $k - 1 + \sigma_2 + 3$ earlier points X_i^r is at most $1/2^n$. Thus the probability that the kth current point coincides with some earlier point is at most $(k + \sigma_2 + 2)/2^n$. So the probability that some current point coincides with some earlier one is at most $\sum_{k=1}^{\sigma_1}(k + \sigma_2 + 2)/2^n = \sigma_1(\sigma_2 + 3)/2^n + \sum_{k=1}^{\sigma_1}(k-1)/2^n \le (\sigma_1\sigma_2 + 3\sigma_1 + 0.5\sigma_1^2)/2^n$. Combining with Equation (12) and the prior arguments we conclude that

$$
\begin{aligned}
\mathbf{Adv}^{\mathrm{dist}}_{\mathbb{OMAC}[\mathcal{R}^n_n],\$_n}(\sigma_1,\sigma_2) &\le \frac{\sigma_1\sigma_2 + 3\sigma_1 + 0.5\sigma_1^2}{2^n} + \frac{\sigma_2(\sigma_1+\sigma_2+3)}{2^n} \\
&= \frac{0.5\sigma_1^2 + 2\sigma_1\sigma_2 + \sigma_2^2 + 3\sigma_1 + 3\sigma_2}{2^n} \\
&\le \frac{(\sigma_1+\sigma_2+3)^2}{2^n}
\end{aligned}
$$

This completes the proof. ▮

E Proof of Security of EAX

Proof of Theorem 5: We begin with the privacy claim. Let A be an adversary using resources (q, σ) that is trying to distinguish EAX$[\mathcal{R}^n_n, \tau]$ from a source of random bits. We construct an adversary B that distinguishes $\mathbb{OMAC}[\mathcal{R}^n_n]$ from a source of random bits. Adversary B has an oracle g that responds to queries $(t, M, s) \in \{0,1,2\} \times \{0,1\}^* \times \mathbb{N}$ with a string $R\, S_0 S_1 \cdots S_{s-1}$, each named component an n-bit string. Adversary B works as follows:

Algorithm B^g
Run A
  When A makes an oracle call $(N_i,\ H_i,\ M_i)$, do the following:
    $s \leftarrow \lceil |M_i|/n \rceil$
    $\mathcal{N}_i\, S_0 \ldots S_{s-1} \leftarrow g(0,\ N_i,\ s)$
    $C_i \leftarrow M_i\ \oplus\ (S_0 \cdots S_{s-1}$ [first $|M_i|$ bits]$)$
    $\mathcal{H}_i \leftarrow g(1,\ H_i,\ 0)$
    $\mathcal{C}_i \leftarrow g(2,\ C_i,\ 0)$
    $T_i \leftarrow \mathcal{N}_i \oplus \mathcal{C}_i \oplus \mathcal{H}_i$ [first τ bits]
    Return, in response to A's query, $C_i \,\|\, T_i$
  When A halts, outputting a bit b, **return** b

We may assume that adversary A makes $q > 1$ queries since, otherwise, the result follows immediately. Then, under our conventions for the data complexity, adversary B uses resources at most $(2\sigma - 3, \sigma)$. Observe that $\Pr[A^{\mathrm{EAX}[\mathcal{R}^n_n,\tau]} = 1] = \Pr[B^{\mathbb{OMAC}[\mathcal{R}^n_n]} = 1]$. Also, since A is nonce respecting, B is length-committing and $\Pr[A^{\$} = 1] = \Pr[B^{\$_n} = 1]$. Using Lemma 4 we conclude that

$$
\begin{aligned}
\mathbf{Adv}^{\mathbf{priv}}_{\mathrm{EAX}[\mathcal{R}^n_n,\tau]}(A) &= \Pr[A^{\mathrm{EAX}[\mathcal{R}^n_n,\tau]} = 1] - \Pr[A^{\$} = 1] \\
&= \Pr[B^{\mathbb{OMAC}[\mathcal{R}^n_n]} = 1] - \Pr[B^{\$_n} = 1] \\
&\le \mathbf{Adv}^{\mathrm{dist}}_{\mathbb{OMAC}[\mathcal{R}^n_n,\$_n]}(2\sigma - 3,\ \sigma) \\
&\le \frac{(3\sigma)^2}{2^n} \\
&\le \frac{9\sigma^2}{2^n}
\end{aligned}
$$

This completes the privacy claim.

Moving on to authenticity and reusing the name, let A be an adversary for attacking the authenticity of $\text{EAX}[\mathcal{R}_n^n, \tau]$ that uses resources at most σ. Let

$$\begin{aligned}
\alpha_1 &= \mathbf{Adv}^{\mathbf{auth}}_{\text{EAX}[\mathcal{R}_n^n,\tau]}(A) \\
\alpha_2 &= \mathbf{Adv}^{\mathbf{auth}}_{\text{EAX2}[\text{CTR}[\mathcal{R}_n^n],\mathcal{R}_n^*,\tau]}(A) \\
\delta &= \alpha_1 - \alpha_2
\end{aligned}$$

By Lemma 2 and known results about the privacy of CTR (cf. [1]) we have

$$\begin{aligned}
\alpha_2 &\leq \frac{1}{2^\tau} + \mathbf{Adv}^{\text{priv}}_{\text{CTR}[\mathcal{R}_n^n]}(\sigma) \\
&\leq \frac{1}{2^\tau} + \frac{\sigma^2}{2^n} \, .
\end{aligned}$$

Hence

$$\alpha_1 \;=\; \alpha_2 + \delta \;\leq\; \delta + \frac{\sigma^2}{2^n} + \frac{1}{2^\tau} \, .$$

We now turn to bounding δ. To do this, reusing the name, we construct from A (the authenticity-attacking adversary) an adversary B (with an oracle for g and intended for distinguishing $\mathbb{OMAC}[\mathcal{R}_n^n]$ from a source of random bits):

Algorithm B^g
Run A
  When A makes an oracle call $(N_i,\ H_i,\ M_i)$, do the following:
    $s \leftarrow \lceil |M_i|/n \rceil$
    $\mathcal{N}_i\, S_0 \ldots S_{s-1} \leftarrow g(0,\ N_i, s)$
    $C_i \leftarrow M_i \;\oplus\; (S_0 \cdots S_{m-1}$ [first $|M_i|$ bits])
    $\mathcal{H}_i \leftarrow g(1,\ H_i,\ 0)$
    $\mathcal{C}_i \leftarrow g(2,\ C_i,\ 0)$
    $T_i \leftarrow \mathcal{N}_i \oplus \mathcal{C}_i \oplus \mathcal{H}_i$
    In response to A's query, return $C_i \parallel T_i$
  When A outputs a forgery attempt $(N,\ H,\ C \parallel T)$ and halts:
    $c \leftarrow \lceil |C|/n \rceil$
    $\mathcal{N} \leftarrow g(0,\ N,\ 0)$
    $\mathcal{H} \leftarrow g(1,\ H,\ 0)$
    $\mathcal{C} \leftarrow g(2,\ C,\ 0)$
    $T' \leftarrow \mathcal{N} \oplus \mathcal{C} \oplus \mathcal{H}$ [first τ bits]
    if $T = T'$ **and** $(N, H, C \parallel T) \neq (N_i, H_i, C_i \parallel T_i)$ for all i
      then return 1 **else return** 0

As before, one may assume that A makes $q > 1$ queries and, according to our conventions, the complexity of B will then be at most $(2\sigma - 3, \sigma)$. Also, $\alpha_1 = \mathbf{Adv}^{\mathbf{auth}}_{\text{EAX}[\mathcal{R}_n^n,\tau]}(A) = \Pr[B^{\mathbb{OMAC}[\mathcal{R}_n^n]} = 1]$.

Next, define the function $E[\rho, f] : \{0,1,2\} \times \{0,1\}^* \times \mathbb{N} \to \{0,1\}^*$ by

Algorithm $E[\rho, f]\ (t, M, s)$
  $R \leftarrow f([t]_n || M)$
  for $j \leftarrow 0$ **to** $s - 1$ **do** $S_j \leftarrow \rho(R + j)$
  return $R\, S_0 S_1 \cdots S_{s-1}$

Note that

$$\alpha_2 = \mathbf{Adv}^{\mathbf{auth}}_{\mathrm{EAX2}[\mathrm{CTR}[\mathcal{R}_n^n],\mathcal{R}_n^*,\tau]}(A) = \Pr[B^{E[\mathcal{R}_n^n,\mathcal{R}_n^*]} = 1].$$

Moreover,

$$\mathbf{Adv}^{\mathrm{dist}}_{\$_n,E[\mathcal{R}_n^n,\mathcal{R}_n^*]} \leq \frac{\sigma_2^2}{2^{n+1}}$$

for all adversaries that request a total of σ_2 keystreams, since $E[\mathcal{R}_n^n, \mathcal{R}_n^*]$ can only be distinguished from $\$_n$ if there is a collision in the inputs to ρ, and there are σ_2 inputs to ρ. As a trivial consequence,

$$\Pr[B^{E[\mathcal{R}_n^n,\mathcal{R}_n^*]} = 1] \geq \Pr[B^{\$_n} = 1] - \frac{\sigma^2}{2^{n+1}}$$

and thus

$$\alpha_2 = \mathbf{Adv}^{\mathbf{auth}}_{\mathrm{EAX2}[\mathrm{CTR}[\mathcal{R}_n^n],\mathcal{R}_n^*,\tau]}(A) \geq \Pr[B^{\$_n} = 1] - \frac{\sigma^2}{2^{n+1}}.$$

Also, adversary B is length-committing, since A is nonce-respecting (we use here the fact that the last three queries B makes all take the form $g(\cdot,\cdot,0)$, so those last three queries cannot violate the length-committing condition). So, using Lemma 4, we conclude that

$$\delta \;=\; \alpha_1 - \alpha_2 \;\leq\; \frac{(3\sigma)^2}{2^n} + \frac{\sigma^2}{2^{n+1}} \;\leq\; \frac{9.5\sigma^2}{2^n}$$

This completes the authenticity claim and the proof. ▮

F Recommended API

Some important features of EAX can only be utilized if one accesses EAX functionality through an appropriate user interface. In this section we therefore put forward an API that permits (a) incremental encryption, (b) incremental decryption, (c) authenticity verification without ciphertext recovery, and (d) static headers with negligible per-message cost. Providing of these features results in an API that is a bit more elaborate than some programmers may want or need, so we also include some simpler, "all-in-one" calls.

```
/*
 * We provide two interfaces:
 * 1. A simple interface that does not support streaming data.
 * 2. An incremental interface that supports streaming data.
 * See below for documentation on both.
 */

/*******************************************************************
 * -- How to encrypt, the simplified interface --
 * First, call
 *     eax_init()
 * to setup the key and set the parameters.
 * Then, for each packet, call
 *    eax_encrypt()
 * When all done, call
 *   eax_zeroize()
 *******************************************************************
 * -- How to decrypt, the simplified interface --
 * First, call
 *   eax_init()
```

```
 * to setup the key and set the parameters.
 * Then, for each packet:
 *   eax_decrypt()
 * When all done, call
 *   eax_zeroize()
 * It is the caller's responsibility to check tag validity
 * by examining the return value of eax_decrypt().
 *********************************************************************/


/*********************************************************************
 * -- How to encrypt, incrementally --
 * First, call
 *     eax_init()
 * to setup the key and set the parameters.
 * Then, for each packet, call
 *    eax_provide_nonce()
 *    {eax_provide_header(), eax_compute_ciphertext()}*
 *    eax_compute_tag()
 * Here {x,y} means x or y, and z* means any number of iterations of z.
 * When all done, call
 *   eax_zeroize()
 *
 * Note that encryption can be done on the fly, and header and message data
 * may be provided in any order and in arbitrary chunks.
 *********************************************************************
 * -- How to decrypt, incrementally --
 * First, call
 *   eax_init()
 * to setup the key and set the parameters.
 * Then, for each packet:
 *   eax_provide_nonce()
 *   {eax_provide_header(), eax_provide_ciphertext()}*
 *   eax_check_tag()
 *   eax_compute_plaintext()    // only do this if tag was valid
 * When all done, call
 *   eax_zeroize()
 * Note that decryption may be done on the fly, and header and message data
 * may be provided in any order and in arbitrary chunks.
 * It is the caller's responsibility to check tag validity
 * by examining the return value of eax_check_tag().
 *********************************************************************/

typedef enum {AES128,AES192,AES256} block_cipher; /* "standard" ciphers */
typedef unsigned char byte;
typedef void eax_state;                      /* EAX context; opaque */


/*********************************************************************
 *   Calls common to incremental and non-incremental API
 *********************************************************************/

/*
 * eax_init
```

```
 *
 * Key and parameter setup to init a EAX context data structure.
 * If you don’t know what to pass for t,E, use t=16, E=AES128.
 */
eax_state *
eax_init(
  byte* Key,        // The key, as a string.
  unsigned int t,   // The tag length, in bytes.
  block_cipher E    // Enumerated that indicates what cipher to use.
);

/*
 * eax_provide_header
 *
 * Supply a message header.  The header "grows" with each call
 * until a eax_provide_header() call is made that follows a
 * eax_encrypt(), eax_decrypt(), eax_provide_plaintext(),
 * eax_provide_ciphertext() or eax_compute_plaintext() call.
 * That starts reinitializes the header.
 */
int
eax_provide_header(
  eax_state *K,      // The EAX context.
  byte *H,           // The header (associated data) (possibly more to come)
  unsigned int h     // having h bytes
);

/*
 * eax_zeroize
 *
 * Session is over; destroy all key material and cleanup!
 */
void
eax_zeroize(
    eax_state *K       // The EAX context to remove
);


/*******************************************************************
 *  All-in-one, non-incremental interface
 ******************************************************************/

/*
 * eax_encrypt
 *
 * Encrypt the given message with the given key, nonce and header.
 * Specify the header (if nonempty) with eax_provide_header().
 */
int
eax_encrypt(
  eax_state *K,        // The caller provides the EAX context,
  byte* N,             // the nonce and
  unsigned int n,      // its length (in bytes), and
  byte* M,             // the plaintext and
```

```
  unsigned int m,     // its length (in bytes).
  byte* C,            // The m-byte ciphertext
  byte* T             // and the tag T are returned.
);


/*
 * eax_decrypt()
 *
 * Decrypt the given ciphertext with the given key, nonce and header.
 * Specify the header (if nonempty) with eax_provide_header().
 * Returns 1 for a valid ciphertext, 0 for an invalid ciphertext.
 */
int
eax_decrypt(
  eax_state *K,       // The caller provides the EAX context,
  byte* N,            // the nonce and
  unsigned int n,     // its length (in bytes), and
  byte* C,            // the ciphertext and
  unsigned int c,     // its length (in bytes), and the
  byte* T,            // tag.
  byte* P             // If valid, return the c-byte plaintext.
);


/*********************************************************************
 *      Incremental interface
 *********************************************************************/

/*
 * eax_provide_nonce
 *
 * Provide a nonce.  For encryption, do this before calling
 * eax_compute_ciphertext() and eax_compute_tag();
 * for decryption, do this before calling
 * eax_provide_ciphertext(), eax_check_tag, or eax_compute_plaintext().
 */
int
eax_provide_nonce(
  eax_state *K,    // The EAX context,
  byte* N,         // the nonce, and
  unsigned int n   // the length of the nonce (in bytes).
);


/*
 * eax_compute_ciphertext
 *
 * Encrypt a message or a part of a message.
 * The nonce needs already to have been
 * specified by a call to eax_provide_nonce().
 */
int
eax_compute_ciphertext(  // Encrypt (part of) a message
```

```
  eax_state *K,     // Given a EAX context K
  byte *M,          // and a message M  (possibly more to come)
  unsigned int m,   // having m bytes.
  byte *C           // Return a ciphertext body C also having m bytes.
);


/*
 * eax_compute_tag
 *
 * Message and header finished: compute the authentication tag that is a part
 * of the complete ciphertext.
 */
int
eax_compute_tag(
  eax_state *K,        // Given a EAX context
  byte *T              // compute the tag T for it.
);


/*
 * eax_provide_ciphertext
 *
 * Supply the ciphertext, or the next piece of ciphertext.
 * This is used to check for the subsequent authenticity check eax_check_tag().
 */
int
eax_provide_ciphertext(
  eax_state *K,        // Given a EAX context
  byte *C,             // and a ciphertext C (possibly more to come)
  unsigned int c       // having c bytes.
);


/*
 * eax_check_tag
 *
 * The nonce, ciphertext and header have all been fully provided; check if
 * they are valid for the given tag.
 * Returns 1 for a valid ciphertext, 0 for an invalid ciphertext
 * (in which case plaintext/ciphertext might be zeroized as well).
 */
int
eax_check_tag(
  eax_state *K,        // Given a EAX context and
  byte *T              // the tag that accompanied the ciphertext.
);


/*
 * eax_compute_plaintext
 *
 * Recover the plaintext from the provided ciphertext.
 * A call to eax_provide_nonce() needs to precede this call.
```

```
 * The caller is responsible for separately checking if the ciphertext is valid.
 * Normally this would be done before computing the plaintext with
 * eax_compute_plaintext().
 */
int
eax_compute_plaintext(
    eax_state *K,   // Given a EAX context
    byte *C,        // and a ciphertext C (possibly more to come)
    unsigned int c, // having c bytes,
    byte *M         // return the corresponding c bytes of plaintext.
);
```

G Test Vectors

The following EAX-AES128 test vectors were graciously provided by Jack Lloyd. They were later verified by Brian Gladman.

```
MSG:
KEY:      233952DEE4D5ED5F9B9C6D6FF80FF478
NONCE:    62EC67F9C3A4A407FCB2A8C49031A8B3
HEADER:   6BFB914FD07EAE6B
CIPHER:   E037830E8389F27B025A2D6527E79D01

MSG:      F7FB
KEY:      91945D3F4DCBEE0BF45EF52255F095A4
NONCE:    BECAF043B0A23D843194BA972C66DEBD
HEADER:   FA3BFD4806EB53FA
CIPHER:   19DD5C4C9331049D0BDAB0277408F67967E5

MSG:      1A47CB4933
KEY:      01F74AD64077F2E704C0F60ADA3DD523
NONCE:    70C3DB4F0D26368400A10ED05D2BFF5E
HEADER:   234A3463C1264AC6
CIPHER:   D851D5BAE03A59F238A23E39199DC9266626C40F80

MSG:      481C9E39B1
KEY:      D07CF6CBB7F313BDDE66B727AFD3C5E8
NONCE:    8408DFFF3C1A2B1292DC199E46B7D617
HEADER:   33CCE2EABFF5A79D
CIPHER:   632A9D131AD4C168A4225D8E1FF755939974A7BEDE

MSG:      40D0C07DA5E4
KEY:      35B6D0580005BBC12B0587124557D2C2
NONCE:    FDB6B06676EEDC5C61D74276E1F8E816
HEADER:   AEB96EAEBE2970E9
CIPHER:   071DFE16C675CB0677E536F73AFE6A14B74EE49844DD

MSG:      4DE3B35C3FC039245BD1FB7D
KEY:      BD8E6E11475E60B268784C38C62FEB22
NONCE:    6EAC5C93072D8E8513F750935E46DA1B
HEADER:   D4482D1CA78DCE0F
CIPHER:   835BB4F15D743E350E728414ABB8644FD6CCB86947C5E10590210A4F

MSG:      8B0A79306C9CE7ED99DAE4F87F8DD61636
KEY:      7C77D6E813BED5AC98BAA417477A2E7D
NONCE:    1A8C98DCD73D38393B2BF1569DEEFC19
HEADER:   65D2017990D62528
```

```
CIPHER:   02083E3979DA014812F59F11D52630DA30137327D10649B0AA6E1C181DB617D7F2

MSG:      1BDA122BCE8A8DBAF1877D962B8592DD2D56
KEY:      5FFF20CAFAB119CA2FC73549E20F5B0D
NONCE:    DDE59B97D722156D4D9AFF2BC7559826
HEADER:   54B9F04E6A09189A
CIPHER:   2EC47B2C4954A489AFC7BA4897EDCDAE8CC33B60450599BD02C96382902AEF7F832A

MSG:      6CF36720872B8513F6EAB1A8A44438D5EF11
KEY:      A4A4782BCFFD3EC5E7EF6D8C34A56123
NONCE:    B781FCF2F75FA5A8DE97A9CA48E522EC
HEADER:   899A175897561D7E
CIPHER:   0DE18FD0FDD91E7AF19F1D8EE8733938B1E8E7F6D2231618102FDB7FE55FF1991700

MSG:      CA40D7446E545FFAED3BD12A740A659FFBBB3CEAB7
KEY:      8395FCF1E95BEBD697BD010BC766AAC3
NONCE:    22E7ADD93CFC6393C57EC0B3C17D6B44
HEADER:   126735FCC320D25A
CIPHER:   CB8920F87A6C75CFF39627B56E3ED197C552D295A7CFC46AFC253B4652B1AF3795B124AB6E
```