Building Secure Cryptographic Transforms, or How to Encrypt and MAC

Tadayoshi Kohno* Adriana Palacio† John Black‡

August 28, 2003

Abstract

We describe several notions of "cryptographic transforms," symmetric schemes designed to meet a variety of privacy and authenticity goals. We consider goals, such as replay-avoidance and in-order packet delivery, that have not been fully addressed in previous works in this area. We then provide an analysis of possible ways to combine standard encryption and message authentication schemes in order to *provably* meet these goals. Our results further narrow the gap between the provable-security results from the theoretical community and the needs of developers who implement real systems.

Keywords: Applied cryptography, cryptographic transforms, authenticated encryption, privacy, authenticity, security proofs.

*Dept. of Computer Science & Engineering, University of California at San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. E-mail: `tkohno@cs.ucsd.edu`. URL: `http://www-cse.ucsd.edu/users/tkohno`. Supported by a National Defense Science and Engineering Graduate Fellowship.

†Dept. of Computer Science & Engineering, University of California at San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. E-mail: `apalacio@cs.ucsd.edu`. URL: `http://www-cse.ucsd.edu/users/apalacio`. Supported by a National Science Foundation Graduate Research Fellowship.

‡Computer Science 430 UCB, University of Colorado at Boulder, Boulder, Colorado 80309-0430, USA. E-mail: `jrblack@cs.colorado.edu`. URL: `http://www.cs.colorado.edu/~jrblack`. Supported by NSF CAREER award CCR-0133985 and start-up funds from the University of Colorado at Boulder.

1 Introduction

Symmetric cryptosystems are generally designed to protect both the privacy and the authenticity of transmitted data. The traditional approach for constructing such cryptosystems has been *ad hoc*, meaning without formal justification or proofs of security. Unfortunately, such *ad hoc* analyses are highly error-prone, as evidenced by the fact that some natural ways of combining standard (privacy-only) encryption schemes with standard message authentication schemes (MACs) are actually insecure (e.g., see [5, 18]). This raises the question: how can one construct symmetric cryptosystems that *provably* provide some form of privacy and authenticity?

Katz and Yung [16], Bellare and Namprempre [5], and Krawczyk [18] were the first to consider this question. They introduced formal notions of security for privacy- and authenticity-providing symmetric constructs (aka *unforgeable encryption* or *authenticated encryption* schemes). They then considered ways of constructing authenticated encryption schemes that provably met their notions of security.

While these and subsequent works took important steps to address the needs of those implementing cryptographic applications, there still remains a gap between what the theory community has proven and what implementors need. For example, the formal notions of security considered in these works do not capture security requirements that many developers have for the symmetric cryptographic portions of their applications, including resistance to replay or out-of-order delivery attacks. Also, in the case of the works that consider ways of combining standard encryption schemes with standard MACs [5, 18, 4, 20], there are a number of natural constructions that fall outside of the proposed models. These observations suggest that developers wishing to design new privacy- and authenticity-providing symmetric cryptosystems have to fall back on *ad hoc* analyses, prove the security of their constructions themselves, or design within the constraints of previous results.

We address these concerns as follows. First, we introduce new formal notions of security capturing common implementation goals. Then we perform an analysis of many natural ways to combine standard encryption schemes and MACs. Because of the generality of our results, we believe that they will be useful to many developers, who will no longer have to argue the security of their constructions themselves or work within the confines of previous provable-security results.

Modeling symmetric cryptosystems. We use the term *cryptographic transform (CT)* to refer to the portion of a cryptographic application that takes application data and turns it into an outgoing packet with the intent of protecting the privacy of a designated portion of the data, and the authenticity of all of the data. The difference between a CT and a more traditional *authenticated encryption* scheme is that the latter is essentially a low-level, application-independent cryptographic primitive, whereas a CT can be application-dependent. For example, an application's CT might preprocess data in some data- or application-dependent way. And a CT might try to enforce some security policies (e.g., replay detection) that are beyond the scope of authenticated encryption schemes.

Focusing on common design goals, we identify five classes of CTs. For four of them, we formalize new notions of security. The first type of CT is essentially an authenticated encryption scheme designed to authenticate more data than it encrypts; for this type we adopt a variant of the security notions in [20]. The second type is designed to protect against replay attacks. The third type is designed to protect against replay and re-ordering attacks. For these three types, packets are allowed to be dropped. The fourth and fifth types are designed to ensure that packets are accepted in exactly the order in which they were generated. For the fourth type, no packet should be accepted after detecting a forgery attempt. For the fifth type, acceptance of legitimate packets should not be affected by forgery attempts. A variant of the fourth type was considered in [4]. We use the labels Type 1–Type 5 to refer to these different types of CTs. Since we believe that the first four types will be the most useful in applications, we defer discussion of Type 5 CTs to the appendices.

BUILDING CRYPTOGRAPHIC TRANSFORMS. After defining the five types of CTs, we consider the problem of designing CTs that provably satisfy the corresponding notions of security. We focus on constructing CTs that use as their underlying building blocks standard encryption schemes and standard MACs.[1]

There are essentially three approaches (or paradigms) for constructing CTs from encryption schemes and MACs. Each approach begins by preprocessing the input in some possibly application-dependent way. Then the approach either (1) runs the encryption and MAC algorithms in parallel on the preprocessed data, (2) runs the MAC algorithm on the preprocessed data, and then runs the encryption algorithm on the preprocessed data and the output of the MAC, or (3) runs the encryption algorithm on the preprocessed data, and then runs the MAC algorithm on the preprocessed data and the output of the encryption algorithm.

The security of the CT depends in part on the initial preprocessing step. In order to be as general as possible, we adopt the approach of [6, 4] and view the preprocessing step as an *encoding* scheme. We specify security properties for these encoding schemes that, if met, guarantee that a transform built using them, in combination with secure encryption and MAC schemes, will provably meet one of our notions of a secure CT. By presenting our results in terms of the security properties of encoding schemes, and not for specific preprocessing algorithms, we give developers the freedom to implement the preprocessing step any way they want, as long as the properties we specify are satisfied.

Since we consider three approaches and five CT types, for a total of 15 combinations, it is impractical to summarize all our results here. Instead, we informally discuss an example that illustrates the generality of our provable-security results. Consider a CT that uses CBC mode as its underlying encryption scheme and UMAC as its underlying MAC. Let M be the payload message for the CT and let H be some fixed-length header or control information. The CT is designed to protect the privacy of M and the integrity of both M and H. It first generates a random CBC mode IV I and a UMAC nonce N. It MACs the message $I\|H\|M$, where $\|$ denotes concatenation, using the nonce N, to get some tag τ. Then it encrypts $M\|\tau$ in CBC mode, using the IV I, to get some intermediate value σ (we assume that $M\|\tau$ is a multiple of the underlying block cipher's block length). Finally, the CT outputs $N\|I\|H\|\sigma$. This message is sent to the receiver, who can recover M and H the natural way. The receiver rejects packets with MAC verification failures or with repeated nonce values. Assuming that the block cipher used in CBC mode is secure and that UMAC is secure, this CT will provably be a secure Type 2 CT. We remark that the provable-security of this CT does not follow from previous results.

HELPING DEVELOPERS. Since we address requirements and goals of real-world systems, and our analyses are performed in a very general way, we believe that our results will be particularly valuable to developers who want to design new (or analyze existing) CTs.

RELATED WORK. Katz and Yung [16] and Bellare and Namprempre [5] formalized the notion of an authenticated encryption scheme. The latter and Krawczyk [18] explored the three basic paradigms for creating such schemes: Encrypt-and-MAC (E&M), MAC-then-Encrypt (MtE), and Encrypt-then-MAC (EtM). The paradigms we consider, called Encode-then-{E&M, MtE, EtM}, are natural

[1]We note that it is also possible to design a CT that uses as its underlying component an authenticated encryption scheme. The main reason we do not consider CTs that are built this way is that, since currently all dedicated authenticated encryption modes are either covered by patents or have comparable software speeds to the combination of standard encryption schemes and standard MACs, and because of the flexibility gained with using standard encryption schemes and MACs as black boxes, a significant population of developers will likely use such CTs in their applications.

extensions of these paradigms, appropriately modified to use encoding schemes. Unfortunately, the research results in [5, 18] do not apply to many real-world CTs since many such CTs are not basic E&M, MtE, or EtM constructions.

Bellare, Kohno and Namprempre [4], noting that the SSH protocol was not one of the basic E&M, MtE, or EtM constructions, analyzed that protocol directly. They formalized a notion similar to, but slightly less general than, our notion of a Type 4 CT. The main difference is that our Type 4 notion specifically addresses the "associated-data problem" (see the next paragraph). The authors also analyzed a variant (again less general) of our Encode-then-E&M paradigm with respect to meeting this Type 4-like notion.

Rogaway [20] introduced the notion of authenticated-encryption with associated-data (AEAD) to address the problem that symmetric cryptosystems must often authenticate more data than they encrypt. Our notion of a Type 1 cryptographic transform essentially corresponds to the AEAD notion. Rogaway considered methods in which one can combine some privacy component with some MAC component to create an AEAD scheme. However, he discussed only two of the three basic approaches for combining these components, and only in the context of achieving the AEAD goal. Furthermore, he made more restrictive requirements on the underlying components than we do. For example, standard encryption schemes, which do not take nonces as input, cannot be used as Rogaway's underlying privacy component, and in some cases we (unlike Rogaway) allow the use of MACs that are not pseudorandom functions (e.g., traditional Carter-Wegman MACs).

The idea of using encodings to capture and model the important properties of some sub-component of a larger scheme comes from [6] and was also used in [4].

In [10] Dodis and An consider methods of constructing authenticated encryption schemes capable of encapsulating long messages from authenticated encryption schemes capable only of encapsulating short messages.

There is a parallel research program exploring the construction of authenticated encryption schemes directly from block ciphers, rather than from existing encryption schemes and MACs (e.g., [16, 12, 15, 21, 7, 17]). Another research program investigates the construction of authenticated encryption primitives (e.g., [11, 13, 1]).

Overview. Our models of cryptographic transforms are presented in Section 2. We discuss the underlying building blocks (encryption schemes and MACs) with which secure CTs can be built in Section 3. Section 4 describes our approach for generalizing the three basic methods for creating cryptographic transforms and, in particular, our use of encoding schemes. The three paradigms (Encode-then-{E&M, MtE, EtM}) and our results for each of them are discussed in Sections 5, 6, and 7, respectively. We present conclusions and discussion of future work in Section 8.

In order to conserve space, we defer some of our formal definitions and theorem statements to the appendices. The details we defer are not critical to the understanding of the body of this paper.

2 Cryptographic Transforms

A cryptographic transform (CT) takes a user's or application's (privacy-critical) payload data and some (non-private) associated data and transforms the input in such a way as to ensure the privacy of the payload data and the integrity[2] of both the payload data and the associated data. An example cryptographic transform is shown in Figure 1. Note that the CT itself may load payload data into packets, add sequence numbers, etc.

In order to ensure the correct interpretation of our results, we must first define what we mean (from an API perspective) by a cryptographic transform. Then we describe our security notions.

[2] We use the terms *integrity* and *authenticity* interchangeably.

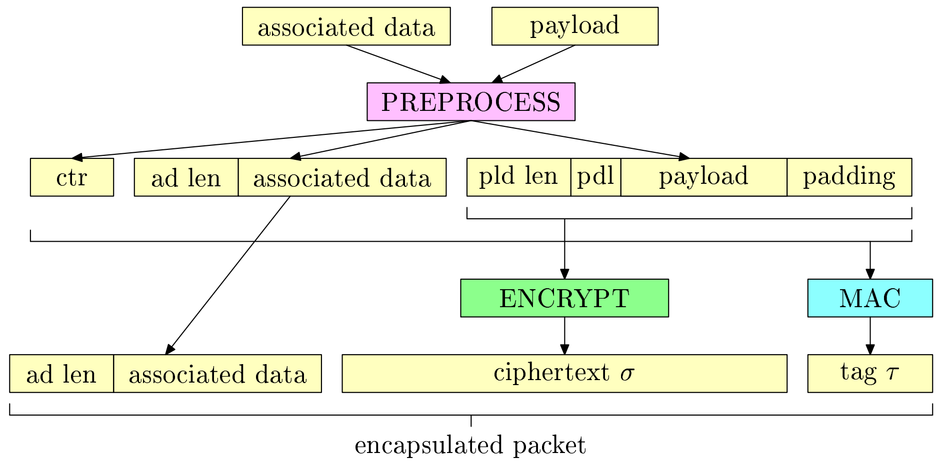

Figure 1: An example cryptographic transform (similar to the SSH CT but with associated data). Note the additional data added by the preprocessing step, the fact that the counter is not included in the encapsulated packet, and the fact that some data is MACed but not encrypted.

PRELIMINARIES. If x and y are strings, then $|x|$ denotes the length of x in *bits* and $x \| y$ denotes their concatenation. The empty string is denoted ε. If $a_1, \ldots, a_m$ are strings, then $\langle a_1, \ldots, a_m \rangle$ denotes an injective encoding from those strings into another string such that $a_1, \ldots, a_m$ are recoverable.

When we say an algorithm is stateful, we mean that it uses and updates its state and that the entity executing it maintains the state between invocations. Let the initial state of any (stateful or stateless) algorithm be ε. If f is a randomized (resp., deterministic) algorithm, then $x \stackrel{R}{\leftarrow} f(y)$ (resp., $x \leftarrow f(y)$) denotes the process of running f on input y and assigning the result to x.

CRYPTOGRAPHIC TRANSFORMS. A *cryptographic transform* $\mathsf{CT} = (\mathsf{KG}, \mathsf{Encap}, \mathsf{Decap})$ consists of three algorithms and is defined for some key space $\mathsf{KeySp}_{\mathsf{CT}}$, associated-data space $\mathsf{AdSp}_{\mathsf{CT}}$, and message space $\mathsf{MsgSp}_{\mathsf{CT}}$. The randomized key-generation algorithm KG returns a key $K \in \mathsf{KeySp}_{\mathsf{CT}}$ (for example, KG might return a random 128- or 256-bit string); we write this as $K \stackrel{R}{\leftarrow} \mathsf{KG}$. The possibly randomized and possibly stateful encapsulation algorithm Encap takes a key $K \in \mathsf{KeySp}_{\mathsf{CT}}$, associated data $M_a \in \mathsf{AdSp}_{\mathsf{CT}}$, and a message $M_s \in \mathsf{MsgSp}_{\mathsf{CT}}$, and outputs an encapsulated message $C \in \{0,1\}^*$; we write this as $C \stackrel{R}{\leftarrow} \mathsf{Encap}_K(M_a, M_s)$. We often refer to C as an *encapsulated packet* or a *ciphertext*. The deterministic and possibly stateful decapsulation algorithm Decap takes a key $K \in \mathsf{KeySp}_{\mathsf{CT}}$ and a message $C \in \{0,1\}^*$, and outputs a pair of messages $(M_a, M_s) \in \mathsf{AdSp}_{\mathsf{CT}} \times \mathsf{MsgSp}_{\mathsf{CT}}$ or the pair $(\perp, \perp)$ on error; we write this as $(M_a, M_s) \leftarrow \mathsf{Decap}_K(C)$. We require that if one of M_a or M_s is $\perp$, then both are $\perp$. We say that Decap_K *accepts* C if $\mathsf{Decap}_K(C) \neq (\perp, \perp)$; otherwise Decap_K *rejects* C. (We return $(\perp, \perp)$ on error instead of a single element $\perp$ because it makes our definitions easier to manipulate.)

We consider five classes of CTs. These types of CTs are designed to provide and run on top of different types of communication channels (e.g., reliable transport, unreliable transport). We shall describe four of them in detail shortly. Type 5 is discussed in the appendices.

SEPARATING FUNCTIONALITY AND SECURITY PROPERTIES. As is tradition in modern cryptography, we distinguish between the *functionality/consistency* requirements for CTs and their *security goals*. In particular, we call *any* object $\mathsf{CT} = (\mathsf{KG}, \mathsf{Encap}, \mathsf{Decap})$ that satisfies our consistency requirements a cryptographic transform. But we *only* call it a *secure* CT if it also satisfies our security requirements. We state the security goals first since in some cases the consistency requirements need only be met if an adversary has not already succeeded in breaching the security of the scheme.[3]

The five types of CTs have different integrity goals (and consistency requirements), but they all share the same privacy goal.[4] We first describe the notion of privacy for CTs. Then we make some general comments about our integrity notions for the five CT types. We then briefly discuss reaction and side-channel attacks. In the subsections that follow, we describe the first four types of CTs and define their integrity properties and consistency requirements. The relevant formal security definitions appear in Appendix B. Here we provide brief descriptions of the notions.

CHOSEN-PLAINTEXT PRIVACY. Our notion of privacy for CTs is based on the notion of left-or-right-indistinguishability under chosen-plaintext attacks [2]. Consider an adversary with access to an encapsulation oracle that on input associated data M_a and messages M_0, M_1 returns the encapsulation of M_a, M_b, where b is a hidden, randomly chosen bit. The adversary "wins" if it guesses bit b, i.e., if it guesses which sequence of messages was encapsulated. A CT is ct-priv-cpa-*secure* if the probability that an adversary with reasonable resources wins is close to 1/2 (i.e., if such an adversary cannot do much better than randomly guess bit b).

INTEGRITY OF CIPHERTEXTS AND CHOSEN-CIPHER-TEXT PRIVACY. The integrity notion for a Type n CT is called ct-int-ctxt$n$. These notions address the integrity of the *ciphertexts* generated by the encapsulation algorithm. This is different from protecting the integrity of the original inputs to the encapsulation method (cf. [5]). Indeed, the latter, in combination with the ct-priv-cpa notion, is insufficient to guarantee privacy under chosen-ciphertext attacks, whereas ct-int-ctxt$n$-security together with ct-priv-cpa-security imply a strong notion of privacy under chosen-ciphertext attacks that we call ct-priv-cca$n$-security. (These results are straightforward extensions of results in [5] for authenticated encryption schemes.) Since ct-priv-cpa and ct-int-ctxt$n$ imply ct-priv-cca$n$, we focus all our discussions on the former two notions.

REACTION AND SIDE-CHANNEL ATTACKS. Vaudenay [22] identified a class of attacks against cryptosystems whose decapsulation algorithms return different error codes, depending on how the decapsulation fails (e.g., the error code returned for bad padding is different than the error code returned for a failed MAC verification). To avoid these attacks, a cryptographic transform should always return the same error code upon failure, regardless of the reason for failure. Our constructions are secure against this type of attack because they always return the same error message, $(\perp, \perp)$.

Furthermore, to avoid Canvel's [9] timing-attack derivatives of [22], one should ensure that the length of time taken by the decapsulation routine does not depend on whether the decapsulation algorithm aborts prematurely. I.e., an adversary should not be able to learn the reason for a decapsulation algorithm's failure by observing the timing characteristics of the decapsulator.

2.1 Type 1 Cryptographic Transforms

For Type 1 CTs, a receiver (or decapsulator) will accept any encapsulated packet sent by the sender (or encapsulator), in any order, and possibly multiple times. A Type 1 CT is essentially an AEAD scheme [20].

INTEGRITY. The integrity notion for Type 1 CTs considers an adversary with chosen-plaintext access to an encapsulator and chosen-ciphertext access to the corresponding decapsulator. The adversary "wins" or "forges" if it can make the decapsulator accept a ciphertext not returned

[3]If an adversary forges a message, it may place the decapsulator in a state that it cannot recover from. Therefore, consistency can only be guaranteed in the absence of a successful adversary.

[4]We comment that this is natural since the differences between the various types of CTs become apparent only when one considers the decapsulation algorithm.

by the encapsulator. Informally, a Type 1 CT is $\mathsf{ct\text{-}int\text{-}ctxt1}$-*secure* if the probability that any adversary with reasonable resources wins is small.

Consistency requirements. For a Type 1 CT, $\mathsf{CT} = (\mathsf{KG}, \mathsf{Encap}, \mathsf{Decap})$, we require that $\mathsf{Decap}_K(\mathsf{Encap}_K(M_a, M_s)) = (M_a, M_s)$ for all messages M_a, M_s in CT's message spaces, all keys in the key space, and all internal states of the encapsulator and decapsulator.

2.2 Type 2 Cryptographic Transforms

Type 2 CTs are designed to protect against replay attacks.

Integrity. Consider an adversary with chosen-plain-text access to an encapsulator and chosen-ciphertext access to the corresponding decapsulator. The adversary "wins" or "forges" if it can make the decapsulator accept a ciphertext that the encapsulator did not generate, or make it accept the same ciphertext twice. Informally, a Type 2 CT is $\mathsf{ct\text{-}int\text{-}ctxt2}$-*secure* if the probability that any adversary with reasonable resources wins is small.

Consistency requirements. For a Type 2 CT, $\mathsf{CT} = (\mathsf{KG}, \mathsf{Encap}, \mathsf{Decap})$, we require that, for all messages M_a, M_s in CT's message spaces and all keys K in the key space, if $C = \mathsf{Encap}_K(M_a, M_s)$ for any internal state of the encapsulator, C has not already been submitted to Decap_K, and an adversary has not already succeeded in breaking the integrity of CT, then $\mathsf{Decap}_K(C) = (M_a, M_s)$.

We also make the following requirement: for any two message pairs (M_a^1, M_s^1), (M_a^2, M_s^2), if the encapsulator computes $C_1 \xleftarrow{R} \mathsf{Encap}_K(M_a^1, M_s^1)$ at some point in time and $C_2 \xleftarrow{R} \mathsf{Encap}_K(M_a^2, M_s^2)$ at some other time, it is the case that $C_1 \neq C_2$ (even if $(M_a^1, M_s^1) = (M_a^2, M_s^2)$). Otherwise, a legitimately encapsulated message might incorrectly be rejected by the receiver.

2.3 Type 3 Cryptographic Transforms

Type 3 CTs are designed to protect against replay attacks and re-ordering attacks, but are not intended to protect against packet loss.

Integrity. Consider an adversary with chosen-plain-text access to an encapsulator and chosen-ciphertext access to the corresponding decapsulator. The adversary "wins" or "forges" if it can make the decapsulator accept a ciphertext that the encapsulator did not generate, accept the same ciphertext twice, or accept a ciphertext that was generated before the last accepted ciphertext. For example, if C_i denotes the i-th ciphertext returned by the encapsulator, the adversary will win if it queries the decapsulator with C_5 followed by C_1 and the decapsulator accepts C_1. Informally, a Type 3 CT is $\mathsf{ct\text{-}int\text{-}ctxt3}$-*secure* if the probability that any adversary with reasonable resources wins is small.

Consistency requirements. For a Type 3 CT, $\mathsf{CT} = (\mathsf{KG}, \mathsf{Encap}, \mathsf{Decap})$, we require that, for all messages M_a, M_s in CT's message spaces and all keys K, if $C = \mathsf{Encap}_K(M_a, M_s)$ for any internal state of the encapsulator, C or a ciphertext generated after C has not already been submitted to Decap_K, and an adversary has not already succeeded in forging, then $\mathsf{Decap}_K(C) = (M_a, M_s)$.

We also make the following requirement: for any two message pairs (M_a^1, M_s^1), (M_a^2, M_s^2), if the encapsulator computes $C_1 \xleftarrow{R} \mathsf{Encap}_K(M_a^1, M_s^1)$ at some point in time and $C_2 \xleftarrow{R} \mathsf{Encap}_K(M_a^2, M_s^2)$ at some other point in time, it is the case that $C_1 \neq C_2$ (even if $(M_a^1, M_s^1) = (M_a^2, M_s^2)$).

2.4 Type 4 Cryptographic Transforms

Type 4 CTs are designed to ensure the in-order delivery of packets. If an adversary tries to forge, the forgery attempt should be detected and all future packets (even if generated by the legitimate encapsulator) should be rejected. Thus a Type 4 CT only has to work if *all* packets are delivered in order (i.e., no bad packet is injected into the communications stream).[5] This type of CT is designed to run on top of a reliable transport protocol like TCP. The notion of a Type 4 CT is closely related to the notion used in [4] to analyze the SSH cryptographic transform; the difference is that a Type 4 CT's encapsulation algorithm can take associated data as input.

INTEGRITY. Consider an adversary with chosen-plain-text access to an encapsulator and chosen-ciphertext access to the corresponding decapsulator. The integrity game for Type 4 CTs begins with a flag phase set to 0. If at any point the sequence of queries to the decapsulation oracle fails to be a prefix of the responses from the encapsulation oracle, phase is set to 1. An adversary wins if it can force the decapsulation oracle to accept a message after phase becomes 1. Informally, a Type 4 CT is $\mathsf{ct\text{-}int\text{-}ctxt4}$-*secure* if the probability that any adversary with reasonable resources wins is small.

CONSISTENCY REQUIREMENTS. Consider some sequence of message pairs $(M_a^1, M_s^1), (M_a^2, M_s^2), \ldots$ and, for $i = 1, 2, \ldots$, let $C_i = \mathsf{Encap}_K(M_a^i, M_s^i)$, starting with Encap_K in its initial state. Then if Decap_K is run on the sequence $C_1, C_2, \ldots$ in order and without the injection of additional packets, we require that $\mathsf{Decap}_K(C_i) = (M_a^i, M_s^i)$.

3 Building Blocks

Composition-based cryptographic transforms are built using two base cryptographic components: encryption schemes and MACs. We consider each of these components in turn.

3.1 Base encryption schemes

A *symmetric encryption scheme* $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ consists of three algorithms and is defined for some key space $\mathsf{KeySp}_{\mathcal{SE}}$, IV-space $\mathsf{IVSp}_{\mathcal{SE}}$, and message space $\mathsf{MsgSp}_{\mathcal{SE}}$. The randomized key-generation algorithm $\mathcal{K}$ returns a key $K \in \mathsf{KeySp}_{\mathcal{SE}}$; we write this as $K \xleftarrow{R} \mathcal{K}$. The possibly randomized and stateful encryption algorithm $\mathcal{E}$ takes a key $K \in \mathsf{KeySp}_{\mathcal{SE}}$, an IV $I \in \mathsf{IVSp}_{\mathcal{SE}}$, and a message $M \in \mathsf{MsgSp}_{\mathcal{SE}}$, and returns a ciphertext $C \in \{0,1\}^*$; we write this as $C \xleftarrow{R} \mathcal{E}_K^I(M)$. Example values for $\mathsf{IVSp}_{\mathcal{SE}}$ are $\{\varepsilon\}$ (when $\mathcal{SE}$ takes no IV) and $\{0,1\}^i$ for some positive integer i. The stateless and deterministic decryption algorithm $\mathcal{D}$ takes a key $K \in \mathsf{KeySp}_{\mathcal{SE}}$, an IV $I \in \mathsf{IVSp}_{\mathcal{SE}}$, and a ciphertext $C \in \{0,1\}^*$, and returns a message $M \in \{0,1\}^*$; we write this as $M \leftarrow \mathcal{D}_K^I(C)$. Note that the decrypted message M may be a string not in $\mathsf{MsgSp}_{\mathcal{SE}}$. The following consistency requirement must be met. $\mathcal{D}_K^I(\mathcal{E}_K^I(M)) = M$ for all $M \in \mathsf{MsgSp}_{\mathcal{SE}}$, $I \in \mathsf{IVSp}_{\mathcal{SE}}$, $K \in \mathsf{KeySp}_{\mathcal{SE}}$, and any internal state of $\mathcal{E}_K$.

Deviating from tradition, we consider three types of base encryption schemes: *nonced* encryption schemes, *length-based IV* encryption schemes, and *random IVed* encryption schemes. For a *nonced encryption scheme* we require that the encryption algorithm is always invoked with a new and distinct IV in $\mathsf{IVSp}_{\mathcal{SE}}$. For a *length-based IV* encryption scheme, we require that the first IV is randomly selected from $\mathsf{IVSp}_{\mathcal{SE}}$, and each subsequent IV is a deterministic function of the initial IV and the lengths of all previous plaintexts. We call this deterministic function the *length-based IV-deriving function* for the encryption scheme. (Our results could easily be extended for use with length-based encryption schemes where the first IV is some fixed constant, like the all zero block.)

[5]We remark that a Type 4 CT may be vulnerable to a DoS attack in which an adversary simply modifies one of the encapsulated packets. Type 5 CTs are similar to Type 4 CTs but are not vulnerable to such a DoS attack. Despite the DoS attack against Type 4 CTs, these CTs more closely match the design goals of a CT for use with a reliable transport protocol (as evidenced, for example, by the SSH protocol's use of a Type 4 CT).

For a *random IVed encryption scheme* we require that the encryption algorithm is always invoked with a randomly selected IV in $\mathsf{IVSp}_{\mathcal{SE}}$. If $\mathsf{IVSp}_{\mathcal{SE}} = \{\varepsilon\}$, then the random IV is always ε, and this is how we model standard encryption schemes, which do not take IVs as input.

Looking ahead, we note that we can enforce these requirements on the IVs through our use of *encodings*. The main reason we do not simply have the underlying encryption scheme in a CT generate its own IVs is that we want to be able to manipulate the IVs before invoking the encryption scheme (e.g., we want to be able to MAC the IV in a MAC-then-Encrypt-style CT).

Privacy. Our notion of privacy for symmetric encryption schemes is based on the notion of left-or-right-indistinguishability from [2] and is closely related to the ct-priv-cpa notion for CTs. Consider an adversary with access to an encryption oracle that on input an IV and a pair of messages, returns the encryption of either the first message or the second message, depending on a hidden random bit. The adversary "wins" if it guesses this bit, i.e., if it guesses which sequence of messages was encrypted. Informally, an encryption scheme is ind-cpa-*secure* if the probability that an adversary, using reasonable resources and respecting the IV properties of the scheme, wins is not much greater than 1/2. The formalization of this notion appears in Appendix B.

Example schemes. There are numerous examples of ind-cpa-secure encryption schemes. An example of a *nonced* encryption scheme is a CTR mode scheme which allocates part of the block cipher input to a nonce and the remainder to a block counter. An example of a *length-based IV* encryption scheme is a CTR mode variant that uses a random b-bit unsigned integer C as its initial counter (where b is the underlying block cipher's block size) and, after encrypting l blocks, uses the integer $C + l \bmod 2^b$ as the IV for the next message. An example of a *random IVed* encryption scheme is a CBC mode scheme that receives a random b-bit IV. Of course, a more traditional encryption scheme is a CBC mode instance that generates its own random b-bit IV (according to our notation, such a scheme would have IV space $\{\varepsilon\}$).

3.2 Message-authentication schemes

A *message-authentication scheme* or *MAC* $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ consists of three algorithms and is defined for some key space $\mathsf{KeySp}_{\mathcal{MA}}$, IV-space $\mathsf{IVSp}_{\mathcal{MA}}$, message space $\mathsf{MsgSp}_{\mathcal{MA}}$, and tag space $\mathsf{TagSp}_{\mathcal{MA}}$. The randomized key-generation algorithm returns a key $K \in \mathsf{KeySp}_{\mathcal{MA}}$; we write this as $K \xleftarrow{R} \mathcal{K}$. The tagging algorithm, which may be randomized and stateful, takes a key $K \in \mathsf{KeySp}_{\mathcal{MA}}$, an IV $I \in \mathsf{IVSp}_{\mathcal{MA}}$, and a message $M \in \mathsf{MsgSp}_{\mathcal{MA}}$, and returns a tag $\tau \in \mathsf{TagSp}_{\mathcal{MA}}$; we write this as $\tau \xleftarrow{R} \mathcal{T}_K^I(M)$. The deterministic and stateless verification algorithm takes a key $K \in \mathsf{KeySp}_{\mathcal{MA}}$, an IV $I \in \mathsf{IVSp}_{\mathcal{MA}}$, a message $M \in \mathsf{MsgSp}_{\mathcal{MA}}$, and a candidate tag $\tau \in \{0,1\}^*$, and returns a bit; we write this as $b \leftarrow \mathcal{V}_K^I(M, \tau)$. The following consistency requirement must be met. $\mathcal{V}_K^I(M, \mathcal{T}_K^I(M)) = 1$ for all $M \in \mathsf{MsgSp}_{\mathcal{MA}}$, $I \in \mathsf{IVSp}_{\mathcal{MA}}$, $K \in \mathsf{KeySp}_{\mathcal{MA}}$, and any internal state of $\mathcal{T}_K$.

As with base encryption schemes, we consider different types of MACs: *nonced* MACs and conventional MACs (i.e., MACs that do not take nonces as input). For a *nonced MAC* we require that the tagging algorithm is always invoked with a new and distinct IV in $\mathsf{IVSp}_{\mathcal{MA}}$. For a conventional MAC, $\mathsf{IVSp}_{\mathcal{MA}} = \{\varepsilon\}$. Explicitly taking a nonce as input is nice because it allows one to share the nonce between, for example, a Carter-Wegman MAC and CTR mode encryption. (Although we could also consider random IV or length-based IV MACs, we do not do so because, unlike with encryption schemes, we have no reason to manipulate such MAC IVs separately, and therefore allowing the caller to supply the random IV or length-based IV provides no clear advantage; the MAC can generate such IVs itself.)

Unforgeability of MACs. The main notion of security for MACs that we consider is strong unforgeability under chosen-message attacks [5]. This notion is described formally in Appendix B. Intuitively, we say that a MAC is uf-*secure* (or *unforgeable*) if the probability is small that any adversary using reasonable resources and respecting the IV properties of the MAC makes the verification algorithm accept some 3-tuple (I, M, τ) such that the tagging algorithm was never run on (I, M) or, if run on (I, M), never generated τ as the tag.

Pseudorandomness of MACs. Another notion of security for MACs is pseudorandomness. This notion only applies when $\mathsf{IVSp}_{\mathcal{MA}} = \{\varepsilon\}$ (or, phrased more appropriately, when the tagging algorithm is a function from $\mathsf{KeySp}_{\mathcal{MA}} \times \mathsf{MsgSp}_{\mathcal{MA}}$ to $\mathsf{TagSp}_{\mathcal{MA}}$). Essentially, a MAC is a *secure pseudorandom function* (*PRF*) if an adversary with chosen-plaintext access to a function f, mapping $\mathsf{MsgSp}_{\mathcal{MA}}$ to $\mathsf{TagSp}_{\mathcal{MA}}$, cannot tell whether the function is an instance of the MAC determined by a randomly selected key, or a randomly selected function from $\mathsf{MsgSp}_{\mathcal{MA}}$ to $\mathsf{TagSp}_{\mathcal{MA}}$. See Appendix B. As shown in [3], if a MAC is a secure PRF, then it is also uf-secure.

Privacy of MACs. The $\mathsf{ind\text{-}cpa}$ notion of privacy for symmetric encryption schemes can also be applied to MACs (see Appendix B). Although most popular MACs are not $\mathsf{ind\text{-}cpa}$-secure, some are (the notable example is Carter-Wegman MACs).

Example schemes. Popular examples of MACs include HMAC [19], OMAC [14], and UMAC [8]. The first two have IV-space $\{\varepsilon\}$ and the third takes a nonce as input. All these examples are uf-secure assuming the IV properties are respected. OMAC (and a number of other MACs) are also provably-secure PRFs, assuming that the underlying block cipher is secure. UMAC is $\mathsf{ind\text{-}cpa}$-secure against nonce-respecting adversaries.

4 The Three Paradigms

We recall the three basic methods to combine encryption schemes with MACs [5, 18]: Encrypt-and-MAC (E&M), MAC-then-Encrypt (MtE), and Encrypt-then-MAC (EtM). Let $\mathcal{E}_{K_e}$ be an encryption algorithm with key K_e, and $\mathcal{T}_{K_t}$ a MAC tagging algorithm with key K_t. The E&M encryption algorithm is defined as $\overline{\mathcal{E}}_{\langle K_e, K_t\rangle}(M) \stackrel{\text{def}}{=} \mathcal{E}_{K_e}(M) \| \mathcal{T}_{K_t}(M)$. The MtE encryption algorithm is defined as $\overline{\mathcal{E}}_{\langle K_e, K_t\rangle}(M) \stackrel{\text{def}}{=} \mathcal{E}_{K_e}(M \| \mathcal{T}_{K_t}(M))$. The EtM encryption algorithm is defined as $\overline{\mathcal{E}}_{\langle K_e, K_t\rangle}(M) \stackrel{\text{def}}{=} \sigma \| \mathcal{T}_{K_t}(\sigma)$, where $\sigma = \mathcal{E}_{K_e}(M)$.

In this work we consider generalizations of the three paradigms, which we call *Encode-then-E&M*, *Encode-then-MtE*, and *Encode-then-EtM*. For each of these paradigms, we consider the five types of cryptographic transforms. The *encodings* play a critical role in the Encode-then-{E&M, MtE, EtM} constructions. In particular, encodings allow us to formally model CTs that preprocess payload data *without* having to specify exactly how applications should do the preprocessing. Also, the encoding schemes are what provide the logic to, for example, detect replay attacks.

4.1 Encodings

An encoding scheme $\mathcal{EC}$ is an *un-keyed* public transformation that consists of four algorithms: Encode, $\mathsf{DecodeA}$, $\mathsf{DecodeB}$, and $\mathsf{DecodeC}$. All algorithms may be stateful and Encode may be randomized. The decoding algorithms $\mathsf{DecodeA}$, $\mathsf{DecodeB}$, and $\mathsf{DecodeC}$ may all share the same state. The specific properties of the algorithms depend on the paradigm in question and the type of CT that is being constructed. We describe them in detail in the following sections. Here we discuss some commonalities between the algorithms of encoding schemes for different paradigms and CT types.

Encoding and encapsulating. Algorithm Encode pre-processes a CT encapsulation algorithm's input messages M_a, M_s. Specifically, on input M_a, M_s, Encode outputs a 5-tuple $(M_p, M_o, M_n, M_e, M_t)$. Intuitively, M_p is cleartext data communicated with the ciphertext, M_o is the IV/nonce for use with the base encryption scheme, M_e is the input for the base encryption scheme, M_n is the IV/nonce for use with the base MAC, and M_t is the input for the base MAC.

The different paradigms then use these five strings in slightly different ways and slightly different orders. For Encode-then-E&M CTs, the encapsulation algorithm encrypts M_e with IV M_o to get a string σ, MACs M_t with IV M_n to get a tag τ, and outputs $\langle M_p, \sigma, \tau\rangle$. For Encode-then-MtE CTs, the encapsulation algorithm MACs M_t with IV M_n to get a tag τ, encrypts $\langle M_e, \tau\rangle$ with IV M_o to get a string σ, and outputs $\langle M_p, \sigma\rangle$. For Encode-then-EtM CTs, the encapsulation algorithm encrypts M_e with IV M_o to get a string σ, MACs $\langle M_t, \sigma\rangle$ with IV M_n to get a tag τ, and outputs $\langle M_p, \sigma, \tau\rangle$.

Decoding and decapsulating. The decoding algorithms $\mathsf{DecodeA}$, $\mathsf{DecodeB}$, and $\mathsf{DecodeC}$ are used in reversing the process. The decapsulation process typically involves first invoking $\mathsf{DecodeA}$ on M_p to get back (at least) M_o, the IV used with the underlying encryption scheme. In the case of Encode-then-EtM constructions, $\mathsf{DecodeA}$ returns the MAC IV M_n and M_t in order to allow for tag verification before decryption. After the underlying encryption scheme recovers the message M_e, the transforms invoke $\mathsf{DecodeB}(M_p, M_e)$ to recover (at least) M_a and M_s. If all goes well, then the transform's decapsulation algorithm returns M_a and M_s to the user or higher-level application.

However, all may not go well in the decapsulation process. For example, $\mathsf{DecodeA}$ or $\mathsf{DecodeB}$ may return the symbol $\perp$, indicating that there was a decoding failure. This can happen, for instance, in Type 2 decoding algorithms if the decoding algorithms detect a replayed message. When $\mathsf{DecodeA}$ or $\mathsf{DecodeB}$ return $\perp$, the decapsulation algorithm does not accept the packet.

It may also be the case that $\mathsf{DecodeA}$ and $\mathsf{DecodeB}$ do not detect any problems (and return strings instead of $\perp$) but the MAC tag verification fails. When this occurs, the decapsulation algorithm invokes $\mathsf{DecodeC}(\perp)$. If the tag verification succeeds, the decapsulation algorithm invokes $\mathsf{DecodeC}(\top)$. By calling $\mathsf{DecodeC}$ in this way, the decapsulation algorithm tells the decoding algorithms whether the packet was accepted. The decoding algorithms can then update their state. For example, for CTs designed to protect against out-of-order delivery attacks, it is prudent to increment the number of packets received *only* if the packet actually decapsulated correctly and passed the tag verification process.

Respecting the IV properties of $\mathcal{SE}$ and $\mathcal{MA}$. Consider the underlying encryption scheme $\mathcal{SE}$ and the underlying MAC $\mathcal{MA}$ that the Encode algorithm is combined with in an Encode-then-{E&M,MtE,EtM} construction. Note that these underlying schemes may have certain IV requirements in order for them to be secure. For example, $\mathcal{SE}$ might require that the IV is a nonce; i.e., that the IV never repeats, or that the IVs be random (or always the empty string ε). Consider any sequence of messages $(M_a^1, M_s^1), (M_a^2, M_s^2), \ldots$, let Encode begin in its initial state, and for $i = 1, 2, \ldots$ let $(M_p^i, M_o^i, M_n^i, M_e^i, M_t^i) = \mathsf{Encode}(M_a^i, M_s^i)$. We call an encoding scheme *nonce-respecting for encryption* if it is the case that $M_o^i \neq M_o^j$ for all distinct i, j. We call an encoding scheme *nonce-respecting for MACing* if $M_n^i \neq M_n^j$ for all distinct i, j. An encoding scheme is *length-based IV-respecting for encryption* with respect to some length-based IV-deriving function if the first M_o value the encoding scheme generates is chosen uniformly at random from $\mathsf{IVSp}_{\mathcal{SE}}$, and all subsequent M_o values are generated according to the length-based IV-deriving function, the initial M_o value, and the lengths of all previous M_e values. An encoding scheme is *random-IV-respecting for encryption* if the encoding algorithm always picks the value M_o uniformly at random from $\mathsf{IVSp}_{\mathcal{SE}}$.

Note that if the IV spaces are finite, then it is impossible to run a nonce-respecting encoding

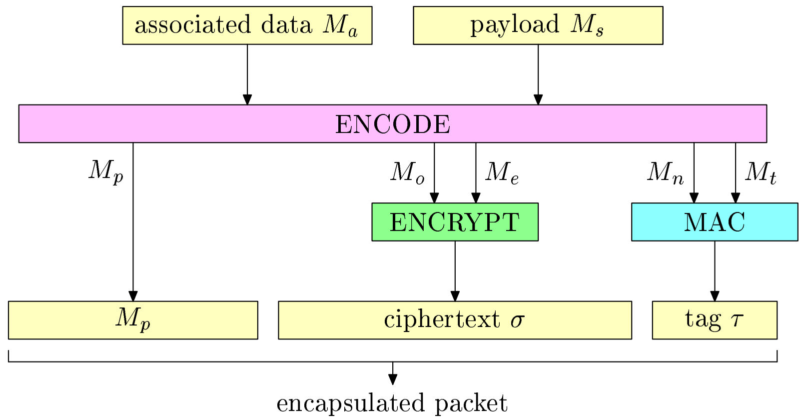

Figure 2: The Encode-then-E&M encapsulation method.

scheme on an infinite number of inputs. Therefore, we associate to any encoding scheme $\mathcal{EC}$ a parameter $\mathsf{MaxNum}_{\mathcal{EC}}$, and we assume that the encoding scheme is not invoked more than $\mathsf{MaxNum}_{\mathcal{EC}}$ times per application (i.e., beginning in its initial state, the encoding algorithm will not be asked to encode more than $\mathsf{MaxNum}_{\mathcal{EC}}$ pairs of messages). In the above discussion and in the following sections, whenever we write "for $i = 1, 2, \ldots$, run Encode," we assume that the iterations stop before i gets larger than $\mathsf{MaxNum}_{\mathcal{EC}}$. (We use the same convention when discussing CTs built from $\mathcal{EC}$.)

5 Encode-then-E&M

We first focus on Encode-then-E&M cryptographic transforms. The encapsulation algorithm of such a CT works as shown in Figure 2. An *E&M encoding scheme* is used to "glue" together the encryption and MAC components of an Encode-then-E&M CT. For an E&M encoding scheme $\mathcal{EC}^{\text{E\&M}} = (\mathsf{Encode}, \mathsf{DecodeA}, \mathsf{DecodeB}, \mathsf{DecodeC})$, Encode behaves as described in Section 4.1. $\mathsf{DecodeA}$, on input a string M_p, outputs a string M_o, or $\perp$ on error. $\mathsf{DecodeB}$, on input two messages M_p, M_e, returns a 4-tuple of messages (M_a, M_s, M_n, M_t), or $(\perp, \perp, \perp, \perp)$ on error (if any one of M_a, M_s, M_n, or M_t is $\perp$, then all of them are $\perp$). $\mathsf{DecodeC}$ takes as input the symbol $\top$ or the symbol $\perp$ and returns nothing.

An encryption scheme, a MAC, and an appropriate E&M encoding scheme can be combined to obtain an Encode-then-E&M CT as follows.

Construction 5.1 (Encode-then-E&M) Let $\mathcal{EC}^{\text{E\&M}} = (\mathsf{Encode}, \mathsf{DecodeA}, \mathsf{DecodeB}, \mathsf{DecodeC})$, $\mathcal{SE} = (\mathcal{K}_e, \mathcal{E}, \mathcal{D})$, and $\mathcal{MA} = (\mathcal{K}_t, \mathcal{T}, \mathcal{V})$ be E&M encoding, encryption, and message-authentication schemes, respectively, with compatible message spaces (e.g., the outputs from Encode are suitable inputs to $\mathcal{E}$ and $\mathcal{T}$). Let all states initially be ε. We associate to these schemes an *Encode-then-E&M cryptographic transform* $\mathsf{CT} = (\mathsf{KG}, \mathsf{Encap}, \mathsf{Decap})$ whose constituent algorithms are defined as follows:

Algorithm KG
$K_e \xleftarrow{R} \mathcal{K}_e$; $K_t \xleftarrow{R} \mathcal{K}_t$
Return $\langle K_e, K_t \rangle$

Algorithm $\mathsf{Encap}_{\langle K_e,K_t\rangle}(M_a, M_s)$
$(M_p, M_o, M_n, M_e, M_t) \xleftarrow{R} \mathsf{Encode}(M_a, M_s)$
$\sigma \xleftarrow{R} \mathcal{E}_{K_e}^{M_o}(M_e)$; $\tau \xleftarrow{R} \mathcal{T}_{K_t}^{M_n}(M_t)$
Return $\langle M_p, \sigma, \tau \rangle$

Algorithm $\mathsf{Decap}_{\langle K_e,K_t\rangle}(C)$
If $st = \perp$ then return $(\perp, \perp)$
If there does not exist M_p, σ, τ s.t. $C = \langle M_p, \sigma, \tau \rangle$ then
$st \leftarrow \boxed{\text{Box}}$; return $(\perp, \perp)$
Parse C as $\langle M_p, \sigma, \tau \rangle$; $M_o \leftarrow \mathsf{DecodeA}(M_p)$
If $M_o = \perp$ then $st \leftarrow \boxed{\text{Box}}$; return $(\perp, \perp)$
$M_e \leftarrow \mathcal{D}_{K_e}^{M_o}(\sigma)$
$(M_a, M_s, M_n, M_t) \leftarrow \mathsf{DecodeB}(M_p, M_e)$
If $M_s = \perp$ then $st \leftarrow \boxed{\text{Box}}$; return $(\perp, \perp)$
$v \leftarrow \mathcal{V}_{K_t}^{M_n}(M_t, \tau)$
If $v = 0$ then $st \leftarrow \boxed{\text{Box}}$; $\mathsf{DecodeC}(\perp)$; return $(\perp, \perp)$
$\mathsf{DecodeC}(\top)$
Return (M_a, M_s)

For a Type 4 CT, each boxed portion of the decapsulator should be $\perp$. For all other types, the boxed portion should be st. Recall that $\langle a_1, \ldots, a_m \rangle$ denotes an encoding of the strings $a_1, \ldots, a_m$ such that $a_1, \ldots, a_m$ are recoverable. For the call to $\mathsf{DecodeB}(M_p, M_e)$, recall that if any one of M_a, M_s, M_n, M_t is $\perp$, then they are all $\perp$. Although only Decap explicitly maintains state in the above pseudocode, the underlying encoding, encryption, and MAC schemes may also maintain state. E.g., the underlying encoding and decoding algorithms may maintain state in order to protect against replay attacks. ▌

Consistency requirements for E&M encoding schemes. Consider any two pairs of messages (M_a, M_s), (M_a, M_s') with $|M_s| = |M_s'|$. Let $(M_p, M_o, M_n, M_e, M_t) \xleftarrow{R} \mathsf{Encode}(M_a, M_s)$ for Encode in some state, and $(M_p', M_o', M_n', M_e', M_t') \xleftarrow{R} \mathsf{Encode}(M_a, M_s')$ for Encode in some (possibly different) state. We require that $|M_e| = |M_e'|$ and $|M_t| = |M_t'|$. If this were not the case, Construction 5.1 might not preserve privacy.

Consider also any two sequences of message pairs $(M_a^1, M_s^1), (M_a^2, M_s^2), \ldots$ and $(N_a^1, N_s^1), (N_a^2, N_s^2), \ldots$. Let Encode begin in its initial state and for $i = 1, 2, \ldots$ let $(M_p^i, M_o^i, M_n^i, M_e^i, M_t^i) = \mathsf{Encode}(M_a^i, M_s^i)$. Similarly, let Encode begin in its initial state and for $i = 1, 2, \ldots$ let $(N_p^i, N_o^i, N_n^i, N_e^i, N_t^i) = \mathsf{Encode}(N_a^i, N_s^i)$. If Encode is randomized, assume that both sequences are generated using the same random tape. Further assume that the randomness used in each invocation is recoverable from the output and that the amount of randomness used per invocation depends only on the lengths of the inputs. Consider any index i. If $|M_s^j| = |N_s^j|$ and $M_a^j = N_a^j$ for all $j \leq i$, then we require that $M_p^i = N_p^i$, $M_o^i = N_o^i$, and $M_n^i = N_n^i$.

Let $(M_a^1, M_s^1), (M_a^2, M_s^2), \ldots$ be a sequence of message pairs and, beginning with Encode in its initial state, let $(M_p^i, M_o^i, M_n^i, M_e^i, M_t^i) = \mathsf{Encode}(M_a^i, M_s^i)$ for $i = 1, 2, \ldots$ up to $\mathsf{MaxNum}_{\mathcal{EC}^{\mathrm{E\&M}}}$. We make the following additional consistency requirements on $\mathcal{EC}^{\mathrm{E\&M}}$, depending on the type of CT in question. In what follows we use the notation $\mathsf{Decode[ABC]}$ to denote any one of the decoding algorithms.

Type 1. For any i and for any state of the decoder, we require that $\mathsf{DecodeA}(M_p^i) = M_o^i$ and $\mathsf{DecodeB}(M_p^i, M_e^i) = (M_a^i, M_s^i, M_n^i, M_t^i)$.

Type 2. For any distinct indices i, j, we require that $(M_p^i, M_e^i) \neq (M_p^j, M_e^j)$.

For any i, we require that for any state of the decoder, $\mathsf{DecodeA}(M_p^i) = M_o^i$. Furthermore, if $\mathsf{DecodeB}$ has not been invoked with (M_p^i, M_e^i) or if $\mathsf{DecodeB}$ has been invoked with (M_p^i, M_e^i) but for each such invocation the next call to $\mathsf{Decode[ABC]}$ was $\mathsf{DecodeC}(\perp)$, then it must be the case that $\mathsf{DecodeB}(M_p^i, M_e^i) = (M_a^i, M_s^i, M_n^i, M_t^i)$.

Type 3. For any distinct indices i, j, we require that $(M_p^i, M_e^i) \neq (M_p^j, M_e^j)$.

For any i, we require that for any state of the decoder, $\mathsf{DecodeA}(M_p^i) = M_o^i$. Furthermore, if DecodeB has not been invoked with (M_p^j, M_e^j) for any $j \geq i$, or if DecodeB has been invoked with (M_p^j, M_e^j), for some $j \geq i$, but for each such invocation the next call to Decode[ABC] was $\mathsf{DecodeC}(\perp)$, then $\mathsf{DecodeB}(M_p^i, M_e^i) = (M_a^i, M_s^i, M_n^i, M_t^i)$.

TYPE 4. For $i = 1, 2, \ldots$ and the decoder beginning in its initial state, let $m_o^i = \mathsf{DecodeA}(M_p^i)$ and $(m_a^i, m_s^i, m_n^i, m_t^i) = \mathsf{DecodeB}(M_p^i, M_e^i)$. We require that $M_a^i = m_a^i$, $M_s^i = m_s^i$, $M_o^i = m_o^i$, $M_n^i = m_n^i$, and $M_t^i = m_t^i$ for all i.

SECURITY REQUIREMENTS FOR E&M ENCODING SCHEMES. The security requirements for E&M encoding schemes are formalized in Appendix C. For all types of CTs we define a property, called e&m-coll-security, that measures the probability of a collision in the M_n, M_t outputs of the encoding scheme. Consider a sequence of inputs $(M_a^1, M_s^1), (M_a^2, M_s^2), \ldots$ to Encode and, beginning with Encode in its initial state, for $i = 1, 2, \ldots$ let $(M_p^i, M_o^i, M_n^i, M_e^i, M_t^i) = \mathsf{Encode}(M_a^i, M_s^i)$. Intuitively, we say that the encoding scheme is e&m-coll-secure if the probability that $(M_n^i, M_t^i) = (M_n^j, M_t^j)$ for distinct indices i, j is small. We note that it is very easy to design an E&M encoding scheme that is e&m-coll-secure: simply include a counter or some random string in one or both of M_n or M_t.

For Type n E&M encoding schemes (i.e., E&M encoding schemes used to construct Type n CTs) we also define a security property called e&m-sec$n$. We distill the important aspects of these security properties here. Essentially, in order for Type 1–Type 3 E&M encoding schemes to be e&m-sec1–e&m-sec3-secure, it should be the case that if (M_p, M_e) and (M_p', M_e') are distinct pairs of strings, then they do not decode (via DecodeB) to identical M_n, M_t strings. For Type 2 E&M encoding schemes it should also be the case that if $\mathsf{DecodeB}(M_p, M_e)$ is called followed by a call $\mathsf{DecodeC}(\top)$, then the next time $\mathsf{DecodeB}(M_p, M_e)$ is called, DecodeB returns $(\perp, \perp, \perp, \perp)$. For Type 3 E&M encoding schemes it should also be the case that if M_p, M_e were in the output of one invocation of Encode, M_p', M_e' were in the output of some later Encode invocation, and $\mathsf{DecodeB}(M_p', M_e')$ is called followed by a call $\mathsf{DecodeC}(\top)$, then a later call $\mathsf{DecodeB}(M_p, M_e)$ returns $(\perp, \perp, \perp, \perp)$.

Consider some interaction with the encoding and decoding algorithms. Let $(M_p^i, M_o^i, M_n^i, M_e^i, M_t^i)$ denote the 5-tuple returned by Encode after its i-th invocation. Let (m_p^j, m_e^j) denote the parameters to the j-th call to DecodeB and let $(m_a^j, m_s^j, m_n^j, m_t^j)$ denote the response. Then for Type 4 encoding schemes it should be the case that $(M_n^i, M_t^i) \neq (m_n^j, m_t^j)$ for all $i \neq j$. And, if $(M_p^j, M_e^j) \neq (m_p^j, m_e^j)$, then it should be the case that $(M_n^j, M_t^j) \neq (m_n^j, m_t^j)$.

5.1 Summary of results

CHOSEN-PLAINTEXT PRIVACY. We are now in a position to describe how to combine a standard encryption scheme with a MAC in an Encrypt-and-MAC fashion in order to yield a CT that preserves privacy under chosen-plaintext attacks. The following summary distills the important properties from Theorem D.1.

Result 5.2 (Privacy of Encode-then-E&M) To construct a Type n Encode-then-E&M scheme CT from an encryption scheme $\mathcal{SE}$ and a MAC $\mathcal{MA}$, one should use a Type n E&M encoding scheme $\mathcal{EC}$ that is e&m-coll-secure and that respects the IV requirements of $\mathcal{SE}$ and $\mathcal{MA}$. If $\mathcal{SE}$ is a secure encryption scheme (ind-cpa-secure), $\mathcal{MA}$ is a secure PRF or privacy preserving (ind-cpa-secure), and all the components satisfy their respective consistency requirements, then CT will be a cryptographic transform that provably provides privacy under chosen-plaintext attacks (i.e., CT will be ct-priv-cpa-secure). ▮

The statement in Theorem D.1 is actually more general than Result 5.2. In particular, the theorem implies that if $\mathcal{MA}$ is ind-cpa-secure, then the encoding scheme need not be e&m-coll-secure. We have chosen to formulate the result as we did because most popular MACs are not ind-cpa-secure, and those that are require a nonce and hence any encoding scheme that respects the IV requirements of the MAC is trivially e&m-coll-secure.

We point out that developers should have no trouble finding secure building blocks. For example, many popular MACs are either proven to be or believed to be secure PRFs. And there are well-known encryption schemes that are provably ind-cpa-secure. (For further discussions of the building blocks, see Section 3.)

As noted above, it is very easy to create encoding schemes that are e&m-coll-secure (for example, the encoding scheme can simply append a counter to the input to the MAC). Looking ahead, we comment that in order to achieve some of our other goals (like resistance to replay attacks), we will have to include counters in the input to the MAC anyway, so requiring such counters for the e&m-coll property does not introduce additional overhead or costs for the CT.

Integrity. We now consider how to design Encode-then-E&M CTs that provably meet the CT integrity goals. The following interprets the results in Theorem D.4.

Result 5.3 (Integrity of Encode-then-E&M) To construct a Type n Encode-then-E&M scheme CT from an encryption scheme $\mathcal{SE}$ and a MAC $\mathcal{MA}$, one should use a Type n E&M encoding scheme $\mathcal{EC}$ that is e&m-sec$n$-secure and that respects the IV requirements of $\mathcal{MA}$. If the $\mathcal{SE}$ encryption algorithm is length-preserving, $\mathcal{MA}$ is unforgeable (uf-secure), and all the components satisfy their respective consistency requirements, then CT will be a cryptographic transform that provably meets the ct-int-ctxt$n$ integrity notion. ▮

It is not hard to find underlying components that satisfy the properties described in Result 5.3.

As with Result 5.2, we comment that the results in Theorem D.4 are more general than Result 5.3. In particular, it is possible for a Type n CT to be ct-int-ctxt$n$-secure even if the underlying encryption algorithm is not length-preserving (see Appendix D for details). However, unless one formally verifies that it is safe to use a specific non-length preserving base encryption scheme, one should closely follow the recommendation for using length-preserving encryption schemes. To see the importance of this, we note that [4] shows that, in the context of SSH, if the underlying encryption scheme is standard CBC mode (which generates the random IV itself and is therefore not length-preserving), then there is an attack on the integrity of the transform. Also, if the underlying encryption scheme is a CTR mode variant that maintains the counter itself (i.e., that doesn't take an IV as input) and includes that counter in the ciphertext, then an attacker with known-plaintext access to the encapsulator can learn the keystream value generated by each initial counter and, since the counter is not included in the input to the MAC, attack the integrity of the ciphertexts.

We believe that our length-preserving restriction on the encryption algorithm will not be a major concern for many developers since many of them will want to avoid the extra packet expansion that comes with using non-length-preserving encryption schemes anyway.

6 Encode-then-MtE

We now turn our attention to the Encode-then-MtE paradigm for CTs. The algorithms that constitute an *MtE encoding scheme* $\mathcal{EC}^{\mathrm{MtE}} = (\mathsf{Encode}, \mathsf{DecodeA}, \mathsf{DecodeB}, \mathsf{DecodeC})$, have the same APIs as those in an E&M encoding scheme.

An encryption scheme, a MAC, and an appropriate MtE encoding scheme can be combined to obtain an Encode-then-MtE CT as follows (see also Figure 3).

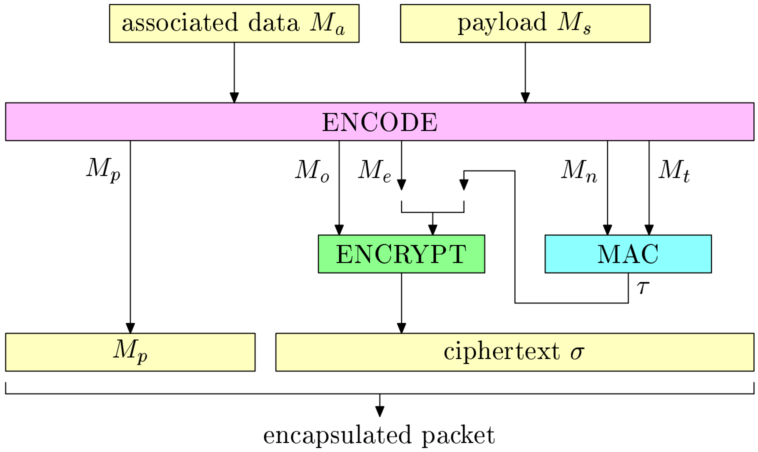

Figure 3: The Encode-then-MtE encapsulation method.

Construction 6.1 (Encode-then-MtE) Let $\mathcal{EC}^{\rm MtE} = (\mathsf{Encode}, \mathsf{DecodeA}, \mathsf{DecodeB}, \mathsf{DecodeC})$, let $\mathcal{SE} = (\mathcal{K}_e, \mathcal{E}, \mathcal{D})$, and let $\mathcal{MA} = (\mathcal{K}_t, \mathcal{T}, \mathcal{V})$ respectively be MtE encoding, encryption, and message-authentication schemes with compatible message spaces (e.g., the outputs from Encode are suitable inputs to $\mathcal{E}$ and $\mathcal{T}$). Assume that $\mathcal{T}$ always produces tags of the same length. Let all states initially be ε. We associate to these schemes an *Encode-then-MtE cryptographic transform* $\mathsf{CT} = (\mathsf{KG}, \mathsf{Encap}, \mathsf{Decap})$ whose constituent algorithms are defined as follows:

Algorithm KG
  $K_e \xleftarrow{R} \mathcal{K}_e$; $K_t \xleftarrow{R} \mathcal{K}_t$
  Return $\langle K_e, K_t \rangle$

Algorithm $\mathsf{Encap}_{\langle K_e, K_t\rangle}(M_a, M_s)$
  $(M_p, M_o, M_n, M_e, M_t) \xleftarrow{R} \mathsf{Encode}(M_a, M_s)$
  $\tau \xleftarrow{R} \mathcal{T}^{M_n}_{K_t}(M_t)$; $\sigma \xleftarrow{R} \mathcal{E}^{M_o}_{K_e}(\langle M_e, \tau\rangle)$
  Return $\langle M_p, \sigma\rangle$

Algorithm $\mathsf{Decap}_{\langle K_e, K_t\rangle}(C)$
  If $st = \perp$ then return $(\perp, \perp)$
  If there does not exist M_p, σ s.t. $C = \langle M_p, \sigma\rangle$ then
    $st \leftarrow \boxed{\text{Box}}$; return $(\perp, \perp)$
  Parse C as $\langle M_p, \sigma\rangle$; $M_o \leftarrow \mathsf{DecodeA}(M_p)$
  If $M_o = \perp$ then $st \leftarrow \boxed{\text{Box}}$; return $(\perp, \perp)$
  $M \leftarrow \mathcal{D}^{M_o}_{K_e}(\sigma)$
  If there does not exist M_e, τ s.t. $M = \langle M_e, \tau\rangle$ then
    $st \leftarrow \boxed{\text{Box}}$; $\mathsf{DecodeC}(\perp)$; return $(\perp, \perp)$
  Parse M as $\langle M_e, \tau\rangle$
  $(M_a, M_s, M_n, M_t) \leftarrow \mathsf{DecodeB}(M_p, M_e)$
  If $M_s = \perp$ then $st \leftarrow \boxed{\text{Box}}$; return $(\perp, \perp)$
  $v \leftarrow \mathcal{V}^{M_n}_{K_t}(M_t, \tau)$
  If $v = 0$ then $st \leftarrow \boxed{\text{Box}}$; $\mathsf{DecodeC}(\perp)$; return $(\perp, \perp)$
  $\mathsf{DecodeC}(\top)$
  Return (M_a, M_s)

For a Type 4 CT, each boxed portion of the decapsulator should be $\perp$. For all other types, the boxed portion should be st. For the call to $\mathsf{DecodeB}(M_p, M_e)$, recall that if any one of M_a, M_s, M_n, M_t is $\perp$, then they are all $\perp$. Although only Decap explicitly maintains state in the above pseudocode, the underlying encoding, encryption, and MAC schemes may also maintain state. We require that the length of the combined string $\langle M_e, \tau\rangle$ depend only on the lengths of M_e and τ. ▮

Consistency requirements for MtE encoding schemes. Consider any two pairs of messages (M_a, M_s), (M_a, M_s'), where $|M_s| = |M_s'|$. Let $(M_p, M_o, M_n, M_e, M_t) \xleftarrow{R} \mathsf{Encode}(M_a, M_s)$ for Encode in some state, and $(M_p', M_o', M_n', M_e', M_t') \xleftarrow{R} \mathsf{Encode}(M_a, M_s')$ for Encode is in some (possibly different) state. We require that $|M_e| = |M_e'|$. Consider also any two sequences of message pairs $(M_a^1, M_s^1), (M_a^2, M_s^2), \ldots$ and $(N_a^1, N_s^1), (N_a^2, N_s^2), \ldots$. Let Encode begin in its initial state and for

$i = 1, 2, \ldots$ let $(M_p^i, M_o^i, M_n^i, M_e^i, M_t^i) = \mathsf{Encode}(M_a^i, M_s^i)$. Similarly, let Encode begin in its initial state and for $i = 1, 2, \ldots$ let $(N_p^i, N_o^i, N_n^i, N_e^i, N_t^i) = \mathsf{Encode}(N_a^i, N_s^i)$. If Encode is randomized, assume that both sequences are generated using the same random tape. Unlike with E&M encoding schemes, we do not require that the randomness used in each invocation be recoverable from the output. Consider any index i. If $|M_s^j| = |N_s^j|$ and $M_a^j = N_a^j$ for all $j \leq i$, then we require that $M_p^i = N_p^i$ and $M_o^i = N_o^i$.

The remainder of the consistency requirements for Type 1–Type 4 MtE encoding schemes are the same as those for the corresponding E&M encoding schemes.

Security requirements for MtE encoding schemes. For Type n MtE encoding schemes we consider a security notion, called mte-sec$n$, that is identical to the e&m-sec$n$ notion defined for Type n E&M encoding schemes. The formal descriptions are in Appendix C.

6.1 Summary of results

Chosen-plaintext privacy. The following shows how to ensure that an Encode-then-MtE CT will provably preserve privacy under chosen-plaintext attacks. It interprets the result in Theorem E.1. This result essentially says that an Encode-then-MtE CT should use an underlying encryption scheme that preserves privacy under chosen-plaintext attacks. As discussed in Section 3, many such encryption schemes exist.

Result 6.2 (Privacy of Encode-then-MtE) To construct a Type n Encode-then-E&M scheme CT from an encryption scheme $\mathcal{SE}$ and a MAC $\mathcal{MA}$ (that always produces tags of the same length), one should use an MtE encoding scheme $\mathcal{EC}$ that respects the IV properties of $\mathcal{SE}$. If $\mathcal{SE}$ is ind-cpa-secure and all the components satisfy their respective consistency requirements, then CT will be a cryptographic transform that provably provides privacy under chosen-plaintext attacks (i.e., CT will be ct-priv-cpa-secure). ▮

Integrity. The following distills the integrity results from Theorem E.4.

Result 6.3 (Integrity of Encode-then-MtE) To construct a Type n Encode-then-E&M scheme CT from an encryption scheme $\mathcal{SE}$ and a MAC $\mathcal{MA}$, one should use a Type n MtE encoding scheme $\mathcal{EC}$ that is mte-sec$n$-secure and that respects the IV requirements of $\mathcal{MA}$. If the $\mathcal{SE}$ encryption algorithm is length-preserving, $\mathcal{MA}$ is unforgeable (uf-secure) and always outputs tags of the same length, and all the components satisfy their respective consistency requirements, then CT will be a cryptographic transform that provably meets the ct-int-ctxt$n$ integrity notion. ▮

We again comment that it is not hard to find base components that satisfy the requirements in Result 6.3.

As with our Encode-then-E&M discussions, we note that the length-preserving requirements on the base encryption scheme are not overly restrictive since developers will likely try to avoid the extra packet expansion associated with non-length-preserving encryption algorithms anyway. In some situations, it seems possible to prove that the use of some non-length-preserving encryption schemes is safe (such proofs will likely make use of the fact that if the MAC is a secure PRF, then part of the plaintext for the base encryption scheme will not be known to an attacker). Exploring this specific scenario would take us afield from our current goal of modeling generic composition-based CTs, and (if there is suitable interest from developers) may be a topic of future work.

7 Encode-then-EtM

We now consider the Encode-then-EtM paradigm. See Figure 4. For an EtM encoding scheme $\mathcal{EC}^{\mathrm{EtM}} = (\mathsf{Encode}, \mathsf{DecodeA}, \mathsf{DecodeB}, \mathsf{DecodeC})$, the encoding algorithm Encode, which may be

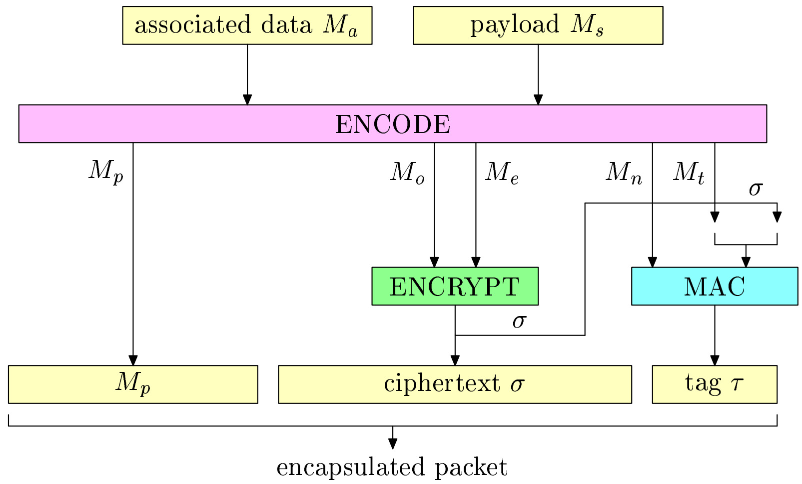

Figure 4: The Encode-then-EtM encapsulation method.

both randomized and stateful, takes as input two messages M_a, M_s and returns a 5-tuple of messages $(M_p, M_o, M_n, M_e, M_t)$. These messages have essentially the same roles as in E&M and MtE encoding schemes. An important difference is that M_t is combined with the output of the encryption algorithm before MACing. The decoding algorithms may also be stateful, but not randomized. They may share state. $\mathsf{DecodeA}$, on input a string M_p, outputs a 3-tuple (M_o, M_n, M_t), or $(\perp, \perp, \perp)$ on error (if one is $\perp$ then all are $\perp$). $\mathsf{DecodeB}$, on input two messages M_p, M_e, returns a pair (M_a, M_s), or $(\perp, \perp)$ on error (if either M_a or M_s is $\perp$, then both are $\perp$). The signature of $\mathsf{DecodeC}$ is as before.

An encryption scheme, a MAC, and an appropriate EtM encoding scheme can be combined to obtain an Encode-then-EtM CT as follows.

Construction 7.1 (Encode-then-EtM) Let $\mathcal{EC}^{\mathrm{EtM}} = (\mathsf{Encode}, \mathsf{DecodeA}, \mathsf{DecodeB}, \mathsf{DecodeC})$, let $\mathcal{SE} = (\mathcal{K}_e, \mathcal{E}, \mathcal{D})$, and let $\mathcal{MA} = (\mathcal{K}_t, \mathcal{T}, \mathcal{V})$ respectively be EtM encoding, encryption, and message-authentication schemes with compatible message spaces (e.g., the outputs from Encode are suitable inputs to $\mathcal{E}$ and $\mathcal{T}$). Let all states initially be ε. We associate to these schemes an *Encode-then-EtM cryptographic transform* $\mathsf{CT} = (\mathsf{KG}, \mathsf{Encap}, \mathsf{Decap})$ whose constituent algorithms are defined as follows:

Algorithm KG
  $K_e \xleftarrow{R} \mathcal{K}_e$; $K_t \xleftarrow{R} \mathcal{K}_t$
  Return $\langle K_e, K_t \rangle$

Algorithm $\mathsf{Encap}_{\langle K_e, K_t \rangle}(M_a, M_s)$
  $(M_p, M_o, M_n, M_e, M_t) \xleftarrow{R} \mathsf{Encode}(M_a, M_s)$
  $\sigma \xleftarrow{R} \mathcal{E}_{K_e}^{M_o}(M_e)$; $\tau \xleftarrow{R} \mathcal{T}_{K_t}^{M_n}(\langle M_t, \sigma \rangle)$
  $C \leftarrow \langle M_p, \sigma, \tau \rangle$
  Return C

Algorithm $\mathsf{Decap}_{\langle K_e, K_t \rangle}(C)$
  If $st = \perp$ then return $(\perp, \perp)$
  If there does not exist M_p, σ, τ s.t. $C = \langle M_p, \sigma, \tau \rangle$ then
    $st \leftarrow \boxed{\text{Box}}$; return $(\perp, \perp)$
  Parse C as $\langle M_p, \sigma, \tau \rangle$; $(M_o, M_n, M_t) \leftarrow \mathsf{DecodeA}(M_p)$
  If $M_o = \perp$ then $st \leftarrow \boxed{\text{Box}}$; return $(\perp, \perp)$
  $v \leftarrow \mathcal{V}_{K_t}^{M_n}(\langle M_t, \sigma \rangle, \tau)$
  If $v = 0$ then $st \leftarrow \boxed{\text{Box}}$; $\mathsf{DecodeC}(\perp)$;
    return $(\perp, \perp)$
  $M_e \leftarrow \mathcal{D}_{K_e}^{M_o}(\sigma)$
  If $M_e = \perp$ then $st \leftarrow \boxed{\text{Box}}$; $\mathsf{DecodeC}(\perp)$;
    return $(\perp, \perp)$
  $(M_a, M_s) \leftarrow \mathsf{DecodeB}(M_p, M_e)$
  If $M_s = \perp$ then $st \leftarrow \boxed{\text{Box}}$; return $(\perp, \perp)$
  $\mathsf{DecodeC}(\top)$
  Return (M_a, M_s)

For a Type 4 CT, each boxed portion of the decapsulator should be $\perp$. For all other types, the boxed portion should be st. For the call to $\mathsf{DecodeA}(M_p)$, recall that if any one of M_o, M_n, M_t is $\perp$, then they are all $\perp$. For the call to $\mathsf{DecodeB}(M_p, M_e)$, recall that if any one of M_a, M_s is $\perp$, then they are both $\perp$. Although only Decap explicitly maintains state in the above pseudocode, the underlying encoding, encryption, and MAC schemes may also maintain state. ▌

CONSISTENCY REQUIREMENTS FOR ETM ENCODING SCHEMES. Consider any two pairs of messages $(M_a, M_s), (M_a, M_s')$ with $|M_s| = |M_s'|$. Let $(M_p, M_o, M_n, M_e, M_t) \stackrel{R}{\leftarrow} \mathsf{Encode}(M_a, M_s)$ for Encode in some state, and $(M_p', M_o', M_n', M_e', M_t') \stackrel{R}{\leftarrow} \mathsf{Encode}(M_a, M_s')$ for Encode in some (possibly different) state. We require that $|M_e| = |M_e'|$. Consider also any two sequences of message pairs $(M_a^1, M_s^1), (M_a^2, M_s^2), \ldots$ and $(N_a^1, N_s^1), (N_a^2, N_s^2), \ldots$. For $i = 1, 2, \ldots$ let $(M_p^i, M_o^i, M_n^i, M_e^i, M_t^i) = \mathsf{Encode}(M_a^i, M_s^i)$ and $(N_p^i, N_o^i, N_n^i, N_e^i, N_t^i) = \mathsf{Encode}(N_a^i, N_s^i)$. Assume that each sequence is generated with Encode starting in its initial state. If Encode is randomized, assume that both sequences are generated using the same random tape. Consider any index i. If $|M_s^j| = |N_s^j|$ and $M_a^j = N_a^j$ for all $j \le i$, then we require that $M_p^i = N_p^i$, $M_o^i = N_o^i$, $M_n^i = N_n^i$, and $M_t^i = N_t^i$.

We make the following additional consistency requirements on $\mathcal{EC}^{\mathrm{EtM}}$, depending on the type of CT in question. Let $(M_a^1, M_s^1), (M_a^2, M_s^2), \ldots$ be a sequence of messages and, beginning with Encode in its initial state, let $(M_p^i, M_o^i, M_n^i, M_e^i, M_t^i) = \mathsf{Encode}(M_a^i, M_s^i)$ for $i = 1, 2, \ldots$ up to $\mathsf{MaxNum}_{\mathcal{EC}^{\mathrm{EtM}}}$. In what follows we use the notation $\mathsf{Decode[ABC]}$ to denote any one of the decoding algorithms.

TYPE 1. For any i and for any state of the decoder, we require that $\mathsf{DecodeA}(M_p^i) = (M_o^i, M_n^i, M_t^i)$ and $\mathsf{DecodeB}(M_p^i, M_e^i) = (M_a^i, M_s^i)$.

TYPE 2. For any distinct indices i, j, we require that $(M_p^i, M_e^i) \neq (M_p^j, M_e^j)$.

For any i, we require that for any state of the decoder, $\mathsf{DecodeA}(M_p^i) = (M_o^i, M_n^i, M_t^i)$. If $\mathsf{DecodeB}$ has not been invoked with (M_p^i, M_e^i) or if $\mathsf{DecodeB}$ has been invoked with (M_p^i, M_e^i) but for each such invocation the next call to $\mathsf{Decode[ABC]}$ was $\mathsf{DecodeC}(\perp)$, then $\mathsf{DecodeB}(M_p^i, M_e^i) = (M_a^i, M_s^i)$.

TYPE 3. For any distinct indices i, j, we require that $(M_p^i, M_e^i) \neq (M_p^j, M_e^j)$.

For any i, we require that for any state of the decoder, $\mathsf{DecodeA}(M_p^i) = (M_o^i, M_n^i, M_t^i)$. Furthermore, if $\mathsf{DecodeB}$ has not been invoked with (M_p^j, M_e^j) for any $j \ge i$, or if $\mathsf{DecodeB}$ has been invoked with (M_p^j, M_e^j), for some $j \ge i$, but for each such invocation the next call to $\mathsf{Decode[ABC]}$ was $\mathsf{DecodeC}(\perp)$, then $\mathsf{DecodeB}(M_p^i, M_e^i) = (M_a^i, M_s^i)$.

TYPE 4. For $i = 1, 2, \ldots$ and the decoder beginning in its initial state, let $(m_o^i, m_n^i, m_t^i) = \mathsf{DecodeA}(M_p^i)$ and $(m_a^i, m_s^i) = \mathsf{DecodeB}(M_p^i, M_e^i)$. We require that $M_a^i = m_a^i$, $M_s^i = m_s^i$, $M_o^i = m_o^i$, $M_n^i = m_n^i$, and $M_t^i = m_t^i$ for all i.

SECURITY REQUIREMENTS FOR ETM ENCODING SCHEMES. The security requirements for EtM encoding schemes are formalized in Appendix C. For Type n EtM encoding schemes (i.e., EtM encoding schemes used to construct Type n CTs) we define a security property called $\mathsf{etm\text{-}sec}n$. In order for Type 1–Type 3 EtM encoding schemes to be $\mathsf{etm\text{-}sec1}$–$\mathsf{etm\text{-}sec3}$-secure, it must be the case that if M_p and M_p' are distinct strings, then they do not decode (via $\mathsf{DecodeA}$) to identical M_n, M_t strings. For Type 2 EtM encoding schemes it should also be the case that if $\mathsf{DecodeB}(M_p, M_e)$ is called followed by a call $\mathsf{DecodeC}(\top)$, then the next time $\mathsf{DecodeB}(M_p, M_e)$ is invoked, the response is $(\perp, \perp)$. For Type 3 EtM encoding schemes it should also be the case that if M_p, M_e were in the output of one invocation of Encode, M_p', M_e' were in the output of some later Encode invocation, and $\mathsf{DecodeB}(M_p', M_e')$ is called followed by a call $\mathsf{DecodeC}(\top)$, then a later call $\mathsf{DecodeB}(M_p, M_e)$ returns $(\perp, \perp)$.

Consider some interaction with the encoding and decoding algorithms. Let $(M_p^i, M_o^i, M_n^i, M_e^i,$

M_t^i) denote the 5-tuple returned by Encode after its i-th invocation. Let m_p^j denote the parameter to the j-th call to DecodeA and let (m_o^j, m_n^j, m_t^j) denote the response. Then for Type 4 encoding schemes it should be the case that $(M_n^i, M_t^i) \neq (m_n^j, m_t^j)$ for all $i \neq j$. And, if $M_p^j \neq m_p^j$, then it should be the case that $(M_n^j, M_t^j) \neq (m_n^j, m_t^j)$.

7.1 Summary of results

Chosen-plaintext privacy. The following result, which interprets Theorem F.1, shows how to design an Encode-then-EtM CT that preserves privacy under chosen-plaintexts attacks.

Result 7.2 (Privacy of Encode-then-EtM) To construct a Type n Encode-then-EtM scheme CT from an encryption scheme $\mathcal{SE}$ and a MAC, one should use a Type n EtM encoding scheme that respects the IV properties of $\mathcal{SE}$. If all the components satisfy their respective consistency requirements and $\mathcal{SE}$ is ind-cpa-secure, then CT will be a cryptographic transform that provably provides privacy under chosen-plaintext attacks (i.e., CT will be ct-priv-cpa-secure). ▮

Integrity. We now show how to construct Encode-then-EtM cryptographic transforms meeting the CT integrity goals. The following distills the results from Theorem F.2.

Result 7.3 (Integrity of Encode-then-EtM) To construct a Type n Encode-then-EtM scheme CT from an encryption scheme $\mathcal{SE}$ and a MAC $\mathcal{MA}$, one should use a Type n etm-sec$n$-secure EtM encoding scheme that respects the IV requirements of $\mathcal{MA}$. If all the components satisfy their respective consistency requirements and $\mathcal{MA}$ is unforgeable (uf-secure), then CT will be a cryptographic transform that provably meets the ct-int-ctxt$n$ integrity notion. ▮

Observe that for an Encode-then-EtM CT, the base encryption scheme is not required to be length preserving. As for the previous paradigms, it is not hard to find base components that satisfy the requirements in the above guidelines.

8 Conclusions and Future Work

In this paper we formalize what it means for different types of cryptographic transforms to be secure, and we present guidelines for developers on how to build such cryptographic transforms. The analyses and recommendations are done in a general way, thereby allowing developers to control the specifics of how to instantiate the recommendations.

Although our results encompass many of the ways developers might naturally construct cryptographic transforms, we do note that there are some ways of constructing CTs that cannot be modeled with any of the three paradigms Encode-then-{E&M, MtE, EtM}. Consider, for example, a cryptographic transform that first MACs some string and then uses the MAC tag as the IV for the underlying encryption scheme. Such a construction falls outside of the three paradigms because it introduces additional interconnections between the encryption and authentication components. We also do not consider encryption schemes with chained initialization vectors since doing so would require feedback from the the encryption component to the encoding component. Considering these and other more advanced composition methods is the topic of future research.

Acknowledgments

We thank David McGrew and Chanathip Namprempre for comments.

References

[1] C. Beaver, T. Draelos, R. Schroeppel, and M. Torgerson. ManTiCore: Encryption with joint cipher-state authentication, 2003. Cryptology ePrint Archive 2003/154, available at `http://eprint.iacr.org/`.

[2] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pages 394–403. IEEE Computer Society Press, 1997.

[3] M. Bellare, J. Kilian, and P. Rogaway. The security of the cipher block chaining message authentication code. In Y. Desmedt, editor, *Advances in Cryptology – CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 341–358. Springer-Verlag, Berlin Germany, Aug. 1994.

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

[5] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In T. Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer-Verlag, Berlin Germany, Dec. 2000.

[6] M. Bellare and P. Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In T. Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 317–330. Springer-Verlag, Berlin Germany, Dec. 2000.

[7] M. Bellare, P. Rogaway, and D. Wagner. A conventional authenticated-encryption mode, 2003. Cryptology ePrint Archive 2003/069, available at `http://eprint.iacr.org/`.

[8] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway. UMAC: Fast and secure message authentication. In M. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 216–233. Springer-Verlag, Berlin Germany, Aug. 1999.

[9] B. Canvel, A. Hiltgen, S. Vaudenay, and M. Vuagnoux. Password interception in a SSL/TLS channel. In D. Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Germany, 2003.

[10] Y. Dodis and J. H. An. Concealment and its applications to authenticated encryption. In E. Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 312–329. Springer-Verlag, Berlin Germany, 2003.

[11] N. Ferguson, D. Whiting, B. Schneier, J. Kelsey, S. Lucks, and T. Kohno. Helix: Fast encryption and authentication in a single cryptographic primitive. In T. Johansson, editor, *Fast Software Encryption 2003*, Lecture Notes in Computer Science. Springer-Verlag, Berlin Germany, 2003.

[12] V. Gligor and P. Donescu. Fast encryption and authentication: XCBC encryption and XECB authentication modes. In *Fast Software Encryption 2001*, Lecture Notes in Computer Science. Springer-Verlag, Berlin Germany, 2001.

[13] P. Hawkes and G. Rose. Primitive specification for SOBER-128, 2003. Cryptology ePrint Archive 2003/081, available at `http://eprint.iacr.org/`.

[14] T. Iwata and K. Kurosawa. OMAC: One-key CBC MAC. In T. Johansson, editor, *Fast Software Encryption 2003*, Lecture Notes in Computer Science. Springer-Verlag, Berlin Germany, 2003.

[15] C. Jutla. Encryption modes with almost free message integrity. In B. Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 529–544. Springer-Verlag, Berlin Germany, May 2001.

[16] J. Katz and M. Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In B. Schneier, editor, *Fast Software Encryption 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 284–299. Springer-Verlag, Berlin Germany, Apr. 2000.

[17] T. Kohno, J. Viega, and D. Whiting. The CWC authenticated encryption (associated data) mode, 2003. Cryptology ePrint Archive 2003/106, available at `http://eprint.iacr.org/`.

[18] H. Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In J. Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 310–331. Springer-Verlag, Berlin Germany, Aug. 2001.

[19] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authenticationa. IETF Internet Request for Comments 2104, Feb. 1997.

[20] P. Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th Conference on Computer and Communications Security*, Nov. 2002.

[21] P. Rogaway, M. Bellare, J. Black, and T. Krovetz. OCB: A block-cipher mode of operation for efficient authenticated encryption. In *Proceedings of the 8th Conference on Computer and Communications Security*, pages 196–205. ACM Press, 2001.

[22] S. Vaudenay. Security flaws induced by CBC padding – applications to SSL, IPSEC, WTLS .... In L. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 534–545. Springer-Verlag, Berlin Germany, 2002.

A Type 5 Cryptographic Transforms

Type 5 CTs are designed to ensure the in-order delivery of packets. Unlike Type 4 CTs, bogus packets should be rejected, but should not cause the CT decapsulation algorithm to reject all future (possibly legitimate) packets.

In what follows we present the consistency requirements for Type 5 cryptographic transforms, as well as the consistency requirements for Type 5 E&M, MtE, and EtM encoding schemes. The notions of privacy and integrity for Type 5 CTs are defined in Appendix B. The notions of security for Type 5 encoding schemes are defined in Appendix C.

CONSISTENCY REQUIREMENTS. For a Type 5 CT, $\mathsf{CT} = (\mathsf{KG}, \mathsf{Encap}, \mathsf{Decap})$, let $(M_a^1, M_s^1), (M_a^2, M_s^2), \ldots$ denote a sequence of message pairs and $C_1, C_2, \ldots$ denote their encapsulation under Encap and any key K. We require that if Decap_K has not yet accepted any message (i.e., Decap_K is in its initial state or has always returned $(\perp, \perp)$), then $\mathsf{Decap}_K(C_1) = (M_a^1, M_s^1)$. For $i \geq 1$, if the only packets accepted by Decap_K are $C_1, C_2, \ldots, C_i$, in that order but with possibly some bad (and rejected) packets in the sequence of messages given to Decap_K, then $\mathsf{Decap}_K(C_{i+1}) = (M_a^{i+1}, M_s^{i+1})$.

Consistency requirements for Type 5 E&M encoding schemes. We use the term *calling sequence* to denote some sequence of calls to $\mathsf{Decode[ABC]}$ as they might appear in Construction 5.1. I.e., a calling sequence consists of a call $\mathsf{DecodeA}(M_p)$ for some M_p and, if the response is not $\perp$, a call $\mathsf{DecodeB}(M_p, M_e)$ for some M_e, and, if the response is not $(\perp,\perp,\perp,\perp)$, a call to $\mathsf{DecodeC}$. We say that (M_p, M_e) is *successfully decoded* if, in a calling sequence, the responses of the first two decoding algorithms are not $\perp$ or $(\perp,\perp,\perp,\perp)$, respectively, and $\mathsf{DecodeC}(\top)$ is called.

Assume that the decoding algorithms are always called as per the calling sequence (e.g., a $\mathsf{DecodeB}$ call is always followed by a $\mathsf{DecodeC}$ call unless $\mathsf{DecodeB}$ returns $(\perp,\perp,\perp,\perp)$). Fix $i \geq 0$ and assume that the only messages that have been successfully decoded are $(M_p^1, M_e^1), \ldots, (M_p^i, M_e^i)$, and that they were decoded in order. We require that after invoking $\mathsf{DecodeA}(M_p^{i+1})$ followed by $\mathsf{DecodeB}(M_p^{i+1}, M_e^{i+1})$ and then $\mathsf{DecodeC}(\top)$, the response to the first call is M_o^{i+1} and the response to the second call is $(M_a^{i+1}, M_s^{i+1}, M_n^{i+1}, M_t^{i+1})$.

Consistency requirements for Type 5 MtE encoding schemes. We use the term *calling sequence* to refer to some sequence of calls to $\mathsf{Decode[ABC]}$ as they might appear in Construction 6.1. I.e., a calling sequence consists of a call $\mathsf{DecodeA}(M_p)$ and, if the response is not $\perp$, either a call $\mathsf{DecodeC}(\perp)$ finalizing the calling sequence, or a call $\mathsf{DecodeB}(M_p, M_e)$ for some M_e and, if the response is not $(\perp,\perp,\perp,\perp)$, a call to $\mathsf{DecodeC}$. We say that (M_p, M_e) is *successfully decoded* if, in a calling sequence, the responses of decoding algorithms $\mathsf{DecodeA}$ and $\mathsf{DecodeB}$ are not $\perp$ or $(\perp,\perp,\perp,\perp)$, respectively, and $\mathsf{DecodeC}(\perp)$ is never called.

Assume that the decoding algorithms are always called in successive calling sequences. Fix $i \geq 0$ and assume that the only messages that have been successfully decoded are $(M_p^1, M_e^1), \ldots, (M_p^i, M_e^i)$, and that they were decoded in order. We require that after invoking $\mathsf{DecodeA}(M_p^{i+1})$ followed by $\mathsf{DecodeB}(M_p^{i+1}, M_e^{i+1})$ and then $\mathsf{DecodeC}(\top)$, the response to the first call is M_o^{i+1} and the response to the second call is $(M_a^{i+1}, M_s^{i+1}, M_n^{i+1}, M_t^{i+1})$.

Consistency requirements for Type 5 EtM encoding schemes. We use the term *calling sequence* to refer to some sequence of calls to $\mathsf{Decode[ABC]}$ as they might appear in Construction 7.1. Note that they have exactly the same form as calling sequences for Type 5 MtE encoding schemes. We say that (M_p, M_e) is *successfully decoded* if, in a calling sequence, the responses of decoding algorithms $\mathsf{DecodeA}$ and $\mathsf{DecodeB}$ are not $(\perp,\perp,\perp)$ or $(\perp,\perp)$, respectively, and $\mathsf{DecodeC}(\perp)$ is never called.

Assume that the decoding algorithms are always called in successive calling sequences. Fix $i \geq 0$ and assume that the only messages that have been successfully decoded are $(M_p^1, M_e^1), \ldots, (M_p^i, M_e^i)$, and that they were decoded in order. We require that after invoking $\mathsf{DecodeA}(M_p^{i+1})$ followed by $\mathsf{DecodeB}(M_p^{i+1}, M_e^{i+1})$ and then $\mathsf{DecodeC}(\top)$, the response to the first call is $(M_o^{i+1}, M_n^{i+1}, M_t^{i+1})$ and the response to the second call is (M_a^{i+1}, M_s^{i+1}).

B Formal Notions of Security

We use a concrete security treatment in order to model schemes based on finite objects such as block ciphers and cryptographic hash functions. To an adversary attacking a given scheme we associate a number, called the *advantage*, that measures its success in breaking the scheme with respect to a particular notion of security. Intuitively, the smaller the adversary's advantage against a scheme, the stronger the scheme is with respect to that adversary. For each of the security notions we consider here and in Appendix C, take "secure" to mean that the advantage (with respect to that security notion) of any adversary with "reasonable" resources is "small".

Cryptographic transforms. In what follows we present chosen-plaintext privacy and integrity notions for cryptographic transforms. As noted in the body of this paper, if a Type n CT meets the $\mathsf{ct\text{-}int\text{-}ctxt}n$ integrity notion and the $\mathsf{ct\text{-}priv\text{-}cpa}$ notion, then it will also provably meet a very strong notion of privacy under chosen-ciphertext attacks (the proof of this fact follows the proof of a similar result for authenticated encryption schemes in [5]). This means that it suffices to consider the notions $\mathsf{ct\text{-}priv\text{-}cpa}$ and $\mathsf{ct\text{-}int\text{-}ctxt}n$. We do not discuss chosen-ciphertext privacy notions further.

Let $\mathsf{CT} = (\mathsf{KG}, \mathsf{Encap}, \mathsf{Decap})$ be a cryptographic transform with key space $\mathsf{KeySp}_{\mathsf{CT}}$, associated data space $\mathsf{AdSp}_{\mathsf{CT}}$, and message space $\mathsf{MsgSp}_{\mathsf{CT}}$. For $K \in \mathsf{KeySp}_{\mathsf{CT}}$ and $b \in \{0,1\}$, we denote by $\mathsf{Encap}_K(\cdot, \mathcal{LR}(\cdot,\cdot,b))$ an oracle that takes input $M_a \in \mathsf{AdSp}_{\mathsf{CT}}$ and $M_0, M_1 \in \mathsf{MsgSp}_{\mathsf{CT}}$, and returns $\mathsf{Encap}_K(M_a, M_b)$ (i.e., the encapsulation of the associated data and either the left message ($b=0$) or the right message ($b=1$)). In the tradition of [2], we call this oracle a *left-or-right (LR) encapsulation oracle.* To define privacy of a cryptographic transform we consider adversaries that have access to an LR encapsulation oracle $\mathsf{Encap}_K(\cdot, \mathcal{LR}(\cdot,\cdot,b))$, for K returned by KG.

Definition B.1 (Privacy for cryptographic transforms) Let $\mathsf{CT} = (\mathsf{KG}, \mathsf{Encap}, \mathsf{Decap})$ be a cryptographic transform and let $b \in \{0,1\}$. Let A be an adversary with access to an LR encapsulation oracle $\mathsf{Encap}_K(\cdot, \mathcal{LR}(\cdot,\cdot,b))$. Assume A returns a bit. Consider the following experiment.

Experiment $\mathbf{Exp}_{\mathsf{CT}}^{\text{ct-priv-cpa-b}}(A)$
  $K \stackrel{R}{\leftarrow} \mathsf{KG}$
  Run $A^{\mathsf{Encap}_K(\cdot,\mathcal{LR}(\cdot,\cdot,b))}$
    Reply to $\mathsf{Encap}_K(M_a, \mathcal{LR}(M_0, M_1, b))$ queries as follows:
      $C \stackrel{R}{\leftarrow} \mathsf{Encap}_K(M_a, M_b)$; $A \Leftarrow C$
  Until A returns a bit d
  Return d

We require that for all queries M_a, M_0, M_1 to $\mathsf{Encap}_K(\cdot, \mathcal{LR}(\cdot,\cdot,b))$, $|M_0| = |M_1|$. We define the $\mathsf{ct\text{-}priv\text{-}cpa}$ *advantage* of $\mathsf{ct\text{-}priv\text{-}cpa}$ adversary A as

$$\mathbf{Adv}_{\mathsf{CT}}^{\text{ct-priv-cpa}}(A) = \Pr\left[\mathbf{Exp}_{\mathsf{CT}}^{\text{ct-priv-cpa-1}}(A) = 1\right] - \Pr\left[\mathbf{Exp}_{\mathsf{CT}}^{\text{ct-priv-cpa-0}}(A) = 1\right]. \ \blacksquare$$

Definition B.2 (Integrity for cryptographic transforms) Let $\mathsf{CT} = (\mathsf{KG}, \mathsf{Encap}, \mathsf{Decap})$ be a cryptographic transform. Let A_1, A_2, A_3, A_4, and A_5 be adversaries each with access to an encapsulation oracle $\mathsf{Encap}_K(\cdot,\cdot)$ and a decapsulation-verification oracle $\mathsf{Decap}^*_K(\cdot)$. The decapsulation-verification oracle, on input C, invokes $\mathsf{Decap}_K(C)$ and returns 1 if $\mathsf{Decap}_K(C) \neq (\perp,\perp)$ and 0 otherwise. Consider the experiments defined below. Each experiment returns 1 if the adversary "wins" and 0 otherwise.

Experiment $\mathbf{Exp}_{\mathsf{CT}}^{\text{ct-int-ctxt1}}(A_1)$
  $K \stackrel{R}{\leftarrow} \mathsf{KG}$; $S \leftarrow \emptyset$
  Run $A_1^{\mathsf{Encap}_K(\cdot,\cdot),\mathsf{Decap}^*_K(\cdot)}$
    Reply to $\mathsf{Encap}_K(M_a, M_s)$ queries as follows:
      $C \stackrel{R}{\leftarrow} \mathsf{Encap}_K(M_a, M_s)$; $S \leftarrow S \cup \{C\}$; $A_1 \Leftarrow C$
    Reply to $\mathsf{Decap}^*_K(C)$ queries as follows:
      $(M_a, M_s) \leftarrow \mathsf{Decap}_K(C)$
      If $(M_a, M_s) \neq (\perp,\perp)$ and $C \notin S$ then return 1 EndIf
      If $(M_a, M_s) \neq (\perp,\perp)$ then $A_1 \Leftarrow 1$
      Else $A_1 \Leftarrow 0$ EndIf
  Until A_1 halts

Return 0

Experiment $\mathbf{Exp}_{\mathsf{CT}}^{\text{ct-int-ctxt2}}(A_2)$
$K \stackrel{R}{\leftarrow} \mathsf{KG}$; $S \leftarrow \emptyset$; $S' \leftarrow \emptyset$
Run $A_2^{\mathsf{Encap}_K(\cdot,\cdot),\mathsf{Decap}^*_K(\cdot)}$
Reply to $\mathsf{Encap}_K(M_a, M_s)$ queries as follows:
$C \stackrel{R}{\leftarrow} \mathsf{Encap}_K(M_a, M_s)$; $S \leftarrow S \cup \{C\}$; $A_2 \Leftarrow C$
Reply to $\mathsf{Decap}^*_K(C)$ queries as follows:
$(M_a, M_s) \leftarrow \mathsf{Decap}_K(C)$
If $(M_a, M_s) \neq (\perp, \perp)$ and $(C \notin S$ or $C \in S')$ then return 1 EndIf
If $(M_a, M_s) \neq (\perp, \perp)$ then $S' \leftarrow S' \cup \{C\}$; $A_2 \Leftarrow 1$
Else $A_2 \Leftarrow 0$ EndIf
Until A_2 halts
Return 0

Experiment $\mathbf{Exp}_{\mathsf{CT}}^{\text{ct-int-ctxt3}}(A_3)$
$K \stackrel{R}{\leftarrow} \mathsf{KG}$; $i \leftarrow 0$; $j \leftarrow 0$
Run $A_3^{\mathsf{Encap}_K(\cdot,\cdot),\mathsf{Decap}^*_K(\cdot)}$
Reply to $\mathsf{Encap}_K(M_a, M_s)$ queries as follows:
$i \leftarrow i+1$; $C_i \stackrel{R}{\leftarrow} \mathsf{Encap}_K(M_a, M_s)$; $A_3 \Leftarrow C_i$
Reply to $\mathsf{Decap}^*_K(C)$ queries as follows:
$(M_a, M_s) \leftarrow \mathsf{Decap}_K(C)$
If $(M_a, M_s) \neq (\perp, \perp)$ and $C \notin \{C_{j+1}, \ldots, C_i\}$then return 1 EndIf
If $(M_a, M_s) \neq (\perp, \perp)$ then $j \leftarrow$ index of C in $\{C_{j+1}, \ldots, C_i\}$; $A_3 \Leftarrow 1$
Else $A_3 \Leftarrow 0$ EndIf
Until A_3 halts
Return 0

Experiment $\mathbf{Exp}_{\mathsf{CT}}^{\text{ct-int-ctxt4}}(A_4)$
$K \stackrel{R}{\leftarrow} \mathsf{KG}$; $i \leftarrow 0$; $j \leftarrow 0$; $\mathsf{phase} \leftarrow 0$
Run $A_4^{\mathsf{Encap}_K(\cdot,\cdot),\mathsf{Decap}^*_K(\cdot)}$
Reply to $\mathsf{Encap}_K(M_a, M_s)$ queries as follows:
$i \leftarrow i+1$; $C_i \stackrel{R}{\leftarrow} \mathsf{Encap}_K(M_a, M_s)$; $A_4 \Leftarrow C_i$
Reply to $\mathsf{Decap}^*_K(C)$ queries as follows:
$j \leftarrow j+1$; $(M_a, M_s) \leftarrow \mathsf{Decap}_K(C)$
If $j > i$ or $C \neq C_j$ then $\mathsf{phase} \leftarrow 1$ EndIf
If $(M_a, M_s) \neq (\perp, \perp)$ and $\mathsf{phase} = 1$ then return 1 EndIf
If $(M_a, M_s) \neq (\perp, \perp)$ then $A_4 \Leftarrow 1$
Else $A_4 \Leftarrow 0$ EndIf
Until A_4 halts
Return 0

Experiment $\mathbf{Exp}_{\mathsf{CT}}^{\text{ct-int-ctxt5}}(A_5)$
$K \stackrel{R}{\leftarrow} \mathsf{KG}$; $i \leftarrow 0$; $j \leftarrow 0$
Run $A_5^{\mathsf{Encap}_K(\cdot,\cdot),\mathsf{Decap}^*_K(\cdot)}$
Reply to $\mathsf{Encap}_K(M_a, M_s)$ queries as follows:
$i \leftarrow i+1$; $C_i \stackrel{R}{\leftarrow} \mathsf{Encap}_K(M_a, M_s)$; $A_5 \Leftarrow C_i$
Reply to $\mathsf{Decap}^*_K(C)$ queries as follows:
$(M_a, M_s) \leftarrow \mathsf{Decap}_K(C)$

If $(M_a, M_s) \neq (\perp, \perp)$ and ($j+1 > i$ or $C \neq C_{j+1}$) then return 1 EndIf
If $(M_a, M_s) \neq (\perp, \perp)$ then $j \leftarrow j+1$; $A_5 \Leftarrow 1$
Else $A_5 \Leftarrow 0$ EndIf
Until A_5 halts
Return 0

For $n = 1, \ldots, 5$, we define the ct-int-ctxt$n$ *advantage* of ct-int-ctxt$n$ adversary A_n as

$$\mathbf{Adv}^{\text{ct-int-ctxt}n}_{\mathsf{CT}}(A_n) \;=\; \Pr\left[\, \mathbf{Exp}^{\text{ct-int-ctxt}n}_{\mathsf{CT}}(A_n) = 1 \,\right]. \quad \blacksquare$$

Privacy for symmetric encryption schemes and MACs. We now describe a notion of chosen-plaintext privacy for encryption schemes and MACs. Although the notion is most intuitive when applied to encryption schemes, there are some situations where having a privacy-preserving MAC is useful.

To define the privacy of a symmetric encryption scheme or MAC $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, we give an adversary access to a left-or-right (LR) encryption (or tagging) oracle $\mathcal{E}_K(\cdot, \mathcal{LR}(\cdot,\cdot,b))$, for some unknown key K returned by $\mathcal{K}$ and a bit b. On input I, M_0, M_1, where $I \in \mathsf{IVSp}_{\mathcal{SE}}$ and $M_0, M_1 \in \mathsf{MsgSp}_{\mathcal{SE}}$, the oracle returns $\mathcal{E}^I_K(M_b)$. The following notion of security extends the notion of left-or-right-indistinguishability from [2] to encryption schemes that explicitly take a nonce or IV as input.

Definition B.3 (Privacy for symmetric encryption and MAC schemes) Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme or a message-authentication scheme, and let $b \in \{0,1\}$. Let A_{cpa} be an adversary with access to a left-or-right encryption (or tagging) oracle $\mathcal{E}_K(\cdot, \mathcal{LR}(\cdot,\cdot,b))$. Assume A_{cpa} returns a bit. Consider the following experiment.

Experiment $\mathbf{Exp}^{\text{ind-cpa-b}}_{\mathcal{SE}}(A_{\text{cpa}})$
$K \stackrel{R}{\leftarrow} \mathcal{K}$
Run $A_{\text{cpa}}^{\mathcal{E}_K(\cdot,\mathcal{LR}(\cdot,\cdot,b))}$
Reply to $\mathcal{E}_K(I, \mathcal{LR}(M_0, M_1, b))$ queries as follows:
$C \stackrel{R}{\leftarrow} \mathcal{E}^I_K(M_b)$; $A_{\text{cpa}} \Leftarrow C$
Until A_{cpa} returns a bit d
Return d

We require that for all queries I, M_0, M_1 to $\mathcal{E}_K(\cdot, \mathcal{LR}(\cdot,\cdot,b))$, $|M_0| = |M_1|$. We call the adversary A_{cpa} *nonce-respecting* if it never queries its oracle with the same nonce twice. We call the adversary *length-based IV-respecting* if it chooses the first IV uniformly at random and independently and if the subsequent IVs are computed using the encryption scheme's length-based IV-deriving function. We call the adversary *random-IV-respecting* if it only queries its oracle with IVs chosen uniformly at random and independently. (As noted in the body, we can consider such adversaries in our reductions because we can control how the encoding algorithms generate the IVs.) We define the *chosen-plaintext (*ind-cpa*) advantage* of ind-cpa adversary A as

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}}(A_{\text{cpa}}) = \Pr\left[\, \mathbf{Exp}^{\text{ind-cpa-1}}_{\mathcal{SE}}(A_{\text{cpa}}) = 1 \,\right] - \Pr\left[\, \mathbf{Exp}^{\text{ind-cpa-0}}_{\mathcal{SE}}(A_{\text{cpa}}) = 1 \,\right].$$

Intuitively, we say that the scheme $\mathcal{SE}$ preserves privacy against nonce-respecting (resp., length-based IV-respecting or random-IV-respecting) adversaries if the advantage of all nonce-respecting (resp., length-based IV-respecting or random-IV-respecting) adversaries with reasonable resources is small. $\blacksquare$

Definition B.4 (Privacy for MACs under distinct chosen-plaintexts.) Let $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be a message-authentication scheme. Let $b \in \{0,1\}$. Let A be an adversary with access to a left-or-right tagging oracle $\mathcal{T}_K(\cdot, \mathcal{LR}(\cdot,\cdot,b))$. Consider the following experiment.

Experiment $\mathbf{Exp}^{\text{ind-dcpa-b}}_{\mathcal{MA}}(A)$
  $K \xleftarrow{R} \mathcal{K}$
  Run $A^{\mathcal{T}_K(\cdot,\mathcal{LR}(\cdot,\cdot,b))}$
    Reply to $\mathcal{T}_K(I, \mathcal{LR}(M_0, M_1, b))$ queries as follows:
      $C \xleftarrow{R} \mathcal{T}^I_K(M_b)$; $A \Leftarrow C$
  Until A returns a bit d
  Return d

We require that for all queries I, M_0, M_1 to the tagging oracle, $|M_0| = |M_1|$. If I^i, M_0^i, M_1^i is the i-th oracle query, we require that for all indices j, k, $j \neq k$, $(I^j, M_0^j) \neq (I^k, M_0^k)$ and $(I^j, M_1^j) \neq (I^k, M_1^k)$ (i.e., all left queries are distinct and all right queries are distinct). We call the adversary A *nonce-respecting* if it never queries its oracle with the same nonce twice. We define the *distinct-chosen-plaintext (*ind-dcpa*) advantage* of ind-dcpa adversary A as

$$\mathbf{Adv}^{\text{ind-dcpa}}_{\mathcal{MA}}(A) = \Pr\left[\mathbf{Exp}^{\text{ind-dcpa-1}}_{\mathcal{MA}}(A) = 1\right] - \Pr\left[\mathbf{Exp}^{\text{ind-dcpa-0}}_{\mathcal{MA}}(A) = 1\right].$$

Intuitively, we say that $\mathcal{MA}$ preserves distinct-chosen-plaintext privacy against (nonce-respecting) adversaries if the advantage of all (nonce-respecting) adversaries with reasonable resources is small. ▮

Unforgeability and pseudorandomness of MACs. We now specify the notions of unforgeability and pseudorandomness for MACs.

Definition B.5 (Unforgeability of MACs) Let $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be a message-authentication scheme. Let F be an adversary with access to a tagging oracle and a verification oracle. Consider the experiment:

Experiment $\mathbf{Exp}^{\text{uf-cma}}_{\mathcal{MA}}(F)$
  $K \xleftarrow{R} \mathcal{K}$; $S \leftarrow \emptyset$
  Run $F^{\mathcal{T}_K(\cdot,\cdot),\mathcal{V}_K(\cdot,\cdot,\cdot)}$
    Reply to $\mathcal{T}_K(I, M)$ queries as follows:
      $\tau \xleftarrow{R} \mathcal{T}^I_K(M)$; $S \leftarrow S \cup \{(I, M, \tau)\}$; $F \Leftarrow \tau$
    Reply to $\mathcal{V}_K(I, M, \tau)$ queries as follows:
      $v \leftarrow \mathcal{V}^I_K(M, \tau)$
      If $v = 1$ and $(I, M, \tau) \notin S$ then return 1
      $F \Leftarrow v$
  Until F halts
  Return 0

We define the uf *advantage* of the forger via

$$\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{MA}}(F) \quad = \quad \Pr\left[\mathbf{Exp}^{\text{uf-cma}}_{\mathcal{MA}}(F) = 1\right]. \text{ ▮}$$

Definition B.6 (Pseudorandom functions) Let $\mathcal{F} : \{0,1\}^k \times \mathcal{M} \to \{0,1\}^L$ be a family of functions from some message space $\mathcal{M}$ to $\{0,1\}^L$, and let $\mathrm{Rand}^{\mathcal{M}\to L}$ denote the family of all functions from $\mathcal{M}$ to $\{0,1\}^L$. Let D be an adversary with access to an oracle. Consider the following experiment.

Experiment $\mathbf{Exp}_{\mathcal{F}}^{\text{prf-b}}(D)$
 If $b = 1$ then $K \xleftarrow{R} \{0,1\}^k$; $g \leftarrow \mathcal{F}_K$
 Else $g \xleftarrow{R} \text{Rand}^{\mathcal{M}\to L}$ EndIf
 Run D^g
  Reply to $g(M)$ queries as follows:
   $D \Leftarrow g(M)$
 Until D returns a bit d
 Return d

We define the prf *advantage* of prf adversary D as

$$\mathbf{Adv}_{\mathcal{F}}^{\text{prf}}(D) = \Pr\left[\mathbf{Exp}_{\mathcal{F}}^{\text{prf-1}}(D) = 1\right] - \Pr\left[\mathbf{Exp}_{\mathcal{F}}^{\text{prf-0}}(D) = 1\right]. \quad \blacksquare$$

Relationships between notions. As shown in [3], if a MAC is a secure PRF, then it is also uf-secure. (When we say a MAC $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ is a secure PRF, we mean that the MAC takes no IVs (i.e., $\mathsf{IVSp}_{\mathcal{MA}} = \{\varepsilon\}$) and the family of functions $F = \{\, \mathcal{T}_K(\varepsilon, \cdot) \;:\; K \in \mathsf{KeySp}_{\mathcal{MA}} \,\}$ is a secure PRF.) We also comment that a number of popular MACs are proven to be secure PRFs. Furthermore, as shown in [4], if a MAC is a secure PRF, then it also ind-dcpa-secure. The reader may ask why we even introduce the ind-dcpa notion if most popular MACs are secure PRFs and the PRF notion implies the ind-dcpa notion. The reason is that in our analysis we want to focus on the minimum properties necessary in order to achieve our goals.

C Security Properties for Encoding Schemes

Definition C.1 (Security of E&M- and MtE-encoding schemes) Consider an E&M or MtE encoding scheme $\mathcal{EC} = (\mathsf{Encode}, \mathsf{DecodeA}, \mathsf{DecodeB}, \mathsf{DecodeC})$. Let A_{cpa} be an adversary with access to an encoding oracle $\mathsf{Encode}(\cdot,\cdot)$ and for $n = 1, \ldots, 5$, let A_n be an adversary with access to an encoding oracle and decoding oracles $\mathsf{DecodeA}(\cdot)$, $\mathsf{DecodeB}(\cdot,\cdot)$, $\mathsf{DecodeC}(\cdot)$ (the adversary may need access to all decoding oracles since these may share state). Let (M_a^i, M_s^i) denote an adversary's i-th encoding query and let $(M_p^i, M_o^i, M_n^i, M_e^i, M_t^i)$ denote the response for that query. Let (m_p^i, m_e^i) denote A_n's i-th $\mathsf{DecodeB}(\cdot,\cdot)$ query and let $(m_a^i, m_s^i, m_n^i, m_t^i)$ denote the response for that query.

Consider the following experiments. (The experiments $\mathbf{Exp}_{\mathcal{EC}}^{\text{mte-sec}n}(A_n)$ for MtE are identical to the $\mathbf{Exp}_{\mathcal{EC}}^{\text{e\&m-sec}n}(A_n)$ experiments for E&M.)

Experiment $\mathbf{Exp}_{\mathcal{EC}}^{\text{e\&m-coll}}(A_{\text{cpa}})$
 Run $A_{\text{cpa}}^{\mathsf{Encode}(\cdot,\cdot)}$ and if it makes two queries (M_a^i, M_s^i) and (M_a^j, M_s^j) to $\mathsf{Encode}(\cdot,\cdot)$ such that $i \neq j$ and
  $(M_n^i, M_t^i) = (M_n^j, M_t^j)$
 then return 1 else return 0

Experiment $\mathbf{Exp}_{\mathcal{EC}}^{\text{e\&m-sec1}}(A_1)$
 Run $A_1^{\mathsf{Encode}(\cdot,\cdot),\mathsf{DecodeA}(\cdot),\mathsf{DecodeB}(\cdot,\cdot),\mathsf{DecodeC}(\cdot)}$ and, if the following occurs:
 — A_1 makes a query (M_a^i, M_s^i) to $\mathsf{Encode}(\cdot,\cdot)$ and a query (m_p^j, m_e^j) to $\mathsf{DecodeB}(\cdot,\cdot)$ such that
  $(M_p^i, M_e^i) \neq (m_p^j, m_e^j)$ and $(M_n^i, M_t^i) = (m_n^j, m_t^j)$
 then return 1 else return 0

Experiment $\mathbf{Exp}_{\mathcal{EC}}^{\text{e\&m-sec2}}(A_2)$
 Run $A_2^{\mathsf{Encode}(\cdot,\cdot),\mathsf{DecodeA}(\cdot),\mathsf{DecodeB}(\cdot,\cdot),\mathsf{DecodeC}(\cdot)}$ and, if one of the following occurs:
 — A_2 makes a query (M_a^i, M_s^i) to $\mathsf{Encode}(\cdot,\cdot)$ and a query (m_p^j, m_e^j) to $\mathsf{DecodeB}(\cdot,\cdot)$ such that
  $(M_p^i, M_e^i) \neq (m_p^j, m_e^j)$ and $(M_n^i, M_t^i) = (m_n^j, m_t^j)$
 — A_2 twice makes a query (m_p^j, m_e^j) to $\mathsf{DecodeB}(\cdot,\cdot)$, the next $\mathsf{Decode[ABC]}$ query following the first of

these queries is a call $\mathsf{DecodeC}(\top)$, and the response for the second of these queries is not $(\perp,\perp,\perp,\perp)$
then return 1 else return 0

Experiment $\mathbf{Exp}_{\mathcal{EC}}^{\text{e\&m-sec3}}(A_3)$
Run $A_3^{\mathsf{Encode}(\cdot,\cdot),\mathsf{DecodeA}(\cdot),\mathsf{DecodeB}(\cdot,\cdot),\mathsf{DecodeC}(\cdot)}$ and, if one of the following occurs:
— A_3 makes a query (M_a^i, M_s^i) to $\mathsf{Encode}(\cdot,\cdot)$ and a query (m_p^j, m_e^j) to $\mathsf{DecodeB}(\cdot,\cdot)$ such that $(M_p^i, M_e^i) \neq (m_p^j, m_e^j)$ and $(M_n^i, M_t^i) = (m_n^j, m_t^j)$
— A_3 makes queries (m_p^j, m_e^j) and (m_p^{j+l}, m_e^{j+l}), $l \geq 1$, to $\mathsf{DecodeB}(\cdot,\cdot)$ such that the next $\mathsf{Decode[ABC]}$ query following the first of these queries is a call $\mathsf{DecodeC}(\top)$, the response for the second of these queries is not $(\perp,\perp,\perp,\perp)$, and for some i,k with $k \leq i, (m_p^j, m_e^j) = (M_p^i, M_e^i)$ and $(m_p^{j+l}, m_e^{j+l}) = (M_p^k, M_e^k)$
then return 1 else return 0

Experiment $\mathbf{Exp}_{\mathcal{EC}}^{\text{e\&m-sec4}}(A_4)$
Run $A_4^{\mathsf{Encode}(\cdot,\cdot),\mathsf{DecodeA}(\cdot),\mathsf{DecodeB}(\cdot,\cdot),\mathsf{DecodeC}(\cdot)}$ and, if one of the following occurs:
— A_4 makes a query (M_a^i, M_s^i) to $\mathsf{Encode}(\cdot,\cdot)$ and a query (m_p^j, m_e^j) to $\mathsf{DecodeB}(\cdot,\cdot)$ such that $i \neq j$ and $(M_n^i, M_t^i) = (m_n^j, m_t^j)$
— A_4 makes a query (M_a^j, M_s^j) to $\mathsf{Encode}(\cdot,\cdot)$ and a query (m_p^j, m_e^j) to $\mathsf{DecodeB}(\cdot,\cdot)$ such that $(M_p^j, M_e^j) \neq (m_p^j, m_e^j)$ and $(M_n^j, M_t^j) = (m_n^j, m_t^j)$
then return 1 else return 0

Experiment $\mathbf{Exp}_{\mathcal{EC}}^{\text{e\&m-sec5}}(A_5)$
Run $A_5^{\mathsf{Encode}(\cdot,\cdot),\mathsf{DecodeA}(\cdot),\mathsf{DecodeB}(\cdot,\cdot),\mathsf{DecodeC}(\cdot)}$ and, if one of the following occurs:
— A_5 makes a query (M_a^i, M_s^i) to $\mathsf{Encode}(\cdot,\cdot)$ and a query (m_p^j, m_e^j) to $\mathsf{DecodeB}(\cdot,\cdot)$ such that $(M_n^i, M_t^i) = (m_n^j, m_t^j)$ and, prior to the j-th $\mathsf{DecodeB}(\cdot,\cdot)$ query, A_5 did *not* make exactly $i-1$ $\mathsf{DecodeB}(\cdot,\cdot)$ queries that returned messages (i.e., not $\perp$) and that were followed by $\mathsf{DecodeC}(\top)$ calls
— A_5 makes a query (M_a^i, M_s^i) to $\mathsf{Encode}(\cdot,\cdot)$ and a query (m_p^j, m_e^j) to $\mathsf{DecodeB}(\cdot,\cdot)$ such that $(M_p^i, M_e^i) \neq (m_p^j, m_e^j)$ and $(M_n^i, M_t^i) = (m_n^j, m_t^j)$, and, prior to the j-th $\mathsf{DecodeB}(\cdot,\cdot)$ query, A_5 made *exactly* $i-1$ $\mathsf{DecodeB}(\cdot,\cdot)$ queries that returned messages (i.e., not $\perp$) and that were followed by $\mathsf{DecodeC}(\top)$ calls
then return 1 else return 0

We define the $\mathsf{e\&m\text{-}coll}$ *advantage* of adversary A_{cpa}, and, for $n = 1,\ldots,5$, the $\mathsf{e\&m\text{-}sec}n$ *advantage* and the $\mathsf{mte\text{-}sec}n$ *advantage* of adversary A_n, respectively, as follows:

$$\begin{aligned}\mathbf{Adv}_{\mathcal{EC}}^{\text{e\&m-coll}}(A_{\text{cpa}}) &= \Pr\left[\mathbf{Exp}_{\mathcal{EC}}^{\text{e\&m-coll}}(A_{\text{cpa}}) = 1\right]\\ \mathbf{Adv}_{\mathcal{EC}}^{\text{e\&m-sec}n}(A_n) &= \Pr\left[\mathbf{Exp}_{\mathcal{EC}}^{\text{e\&m-sec}n}(A_n) = 1\right]\\ \mathbf{Adv}_{\mathcal{EC}}^{\text{mte-sec}n}(A_n) &= \Pr\left[\mathbf{Exp}_{\mathcal{EC}}^{\text{mte-sec}n}(A_n) = 1\right].\end{aligned}$$ ▮

Definition C.2 (Security of EtM encoding schemes) Consider an EtM encoding scheme $\mathcal{EC} = (\mathsf{Encode}, \mathsf{DecodeA}, \mathsf{DecodeB}, \mathsf{DecodeC})$. For $n = 1,\ldots,5$, let A_n be an adversary with access to an encoding oracle $\mathsf{Encode}(\cdot,\cdot)$ and decoding oracles $\mathsf{DecodeA}(\cdot)$, $\mathsf{DecodeB}(\cdot,\cdot)$, $\mathsf{DecodeC}(\cdot)$ (the adversary may need access to all decoding oracles since these may share state). Let (M_a^i, M_s^i) denote an adversary's i-th encoding query and let $(M_p^i, M_o^i, M_n^i, M_e^i, M_t^i)$ denote the response for that query. Let m_p^i denote A_n's i-th $\mathsf{DecodeA}(\cdot)$ query and let (m_o^i, m_n^i, m_t^i) denote the response for that query. Consider the following experiments.

Experiment $\mathbf{Exp}_{\mathcal{EC}}^{\text{etm-sec1}}(A_1)$
Run $A_1^{\mathsf{Encode}(\cdot,\cdot),\mathsf{DecodeA}(\cdot),\mathsf{DecodeB}(\cdot,\cdot),\mathsf{DecodeC}(\cdot)}$ and, if the following occurs:
— A_1 makes a query (M_a^i, M_s^i) to $\mathsf{Encode}(\cdot,\cdot)$ and a query m_p^j to $\mathsf{DecodeA}(\cdot)$ such that $M_p^i \neq m_p^j$ and $(M_n^i, M_t^i) = (m_n^j, m_t^j)$
then return 1 else return 0

Experiment $\mathbf{Exp}_{\mathcal{EC}}^{\text{etm-sec2}}(A_2)$
Run $A_2^{\mathsf{Encode}(\cdot,\cdot),\mathsf{DecodeA}(\cdot),\mathsf{DecodeB}(\cdot,\cdot),\mathsf{DecodeC}(\cdot)}$ and, if one of the following occurs:
— A_2 makes a query (M_a^i, M_s^i) to $\mathsf{Encode}(\cdot,\cdot)$ and a query m_p^j to $\mathsf{DecodeA}(\cdot)$ such that $M_p^i \neq m_p^j$ and $(M_n^i, M_t^i) = (m_n^j, m_t^j)$
— A_2 twice makes a query (m_p^j, m_e^j) to $\mathsf{DecodeB}(\cdot,\cdot)$, the next $\mathsf{Decode[ABC]}$ query following the first of these queries is a call $\mathsf{DecodeC}(\top)$, and the response for the second of these queries is not $(\perp,\perp)$
then return 1 else return 0

Experiment $\mathbf{Exp}_{\mathcal{EC}}^{\text{etm-sec3}}(A_3)$
Run $A_3^{\mathsf{Encode}(\cdot,\cdot),\mathsf{DecodeA}(\cdot),\mathsf{DecodeB}(\cdot,\cdot),\mathsf{DecodeC}(\cdot)}$ and, if one of the following occurs:
— A_3 makes a query (M_a^i, M_s^i) to $\mathsf{Encode}(\cdot,\cdot)$ and a query m_p^j to $\mathsf{DecodeA}(\cdot)$ such that $M_p^i \neq m_p^j$ and $(M_n^i, M_t^i) = (m_n^j, m_t^j)$
— A_3 makes queries (m_p^j, m_e^j) and (m_p^{j+l}, m_e^{j+l}), $l \geq 1$, to $\mathsf{DecodeB}(\cdot,\cdot)$ such that the next $\mathsf{Decode[ABC]}$ query following the first of these queries is a call $\mathsf{DecodeC}(\top)$, the response for the second of these queries is not $(\perp,\perp)$, and for some i,k with $k \leq i$, $(m_p^j, m_e^j) = (M_p^i, M_e^i)$ and $(m_p^{j+l}, m_e^{j+l}) = (M_p^k, M_e^k)$
then return 1 else return 0

Experiment $\mathbf{Exp}_{\mathcal{EC}}^{\text{etm-sec4}}(A_4)$
Run $A_4^{\mathsf{Encode}(\cdot,\cdot),\mathsf{DecodeA}(\cdot),\mathsf{DecodeB}(\cdot,\cdot),\mathsf{DecodeC}(\cdot)}$ and, if one of the following occurs:
— A_4 makes a query (M_a^i, M_s^i) to $\mathsf{Encode}(\cdot,\cdot)$ and a query m_p^j to $\mathsf{DecodeA}(\cdot)$ such that $i \neq j$ and $(M_n^i, M_t^i) = (m_n^j, m_t^j)$
— A_4 makes a query (M_a^j, M_s^j) to $\mathsf{Encode}(\cdot,\cdot)$ and a query m_p^j to $\mathsf{DecodeA}(\cdot)$ such that $M_p^j \neq m_p^j$ and $(M_n^j, M_t^j) = (m_n^j, m_t^j)$
then return 1 else return 0

Experiment $\mathbf{Exp}_{\mathcal{EC}}^{\text{etm-sec5}}(A_5)$
Run $A_5^{\mathsf{Encode}(\cdot,\cdot),\mathsf{DecodeA}(\cdot),\mathsf{DecodeB}(\cdot,\cdot),\mathsf{DecodeC}(\cdot)}$ and, if A_5 only invokes $\mathsf{Decode[ABC]}$ in legitimate EtM calling sequences (see Appendix A), and one of the following occurs:
— A_5 makes a query (M_a^i, M_s^i) to $\mathsf{Encode}(\cdot,\cdot)$ and a query m_p^j to $\mathsf{DecodeA}(\cdot)$ such that $(M_n^i, M_t^i) = (m_n^j, m_t^j)$ and, prior to the j-th $\mathsf{DecodeA}(\cdot)$ query, A_5 did *not* make exactly $i-1$ $\mathsf{Decode[ABC]}$ calling sequences that ended in the call $\mathsf{DecodeC}(\top)$
— A_5 makes a query (M_a^i, M_s^i) to $\mathsf{Encode}(\cdot,\cdot)$ and a query m_p^j to $\mathsf{DecodeA}(\cdot)$ such that $M_p^i \neq m_p^j$ and $(M_n^i, M_t^i) = (m_n^j, m_t^j)$, and, prior to the j-th $\mathsf{DecodeA}(\cdot)$ query, A_5 made *exactly* $, i-1$ $\mathsf{Decode[ABC]}$ calling sequences that ended in the call $\mathsf{DecodeC}(\top)$
then return 1 else return 0

For $n = 1, \ldots, 5$, we define the etm-sec$n$ *advantage* of adversary A_n as

$$\mathbf{Adv}_{\mathcal{EC}}^{\text{etm-sec}n}(A_n) \;=\; \Pr\left[\mathbf{Exp}_{\mathcal{EC}}^{\text{etm-sec}n}(A_n) = 1\right]. \quad \blacksquare$$

D Encode-then-E&M

Privacy. The following is our privacy result for Encode-then-E&M CTs. This theorem is interpreted in Result 5.2.

Theorem D.1 (Privacy of Encode-then-E&M) Let $\mathcal{SE}$, $\mathcal{MA}$, and $\mathcal{EC}$ be an encryption, a message authentication, and an E&M encoding scheme, respectively. Let CT be the cryptographic transform associated to them as per Construction 5.1. Then, given any ct-priv-cpa adversary S against CT, there exist adversaries A, B, D, and C such that

$$\mathbf{Adv}_{\mathsf{CT}}^{\text{ct-priv-cpa}}(S) \leq \mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cpa}}(A) + \mathbf{Adv}_{\mathcal{MA}}^{\text{ind-dcpa}}(D) + 2 \cdot \mathbf{Adv}_{\mathcal{EC}}^{\text{e\&m-coll}}(C)$$

and

$$\mathbf{Adv}^{\text{ct-priv-cpa}}_{\mathsf{CT}}(S) \le \mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}}(A) + \mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{MA}}(B)\ .$$

Furthermore, A, B, D, and C use the same resources as S except that A's, B's, and D's inputs to their respective oracles may be of different lengths than those of S (due to the encoding). If $\mathcal{EC}$ is nonce-respecting-for-encryption (resp., length-based IV-respecting-for-encryption or random-IV-respecting-for-encryption), then A will be nonce-respecting (resp., length-based IV-respecting or random-IV-respecting). Similarly, if $\mathcal{EC}$ is nonce-respecting-for-MACing, then B and D will be nonce-respecting. ▮

The proof of Theorem D.1 is similar to the proof of Lemma 6.4 in [4] and is omitted here. Differences between Theorem D.1 and Lemma 6.4 in [4] include the following: we consider cryptographic transforms that take associated data; we allow $\mathcal{SE}$ to take nonces, length-based IVs, or random-IVs as input, and $\mathcal{MA}$ to take nonces as input; in order for the hybrid argument to work, we use the fact that we can recover the randomness from the output of $\mathcal{EC}$'s encoding function.

We remark that if the underlying MAC requires a nonce, then $\mathbf{Adv}^{\text{e\&m-coll}}_{\mathcal{EC}}(C) = 0$. We also note that some MACs (e.g., Carter-Wegman MACs) are ind-cpa- and ind-dcpa-secure.

INTEGRITY. We begin by formalizing a new property for Encode-then-E&M CTs. As with our use of the ind-dcpa notion, we use this security notion because we feel it important to accurately describe the specific properties we require from the CT. In most situations, however, one does not actually need to manipulate this definition but must merely invoke Lemma D.3.

Definition D.2 Fix $n \in \{1, \ldots, 5\}$. Let $\mathcal{SE}$, $\mathcal{MA}$, and $\mathcal{EC}$, respectively, be an encryption, a message authentication, and an E&M encoding scheme. Let $\mathsf{CT} = (\mathsf{KG}, \mathsf{Encap}, \mathsf{Decap})$ be a Type n cryptographic transform associated to them as per Construction 5.1. Let A be an adversary with access to an encapsulation oracle $\mathsf{Encap}_K(\cdot,\cdot)$ and a decapsulation oracle $\mathsf{Decap}_K(\cdot)$. Let (M^i_a, M^i_s) denote the adversary's i-th encapsulation oracle query, $(M^i_p, M^i_o, M^i_n, M^i_e, M^i_t)$ denote the encoding of that query, and $\langle M^i_p, \sigma_i, \tau_i\rangle$ denote the returned ciphertext. Let $\langle m^i_p, \sigma'_i, \tau'_i\rangle$ denote the i-th decapsulation query (assuming it is parseable), and $m^i_o, m^i_n, m^i_e, m^i_t, m^i_a, m^i_s$ denote the internal values in the decapsulation process (or $\perp$ if an error occurs during decapsulation). A "wins" if it makes a decapsulation query $\langle m^j_p, \sigma'_j, \tau'_j\rangle$ such that $(m^j_o, m^j_e) = (M^i_o, M^i_e)$ for some $i \in \{1, \ldots, k\}$ but $\sigma'_j \ne \sigma_i$ (where k is the number of $\mathsf{Encap}_K(\cdot,\cdot)$ oracle queries made by A before A's j-th decapsulation query). We define the e&m-sp *advantage* of e&m-sp adversary A as

$$\mathbf{Adv}^{\text{e\&m-sp}}_{\mathsf{CT}}(A) \;=\; \Pr\left[\, K \stackrel{R}{\leftarrow} \mathsf{KG} \,:\, A \text{ ``wins''} \,\right]\ . \ ▮$$

The following lemma shows that if the underlying encryption scheme is length preserving (such as random-IV CBC mode as defined in the first example of a random IVed encryption scheme in Section 3), then an adversary cannot win the game described in the above definition.

Lemma D.3 Fix $n \in \{1, \ldots, 5\}$. Let $\mathcal{SE}$, $\mathcal{MA}$, and $\mathcal{EC}$, respectively, be an encryption, a MAC, and a Type n E&M encoding scheme. Let $\mathsf{CT} = (\mathsf{KG}, \mathsf{Encap}, \mathsf{Decap})$ be a Type n cryptographic transform associated to them as per Construction 5.1. Let A be an e&m-sp adversary. If $\mathcal{SE}$'s encryption operation is length-preserving, then

$$\mathbf{Adv}^{\text{e\&m-sp}}_{\mathsf{CT}}(A) \;=\; 0\ . \ ▮$$

Proof: If $\mathcal{SE}$'s encryption operation is length-preserving, then given any IV I, the encryption operation is bijective. This means A can never win. ▮

We can now state our integrity result for Encode-then-E&M constructions. This theorem is interpreted in Result 5.3.

Theorem D.4 (Integrity of Encode-then-E&M) Fix $n \in \{1, \ldots, 5\}$. Let $\mathcal{SE}$, $\mathcal{MA}$, and $\mathcal{EC}$, respectively, be an encryption, a MAC, and a Type n E&M encoding scheme. Let CT be a Type n cryptographic transform associated to them as per Construction 5.1. Then, given any ct-int-ctxt$n$ adversary I against CT, there exist adversaries F, C, and S such that

$$\mathbf{Adv}^{\text{ct-int-ctxt}n}_{\mathsf{CT}}(I) \leq \mathbf{Adv}^{\text{uf-cma}}_{\mathcal{MA}}(F) + \mathbf{Adv}^{\text{e\&m-sec}n}_{\mathcal{EC}}(C) + \mathbf{Adv}^{\text{e\&m-sp}}_{\mathsf{CT}}(S) .$$

Furthermore, F, C, and S use the same resources as I except that F's messages to its oracles may be of different lengths than I's queries to its oracles (due to encoding) and C's messages to its decoding oracle may have slightly different lengths than I's decapsulation queries. If $\mathcal{EC}$ is nonce-respecting-for-MACing, then F will be nonce-respecting. ▌

We remark that the proof of the above for Type 4 CTs is similar to the proof of Theorem 6.5 of [4] except that we consider cryptographic transforms that accept associated data. Let us now consider the proof for all types $n \in \{1, \ldots, 5\}$.

Proof: Let F, C, and S be adversaries that run I and reply to I's oracle queries using their own oracles. In more detail, F presents I with encapsulation and decapsulation-verification oracles exactly as in Construction 5.1 except that F uses its own oracles for handling tagging and verification portions of Construction 5.1. Similarly, C runs I exactly as in Construction 5.1 except that it runs all encoding and decoding operations through its own oracles. In the case of S, S simply passes all of I's encapsulation and decapsulation queries to its (S's) own oracles.

Let (M_a^i, M_s^i) denote I's i-th oracle query, let $(M_p^i, M_o^i, M_n^i, M_e^i, M_t^i)$ denote the encoding of that query, and let $\langle M_p^i, \sigma_i, \tau_i \rangle$ denote the returned ciphertext. Additionally, let $\langle m_p^i, \sigma_i', \tau_i' \rangle$ denote the i-th decapsulation-verification query (assuming it is parseable), $m_o^i, m_n^i, m_e^i, m_t^i, m_a^i, m_s^i$ denote the internal values in the decapsulation process (or $\perp$ if an error occurs during decapsulation). Let j denote the index of I's (first) winning query and let k denote the number of encapsulation oracle queries performed at the time I wins.

Let E be the event that I wins. By partitioning the event E, we see that if I succeeds in forging, then one of F, C, and S will also win their game.

For a Type 1 CT, let the event E be partitioned as follows:

$$
\begin{array}{ll}
E & : \ I \text{ wins} \\
E_1 & : \ E \text{ occurs and } (m_p^j, m_e^j, \tau_j') \in \{\, (M_p^i, M_e^i, \tau_i) \,:\, 1 \leq i \leq k \,\} \quad // \quad S \text{ wins} \\
E_2 & : \ E \text{ occurs and } (m_p^j, m_e^j, \tau_j') \notin \{\, (M_p^i, M_e^i, \tau_i) \,:\, 1 \leq i \leq k \,\} \\
E_{2,1} & : \ E_2 \text{ occurs and } (m_n^j, m_t^j, \tau_j') \notin \{\, (M_n^i, M_t^i, \tau_i) \,:\, 1 \leq i \leq k \,\} \quad // \quad F \text{ wins} \\
E_{2,2} & : \ E_2 \text{ occurs and } (m_n^j, m_t^j, \tau_j') \in \{\, (M_n^i, M_t^i, \tau_i) \,:\, 1 \leq i \leq k \,\} \quad // \quad C \text{ wins}
\end{array}
$$

The above partitioning shows that if the event E occurs, then one of E_1, $E_{2,1}$, or $E_{2,2}$ must occur. Note that if E_1 occurs then S wins its game. This is because $m_p^j = M_p^i$ (and therefore $m_o^j = M_o^i$ by consistency requirements on the encoding scheme) and $\tau_j' = \tau_i$ but $\sigma_j' \neq \sigma_i$ (otherwise this would not be a winning forgery for I). Consequently $(m_o^j, m_e^j) = (M_o^i, M_e^i)$ but $\sigma_j' \neq \sigma_i$. Also, if $E_{2,1}$ occurs, then F forges. This is clear from the fact that F never queried its tagging oracle with (m_n^j, m_t^j) (or, if it did, the response wasn't τ_j'). Lastly, if $E_{2,2}$ occurs, then C wins its game. This is because we know that there is some index i such that $(m_n^j, m_t^j) = (M_n^i, M_t^i)$ but $(m_p^j, m_e^j) \neq (M_p^i, M_e^i)$ (the latter comes from the event E_2). Together, this means that the probability that I wins is upper bounded by the sum of the probabilities that C, F, and S win their respective games. The theorem follows for Type 1 CTs.

Let us now consider the other types of cryptographic transforms. For Type 2 we partition E as follows:

E : I wins
E_1 : E occurs and $(m_p^j, m_e^j, \tau_j') \in \{\, (M_p^i, M_e^i, \tau_i) \,:\, 1 \le i \le k \,\}$
$E_{1,1}$: E_1 occurs and there does not exist i such that $(m_p^j, \sigma_j', \tau_j') = (M_p^i, \sigma_i, \tau_i)$ // S wins
$E_{1,2}$: E_1 occurs and there exists i such that $(m_p^j, \sigma_j', \tau_j') = (M_p^i, \sigma_i, \tau_i)$ // C wins
E_2 : E occurs and $(m_p^j, m_e^j, \tau_j') \notin \{\, (M_p^i, M_e^i, \tau_i) \,:\, 1 \le i \le k \,\}$
$E_{2,1}$: E_2 occurs and $(m_n^j, m_t^j, \tau_j') \notin \{\, (M_n^i, M_t^i, \tau_i) \,:\, 1 \le i \le k \,\}$ // F wins
$E_{2,2}$: E_2 occurs and $(m_n^j, m_t^j, \tau_j') \in \{\, (M_n^i, M_t^i, \tau_i) \,:\, 1 \le i \le k \,\}$ // C wins

Above the partitioning of event E is the same as with Type 1 except that we further partition event E_1. If the event $E_{1,1}$ occurs then S wins (since $(m_o^j, m_e^j) = (M_o^i, M_e^i)$ for some index i but $\sigma_j' \neq \sigma_i$). In the case of $E_{1,2}$, in order for I's j-th decapsulation query to be considered a forgery, it must be a replayed packet. The first it would have been accepted (by the consistency requirements on cryptographic transforms). This means that the DecodeB failed to return all $\perp$s in response to its second query with m_p^j, m_e^j, allowing C to win.

For Type 3 we partition E as with Type 2. As with Type 2, when $E_{1,2}$ occurs C will win its game (although C's game with Type 3 encoding schemes is different than its game with Type 2 encoding schemes).

For Type 4 we partition E as follows:

E : I wins
E_1 : E occurs and $(m_n^j, m_t^j) \notin \{(M_n^1, M_t^1), \ldots, (M_n^k, M_t^k)\}$ // F wins
E_2 : E occurs and $(m_n^j, m_t^j) \in \{(M_n^1, M_t^1), \ldots, (M_n^k, M_t^k)\}$
$E_{2,1}$: E_2 occurs and either $k < j$ or $(m_p^j, m_e^j) \neq (M_p^j, M_e^j)$ // C wins
$E_{2,2}$: E_2 occurs and $k \ge j$ and $(m_p^j, m_e^j) = (M_p^j, M_e^j)$
$E_{2,2,1}$: $E_{2,2}$ occurs and $\tau_j' \neq \tau_j$ and $(m_n^j, m_t^j) \notin \{(M_n^1, M_t^1), \ldots, (M_n^{j-1}, M_t^{j-1}),$
$(M_n^{j+1}, M_t^{j+1}), \ldots, (M_n^k, M_t^k)\}$ // F wins
$E_{2,2,2}$: $E_{2,2}$ occurs and $\tau_j' \neq \tau_j$ and $(m_n^j, m_t^j) \in \{(M_n^1, M_t^1), \ldots, (M_n^{j-1}, M_t^{j-1}),$
$(M_n^{j+1}, M_t^{j+1}), \ldots, (M_n^k, M_t^k)\}$ // C wins
$E_{2,2,3}$: $E_{2,2}$ occurs and $\tau_j' = \tau_j$. // S wins

If events E_1 or $E_{2,2,1}$ occur then F wins its game; if events $E_{2,1}$ or $E_{2,2,2}$ occur, then C wins its game; if event $E_{2,2,3}$ occurs, S wins its game. Note that, for $E_{2,2,3}$, we make use of the fact that, as per Construction 5.1, once a forgery attempt is detected, the decapsulation algorithm enters the state $\perp$. This means that prior to the first forgery attempt all the decapsulation-verification queries were in order and, since I's j-th decapsulation-verification oracle query is a forgery, it must be the case that $\sigma_j' \neq \sigma_j$. (Note that, for Type 4 constructions, if the construction didn't enter a halting state we could not guarantee that $\sigma_j' \neq \sigma_j$.) Additionally, by the consistency requirements on the encoding scheme, $m_o^j = M_o^j$.

Let us now consider Type 5. As before, let j denote the index of I's winning decapsulation-verification-oracle query. Let l be the number of decapsulation-verification oracle queries (including the j-th query) that succeeded in decapsulating (i.e., not returning $(\perp, \perp)$). We now partition E as follows:

$$
\begin{array}{ll}
E & : I \text{ wins} \\
E_1 & : E \text{ occurs and } (m_n^j, m_t^j) \notin \{(M_n^1, M_t^1), \ldots, (M_n^k, M_t^k)\} \ // \ \ F \text{ wins} \\
E_2 & : E \text{ occurs and } (m_n^j, m_t^j) \in \{(M_n^1, M_t^1), \ldots, (M_n^k, M_t^k)\} \\
E_{2,1} & : E_2 \text{ occurs and either } k < l \text{ or } (m_p^j, m_e^j) \neq (M_p^l, M_e^l) \ // \ \ C \text{ wins} \\
E_{2,2} & : E_2 \text{ occurs and } k \geq l \text{ and } (m_p^j, m_e^j) = (M_p^l, M_e^l) \\
E_{2,2,1} & : E_{2,2} \text{ occurs and } \tau'_j \neq \tau_l \text{ and } (m_n^j, m_t^j) \notin \{(M_n^1, M_t^1), \ldots, (M_n^{l-1}, M_t^{l-1}), \\
& \quad (M_n^{l+1}, M_t^{l+1}), \ldots, (M_n^k, M_t^k)\} \ // \ \ F \text{ wins} \\
E_{2,2,2} & : E_{2,2} \text{ occurs and } \tau'_j \neq \tau_l \text{ and } (m_n^j, m_t^j) \in \{(M_n^1, M_t^1), \ldots, (M_n^{l-1}, M_t^{l-1}), \\
& \quad (M_n^{l+1}, M_t^{l+1}), \ldots, (M_n^k, M_t^k)\} \ // \ \ C \text{ wins} \\
E_{2,2,3} & : E_{2,2} \text{ occurs and } \tau'_j = \tau_l. \ // \ \ S \text{ wins}
\end{array}
$$

If events E_1 or $E_{2,2,1}$ occur then F wins its game. Furthermore, if events $E_{2,1}$ or $E_{2,2,2}$ occur, then C wins its game. And if event $E_{2,2,3}$ occurs, S wins its game. To see that S wins when $E_{2,2,3}$ occurs, we use the consistency requirement on Type 5 encoding schemes that tell us that $m_o^j = M_o^l$. Furthermore, it must be the case that $\sigma'_j \neq \sigma_l$ since otherwise the j-th decapsulation-verification query would not be a forgery. ▌

E Encode-then-MtE

Privacy. We now state out result for Encode-then-MtE constructions. This theorem is interpreted in Result 6.2.

Theorem E.1 (Privacy of Encode-then-MtE) Let $\mathcal{SE}$, $\mathcal{MA}$, and $\mathcal{EC}$, respectively, be an encryption, a message authentication, and an MtE encoding scheme. Let CT be the cryptographic transform associated to them as per Construction 6.1. Then, given any ct-priv-cpa adversary S against CT, there exists an adversary A such that

$$\mathbf{Adv}_{\mathsf{CT}}^{\text{ct-priv-cpa}}(S) \ \leq \ \mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cpa}}(A) \ .$$

Furthermore, A use the same resources as S except that its input to its oracle may be of different lengths than those of S (due to the encoding). If $\mathcal{EC}$ is nonce-respecting-for-encryption (resp., length-based IV-respecting-for-encryption or random-IV-respecting-for-encryption), then A will be nonce-respecting (resp., length-based IV-respecting or random-IV-respecting). ▌

The proof is similar to that of Theorem 4.5 in [5] and is omitted here. We remark that the proof relies on the fact that M_p is independent of the content of the messages and that, when run with the same random tape, the M_o values will also be the same. (These are consistency requirements for MtE encoding schemes specified in Section 6.)

Integrity. We begin by formalizing a new property for Encode-then-MtE CTs, analogous to the e&m-sp property for Encode-then-E&M CTs. In most situations, one does not actually need to manipulate this definition but must merely invoke Lemma E.3.

Definition E.2 Fix $n \in \{1, \ldots, 5\}$. Let $\mathcal{SE}$, $\mathcal{MA}$, and $\mathcal{EC}$, respectively, be an encryption, a message authentication, and an MtE encoding scheme. Let $\mathsf{CT} = (\mathsf{KG}, \mathsf{Encap}, \mathsf{Decap})$ be a Type n cryptographic transform associated to them as per Construction 6.1. Let A be an adversary with access to an encapsulation oracle $\mathsf{Encap}_K(\cdot, \cdot)$ and a decapsulation oracle $\mathsf{Decap}_K(\cdot)$. Let (M_a^i, M_s^i)

denote the adversary's i-th encapsulation oracle query, $(M_p^i, M_o^i, M_n^i, M_e^i, M_t^i)$ denote the encoding of that query, τ_i denote the intermediate tag, and $\langle M_p^i, \sigma_i \rangle$ denote the returned ciphertext. Let $\langle m_p^i, \sigma_i' \rangle$ denote the i-th decapsulation query (assuming it is parseable), τ_i' denote the intermediate tag, and $m_o^i, m_n^i, m_e^i, m_t^i, m_a^i, m_s^i$ denote the internal values in the decapsulation process (or $\perp$ if an error occurs during decapsulation). A "wins" if it makes a decapsulation query $\langle m_p^j, \sigma_j' \rangle$ such that $(m_o^j, m_e^j, \tau_j') = (M_o^i, M_e^i, \tau_i)$ for some $i \in \{1, \ldots, k\}$ but $\sigma_j' \neq \sigma_i$ (where k is the number of $\mathsf{Encap}_K(\cdot,\cdot)$ oracle queries made by A before A's j-th decapsulation query). We define the $\mathsf{mte\text{-}sp}$ *advantage* of $\mathsf{mte\text{-}sp}$ adversary A as

$$\mathbf{Adv}^{\text{mte-sp}}_{\mathsf{CT}}(A) \ = \ \Pr\left[\, K \xleftarrow{R} \mathsf{KG} \,:\, A \text{ "wins" } \right] . \ ▮$$

As in Appendix D, we present a lemma showing that if the underlying encryption scheme is length preserving, then an adversary cannot win the game described above.

Lemma E.3 Fix $n \in \{1, \ldots, 5\}$. Let $\mathcal{SE}$, $\mathcal{MA}$, and $\mathcal{EC}$, respectively, be an encryption, a MAC, and a Type n MtE encoding scheme. Let $\mathsf{CT} = (\mathsf{KG}, \mathsf{Encap}, \mathsf{Decap})$ be a Type n cryptographic transform associated to them as per Construction 6.1. Let A be an $\mathsf{mte\text{-}sp}$ adversary. If $\mathcal{SE}$'s encryption operation is length-preserving, then

$$\mathbf{Adv}^{\text{mte-sp}}_{\mathsf{CT}}(A) \ = \ 0 \ . \ ▮$$

We now state our integrity result for Encode-then-MtE constructions, which is interpreted in Result 6.3.

Theorem E.4 (Integrity of Encode-then-MtE) Let $\mathcal{SE}$, $\mathcal{MA}$, and $\mathcal{EC}$, respectively, be an encryption, a message authentication, and an MtE encoding scheme. Let CT be a Type n cryptographic transform associated to them as per Construction 6.1. Then, given any ct-int-ctxt$n$ adversary I against CT, there exist adversaries F, C, and S such that

$$\mathbf{Adv}^{\text{ct-int-ctxt}n}_{\mathsf{CT}}(I) \leq \mathbf{Adv}^{\text{uf-cma}}_{\mathcal{MA}}(F) + \mathbf{Adv}^{\text{mte-sec}n}_{\mathcal{EC}}(C) + \mathbf{Adv}^{\text{mte-sp}}_{\mathsf{CT}}(S) \ .$$

Furthermore, F, C, and S use the same resources as I except that F's messages to its oracles may be of different lengths than I's queries to its oracles (due to encoding) and C's messages to its decoding oracle may have slightly different lengths than I's decapsulation queries. If $\mathcal{EC}$ is nonce-respecting-for-MACing, then F will be nonce-respecting. ▮

Proof: The proof is based on the proof of Theorem D.4 for Encode-then-E&M constructions. The partitioning of event E for Type 2 and Type 3 differs slightly from the partitioning we used in the proof of Theorem D.4. The difference is because in the Encode-then-MtE construction the tag is not sent in the clear. The revised partitioning is as follows:

E : I wins
E_1 : E occurs and $(m_p^j, m_e^j, \tau_j') \in \{\, (M_p^i, M_e^i, \tau_i) \,:\, 1 \leq i \leq k \,\}$
$E_{1,1}$: E_1 occurs and there does not exist i such that $(m_p^j, \sigma_j') = (M_p^i, \sigma_i)$ // S wins
$E_{1,2}$: E_1 occurs and there exists i such that $(m_p^j, \sigma_j') = (M_p^i, \sigma_i)$ // C wins
E_2 : E occurs and $(m_p^j, m_e^j, \tau_j') \notin \{\, (M_p^i, M_e^i, \tau_i) \,:\, 1 \leq i \leq k \,\}$
$E_{2,1}$: E_2 occurs and $(m_n^j, m_t^j, \tau_j') \notin \{\, (M_n^i, M_t^i, \tau_i) \,:\, 1 \leq i \leq k \,\}$ // F wins
$E_{2,1}$: E_2 occurs and $(m_n^j, m_t^j, \tau_j') \in \{\, (M_n^i, M_t^i, \tau_i) \,:\, 1 \leq i \leq k \,\}$ // C wins

The partitioning of E for Type 1, Type 4, and Type 5 is the same as in the proof of Theorem D.4. ∎

F Encode-then-EtM

Privacy. We now state our result for Encode-then-EtM constructions. This theorem is interpreted in Result 7.2.

Theorem F.1 (Privacy of Encode-then-EtM) Let $\mathcal{SE}$, $\mathcal{MA}$, and $\mathcal{EC}$, respectively, be an encryption, a message authentication, and an EtM encoding scheme. Let CT be the cryptographic transform associated to them as per Construction 7.1. Then, given any ct-priv-cpa adversary S against CT, there exists an adversary A such that

$$\mathbf{Adv}_{\mathsf{CT}}^{\text{ct-priv-cpa}}(S) \;\leq\; \mathbf{Adv}_{\mathcal{SE}}^{\text{ind-cpa}}(A)\;.$$

Furthermore, A use the same resources as S except that its inputs to its oracle may be of different lengths than those of S (due to the encoding). If $\mathcal{EC}$ is nonce-respecting-for-encryption (resp., length-based IV-respecting-for-encryption or random-IV-respecting-for-encryption), then A will be nonce-respecting (resp., length-based IV-respecting or random-IV-respecting). ∎

The proof is similar to that of Theorem 4.7 in [5]. We note that the proof relies on the fact that if the encoding algorithm is run using the same random tape, on two pairs of messages $(M_a, M_s), (M_a, N_s)$ such that $|M_s| = |N_s|$, then the resulting values for M_p, M_o, M_n and M_t will be the same. (These are consistency requirements for EtM encoding schemes specified in Section 7.)

Integrity. Our integrity results for Encode-then-EtM CTs is presented below. This theorem is interpreted in Result 7.3.

Theorem F.2 (Integrity of Encode-then-EtM) Fix $n \in \{1, \ldots, 5\}$. Let $\mathcal{SE}$, $\mathcal{MA}$, and $\mathcal{EC}$, respectively, be an encryption, a message authentication, and an EtM encoding scheme. Let CT be a Type n cryptographic transform associated to them as per Construction 7.1. Then, given any ct-int-ctxt$n$ adversary I against CT, there exist adversaries F and C such that

$$\mathbf{Adv}_{\mathsf{CT}}^{\text{ct-int-ctxt}n}(I) \leq \mathbf{Adv}_{\mathcal{MA}}^{\text{uf-cma}}(F) + \mathbf{Adv}_{\mathcal{EC}}^{\text{etm-sec}n}(C).$$

Furthermore, F and C use the same resources as I except that F's messages to its oracles may be of different lengths than I's queries to its oracles (due to encoding) and C's messages to its decoding oracle may have slightly different lengths than I's decapsulation queries. If $\mathcal{EC}$ is nonce-respecting-for-MACing, then F will be nonce-respecting. ∎

Proof: The proof is similar to that of Theorem D.4 and Theorem E.4.

Let F and C be adversaries that run I and reply to I's oracle queries using their own oracles. Let (M_a^i, M_s^i) denote I's i-th encapsulation query, let $(M_p^i, M_o^i, M_n^i, M_e^i, M_t^i)$ denote the encoding of that query, and let $\langle M_p^i, \sigma_i, \tau_i\rangle$ denote the returned ciphertext. Let $\langle m_p^i, \sigma_i', \tau_i'\rangle$ denote the i-th decapsulation-verification query (assuming it can be parsed), and $m_o^i, m_n^i, m_t^i, m_e^i, m_a^i, m_s^i$ denote the internal values in the decapsulation process (or $\perp$ if an error occurs during decapsulation). Assume that I wins and let j denote the index of its (first) winning decapsulation-verification query and k denote the number of encapsulation queries performed at the time I wins. We will prove that either F or C also wins its game.

For Type 1, Type 2, Type 3, and Type 5 CTs, we consider the following events:

E : I wins
E_1 : E occurs and $(m_n^j, m_t^j, \sigma'_j, \tau'_j) \notin \{\, (M_n^i, M_t^i, \sigma_i, \tau_i) \,:\, 1 \leq i \leq k \,\}$ // F wins
E_2 : E occurs and $(m_n^j, m_t^j, \sigma'_j, \tau'_j) \in \{\, (M_n^i, M_t^i, \sigma_i, \tau_i) \,:\, 1 \leq i \leq k \,\}$ // C wins

Note that if event E occurs then either E_1 or E_2 must occur. Event E_1 implies that the query $m_n^j, \langle m_t^j, \sigma'_j \rangle, \tau'_j$ is accepted by the verification oracle (otherwise $\langle m_p^j, \sigma'_j, \tau'_j \rangle$ would not be a winning query for I) and is such that τ'_j was never returned by the tagging oracle as an answer to query $m_n^j, \langle m_t^j, \sigma'_j \rangle$. Therefore, if E_1 occurs then F forges.

Assume that event E_2 occurs. Then there exists an index $i \leq k$ such that $(m_n^j, m_t^j, \sigma'_j, \tau'_j) = (M_n^i, M_t^i, \sigma_i, \tau_i)$. For Type 1 CTs, it must be the case that $m_p^j \neq M_p^i$ (otherwise $\langle m_p^j, \sigma'_j, \tau'_j \rangle$ would not be a winning query for I). Since $M_p^i \neq m_p^j$ and $(M_n^i, M_t^i) = (m_n^j, m_t^j)$, C wins. For Type 2 and Type 3 CTs, C also wins if $m_p^j \neq M_p^i$. If $m_p^j = M_p^i$ then for Type 2 CTs, it must be the case that $\langle m_p^j, \sigma'_j, \tau'_j \rangle$ is a replayed packet (otherwise this would not be a winning query for I). The consistency requirements for the encoding scheme and the encryption scheme, imply that (m_p^j, m_e^j) was decoded correctly (i.e., without returning $(\perp, \perp)$) twice. Therefore, C also wins in this case. For Type 3 CTs, $m_p^j = M_p^i$ implies that $\langle m_p^j, \sigma'_j, \tau'_j \rangle$ is a replayed or out-of-order packet (otherwise this would not be a winning query for I). Again, the consistency requirements for the encoding scheme and the encryption scheme, imply that C wins. For Type 4 CTs, it must be the case that either $i \neq j$ or $m_p^j \neq M_p^j$ (if $i = j$ and $m_p^j = M_p^j$, then $j \leq k$ and $\langle m_p^j, \sigma'_j, \tau'_j \rangle = \langle M_p^j, \sigma_j, \tau_j \rangle$, which contradicts the assumption that $\langle m_p^j, \sigma'_j, \tau'_j \rangle$ is a winning query for I). In both of these cases C wins. Finally, for Type 5 CTs, let l be the number of decapsulation-verification oracle queries prior to the j-th one that succeeded in decapsulating (i.e., did not return $(\perp, \perp)$). Then it must be the case that either $l \neq i - 1$ or $m_p^j \neq M_p^i$ (if $l = i - 1$ and $m_p^j = M_p^i$, then $l + 1 \leq k$ and $\langle m_p^j, \sigma'_j, \tau'_j \rangle = \langle M_p^{l+1}, \sigma_{l+1}, \tau_{l+1} \rangle$, contradicting the assumption that $\langle m_p^j, \sigma'_j, \tau'_j \rangle$ is a winning query for I). In both of these cases C wins. Hence for all CT types, E_2 implies that C wins. ▮