Fast Message Franking: From Invisible Salamanders to Encryptment[*]

Yevgeniy Dodis[1], Paul Grubbs[2,†], Thomas Ristenpart[2], Joanne Woodage[3,†]

[1] New York University [2] Cornell Tech

[3] Royal Holloway, University of London

Abstract

Message franking enables cryptographically verifiable reporting of abusive content in end-to-end encrypted messaging. Grubbs, Lu, and Ristenpart recently formalized the needed underlying primitive, what they call compactly committing authenticated encryption (AE), and analyzed the security of a number of approaches. But all known secure schemes are still slow compared to the fastest standard AE schemes. For this reason Facebook Messenger uses AES-GCM for franking of attachments such as images or videos.

We show how to break Facebook's attachment franking scheme: a malicious user can send an objectionable image to a recipient but that recipient cannot report it as abuse. The core problem stems from use of fast but non-committing AE, and so we build the fastest compactly committing AE schemes to date. To do so we introduce a new primitive, called encryptment, which captures the essential properties needed. We prove that, unfortunately, schemes with performance profile similar to AES-GCM won't work. Instead, we show how to efficiently transform Merkle-Damgärd-style hash functions into secure encryptments, and how to efficiently build compactly committing AE from encryptment. Ultimately our main construction allows franking using just a single computation of SHA-256 or SHA-3. Encryptment proves useful for a variety of other applications, such as remotely keyed AE and concealments, and our results imply the first single-pass schemes in these settings as well.

1 Introduction

End-to-end encrypted messaging systems including WhatsApp [49], Signal [46], and Facebook Messenger [15] have increased in popularity — billions of people now rely on them for security. In these systems, intermediaries including the messaging service provider should not be able to read or modify messages. Providers simultaneously want to support abuse reporting: should one user send another a harmful message, image, or video, the recipient should be able to report the content to the provider. End-to-end encryption would seem to prevent the provider from verifying that the reported message was the one sent.

Facebook suggested a way to navigate this tension in the form of message franking [16, 36]. The idea is to enable the recipient to cryptographically prove to the service provider that the reported message was the one sent. Grubbs, Lu, and Ristenpart (GLR) [19] provided the first formal treatment of the problem, and introduced compactly committing authenticated encryption

[*]A preliminary version of this paper appeared at CRYPTO 2018. This is the full version.

[†]Contact authors.

with associated data (ccAEAD) as the key primitive. A secure ccAEAD scheme is symmetric encryption for which a short portion of the ciphertext serves as a cryptographic commitment to the underlying message (and associated data). GLR detailed appropriate security notions, and proved the main Facebook message franking approach achieves them. They also introduced a faster custom ccAEAD scheme called Committing Encrypt-and-PRF (CEP).

The Facebook scheme composes HMAC (serving the role of a commitment) with a standard encrypt-then-MAC AEAD scheme. Their scheme therefore requires a full three cryptographic passes over messages. The CEP construction gets this down to two. But even that does not match the fastest standard AE schemes such as AES-GCM [33] and OCB [40]. These require at most one blockcipher call (on the same key) per block of message and some arithmetic operations in GF(2^n), which are faster than a blockcipher invocation. As observed by GLR, however, these schemes are not compactly committing: one can find two distinct messages and two encryption keys that lead to the same tag. This violates what they call receiver binding, and could in theory allow a malicious recipient to report a message that was never sent.

Existing ccAEAD schemes are not considered fast enough for all applications of message franking by practitioners [36]. Facebook Messenger does not use the ccAEAD scheme mentioned above to directly encrypt attachments, rather using a kind of hybrid encryption combining ccAEAD of a symmetric key that is in turn used with AES-GCM to encrypt the attachment. Use of AES-GCM does not necessarily seem problematic despite the GLR results; the latter do not imply any concrete attack on Facebook's system.

Breaking Facebook's attachment franking. Our first contribution is to show an attack against Facebook's attachment franking scheme. The attack enables a malicious sender to transmit an abusive attachment (e.g., an objectionable image or video) to a receiver so that: (1) the recipient receives the attachment (it decrypts correctly), yet (2) reporting the abusive message fails — Facebook's systems essentially "lose" the abusive image, rendering them invisible from the abuse handling team. Instead what gets reported to Facebook is a different, innocuous image. See Figure 3.

Perhaps confusingly, our attack does not violate the primary reason for requiring receiver binding in committing AE, namely preventing a malicious recipient from framing a user as having sent a message they didn't send. Facebook's attachment franking appears to prevent this. Instead, the attack violates what GLR call sender binding security: a malicious sender should not be able to force an abusive message to be received by the recipient, yet that recipient can't report it properly. Nevertheless, the root cause of this vulnerability in Facebook's case is the use of an AE scheme that is not a binding commitment to its message or, equivalently in this context, that is not a robust encryption scheme [1, 17, 18].

Briefly, Facebook uses a cryptographic hash of the AES-GCM ciphertext, along with a randomly-generated value, as an identifier for the attachment. For a given abusive message, our attack efficiently finds two keys and a ciphertext, such that the first key decrypts the ciphertext to the abusive attachment while the other key successfully decrypts the same ciphertext, but to another innocuous attachment. The malicious sender transmits two messages with the different keys but the same attachment ciphertext. Facebook's systems deduplicate the two attachments, and the report will only include the non-abusive image.

We responsibly disclosed this vulnerability to Facebook, and in fact they helped us understand how our attack works against their systems (much of the abuse handling code is server-side and closed source). The severity of the issue led them to patch their (server-side) systems and to award us a bug bounty. Their fix is ad hoc and involves deduplicating more carefully. But the vulnerability would have been avoided in the first place by using a fast ccAEAD scheme that provided the binding security properties implicitly assumed of, but not actually provided by, AES-GCM.

Towards faster ccAEAD schemes: encryptment. This message franking failure motivates the need for faster schemes. As mentioned, the best known secure ccAEAD scheme from GLR is two pass, requiring computing both HMAC and AES-CTR mode (or similar) over the message. The fastest standard AE schemes [25,33,40], however, require just a single pass using a blockcipher with a single key. Can we build ccAEAD schemes that match this performance?

To tackle this question we first abstract out the core technical challenge underlying ccAEAD: building a one-time encryption mechanism that simultaneously encrypts and compactly commits to the message. We formalize this in a new primitive that we call *encryptment*. An encryptment of a message using a key K_{EC} is a pair $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$ where C_{EC} is a ciphertext and B_{EC} is a binding tag. By compactness we require that $|B_{\mathsf{EC}}|$ is independent of the length of the message. Decryption takes as input $K_{\mathsf{EC}}, C_{\mathsf{EC}}, B_{\mathsf{EC}}$ and returns a message (or $\perp$). Finally, there is a verification algorithm that takes a key, a message, and a binding tag, and determines whether the tag is a commitment to the message. Encryptment supports associated data also, but we defer the details to the body.

We introduce security notions for encryptment. These include a real-or-random style confidentiality goal in which the adversary must distinguish between a single encryptment and an appropriate-length sequence of random bits. Additionally we require sender binding and receiver binding notions like those from GLR (but adapted to the encryptment syntax), and finally a strong correctness property that is easy to meet. Comparatively, GLR require many-time confidentiality and integrity notions in addition to various binding notions.

Therefore encryptment is substantially simpler than ccAEAD, making analyses easier and, we think, design of constructions more intuitive. At the same time, we will be able to build ccAEAD from encryptment using simple, efficient transforms. In the other direction, we show that one can also build encryptment from ccAEAD, making the two primitives equivalent from a theoretical perspective. Encryptment also turns out to be the "right" primitive for a number of other applications: robust authenticated encryption [1, 17, 18], concealments [14], remotely keyed authenticated encryption [14], and perhaps even more.

Fast encryptment from fixed-key blockciphers? Given a simpler formulation in hand, we turn to building fast schemes. First, we show a negative result: encryptment schemes cannot match the efficiency profile of OCB or AES-GCM. In fact we rule out any scheme that uses just a single blockcipher invocation for each block of message, with some fixed small set of keys.

The negative result makes use of a connection between encryptment and collision-resistant (CR) hashing. Because encryptment schemes are deterministic, we can think of the computation of a binding tag B_{EC} as a deterministic function $F(K_{\mathsf{EC}}, M)$ applied to the key and message; verification simply checks that $F(K_{\mathsf{EC}}, M) = B_{\mathsf{EC}}$. Then, receiver binding is achieved if and only if F is CR: the adversary shouldn't be able to find $(K_{\mathsf{EC}}, M) \neq (K'_{\mathsf{EC}}, M')$ such that $F(K_{\mathsf{EC}}, M) = F(K'_{\mathsf{EC}}, M')$.

Given this connection, we can exploit previous work on ruling out fixed-key blockcipher-based CR hashing [42,43,45]. A simple corollary of [43, Thm. 1] is that one cannot prove receiver binding security for any rate-1 fixed-key blockcipher-based encryptment. (Rate-1 meaning one blockcipher call per block of message.) Since OCB and AES-GCM fall into this category of rate-1, they don't work, but neither do other similar blockcipher-based schemes. Our negative result also rules out rate-1 ccAEAD, due to our aforementioned result that (fast) ccAEAD implies (fast) encryptment.

One-pass encryptment from hashing. Given the connection just mentioned, it is natural to turn to CR hashing as a starting point for building as-fast-as-possible encryptment. We do so and show how to achieve secure encryptment using just a single pass of a secure cryptographic hash function. The encryptment can be viewed as a mode of operation of a fixed-input-length compression function, such as the one underlying SHA-256 or other Merkle-Damgärd style constructions.

Let $f(x, y)$ be a compression function on two n-bit inputs and with output an n-bit string. Then our HFC (hash function chaining) encryptment works as shown in Figure 9. Basically one hashes $K_{\mathsf{EC}} \,\|\, (M_1 \oplus K_{\mathsf{EC}}) \,\|\, \cdots \,\|\, (M_2 \oplus K_{\mathsf{EC}})$ using a standard iteration of f. But, additionally, one uses the intermediate chaining values as pads to encrypt the message blocks. Decryption simply computes the hash, recovering message blocks as it goes.

We prove that our HFC scheme is a secure encryptment. Binding is inherited from the CR of the underlying hash function. We show confidentiality assuming $f(x, y \oplus K_{\mathsf{EC}})$ is a related-key-attack-secure pseudorandom function (RKA-PRF) [4] when keyed by K_{EC}. For standard designs, such as the Davies-Meyer construction $f(x, y \oplus K_{\mathsf{EC}}) = E(y \oplus K_{\mathsf{EC}}, x) \oplus x$, we can reduce RKA-PRF security to RKA-PRP security of the underlying blockcipher E. This property is already an active target of cryptographic analysis for standard E (such as AES), giving us confidence in the assumption. Because SHA-256 uses a DM-style compression function, this also gives confidence for using SHA-256 (or SHA-384, SHA-512).

From a theoretical perspective, one might want to avoid relying on RKA security (compared to standard PRF security). We discuss approaches for doing so in the body, but the resulting constructions are not as fast or elegant as HFC.

HFC has some features in common with the Duplex authenticated-encryption mode [8] using Keccak (SHA-3) [7]. In fact the Duplex mode gives rise to a secure encryptment scheme as well; see Appendix F for a discussion. The way we key in HFC is also similar to the Halevi-Krawczyk construction for reducing the assumptions needed on hash functions in digital signature settings [22], but the keying serves a different role here and their analysis techniques are not applicable.

From encryptment to ccAEAD. We show two efficient transforms for building a ccAEAD scheme given a secure encryptment. First consider doing so given also a secure (standard) AE scheme. To encrypt a message M, first generate a random key K_{EC} and then compute an encryptment $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$ for K_{EC}, M. Encrypt K_{EC} under the long-lived AE key K using as associated data the binding tag B_{EC}. The resulting ciphertext is the AE ciphertext (including its authentication tag) along with $C_{\mathsf{EC}}, B_{\mathsf{EC}}$. We prove that this transformation provides the multi-opening confidentiality and integrity goals for ccAEAD of GLR, assuming the standard security of the AE scheme and the aforementioned security goals are met for the encryptment scheme.

One can instead use just two additional PRF calls to securely convert an encryptment scheme to a ccAEAD scheme. One can, for example, instantiate the PRF with the SHA-256 compression function, to have a total cost of at most $m + 4$ SHA-256 compression function calls for a message that can be parsed into m blocks of 256 bits. See Section 7.3.

Our approach of hashing-based ccAEAD has a number of attractive features. HFC works with any hash function that iterates a secure compression function, giving us a wide variety of options for instantiation. Because of our simplified formalization via encryptment, the security proofs are modular and conceptually straightforward. As already mentioned it is fast in terms of the number of underlying primitive calls. If instantiated using SHA-256, one can use the SHA hardware instructions [20] now supported on some AMD and ARM processors, and that are likely to be incorporated in future Intel processors. Finally, HFC-based ccAEAD is simple to implement.

Other applications. Encryptment proves a useful abstraction for other applications as well. In Section 8, we show how it suffices for building concealments [14] (a conceptually similar, but distinct, primitive) which, in turn, can be used to build remotely keyed AE [14]. Previous constructions of these required two passes over the message. Our new encryptment-based approach gives the first single-pass concealments and remotely keyed AE. Finally, encryptment schemes give rise to robust AE [17] via some of our transforms mentioned above. We expect that encryptment will find further applications in the future.

2 Definitions and Preliminaries

Preliminaries. For an alphabet Σ, we let Σ^* denote the set of all strings of symbols from that alphabet, and let Σ^n denote the set of all such strings of length n. For a string $x \in \Sigma^*$, we write $|x|$ to denote the length of x. We let ε denote the empty string, and $\perp$ denote a distinguished error symbol. We write $x \leftarrow\!\!\$\, \mathcal{X}$ to denote choosing a uniformly random element from the set $\mathcal{X}$.

We define the XOR of two strings of different lengths to return the XOR of the shorter string and the truncation of the longer string to the length of the shorter string. Our proofs assume a RAM model of computation where most operations are unit cost. We use big-O notation $\mathcal{O}(\cdot)$ to hide small constants related to data structures (e.g., tables of queries) used by reductions.

For a deterministic algorithm A, we write $y \leftarrow A(x_1, \dots)$ to denote running A on inputs $x_1, \dots$ to produce output y. For a probabilistic algorithm A with coin space $\mathcal{C}$, we write $y \leftarrow\!\!\$\, A(x_1, \dots)$ to denote choosing coins $c \leftarrow\!\!\$\, \mathcal{C}$ and returning $y \leftarrow A(x_1, \dots; c)$, where $y \leftarrow A(x_1, \dots; c)$ denotes running A on the given inputs with coins c fixed, to deterministically produce output y.

Collision-resistant functions. Let $\mathcal{H} \colon \mathit{Dom} \to \{0,1\}^n$ be a function on domain $\mathit{Dom} \subset \{0,1\}^*$. The collision resistance game CR has $\mathcal{A}$ run and output a pair of messages X, X'. If analysis is with respect to an ideal primitive such as an ideal cipher, then $\mathcal{A}$ is given oracle access to this primitive also. The game outputs true if $\mathcal{H}(X) = \mathcal{H}(X')$ and $X \neq X'$. The CR advantage of an adversary $\mathcal{A}$ against $\mathcal{H}$ is defined $\mathbf{Adv}^{\mathrm{cr}}_{\mathcal{H}}(\mathcal{A}) = \Pr\left[\, \mathrm{CR}^{\mathcal{A}}_{\mathcal{H}} \Rightarrow \mathsf{true} \,\right]$, where the probability is over the coins of $\mathcal{A}$ and those of any ideal primitive. We measure the efficiency of the attacker in terms of their resources, e.g., run time or number of queries made to some underlying primitive.

Authenticated encryption. A nonce-based authenticated encryption scheme with associated data (AEAD) scheme is a tuple of algorithms $\mathsf{SE} = (\mathsf{kg}, \mathsf{enc}, \mathsf{dec})$, with associated key space $\mathcal{K} \subseteq \Sigma^*$, nonce space $\mathcal{N} \subseteq \Sigma^*$, header space $\mathcal{H} \subseteq \Sigma^*$, message space $\mathcal{M} \subseteq \Sigma^*$, and ciphertext space $\mathcal{C} \subseteq \Sigma^*$, defined as follows. The key generation algorithm takes random coins as input, and outputs a secret key $K \in \mathcal{K}$. We typically have kg choose $K \leftarrow\!\!\$\, \mathcal{K}$ and return K. Encryption enc is a deterministic algorithm which takes as input a key $K \in \Sigma^*$, a nonce $N \in \Sigma^*$, a header $H \in \Sigma^*$, and a message $M \in \Sigma^*$, and outputs either a ciphertext $C \in \mathcal{C}$ or the error symbol $\perp$. We require that for all tuples $(K, N, H, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{H} \times \mathcal{M}$ it holds that $\mathsf{enc}(K, N, H, M) \neq \perp$. Decryption dec is a deterministic algorithm which takes as input a key $K \in \Sigma^*$, header $H \in \Sigma^*$, and a ciphertext $C \in \Sigma^*$, and outputs either a message M or the error symbol $\perp$. We say that an AEAD scheme is *perfectly correct* if for all $(K, N, H, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{H} \times \mathcal{M}$ it holds that $\mathsf{dec}(K, N, H, \mathsf{enc}(K, N, H, M)) = M$.

A *randomized AEAD scheme* is defined analogously to the definition of a nonce-based AEAD scheme above, except all references to use of a nonce are removed, and instead enc takes as input random coins R sampled from the coin space $\mathcal{R} \subseteq \Sigma^*$ associated to the scheme. We say that a randomized AEAD scheme is perfectly correct if for all $(K, R, H, M) \in \mathcal{K} \times \mathcal{R} \times \mathcal{H} \times \mathcal{M}$ it holds that $\mathsf{dec}(K, H, \mathsf{enc}(K, H, M, R)) = M$.

Commitment schemes. A commitment scheme with verification $\mathsf{CS} = (\mathsf{Com}, \mathsf{VerC})$ is a pair of algorithms. Associated to any such scheme is a message space $\mathcal{M} \subseteq \Sigma^*$, an opening space $\mathcal{D} \subseteq \Sigma^*$, and a commitment space $\mathcal{C} \subseteq \Sigma^*$. The randomized algorithm Com takes as input a message $M \in \mathcal{M}$ and outputs a pair $(c, d) \in \mathcal{C} \times \mathcal{D}$. The deterministic algorithm VerC takes as input a tuple $(c, d, M) \in \mathcal{C} \times \mathcal{D} \times \mathcal{M}$ and outputs a bit. We assume that VerC returns 0 if $(c, d, M) \notin \mathcal{C} \times \mathcal{D} \times \mathcal{M}$. We require that the commitments c returned by CS are of some fixed length t. A commitment scheme is correct if for all $M \in \mathcal{M}$ it holds that $\Pr[\mathsf{VerC}(\mathsf{Com}(M), M) = 1] = 1$, where the probability is over the coins of Com. One can formalize the binding security of a commitment scheme as a game. Game $\mathrm{vBIND}^{\mathcal{A}}_{\mathsf{CS}}$ runs the adversary $\mathcal{A}$, who outputs a tuple $((d, M), (d', M'), c)$. The game then runs $b \leftarrow \mathsf{VerC}(c, d, M)$ and $b' \leftarrow \mathsf{VerC}(c, d', M')$. Finally, the game outputs true if $M \neq M'$ and $b = b' = 1$, and false otherwise. To a commitment scheme CS and adversary $\mathcal{A}$ we associate the vBIND advantage

$$\mathbf{Adv}^{\text{v-bind}}_{\mathsf{CS}}(\mathcal{A}) = \Pr\left[\, \mathrm{vBIND}^{\mathcal{A}}_{\mathsf{CS}} \Rightarrow \mathsf{true} \,\right] ,$$

where the probability is over the coins of $\mathcal{A}$ and Com.

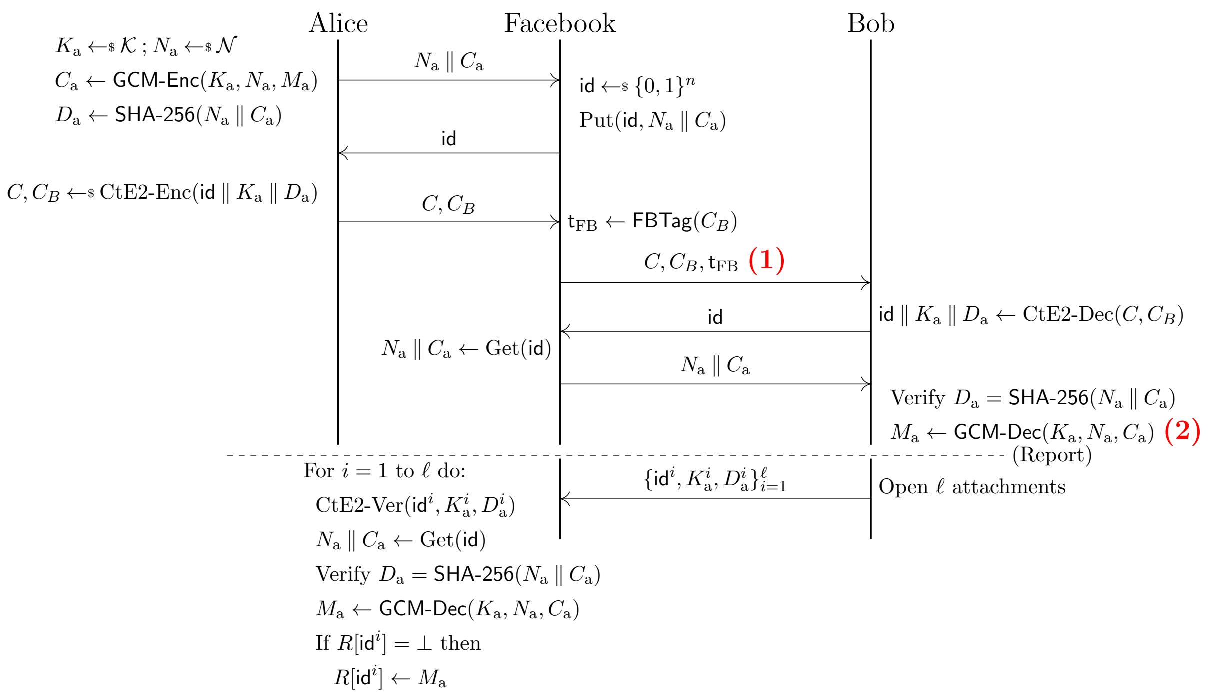
3 Invisible Salamanders: Breaking Facebook's Franking

In this section we demonstrate an attack against Facebook's message franking. Facebook uses AES-GCM to encrypt attachments sent via Secret Conversations [16], the end-to-end encryption feature in Messenger. The attack creates a "colliding" GCM ciphertext which decrypts to an abusive attachment via one key and an innocuous attachment via another. This combined with the behavior of Facebook's server-side abuse report generation code prevents abusive messages from being reported to Facebook. Since messages in Secret Conversations are called "salamanders" by Facebook (perhaps inspired by the Axolotl ratchet used in Signal, named for an endangered salamander), ensuring Facebook does not see a message essentially makes it an *invisible salamander*. We responsibly disclosed the vulnerability to Facebook. They have remediated it and have given us a bug bounty for reporting the issue.

Facebook's attachment franking. A diagram of Facebook's franking protocol for attachments (e.g., images and videos) is in Figure 1. The protocol uses Facebook's ccAEAD scheme for chat messages described in [16, 36] and analyzed in [19] (there called CtE2) as a subroutine. (Readers interested in the specifics of CtE2 should consult GLR [19]; the only salient detail is that it is secure as a ccAEAD scheme.) Some encryption and HMAC keys, as well as some other details like headers and associated data not important to the presentation of the protocol or our attack, have been removed for simplicity in the diagram and prose below. Consult [16, 19] for additional details. For ease of exposition we divide the protocol into three phases: the *sending phase* involving the sender Alice and Facebook, the *receiving phase* involving the receiver Bob and Facebook, and the *reporting phase* between Bob and Facebook.

Sending phase: Alice begins the sending phase with an attachment M_a to send to Bob. In the first part of the sending phase, Alice generates a key K_a and nonce N_a and encrypts M_a using AES-GCM (described in pseudocode in Figure 2) to obtain a ciphertext C_a. The sender computes the SHA-256 digest D_a of $N_\mathrm{a} \parallel C_\mathrm{a}$ and sends Facebook $N_\mathrm{a} \parallel C_\mathrm{a}$ for storage. Facebook generates a random identifier id and puts $N_\mathrm{a} \parallel C_\mathrm{a}$ in a key-value data structure with key id. Facebook then sends id to Alice. In the second part of the sending phase, Alice encrypts the message $\mathsf{id} \parallel K_\mathrm{a} \parallel D_\mathrm{a}$ using CtE2 to obtain the ccAEAD ciphertext C, C_B. Below, we will call a message containing an identifier, key and digest an *attachment metadata* message. Alice sends C, C_B to Facebook, which runs FBTag on C_B (this amounts to HMAC-SHA256 with an internal Facebook key and some metadata) as in the standard message franking protocol [16, 19] to obtain t_{FB}. Facebook sends $C, C_B, \mathsf{t}_{\mathrm{FB}}$ to the receiver.

Receiving phase: Upon receiving a message $C, C_B, \mathsf{t}_{\mathrm{FB}}$ from Alice (via Facebook), Bob runs CtE2-Dec on C, C_B to obtain $\mathsf{id}\|K_\mathrm{a}\|D_\mathrm{a}$. Bob then sends id to Facebook, which gets the value $N_\mathrm{a}\|C_\mathrm{a}$ associated with id in its key-value store and sends it to Bob. Bob verifies that $D_\mathrm{a} = \mathsf{SHA\text{-}256}(N_\mathrm{a}\|C_\mathrm{a})$



Alice
Facebook
Bob
$K_a \leftarrow_\$ \mathcal{K}$; $N_a \leftarrow_\$ \mathcal{N}$
$C_a \leftarrow \mathsf{GCM\text{-}Enc}(K_a, N_a, M_a)$
$D_a \leftarrow \mathsf{SHA\text{-}256}(N_a \parallel C_a)$
$N_a \parallel C_a$
$\mathsf{id} \leftarrow_\$ \{0,1\}^n$
$\mathrm{Put}(\mathsf{id}, N_a \parallel C_a)$
id
$C, C_B \leftarrow_\$ \mathrm{CtE2\text{-}Enc}(\mathsf{id} \parallel K_a \parallel D_a)$
C, C_B
$\mathsf{t}_{\mathrm{FB}} \leftarrow \mathsf{FBTag}(C_B)$
$C, C_B, \mathsf{t}_{\mathrm{FB}}$ (1)
id
$\mathsf{id} \parallel K_a \parallel D_a \leftarrow \mathrm{CtE2\text{-}Dec}(C, C_B)$
$N_a \parallel C_a \leftarrow \mathrm{Get}(\mathsf{id})$
$N_a \parallel C_a$
Verify $D_a = \mathsf{SHA\text{-}256}(N_a \parallel C_a)$
$M_a \leftarrow \mathsf{GCM\text{-}Dec}(K_a, N_a, C_a)$ (2)
(Report)
For $i = 1$ to ℓ do:
$\mathrm{CtE2\text{-}Ver}(\mathsf{id}^i, K_a^i, D_a^i)$
$N_a \parallel C_a \leftarrow \mathrm{Get}(\mathsf{id})$
Verify $D_a = \mathsf{SHA\text{-}256}(N_a \parallel C_a)$
$M_a \leftarrow \mathsf{GCM\text{-}Dec}(K_a, N_a, C_a)$
If $R[\mathsf{id}^i] = \perp$ then
$R[\mathsf{id}^i] \leftarrow M_a$
$\{\mathsf{id}^i, K_a^i, D_a^i\}_{i=1}^{\ell}$
Open ℓ attachments

Figure 1: Facebook's attachment franking protocol [35, 36]. The sending phase consists of everything from the upper-left corner to the message marked **(1)**. The receiving phase consists of everything strictly after **(1)** and before **(2)**. The reporting phase is below the dashed line. The descriptions of Facebook's behavior during the reporting phase were paraphrased (with permission) from conversations with Jon Millican, whom the authors thank profusely.

and decrypts C_a to obtain the attachment content M_a.

Reporting phase: Bob sends all recent messages to Facebook along with their commitment openings and t_{FB} values (not pictured in the diagram). For each message, Facebook verifies the commitment using CtE2-Ver and the authentication tag t_{FB} using its internal HMAC key. Then, if the commitment verifies correctly and the message contains attachment metadata, Facebook gets the attachment ciphertext and nonce $N_a \| C_a$ from its key-value store using its identifier id. Facebook verifies that $D_a = \mathsf{SHA\text{-}256}(N_a \parallel C_a)$ and decrypts C_a with K_a and N_a to obtain the attachment content M_a. If no other attachment metadata message containing identifier id has already been seen, the plaintext M_a is added to the abuse report R. (Looking ahead, this is the application-level behavior that enables the attack, which will violate the one-to-one correspondence between id and plaintext that is assumed here.)

Attack intuition. The threat model of this attack is a malicious Alice who wants to send an abusive attachment to Bob, but prevent Bob from reporting it to Facebook. The attachment can be an offensive image (e.g., a picture of abusive text or of a gun) or video. We focus our discussion below on images.

The attack has two main steps: (1) generating the colliding ciphertext and (2) sending it twice to Bob. In step (1), Alice creates two GCM keys and a single GCM ciphertext which decrypts (correctly) to the abusive attachment under one key and to a different attachment under the other key. In step (2), Alice sends the ciphertext to Facebook and gets an identifier back. Alice then sends the identifier to Bob twice, once with each key.

On receiving the two messages, Bob decrypts the image twice and sees both the abusive attachment and the other one. When Bob reports the conversation to Facebook, its server-side code verifies both decryptions of the image ciphertext but only inserts the other decryption into the abuse report—the human making the abusive-or-not judgment will have no idea Bob saw the abusive attachment.

We will describe two variants of the attack. We will begin with the case where the second decryption of the colliding ciphertext is junk bytes with no particular structure. This variant is simple but easily detectable, since the junk bytes will not display correctly. Then we give a more advanced variant where the second decryption correctly displays an innocuous attachment, like a picture of a kitten. Before describing the attack variants, we will explain GCM at a high level for unfamiliar readers.

An Overview of GCM. For completeness, we will briefly describe GCM. Pseudocode for GCM encryption and decryption can be found in Figure 2. GCM is a mode of operation that builds a randomized authenticated encryption scheme from a block cipher E on n-bit inputs and outputs [33, 34]. Call the block cipher key K. GCM is an encrypt-then-MAC (EtM) composition [5] of counter mode and a Carter-Wegman (CW) MAC [48]. GCM's MAC is based on arithmetic in the finite field $\mathrm{GF}(2^n)$. The MAC is computed by taking the associated data and ciphertext blocks (as well as a block encoding the input length) to be the coefficients of a polynomial over $\mathrm{GF}(2^n)$, then evaluating the polynomial at the point $E_K(0^n)$. In the left two algorithms of Figure 2, counter mode encryption/decryption happens in the lines assigning $C[i]$ and $M[i]$, respectively. The MAC computation corresponds to the lines assigning the tag T. GCM is *not* a robust encryption scheme in the sense of [18]—it is possible to construct ciphertexts that decrypt correctly under two distinct keys.

3.1 A Simple Attack

Alice begins the attack with an abusive attachment $M_{\mathrm{a}}^{\mathrm{ab}}$. Alice chooses two distinct 128-bit GCM keys K_1 and K_2 and a nonce N_{a}, then computes a ciphertext C_{a} via $\mathrm{CTR\text{-}Enc}(K_1, N_{\mathrm{a}} + 2, M_{\mathrm{a}}^{\mathrm{ab}})$, where CTR-Enc denotes CTR-mode encryption with the given key and nonce. The nonce is $N_{\mathrm{a}} + 2$ to match GCM, see Figure 2. In Facebook's scheme Alice can choose the keys and the nonce, but this is not necessary—any combination of two keys and a nonce will work.

The ciphertext C_{a} is almost, but not quite, the ciphertext Alice will use in the attack. To ensure GCM decryption is correct for both keys, Alice generates the colliding GCM tag and final ciphertext block using $\mathsf{Collide\text{-}GCM}(K_1, K_2, N_{\mathrm{a}}, C_{\mathrm{a}})$ (described in Figure 2). The function $\mathsf{Collide\text{-}GCM}$ works by computing the tags for the two keys then solving a linear equation to find the value of the last ciphertext block. We use the final ciphertext block as the variable, but a different ciphertext block or a block of associated data could be used instead. The output $N_{\mathrm{a}} \parallel C_{\mathrm{a}} \parallel T$ correctly decrypts to $M_{\mathrm{a}}^{\mathrm{ab}}$ under K_1 and to another plaintext M_{j} under K_2. However, the plaintext M_{j} will be random bytes with no structure. The advanced variant of our attack (in Section 3.2) will ensure M_{j} is a correctly-formatted plaintext.

Sending the colliding ciphertext. Alice continues the sending phase with Facebook, obtaining an identifier id for the ciphertext $N_{\mathrm{a}} \parallel C_{\mathrm{a}}$. Alice then creates two attachment metadata messages: $\mathrm{MD}_1 = \mathsf{id} \parallel K_2 \parallel D_{\mathrm{a}}$ and $\mathrm{MD}_2 = \mathsf{id} \parallel K_1 \parallel D_{\mathrm{a}}$. Alice completes the remainder of the sending phase twice, first with MD_1 and then with MD_2. (The first message sent is associated to the junk message.) After finishing the receiving phase for MD_1, Bob will decrypt C_{a} with K_2, giving M_{j}. After finishing the receiving phase with MD_2, Bob will decrypt C_{a} with K_1 and see $M_{\mathrm{a}}^{\mathrm{ab}}$. We emphasize that both attachment metadata messages are valid, and no security properties of CtE2

```
GCM-Enc($K, N, \mathrm{AD}, M$):
$H \leftarrow E_K(0^{128})$
$P \leftarrow E_K(N+1)$
$\mathsf{lens} \leftarrow \mathsf{encode}_{64}(|\mathrm{AD}|) \| \mathsf{encode}_{64}(|M|)$
$T \leftarrow \mathsf{lens} \times_{GF} H \oplus P$
$\mathsf{mlen} \leftarrow |M|/128$
$\mathsf{adlen} \leftarrow |\mathrm{AD}|/128$
$\mathsf{blen} \leftarrow \mathsf{mlen} + \mathsf{adlen}$
For $i = 1$ to $\mathsf{adlen}$:
    $T \leftarrow T \oplus \mathrm{AD}[i] \times_{GF} H^{\mathsf{blen}+2-i}$
For $i = 1$ to $\mathsf{mlen}$:
    $C[i] \leftarrow E_K(N+1+i) \oplus M[i]$
    $T \leftarrow T \oplus C_i \times_{GF} H^{\mathsf{blen}+2-i-\mathsf{adlen}}$
Return $N \| C \| T$
```

```
GCM-Dec($K, \mathrm{AD}, N \| C \| T'$):
$H \leftarrow E_K(0^{128})$
$P \leftarrow E_K(N+1)$
$\mathsf{lens} \leftarrow \mathsf{encode}_{64}(|\mathrm{AD}|) \| \mathsf{encode}_{64}(|M|)$
$T \leftarrow \mathsf{lens} \times_{GF} H \oplus P$
$\mathsf{mlen} \leftarrow |C|/128$
$\mathsf{adlen} \leftarrow |\mathrm{AD}|/128$
$\mathsf{blen} \leftarrow \mathsf{mlen} + \mathsf{adlen}$
For $i = 1$ to $\mathsf{adlen}$:
    $T \leftarrow T \oplus \mathrm{AD}[i] \times_{GF} H^{\mathsf{blen}+2-i}$
For $i = 1$ to $\mathsf{mlen}$:
    $M[i] \leftarrow E_K(N+1+i) \oplus C[i]$
    $T \leftarrow T \oplus C_i \times_{GF} H^{\mathsf{blen}+2-i-\mathsf{adlen}}$
If $T \neq T'$ then Return $\perp$
Return $M$
```

```
Collide-GCM($K_1, K_2, N_\mathrm{a}, C$):
$H_1 \leftarrow E_{K_1}(0^{128})$ ; $H_2 \leftarrow E_{K_2}(0^{128})$
$P_1 \leftarrow E_{K_1}(N_\mathrm{a}+1)$ ; $P_2 \leftarrow E_{K_2}(N_\mathrm{a}+1)$
$\mathsf{mlen} \leftarrow |C|/128 + 1$
$\mathsf{lens} \leftarrow \mathsf{encode}_{64}(0) \| \mathsf{encode}_{64}(|C|+128)$
$\mathsf{acc} \leftarrow \mathsf{lens} \times_{GF} (H_1 \oplus H_2) \oplus P_1 \oplus P_2$
For $i = 1$ to $\mathsf{mlen} - 1$:
    $H_i \leftarrow H_1^{\mathsf{mlen}+2-i} \oplus H_2^{\mathsf{mlen}+2-i}$
    $\mathsf{acc} \leftarrow \mathsf{acc} \oplus C[i] \times_{GF} H_i$
$\mathsf{inv} \leftarrow (H_1^2 \oplus H_2^2)^{-1}$
$C_{\mathsf{mlen}} \leftarrow \mathsf{acc} \times_{GF} \mathsf{inv}$
$C_\mathrm{a} \leftarrow C \| C_{\mathsf{mlen}}$
$T \leftarrow \mathsf{GHASH}(H_1, C_\mathrm{a}) \oplus P_1$
Return $N_\mathrm{a} \| C_\mathrm{a} \| T$
```

Figure 2: **(Left)** The Galois/Counter block cipher mode (GCM) encryption algorithm. **(Middle)** The GCM decryption algorithm. **(Right)** The Collide-GCM algorithm, which takes a partial ciphertext C, a nonce N_a, and two keys K_1 and K_2 and computes a tag T and final ciphertext block so that the output nonce-ciphertext-tag triple $N_\mathrm{a} \| C_\mathrm{a} \| T$ decrypts correctly under both keys. Array indexing is done in terms of 128-bit blocks. We assume all input bit lengths are multiples of 128 for simplicity, and that the input M_a to Collide-GCM is at least two blocks in length. The function GHASH is the standard GCM polynomial hash (the lines which assign to T on the left). The function $\mathsf{encode}_{64}(\cdot)$ returns a 64-bit representation of its input. Arithmetic (addition and multiplication) in GF(2^{128}) is denoted $\oplus$ and $\times_{GF}$, respectively. The function Collide-GCM can take arbitrary headers, but we elide them for simplicity.

are violated.

When Bob reports the recent messages, Facebook will verify both MD_1 and MD_2 and check the digest D_a matches the value $N_\mathrm{a} \| C_\mathrm{a}$ stored with identifier id. ***However, it will only insert the first decryption, the plaintext*** M_j***, into the abuse report.*** The system sees the second ciphertext has the same SHA-256 hash and identifier, and assumes it's a duplicate: the report contains no trace of the message M_a^ab.

3.2 Advanced Variant and Proof of Concept

Next we will describe the advanced variant of the attack (in which both decryptions correctly display as attachments) and our proof-of-concept implementation. Ensuring both decryptions are valid attachments is important because the simple variant (where one decryption is random bytes) may not have sufficed for a practical exploit if Facebook only inserted valid images into their abuse reports. We implemented the advanced variant and crafted a colliding ciphertext for which the "abusive" decryption M_a^ab is the image of an Axolotl salamander in Figure 3. The innocuous decryption M_j is the image of a kitten in that figure. We verified both display correctly in Facebook Messenger's browser client. Code for our proof of concept is available on request.

The only difference between the advanced variant and the one described above is the way Alice generates the ciphertext C_a which is input to Collide-GCM. Instead of simply encrypting the abusive attachment M_a^ab, Alice first merges M_a^ab and another innocuous attachment M_j using an algorithm Att-Merge($K_1, K_2, M_\mathrm{a}^\mathrm{ab}, \mathsf{M}_\mathrm{j}$) which takes the two keys and attachments and outputs a nonce N_a and C_a so that CTR-Dec($K_1, N_\mathrm{a}+2, C_\mathrm{a}$) displays M_a^ab and CTR-Dec($K_2, N_\mathrm{a}+2, C_\mathrm{a}$) displays M_j. The exact implementation of Att-Merge is file-format-specific, but for most formats Att-Merge has two main steps: (1) a *nonce search* yielding a nonce which preserves certain file structures in

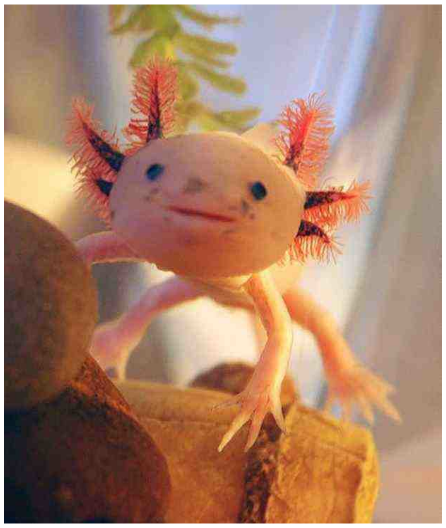
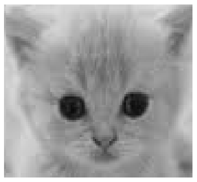



Figure 3: Two images with the same GCM ciphertext $C_\text{a} \parallel T$ when encrypted using 16-byte key $K_1 =$ $(\texttt{03})^{16}$ or $K_2 = (\texttt{02})^{16}$, nonce $N_\text{a} = 10606665379$, and associated data $H = (\texttt{ad})^{32}$ (all given in hex where exponentiation indicates repetition). **(Left)** The titular invisible salamander, which is delivered to the recipient but not inserted into the abuse report. **(Right)** An image of a kitten that is put in the recipient's abuse report instead of the salamander.

the plaintexts, and (2) a *plaintext restructuring* that expands the plaintexts with random bytes in locations that are ignored by parsers for their respective file formats. We implemented Att-Merge for JPEG and BMP images (the salamander image and the kitten image, respectively), so our discussion will focus on these formats.

Before discussing our implementation of Att-Merge we will briefly describe the JPEG and BMP file formats. JPEG files are of the form `ff` $\parallel$ `d8` $\parallel$ `JPEG data` $\parallel$ `ff` $\parallel$ `d9`. The two-byte sequence `ffd8` *must* be the first two bytes, and the two-byte sequence `ffd9` *must* be the final two bytes. There is no file length block in JPEG files—internal data structures have length fields but the total size of the file can be determined only after parsing. JPEG files can also contain comments of up to 2^{16} bytes that are ignored by JPEG parsers. JPEG Comments are indicated with the two-byte sequence `fffe` followed by a big-endian two-byte encoding of the comment length. BMP files are of the form `42` $\parallel$ `4d` $\parallel$ `<length>` $\parallel$ `BMP data`. BMP files *must* begin with `424d`, and the next four bytes *must* be the length block. The length block in a BMP file is a four-byte (little-endian) encoding of the file length. All the BMP parsers we used only read the number of bytes indicated in the header and ignore trailing bytes.

Some intuition. At a high level, our Att-Merge proof-of-concept will craft the colliding ciphertext C_a by putting the encryption of BMP under K_2 and the encryption of the JPEG under K_1 at different byte offsets in C_a. This will, of course, result in some portions of both plaintexts being randomized, but we use several features of the JPEG and BMP file formats which will ensure these random-looking bytes do not prevent the image from being correctly parsed and displayed (see Figure 4).

Nonce search. Because the actual image data occurs at different offsets in the two plaintexts the nonce search need only find a "true" collision for those bytes of the ciphertext that are semantically meaningful in *both* plaintext files. JPEG and BMP files must begin with different fixed two-byte sequences, so the keystreams XORed with those sequences must result in a collision for the first

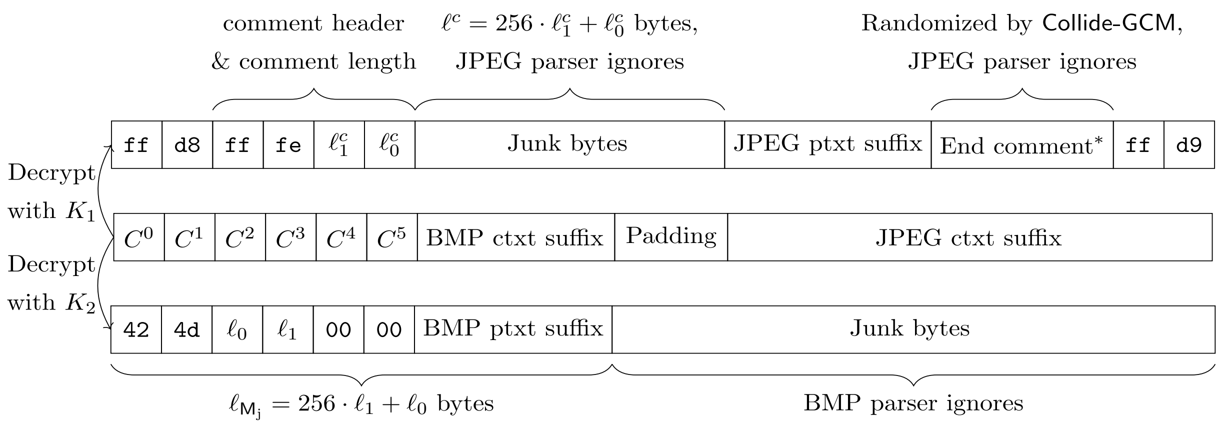


Figure 4: Diagram of the JPEG $M_{\mathrm{a}}^{\mathrm{ab}}$ **(top)** and BMP $\mathsf{M_j}$ **(bottom)** plaintexts output by the plaintext restructuring step, and their ciphertext **(middle)**. The leftmost block of each file is the first byte. The "BMP ptxt suffix" is the suffix of the original BMP starting at byte 6. The "JPEG ptxt suffix" is the bytes of the original JPEG starting at byte 2 and ending before the final two bytes. The fifth and sixth bytes of the JPEG (marked ℓ_1^c and ℓ_0^c) are randomized during nonce search. (*) The region marked "End comment" begins with the comment header and comment length bytes (which are *not* randomized by Collide-GCM), but we do not depict them for simplicity.

two bytes. The plaintext restructuring step will need the JPEG to have a comment header in the next two bytes, which in the BMP plaintext contain the file length. Thus, the nonce output by Att-Merge must produce keystreams (under K_1 and K_2) so the encryptions of the first four bytes (marked C^0 through C^4 in Figure 4) of either file result in the same value. This happens for about one in 2^{32} nonces. We wrote a simple Python script to search through nonces until we found 10606665379, which produces the required collision. Finding that nonce took roughly three hours on a 3.4GHz quad-core Intel i7. Though very fast, fixing the keys and searching through nonces is not the fastest way to find a collision. Since the keys can be chosen arbitrarily, fixing a nonce and doing a birthday attack on keys would produce a collision after only about 2^{17} encryptions.

Plaintext restructuring. After the nonce search, the two plaintexts can be restructured. For JPEG and BMP images Att-Merge performs the following steps: (1) inserting the decryption (under K_1) of the BMP ciphertext into a comment region at the beginning of the JPEG, (2) inserting an additional comment at the end of the JPEG so the bytes randomized by Collide-GCM are ignored by the JPEG parser, and (3) appending the decryption (under K_2) of the JPEG ciphertext to the end of the BMP plaintext. See Figure 4 for a diagram of the JPEG and BMP plaintexts after restructuring.

To put the decryption of the BMP ciphertext in a JPEG comment (step (1)), we first insert the comment region in between the two-byte header `ffd8` and the rest of the JPEG data by placing the two-byte comment sequence `fffe` after `ffd8`. These are the first four bytes of the JPEG file in Figure 4. The four-byte collision in the keystream ensures the ciphertext of these four bytes (denoted C^0 through C^3 in Figure 4) will be the same as the first four bytes of the BMP ciphertext.

The next two bytes of the modified JPEG are the comment's length $\ell_1^c \parallel \ell_0^c$ (the fifth and sixth bytes of the JPEG file in Figure 4). To ensure the BMP parses correctly, these bytes must be fixed as the XOR of the fifth and sixth JPEG keystream bytes and the fifth and sixth BMP keystream bytes. If we let P_{K_2} be the keystream for the BMP key and P_{K_1} be the keystream for the JPEG key (indexing from zero), $\ell_1^c = P_{K_2}[4] \oplus P_{K_1}[4]$ and $\ell_0^c = P_{K_2}[5] \oplus P_{K_1}[5]$. This is because bytes five and six of the ciphertext must be the fifth and sixth BMP keystream bytes to ensure the BMP plaintext has the correct length—the fifth and sixth bytes of a BMP file are the high-order bytes of the file length and must both be zero for the BMP plaintext $\mathsf{M_j}$ (see Figure 4).

Once we know the length of the comment, we can make the next bytes the "decryption" of the BMP data's ciphertext under the JPEG key K_1. To write this more formally, first define $b[a \cdots c]$ to be the bytes of b from index a to c, inclusive. Then if we let $\mathsf{M}_\mathsf{j}^{\mathsf{suff}}$ be everything but the first six bytes of the BMP $\mathsf{M_j}$ and $\ell_{\mathsf{M_j}} = |\mathsf{M_j}|$, the next bytes of the JPEG will be $\mathsf{M}_\mathsf{j}^{\mathsf{suff}} \oplus P_{K_2}[6 \cdots \ell_{\mathsf{M_j}}] \oplus P_{K_1}[6 \cdots \ell_{\mathsf{M_j}}]$. If the BMP data is too short, we append $\ell^c - \ell_{\mathsf{M_j}}$ random bytes. The comment will be random-looking bytes, but the JPEG parser will ignore it and jump to the byte after the comment.

Before discussing step (2), we will make a few observations. First, we place essentially the entire BMP file in a JPEG comment. This means the file cannot be longer than the maximum length of a JPEG comment. Second, without additional brute-forcing we cannot choose the comment length $\ell^c = 256 \cdot \ell_1^c + \ell_0^c$ — it is a random number in the range $[0, \ldots, 2^{16} - 1]$ and is fixed by the choice of the nonce and the two keys. Thus, smaller BMP files are better than large ones: if the length of the BMP file is $\ell_{\mathsf{M_j}}$ and we model AES as an ideal cipher, each nonce with the four-byte collision we need gives $\Pr\left[\, \ell^c \geq \ell_{\mathsf{M_j}} - 6 \,\right] \approx (2^{16} - (\ell_{\mathsf{M_j}} - 6))/2^{16}$ (where the probability is taken over the choice of random permutation for the two keys). In words, the probability of the comment length being greater than or equal to the file length is inversely proportional to the file length. The byte length of a BMP file is directly related to the number of pixels in the image, so the chosen BMP files should be fairly small in dimension to reduce the work required to find a nonce because a nonce resulting in a too-short comment must be discarded. For example, the kitten image in Figure 3 is about one hundred pixels by eighty pixels and is in grayscale so that the number of bytes needed to describe each pixel is minimized. The kitten BMP file is $\ell_{\mathsf{M_j}} = 9502$ bytes, and one of the two nonces we found during our search did *not* result in $\ell^c \geq \ell_{\mathsf{M_j}} - 6$.

Our third observation is that nothing in step (1) has depended on the contents of either image, only on the length of the BMP. Thus, the nonce output by the nonce search phase can be re-used: for K_2, K_1 and N_a a colliding ciphertext for any valid JPEG and any BMP of the same length as $\mathsf{M_j}$ can be created.

In step (2) we again expand the JPEG with an additional comment region. This comment region is placed immediately before the end-of-file indicator `ffd9`. The comment region's length is 44 bytes. With the comment header and length bytes we add 48 total bytes to the end of the JPEG. This ensures the second-to-last sixteen-byte block of the JPEG is always ignored by the parser. This is the block of ciphertext we will use to ensure a tag collision in $\mathsf{Collide\text{-}GCM}$. We could have used a block of associated data and dispensed with the second JPEG comment entirely. Using a block of ciphertext makes our proof-of-concept more realistic, since Facebook's GCM ciphertexts all have fixed associated data that cannot be modified. Define the modified JPEG file resulting from steps (1) and (2) to be M'_a.

In step (3) we append the "decryption" under K_2 of the suffix of $\text{CTR-Enc}(K_1, N_\mathrm{a} + 2, M'_\mathrm{a})$ beginning at byte $\ell^c + 6$ to the BMP plaintext $\mathsf{M_j}$. This step is straightforward—BMP parsers ignore trailing bytes, so our BMP image will still display correctly even when random-looking bytes are appended.

Implementing Collide-GCM. We implemented $\mathsf{Collide\text{-}GCM}$ in Python 2.7 and verified that colliding ciphertexts can be generated in roughly 45 seconds using an unoptimized implementation of $\mathrm{GF}(2^{128})$ arithmetic. We checked decryption correctness using cryptography.io, a Python cryptography library which uses OpenSSL's GCM implementation. This sufficed as a proof-of-concept exploit for Facebook's engineering team.

3.3 Discussion And Mitigation

We chose JPEG and BMP files for our Att-Merge proof of concept because their formats can tolerate random bytes in different regions of the file (the beginning and the end, respectively). We did not try to extend the Att-Merge to other common image formats (like PNG, TIFF or GIF) but file format tricks similar to the ones described above can be used to craft ciphertexts that decrypt to images in those formats. As an example, we will sketch a variant of the attack for which the colliding plaintexts are both JPEG files. It is similar to the JPEG/BMP collision described above except for two differences. First, the JPEG taking the place of the BMP (i.e., the one put in the beginning comment of the other JPEG) must end in another comment instead of `ffd9`, which ensures the JPEG parser will ignore the trailing random bytes. Second, a two-byte collision must be found in the final two bytes of the keystream so both JPEGs end in `ffd9`. A birthday attack on keys should find keystreams with the correct structure (i.e., the first four bytes and the last two are the same) in roughly 2^{25} encryptions. We did not try to implement Att-Merge for video file formats. Such formats are more complex than image formats, but we conjecture it is possible to extend the attack to video files.

Relation to GLR. In [19] GLR proved CtE2 is a ccAEAD scheme. Their proof only applies to CtE2 itself, not to the composition of CtE2 and GCM. Concretely, GLR analyzed CtE2 as it is used for text chat messages in Messenger, but did not analyze how it is used for attachments. The Collide-GCM algorithm in Figure 2 is related to the r-BIND attack against GCM given by GLR [19]. However, their attack is insufficient to exploit Facebook's attachment franking—it only creates ciphertexts with colliding tags, but not the same ciphertext. Thus using it against Facebook wouldn't work, because the SHA-256 hashes of the two images would not collide. The Collide-GCM algorithm works even if the entire ciphertext, including any headers and the nonce, act as the commitment and the only opening is the encryption key.

If Facebook's attachment franking protocol is viewed as a ccAEAD by taking the CtE2 binding tag (i.e., the value C_B output by running CtE2-Enc($\mathsf{id} \parallel K_\mathrm{a} \parallel D_\mathrm{a}$) during the sending phase) to be the compact commitment to the attachment plaintext, the resulting scheme is not vulnerable to GLR's r-BIND attack. This is because C_B commits both to the hash D_a of the nonce/ciphertext pair and to the GCM key K_a.

Mitigating the attack. There are two main ways this attack can be mitigated. The first is a server-software-only patch that ensures abuse reports containing attachments are not deduplicated by attachment identifier. The second is changing the Messenger clients to use a ccAEAD scheme instead of GCM to encrypt attachments. In response to our bug report, Facebook deployed the first mitigation, for two main reasons: (1) it did not require patching the Messenger clients (an expensive and time-consuming process) and (2) changing the client-side crypto while maintaining backwards compatibility with old clients is difficult. Despite requiring less engineering effort, we believe this mitigation has some important drawbacks. Most notably, it leaves the underlying cryptographic issue intact: attachments are still encrypted using GCM. This means future changes to either the Messenger client or Facebook's server-side code could re-expose the vulnerability. Using a ccAEAD in place of GCM for attachment encryption would immediately prevent any deduplication behavior from being exploited, since the binding security of ccAEAD implies attachment identifiers uniquely identify the attachment *plaintexts*.

4 A New Primitive: Encryptment

In this section, we introduce a new primitive called an *encryptment scheme*. Encryptment schemes allow both encryption of, and commitment to, a message. (See Section 2 for the exact definition of commitment we will use below.) Moreover, the schemes which we target and ultimately build achieve both security goals with only a *single* pass over the underlying data.

While the syntax of encryptment schemes is similar to that of the ccAEAD schemes we ultimately look to build, the key difference is that we expect far more minimal security notions from encryptment schemes (see Section 7 for a more detailed discussion). Looking ahead, we shall see that a secure encryptment scheme is the key building block for more complex primitives such as ccAEAD schemes, robust encryption [1, 17, 18], cryptographic concealments [14], and domain extension for authenticated encryption and remotely keyed AE [14], facilitating the construction of very efficient instantiations of these primitives. In Section 7.3 we show how to build ccAEAD from encryptment, and discuss the other primitives in Section 8.

Encryptment schemes. Applying the encryptment algorithm to a given key, header and message tuple (K_{EC}, H, M) returns a pair $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$ which we call an *encryptment.* We refer to encryptment component C_{EC} as the *ciphertext*, and to B_{EC} as the *binding tag.* Together the ciphertext / binding tag pair $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$ function as an encryption of M under key K_{EC}, so that given $(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$, the opening algorithm DO can recover the underlying message M. The binding tag B_{EC} simultaneously acts as a commitment to the underlying header and message, with opening K_{EC}; the validity of this commitment to a given pair (H, M) is checked by the verification algorithm EVer. Looking ahead, we will actually require that B_{EC} acts as a commitment to the opening K_{EC} also, in that it should be infeasible to find $K_{\mathsf{EC}} \neq K'_{\mathsf{EC}}$ which verify the same B_{EC}.

Formally an encryptment scheme is a tuple $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EC}, \mathsf{DO}, \mathsf{EVer})$ defined as follows. Associated to the scheme is a key space $\mathcal{K}_{\mathsf{EC}} \subseteq \Sigma^*$, header space $\mathcal{H}_{\mathsf{EC}} \subseteq \Sigma^*$, message space $\mathcal{M}_{\mathsf{EC}} \subseteq \Sigma^*$, ciphertext space $\mathcal{C}_{\mathsf{EC}} \subseteq \Sigma^*$, and binding tag space $\mathcal{T}_{\mathsf{EC}} \subseteq \Sigma^*$.

- The randomized key generation EKg algorithm takes no input, and outputs a key $K_{\mathsf{EC}} \in \mathcal{K}_{\mathsf{EC}}$.
- The encryptment algorithm EC is a deterministic algorithm which takes as input a key $K_{\mathsf{EC}} \in \mathcal{K}_{\mathsf{EC}}$, a header $H \in \mathcal{H}_{\mathsf{EC}}$, and a message $M \in \mathcal{M}_{\mathsf{EC}}$, and outputs an encryptment $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \in \mathcal{C}_{\mathsf{EC}} \times \mathcal{T}_{\mathsf{EC}}$.
- The decryptment algorithm DO is a deterministic algorithm which takes as input a key $K_{\mathsf{EC}} \in \mathcal{K}_{\mathsf{EC}}$, a header $H \in \mathcal{H}_{\mathsf{EC}}$, and an encryptment $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \in \mathcal{C}_{\mathsf{EC}} \times \mathcal{T}_{\mathsf{EC}}$, and outputs a message $M \in \mathcal{M}_{\mathsf{EC}}$ or the error symbol $\perp$. We assume that if $(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}}) \notin \mathcal{K}_{\mathsf{EC}} \times \mathcal{H}_{\mathsf{EC}} \times \mathcal{C}_{\mathsf{EC}} \times \mathcal{T}_{\mathsf{EC}}$, then $\perp \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$.
- The verification algorithm EVer is a deterministic algorithm which takes as input a header $H \in \mathcal{H}_{\mathsf{EC}}$, a message $M \in \mathcal{M}_{\mathsf{EC}}$, a key $K_{\mathsf{EC}} \in \mathcal{K}_{\mathsf{EC}}$, and a binding tag $B_{\mathsf{EC}} \in \mathcal{T}_{\mathsf{EC}}$, and returns a bit b. We assume that if $(H, M, K_{\mathsf{EC}}, B_{\mathsf{EC}}) \notin \mathcal{H}_{\mathsf{EC}} \times \mathcal{M}_{\mathsf{EC}} \times \mathcal{K}_{\mathsf{EC}} \times \mathcal{T}_{\mathsf{EC}}$ then $0 \leftarrow \mathsf{EVer}(H, M, K_{\mathsf{EC}}, B_{\mathsf{EC}})$.

Length regularity and compactness. We impose two requirements on the lengths of the encryptments output by encryptment schemes. First, we require *compactness*: that the binding tags B_{EC} output by an encryptment scheme are of constant length btlen *regardless* of the length of the underlying message, and that btlen is linear in the key size. Second, we require *length regularity*: that the length of ciphertexts C_{EC} depend only on the length of the underlying message. Formally, we require there exists a function $\mathsf{clen} : \mathbb{N} \to \mathbb{N}$ such that for all $(H, M) \in \mathcal{H}_{\mathsf{EC}} \times \mathcal{M}_{\mathsf{EC}}$ it holds that $|C_{\mathsf{EC}}| = \mathsf{clen}(|M|)$ with probability one for the sequence of algorithm executions: $K_{\mathsf{EC}} \leftarrow\$ \mathsf{EKg}$; $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$.

```
$\underline{\text{otROR0}^{\mathcal{A}}_{\mathsf{EC}}}$:
$K_{\mathsf{EC}} \leftarrow\!\!\$\ \mathsf{EKg}$
$\mathsf{query\text{-}made} \leftarrow \mathsf{false}$
$b \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{enc}(\cdot,\cdot)}$
Return $b$

$\underline{\mathsf{enc}(H, M)}$:
If $\mathsf{query\text{-}made} = \mathsf{true}$ then
    Return $\perp$
$\mathsf{query\text{-}made} \leftarrow \mathsf{true}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
Return $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$
```

```
$\underline{\text{otROR1}^{\mathcal{A}}_{\mathsf{EC}}}$:
$\mathsf{query\text{-}made} \leftarrow \mathsf{false}$
$b \leftarrow\!\!\$\ \mathcal{A}^{\$(\cdot,\cdot)}$
Return $b$

$\underline{\$(H, M)}$:
If $\mathsf{query\text{-}made} = \mathsf{true}$ then
    Return $\perp$
$\mathsf{query\text{-}made} \leftarrow \mathsf{true}$
$C_{\mathsf{EC}} \leftarrow\!\!\$\ \{0,1\}^{\mathsf{clen}(|M|)}$
$B_{\mathsf{EC}} \leftarrow\!\!\$\ \{0,1\}^{\mathsf{btlen}}$
Return $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$
```

```
$\underline{\text{SCU}^{\mathcal{A}}_{\mathsf{EC}}}$:
$K_{\mathsf{EC}} \leftarrow\!\!\$\ \mathsf{EKg}$
$\mathsf{win} \leftarrow \mathsf{false}$
$\mathsf{query\text{-}made} \leftarrow \mathsf{false}$
$\varepsilon \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{enc}(\cdot,\cdot),\mathsf{dec}(\cdot,\cdot)}$
Return $\mathsf{win}$

$\underline{\mathsf{enc}(H, M)}$
If $\mathsf{query\text{-}made} = \mathsf{true}$ then
    Return $\perp$
$\mathsf{query\text{-}made} \leftarrow \mathsf{true}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
Return $((C_{\mathsf{EC}}, B_{\mathsf{EC}}), K_{\mathsf{EC}})$

$\underline{\mathsf{dec}(H', C'_{\mathsf{EC}})}$
If $\mathsf{query\text{-}made} = \mathsf{false}$ then
    Return $\perp$
If $(H', C'_{\mathsf{EC}}) = (H, C_{\mathsf{EC}})$ then
    Return $\perp$
$M' \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H', C'_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M' \neq \perp$ then $\mathsf{win} \leftarrow \mathsf{true}$
Return $M'$
```

```
$\underline{\text{sr-BIND}^{\mathcal{A}}_{\mathsf{CE}}}$:
$(V_1, V_2, B_{\mathsf{EC}}) \leftarrow\!\!\$\ \mathcal{A}$
$(H, M, K_{\mathsf{EC}}) \leftarrow V_1$
$(H', M', K'_{\mathsf{EC}}) \leftarrow V_2$
$b \leftarrow \mathsf{EVer}(H, M, K_{\mathsf{EC}}, B_{\mathsf{EC}})$
$b' \leftarrow \mathsf{EVer}(H', M', K'_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $V_1 = V_2$ then
    Return $\mathsf{false}$
Return $(b = b' = 1)$
```

```
$\underline{\text{s-BIND}^{\mathcal{A}}_{\mathsf{CE}}}$:
$(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow\!\!\$\ \mathcal{A}$
$M' \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M' = \perp$ then Return $\mathsf{false}$
$b \leftarrow \mathsf{EVer}(H, M', K_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $b = 0$ then
    Return $\mathsf{true}$
Return $\mathsf{false}$
```

Figure 6: One-time real-or-random (otROR), second-ciphertext unforgeability (SCU), and binding notions for an encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EC}, \mathsf{DO}, \mathsf{EVer})$.

```
$\underline{\text{COR}_{\mathsf{EC}}(H, M)}$:
$K_{\mathsf{EC}} \leftarrow\!\!\$\ \mathsf{EKg}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$M' \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
$b \leftarrow \mathsf{EVer}(H, M', K_{\mathsf{EC}}, B_{\mathsf{EC}})$
Return $(M = M' \wedge b = 1)$

$\underline{\text{S-COR}_{\mathsf{EC}}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})}$:
$M \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
$(C'_{\mathsf{EC}}, B'_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
Return $((C_{\mathsf{EC}}, B_{\mathsf{EC}}) = (C'_{\mathsf{EC}}, B'_{\mathsf{EC}}))$
```

Figure 5: Correctness games for an encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EC}, \mathsf{DO}, \mathsf{EVer})$.

Correctness. We define two correctness notions for encryptment schemes, which we formalize via the games COR and S-COR shown in Figure 5. We require that *all* encryptment schemes satisfy our all-in-one *correctness* notion, which requires that honestly generated encryptments both decrypt to the correct underlying message, and successfully verify, with probability one. Formally, we say that an encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EC}, \mathsf{DO}, \mathsf{EVer})$ is correct if for all header / message pairs $(H, M) \in \mathcal{H}_{\mathsf{EC}} \times \mathcal{M}_{\mathsf{EC}}$, it holds that $\Pr[\,\text{COR}_{\mathsf{EC}}(H, M) \Rightarrow 1\,] = 1$, where the probability is over the coins of EKg.

We additionally define *strong correctness*, which requires that for each tuple $(K_{\mathsf{EC}}, H, M) \in \mathcal{K}_{\mathsf{EC}} \times \mathcal{H}_{\mathsf{EC}} \times \mathcal{M}_{\mathsf{EC}}$ there is a unique encryptment $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$ such that $M \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$. We formalize this in game S-COR, and say that an encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EC}, \mathsf{DO}, \mathsf{EVer})$ is strongly correct if for all tuples $(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}}) \in \mathcal{K}_{\mathsf{EC}} \times \mathcal{H}_{\mathsf{EC}} \times \mathcal{C}_{\mathsf{EC}} \times \mathcal{T}_{\mathsf{EC}}$, it holds that $\Pr[\,\text{S-COR}_{\mathsf{EC}}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}}) \Rightarrow 1\,] = 1$. While we only require that encryptment schemes satisfy correctness, the schemes we build will also possess the stronger property (which simplifies their security proofs). We note that strong correctness can be added to any encryptment scheme by making DO recompute an encryptment after decrypting, and returning $\perp$ if the two do not match; however for efficiency we target schemes which achieve strong correctness without this.

4.1 Security Goals for Encryptment

We require encryptment schemes to satisfy both one-time real-or-random (otROR) security, and a variant of one-time ciphertext integrity (SCU) which requires forging a ciphertext for a given binding tag with a known key; we motivate this variant below. The security games for both notions are shown in Figure 6.

Confidentiality. We define otROR security for an encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EC}, \mathsf{DO}, \mathsf{EVer})$ in terms of games otROR0 and otROR1. Each game allows an attacker $\mathcal{A}$ to make one query of the form (H, M) to his real-or-random encryption oracle; in game otROR0 he receives back the real encryptment $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$ encrypting the input under a secret key, and in game otROR1 he receives back random bit strings. For an encryptment scheme EC and adversary $\mathcal{A}$, we define the otROR advantage of $\mathcal{A}$ against EC as

$$\mathbf{Adv}^{\text{ot-ror}}_{\mathsf{EC}}(\mathcal{A}) = \left| \Pr\left[\, \text{otROR0}^{\mathcal{A}}_{\mathsf{EC}} \Rightarrow 1 \,\right] - \Pr\left[\, \text{otROR1}^{\mathcal{A}}_{\mathsf{EC}} \Rightarrow 1 \,\right] \right| ,$$

where the probability is over the coins of EKg and $\mathcal{A}$.

Second-ciphertext unforgeability. We also ask that encryptment schemes meet an unforgeability goal that we call second-ciphertext unforgeability (SCU). In this game, the attacker first learns an encryptment $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$ corresponding to a chosen header / message pair (H, M) under key K_{EC}. We then require that the attacker shouldn't be able to find a *distinct* header and ciphertext pair $(H', C'_{\mathsf{EC}}) \neq (H, C_{\mathsf{EC}})$ such that $\mathsf{DO}(K_{\mathsf{EC}}, H', C'_{\mathsf{EC}}, B_{\mathsf{EC}})$ does not return an error. This should hold even if the attacker knows K_{EC}. Looking ahead, this is a necessary and sufficient condition needed from encryptment when using it to build ccAEAD schemes from fixed domain authenticated encryption.

Formally, the game SCU is shown in Figure 6. To an encryptment scheme EC and adversary $\mathcal{A}$, we define the second-ciphertext unforgeability (SCU) advantage to be $\mathbf{Adv}^{\text{scu}}_{\mathsf{EC}}(\mathcal{A}) = \Pr\left[\, \text{SCU}^{\mathcal{A}}_{\mathsf{EC}} \Rightarrow \mathsf{true} \,\right]$, where the probability is again over the coins of EKg and $\mathcal{A}$.

Binding security. We finally require that encryptment schemes satisfy certain binding notions. We start by generalizing the receiver binding notion r-BIND for ccAEAD schemes from [19], and adapting the syntax to the encryptment setting. r-BIND security requires that no computationally efficient adversary can find two keys, message, header triples (K_{EC}, H, M),$(K'_{\mathsf{EC}}, H', M')$ and a binding tag B_{EC} such that $(H, M) \neq (H', M')$ and $\mathsf{EVer}(H, M, K_{\mathsf{EC}}, B_{\mathsf{EC}}) = \mathsf{EVer}(H', M', K'_{\mathsf{EC}}, B_{\mathsf{EC}}) = 1$. A simple strengthening of this notion — which we denote sr-BIND (for *strong* receiver binding) — allows the adversary to instead win if $(H, M, K_{\mathsf{EC}}) \neq (H', M', K'_{\mathsf{EC}})$. The pseudocode game sr-BIND is shown in Figure 6, where we define the sr-BIND advantage of an adversary $\mathcal{A}$ against EC as $\mathbf{Adv}^{\text{sr-bind}}_{\mathsf{EC}}(\mathcal{A}) = \Pr\left[\, \text{sr-BIND}^{\mathcal{A}}_{\mathsf{EC}} \Rightarrow \mathsf{true} \,\right]$. The corresponding game and advantage term for r-BIND security are defined analogously. The stronger receiver binding notion implies the prior notion, and indeed is strictly stronger, as we detail in Appendix A. For our purposes, it will simplify our negative results about rate-1 blockcipher-based encryptment.

We additionally define the notion of sender binding. It ensures that a sender must itself commit to the message underlying an encryptment, by requiring that it is infeasible to find an encryptment which decrypts correctly but for which verification fails. Without this requirement, a malicious sender may be able to send an abusive message to a receiver with a faulty commitment such that a receiver is unable to report it. We define sender binding security formally via the game s-BIND in Figure 6. We define the s-BIND advantage of an adversary $\mathcal{A}$ against an encryptment scheme EC as $\mathbf{Adv}^{\text{s-bind}}_{\mathsf{EC}}(\mathcal{A}) = \Pr\left[\, \text{s-BIND}^{\mathcal{A}}_{\mathsf{EC}} \Rightarrow \mathsf{true} \,\right]$.

Integrity. We may also define a one-time notion of ciphertext integrity for encryptment schemes. Here, the attacker is challenged to output a fresh encryptment which decrypts correctly given a single query to an encryptment oracle. The game otCTXT in Figure 7 is defined identically to game SCU in Figure 6, except for two changes: first, we do not return the key K_{EC} from enc. Second, we modify the specification of oracle dec as follows. In game otCTXT, dec takes as input a tuple $(H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$, returns $\perp$ if $\mathsf{query\text{-}made} = \mathsf{false}$, or if $\mathsf{query\text{-}made} = \mathsf{true}$ and $(H, B_{\mathsf{EC}}, C_{\mathsf{EC}})$ is equal to the encryptment returned in the query to enc. Otherwise, dec returns the output of $\mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$, and the attacker wins if any dec query does not return $\perp$. To an encryptment scheme EC and adversary $\mathcal{A}$, we define the otCTXT advantage to be $\mathbf{Adv}^{\text{ot}-\text{ctxt}}_{\mathsf{EC}}(\mathcal{A}) = \Pr\left[\,\text{otCTXT}^{\mathcal{A}}_{\mathsf{EC}} \Rightarrow \mathsf{true}\,\right]$.

While we do not require that encryptment schemes satisfy the notion of otCTXT security, looking ahead to Section 7, those that do (when reframed in the ccAEAD syntax) constitute a secure *one-time ccAEAD scheme*. By this, we mean a ccAEAD scheme for which a fresh secret key is chosen for each encryption. A one-time ccAEAD scheme is suitable for use in applications such as Signal, where ratcheting ensures that each encryption is essentially under a fresh key.

```
otCTXT$^{\mathcal{A}}_{\mathsf{EC}}$:
$K_{\mathsf{EC}} \leftarrow\!\$\ \mathsf{EKg}$
$\mathsf{win} \leftarrow \mathsf{false}$
$\mathsf{query\text{-}made} \leftarrow \mathsf{false}$
$\varepsilon \leftarrow\!\$\ \mathcal{A}^{\mathsf{enc}(\cdot,\cdot),\mathsf{dec}(\cdot,\cdot,\cdot)}$
Return $\mathsf{win}$

$\mathsf{enc}(H, M)$
If $\mathsf{query\text{-}made} = \mathsf{true}$ then
    Return $\perp$
$\mathsf{query\text{-}made} \leftarrow \mathsf{true}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$\mathbf{H} \leftarrow H$ ; $\mathbf{C}_{\mathsf{EC}} \leftarrow C_{\mathsf{EC}}$ ; $\mathbf{B}_{\mathsf{EC}} \leftarrow B_{\mathsf{EC}}$
Return $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$

$\mathsf{dec}(H', C'_{\mathsf{EC}}, B'_{\mathsf{EC}})$
If $\mathsf{query\text{-}made} = \mathsf{false}$ then
    Return $\perp$
$b \leftarrow (H', C'_{\mathsf{EC}}, B'_{\mathsf{EC}}) = (\mathbf{H}, \mathbf{C}_{\mathsf{EC}}, \mathbf{B}_{\mathsf{EC}})$
If $\mathsf{query\text{-}made} = \mathsf{true} \wedge b$ then
    Return $\perp$
$M' \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H', C'_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M' \neq \perp$ then $\mathsf{win} \leftarrow \mathsf{true}$
Return $M'$
```

Figure 7: otCTXT game for an encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EC}, \mathsf{DO}, \mathsf{EVer})$.

Relation to ccAEAD. Given the simpler security properties expected of them, building highly efficient secure encryptment schemes is a more straightforward task than constructing a ccAEAD scheme directly. However, as we shall see, encryptment isolates the core complexity of building ccAEAD schemes with multi-opening security. In particular, in Section 7.3 we give a generic transform which allows one to build a multi-opening secure ccAEAD schemes from a secure encryptment scheme and secure AEAD scheme. Armed with this transform, in Section 6 we show how to construct a secure encryptment scheme from cryptographic hash functions. Together, our results will yield the first single-pass, single-primitive constructions of ccAEAD.

Binding and correctness imply second-ciphertext unforgeability. One reason we have introduced encryptment as a standalone primitive (instead of directly working with the ccAEAD formulation from GLR) is that it simplifies security analyses. A useful tool for these analyses is the following lemma, which states that for any encryptment scheme EC that enjoys strong correctness, the combination of r-BIND and s-BIND security suffice to prove SCU security.

Lemma 1 *Let* $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EC}, \mathsf{DO}, \mathsf{EVer})$ *be a strongly correct encryptment scheme, and consider an attacker* $\mathcal{A}$ *in the* SCU *game against* EC*. Then there exist attackers* $\mathcal{B}$ *and* $\mathcal{C}$ *such that* $\mathbf{Adv}^{\text{scu}}_{\mathsf{EC}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{s-bind}}_{\mathsf{EC}}(\mathcal{B}) + \mathbf{Adv}^{\text{r-bind}}_{\mathsf{EC}}(\mathcal{C})$*, and moreover* $\mathcal{B}$ *and* $\mathcal{C}$ *both run in the same time as* $\mathcal{A}$*.*

We give a proof sketch and defer the details to Appendix B. Let $((C_{\mathsf{EC}}, B_{\mathsf{EC}}), K_{\mathsf{EC}})$ be the tuple corresponding to $\mathcal{A}$'s single encryption query (H, M) in the SCU game, and suppose that $\mathcal{A}$ wins the game with decryption oracle query (H', C'_{EC}), meaning that $\mathsf{DO}(K_{\mathsf{EC}}, H', C'_{\mathsf{EC}}, B_{\mathsf{EC}}) = M' \neq \perp$ and $(H', C'_{\mathsf{EC}}) \neq (H, C_{\mathsf{EC}})$. The proof first argues that if the scheme is s-BIND-secure, then any ciphertext which decrypts correctly must also verify correctly. As such, it follows that if $(H, M) \neq (H', M')$ for the winning query, then this can be used to construct a winning tuple for an attacker in the r-BIND game against EC; we bound the probability that this occurs with a reduction to r-BIND security. On the other hand, if $(H, M) = (H', M')$, then it must be the case that $C_{\mathsf{EC}} \neq C'_{\mathsf{EC}}$ — but this in turn implies that we have found two distinct encryptments which decrypt to the same header and message under K_{EC}, violating strong correctness.

A simple encryptment construction. It is straightforward to construct an encryptment scheme by composing a secure encryption scheme and a commitment scheme. One can just use a simple adaptation of the CtE2 ccAEAD scheme from [19]. We defer the details to Appendix C. But such generic compositions are inherently two pass and we seek faster schemes.

5 On Efficient Fixed-key Blockcipher-Based Encryptment

We are interested in building encryptment schemes — and ultimately, more complex primitives such as ccAEAD schemes — from just a blockcipher used on a small number of keys and other primitive arithmetic operations (XOR, finite field arithmetic, etc.). Beyond being an interesting theoretical question, there is the practical motivation that the current fastest AEAD schemes, such as OCB [40], fall into this category.

As a simple motivating example illustrating the challenging nature of this task, we note that OCB does *not* satisfy r-BIND security (see Section 4) when reframed as an encryptment scheme in the natural way. The high level reason for this (modulo a number of details), is that in OCB the binding tag is computed as a function over the XOR of the message blocks. As such, it is straightforward to construct two distinct messages such that the blocks XOR to the same value (and thus produce the same binding tag), thereby violating r-BIND security. Full details of the scheme and attack are given in Appendix E.

For the remainder of this section, we formally define high-rate encryptment schemes, and show how prior results on the impossibility of high-rate CR functions can be used to rule out high-rate encryptment schemes as well.

A connection between hashing and encryptment. Towards showing negative results, we must first define more carefully what we mean by the rate of encryptment schemes. We are inspired by (and will later exploit connections to) the definitions of rate from the blockcipher-based hash function literature [11, 42, 43].

Consider a compression function $\mathcal{H}\colon \{0,1\}^{mn} \to \{0,1\}^{rn}$ for $m > r \geq 1$ and $n \geq 1$, which uses $k \geq 1$ calls of a blockcipher $E\colon \{0,1\}^{\kappa} \times \{0,1\}^{n} \to \{0,1\}^{n}$ $(m, r, n, k, \kappa \in \mathbb{N})$. Then following [43], we may write $\mathcal{H}$ as shown in Figure 8, where we let $K_1, \ldots, K_k$ be any *fixed* strings. Further, we let $f_i\colon \{0,1\}^{(m+(i-1))n} \to \{0,1\}^{n}$ (where $i \in [1, \ldots, k]$) and $g\colon \{0,1\}^{(m+k)n} \to \{0,1\}^{rn}$ be functions.

The *rate* of $\mathcal{H}$ is defined to be m/k; so a rate-$\frac{1}{\beta}$ function $\mathcal{H}$ makes β blockcipher calls per n-bits of input. For example, a rate-1 $\mathcal{H}$ would achieve a single blockcipher call per n-bit block of input. A consequence of the more general results of [43] (see below) is that they rule out rate-1 functions achieving security past $2^{n/4}$ queries to E by an adversary, when modeling E as an ideal cipher. We would like to exploit their negative results to similarly rule out rate-1 encryptment schemes.

```
$\underline{\mathcal{H}(V)}$:
For $i = 1$ to $k$ do
    $X_i \leftarrow f_i(V, Y_1, \ldots, Y_{i-1})$
    $Y_i \leftarrow E_{K_i}(X_i)$
$W \leftarrow g(V, Y_1, \ldots, Y_k)$
Return $W$
```

Figure 8: A blockcipher-based compression function.

We now focus attention on encryptment schemes that fall into a certain form. Consider an encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EC}, \mathsf{DO}, \mathsf{EVer})$. Because EC is deterministic, we can view computing the binding tag as a function $F(K_{\mathsf{EC}}, H, M)$ defined by outputting the binding tag B_{EC}

computed via $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$. The verification algorithm $\mathsf{EVer}(H, M, K_{\mathsf{EC}}, B_{\mathsf{EC}})$ checks that $F(K_{\mathsf{EC}}, H, M) = B_{\mathsf{EC}}$. (One can generalize this definition by allowing EC and EVer to use different functions F,F' to compute the binding tag; the lower bounds given in this section on the rate of such functions readily extend to this case also.)

With this in place, we can define the rate of verification for encryptment analogously to defining the rate of a hash function $\mathcal{H}$, by saying that an encryptment scheme has rate-$\frac{1}{\beta}$ if the associated function F makes β blockcipher calls per n-bits of header and message data (or equivalently, can process (H, M) of combined length mn-bits using βm blockcipher calls).

Now we can give a generic, essentially syntactic, transform from an encryptment scheme to a hash function. For an encryptment scheme EC, let F be the associated binding tag computation function as per above. Let $\mathcal{H}\colon \{0,1\}^* \to \{0,1\}^n$ be the function defined as $\mathcal{H}(X) = F(K_{\mathsf{EC}}, \varepsilon, X)$ for K_{EC} an arbitrary, fixed bit string. Here we take $H = \varepsilon$, so that the number of block cipher calls required to compute F is solely determined by the length of the input X. Note also that any two $X, X' \notin \mathcal{M}$ (where $\mathcal{M}$ is the domain of the encryptment scheme) will trivially collide under $\mathcal{H}$, since $F(K_{\mathsf{EC}}, \varepsilon, X) = F(K_{\mathsf{EC}}, \varepsilon, X') = \bot$. We will therefore assume $\mathcal{M} = \{0,1\}^*$ below. With this, the following theorem is simple to prove.

Theorem 1 *Let EC be a encryptment scheme, and let $\mathcal{H}$ be defined as above. For any collision-resistance adversary $\mathcal{A}$, we give an* r-BIND *adversary $\mathcal{B}$ so that $\mathbf{Adv}^{\mathrm{cr}}_{\mathcal{H}}(\mathcal{A}) \le \mathbf{Adv}^{\mathrm{r\text{-}bind}}_{\mathsf{EC}}(\mathcal{B})$. The adversary $\mathcal{B}$ runs in the same amount of time as $\mathcal{A}$.*

Theorem 1 allows us to apply known negative results about efficient CR-hashing. For example, we have the following corollary of Theorem 1 and [43, Th. 1]:

Corollary 2 *Fix $m > r \ge 1$ and $n > 0$ ($m, r, n \in \mathbb{N}$). Let $N = 2^n$. Let EC be an encryptment scheme for which (1) B_{EC} is computed with an ideal cipher, (2) $|B_{\mathsf{EC}}| = rn$, and (3) that has message space including strings of length mn. Then there is a runnable adversary $\mathcal{A}$ making $q = k(N^{1-(m-r)/k} + 1)$ ideal cipher queries and achieving $\mathbf{Adv}^{\mathrm{r\text{-}bind}}_{\mathsf{EC}}(\mathcal{A}) = 1$, where $k \in \mathbb{N}$ denotes the number of ideal cipher calls required to compute the binding tag for an mn-bit input.*

This immediately rules out security of rate-1 schemes that achieve the efficiency of OCB, i.e., having $k = m$, m arbitrarily large, and $r = 1$. Consider the minimal case that $m = 2$ (two block messages), then $\mathcal{A}$ only requires $q = 2$ queries to succeed. Stronger results ruling out rate-$\frac{1}{2}$ verification can be similarly lifted from [43, Th. 2] under some technical conditions about the verification function and the adversary.

One can modify our definitions so the key K_i for the ith cipher call can be picked from the set $\{K_1, \ldots, K_k\}$ as a function of the current round and messages instead of being used in a fixed order; Rogaway and Steinberger refer to this as the no-fixed order model (though it first appeared in [11]). A negative result based on [11, Th. 5] would rule out encryptment using any rate-1 no-fixed order verification algorithm.

The results above were cast in terms of r-BIND security, but extend to sr-BIND security because the latter implies the former. We conjecture that these lower bounds can be extended to blockcipher-based *robust* encryption schemes (in the sense of [18]); we leave this as an open problem for future work.

Ultimately these negative results indicate that for an r-BIND-secure encryptment scheme, the best we can hope for is either a rate-$\frac{1}{3}$ construction with a small set of keys, or to allow rekeying with each block of message. We therefore turn to building as efficient-as-possible constructions.

In Appendix I, we will describe how the existence of an r-BIND-secure ccAEAD scheme of a given rate implies the existence of an r-BIND-secure encryptment scheme of the same rate, and so the results of this section exclude the existence of rate-1 or rate-$\frac{1}{2}$ ccAEAD schemes also.

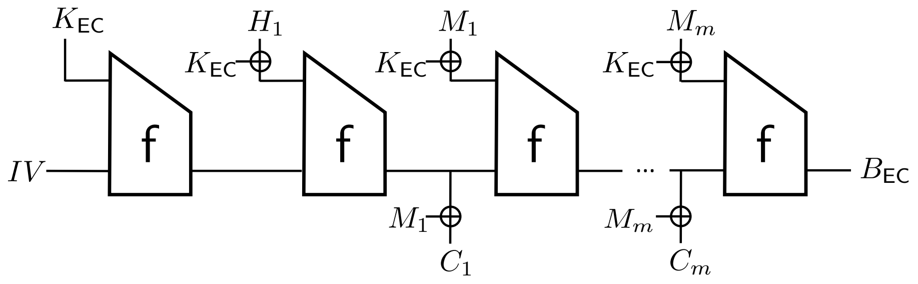


Figure 9: Encryptment in the HFC scheme for a 1-block header and m-block message. For simplicity the diagram does not show the details of padding.

6 Encryptment from Hashing

In this section, we turn our attention to building secure and efficient encryptment schemes. As we shall see in Section 7, these can be lifted to multi-opening, many-time secure ccAEAD via simple and efficient transforms.

As one might expect given the close relationship between binding and CR hashing discussed previously in Section 5, our starting point will be cryptographic hashing. A slightly simplified version of our construction is shown in Figure 9 (padding details are omitted), where f is a compression function. In summary, the scheme hashes the key, associated data and message data (the latter two of which are repeatedly XOR'd with the key). Intermediate chaining variables from the hash computation are used as pads to encrypt the message data, while the final chaining variable constitutes the binding tag.

Intuitively, (strong) receiver binding derives from the collision resistance of the underlying hash function. We XOR the key into all the associated data and message blocks to ensure that every application of the compression function is keyed. This is critical; just prepending (or both prepending and appending) the key to the data leads to a scheme whose confidentiality is easily broken. Likewise one cannot dispense with the additional initial block that simply processes the key, otherwise the encoding of the key, associated data, and message would not be injective and binding attacks result.

Some notation. Before defining the full scheme, we first give some additional notation which will simplify the presentation.

The algorithm Parse_d is used to partition a string into d-bit blocks. Formally, we define Parse_d to be the algorithm which on input X outputs $(X_1, \ldots, X_\ell)$ such that $|X_i| = d$ for $1 \le i \le \ell - 1$ and $|X_\ell| = |X| \mod d$. For correctness, we require that $X = X_1 \| \ldots \| X_\ell$. Similarly, we define Trunc_r to be the algorithm which on input X outputs the r leftmost bits of X. We write $\langle y \rangle_{64}$ to be the encoding of y as a 64-bit string.

The padding scheme is parameterized by positive integers d and n, but we omit these in the notation for simplicity. We assume $d \ge n \ge 128$. Our scheme utilizes a padding scheme $\mathsf{PadS} = (\mathrm{PadH}, \mathrm{PadM}, \overline{\mathrm{PadM}}, \mathrm{PadSuf}, \mathrm{Pad})$. The header and message padding functions PadH, PadM take as input a pair (H, M), and return tuples $(H_1, \ldots, H_h)$ and $(M'_1, \ldots, M'_{m-1})$ respectively. We require that $H_i, M'_j \in \{0,1\}^d$ for $i = 1, \ldots, h$, and $j = 1, \ldots, m-1$. We abuse notation to let $\mathrm{PadH}(H, M)$ (resp. $\mathrm{PadM}(H, M)$) denote the concatenation of the blocks returned by these algorithms. The online padding algorithm $\overline{\mathrm{PadM}}$ takes as input a tuple (H, M_i, i) where $M_i \in \{0,1\}^n$, and outputs $M'_i \in \{0,1\}^d$. We require that for any pair (H, M), it holds that $\overline{\mathrm{PadM}}(H, M_i, i) = M'_i$ for $i = 1, \ldots, m-1$, where $(M'_1, \ldots, M'_{m-1}) \leftarrow \mathrm{PadM}(H, M)$. This allows PadM to be computed in an online manner, with the message data being delivered in n-bit blocks. Finally, we define PadSuf to be the algorithm which takes as input $\ell_H, \ell_M \in \mathbb{N}^2$ and a string $X \in \{0,1\}^{\leq d}$, and outputs a string Y such that $\mathrm{Trunc}_{|X|}(Y) = X$, and Y is a multiple of d-bits. The full padding function is then defined to be $\mathrm{Pad}(H, M) = \mathrm{PadH}(H, M) \parallel \mathrm{PadM}(H, M) \parallel \mathrm{PadSuf}(|H|, |M|, M_m)$ where M_m denotes the final message block in the output of $\mathrm{Parse}_n(M)$. Note that d divides $|\mathrm{Pad}(H, M)|$.

Padding scheme properties. To prove the security of HFC as an encryptment scheme, the padding scheme Pad must be *injective*. An example of such a padding scheme is shown in Figure 10, and we shall assume that HFC is instantiated with this scheme unless stated otherwise. Our padding scheme is a variant of MD strengthening. We will not rely on the strengthening for its traditional purpose of forming a suffix-free padding scheme; we use strengthening only for injectivity and will assume more of f. If we would additionally like to prove the otCTXT-security of HFC, we require that the padding scheme satisfies the stronger property of *prefix-freeness*, meaning that if $(H, M) \neq (H', M')$ then $\mathrm{Pad}(H, M)$ is not a prefix of $\mathrm{Pad}(H', M')$. This can be achieved by, for example, adding frame-bits to distinguish the final block from the rest of the header and message data.

```
$\underline{\mathrm{PadH}(H, M)}$:
$(H_1, \ldots, H_h) \leftarrow \mathrm{Parse}_d(H \parallel 0^{d-|H| \bmod d})$
Return $(H_1, \ldots, H_h)$

$\underline{\mathrm{PadM}(H, M)}$:
$(M_1, \ldots, M_m) \leftarrow \mathrm{Parse}_n(M)$
For $i = 1, \ldots, m-1$
    $M'_i \leftarrow M_i \parallel 0^{d-n}$
Return $(M'_1, \ldots, M'_{m-1})$

$\underline{\overline{\mathrm{PadM}}(H, M_i, i)}$:
$M'_i \leftarrow M_i \parallel 0^{d-n}$
Return $M'_i$

$\underline{\mathrm{PadSuf}(\ell_H, \ell_M, M_m)}$:
$p \leftarrow \min\{i \in \mathbb{N} \,:\, d \mid ((\ell_M \bmod n) + i + 128)\}$
Return $M_m \parallel 0^p \parallel \langle \ell_H \rangle_{64} \parallel \langle \ell_M \rangle_{64}$
```

Figure 10: Padding scheme $\mathsf{PadS} = (\mathrm{PadH}, \mathrm{PadM}, \overline{\mathrm{PadM}}, \mathrm{PadSuf}, \mathrm{Pad})$. We require that $\ell_H, \ell_M \in \mathbb{N}$.

Iterated functions. Next we define iterated functions. Let $\mathsf{f}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$ be a function for some $d \geq n \geq 128$, let $D^+ = \cup_{i \geq 1} \{0,1\}^{id}$ and let $V_0 \in \{0,1\}^n$. Then $\mathsf{f}^+\colon \{0,1\}^n \times D^+ \to \{0,1\}^n$ denotes the *iteration* of f, where $\mathsf{f}^+(V_0, X_1 \parallel \cdots \parallel X_m) = V_m$ is computed via $V_i = \mathsf{f}(V_{i-1}, X_i)$ for $1 \leq i \leq m$.

The HFC encryptment scheme. The hash-function-chaining encryptment scheme $\mathsf{HFC} = (\mathsf{HFCKg}, \mathsf{HFCEnc}, \mathsf{HFCDec}, \mathsf{HFCVer})$ is based on a compression function $\mathsf{f}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$. The pseudocode for the encryptment and decryptment algorithms is presented in Figure 11.

Key generation HFCKg chooses $K_{\mathsf{EC}} \leftarrow\!\!\$\ \{0,1\}^d$. Encryptment first pads the header and message using the padding functions PadH and PadM respectively. We let $IV \in \{0,1\}^n$ be a fixed constant value (also called an initialization vector). The scheme computes an initial chaining variable as $V_0 = \mathsf{f}(IV, K_{\mathsf{EC}})$. It then hashes $\mathrm{PadH}(H, M) \parallel \mathrm{PadM}(H, M) \parallel \mathrm{PadSuf}(|H|, |M|, M_m)$ (where M_m denotes the final block returned by $\mathrm{Parse}_n(M)$) with f^+, the iteration of the compression function f, where the secret encryptment key K_{EC} is XORed into each d-bit block prior to hashing. The final chaining variable produced by this process forms the binding tag B_{EC}. Notice that while the compression function takes d-bit inputs, the way in which the message data is padded means we only process n-bits of message in each compression function call; looking ahead, this is to ensure decryptability. We will see that the collision resistance of the iterated hash function when instantiated with an appropriate compression function implies the sr-BIND security of the construction.

Rather than running a separate encryption algorithm alongside this process to encrypt the message, we instead generate ciphertext blocks by XORing the message blocks M_i with intermediate chaining variables, yielding $C_i = V_{h+i-1} \oplus M_i$ for $1 \leq i \leq m$ where h denotes the number of header blocks. Recall that in our notation $X \oplus Y$ silently truncates the longer string to the length of

```
HFCEnc(K_EC, H, M):
$(H_1, \ldots, H_h) \leftarrow \mathrm{PadH}(H, M)$
$(M_1, \ldots, M_m) \leftarrow \mathrm{Parse}_n(M)$
$(M'_1, \ldots, M'_{m-1}) \leftarrow \mathrm{PadM}(H, M)$
$V_0 \leftarrow \mathsf{f}(IV, K_{\mathsf{EC}})$
$V_h \leftarrow \mathsf{f}^+\big(V_0, (K_{\mathsf{EC}} \oplus H_1) \,\|\, \cdots \,\|\, (K_{\mathsf{EC}} \oplus H_h)\big)$
$C_{\mathsf{EC}} \leftarrow \varepsilon$
For $i = 1, \ldots, m-1$ do
    $C_{\mathsf{EC}} \leftarrow C_{\mathsf{EC}} \,\|\, (V_{h+i-1} \oplus M_i)$
    $V_{h+i} \leftarrow \mathsf{f}(V_{h+i-1}, (K_{\mathsf{EC}} \oplus M'_i))$
$C_{\mathsf{EC}} \leftarrow C_{\mathsf{EC}} \,\|\, (V_{h+m-1} \oplus M_m)$
$M'_m, M'_{m+1} \leftarrow \mathrm{Parse}_d(\mathrm{PadSuf}(|H|, |M|, M_m))$
$B_{\mathsf{EC}} \leftarrow \mathsf{f}^+(V_{h+m-1}, (K_{\mathsf{EC}} \oplus M'_m) \,\|\, (K_{\mathsf{EC}} \oplus M'_{m+1}))$
Return $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$
```

```
HFCDec(K_EC, H, C_EC, B_EC):
$(H_1, \ldots, H_h) \leftarrow \mathrm{PadH}(H, C_{\mathsf{EC}})$
$(C_1, \ldots, C_m) \leftarrow \mathrm{Parse}_n(C_{\mathsf{EC}})$
$V_0 \leftarrow \mathsf{f}(IV, K_{\mathsf{EC}})$
$V_h \leftarrow \mathsf{f}^+\big(V_0, (K_{\mathsf{EC}} \oplus H_1) \,\|\, \cdots \,\|\, (K_{\mathsf{EC}} \oplus H_h)\big)$
For $i = 1, \ldots, m-1$ do
    $M_i \leftarrow V_{h+i-1} \oplus C_i$ ; $M'_i \leftarrow \overline{\mathrm{PadM}}(H, M_i, i)$
    $V_{h+i} \leftarrow \mathsf{f}(V_{h+i-1}, (K_{\mathsf{EC}} \oplus M'_i))$
$M_m \leftarrow V_{h+m-1} \oplus C_m$
$M'_m, M'_{m+1} \leftarrow \mathrm{Parse}_d(\mathrm{PadSuf}(|H|, |C_{\mathsf{EC}}|, M_m))$
$B'_{\mathsf{EC}} \leftarrow \mathsf{f}^+(V_{h+m-1}, (K_{\mathsf{EC}} \oplus M'_m) \,\|\, (K_{\mathsf{EC}} \oplus M'_{m+1}))$
If $B'_{\mathsf{EC}} \neq B_{\mathsf{EC}}$ then
    Return $\perp$
Return $M_1 \,\|\, \cdots \,\|\, M_m$
```

Figure 11: The HFC encryptment scheme HFC built from a compression function $\mathsf{f}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$ and padding scheme $\mathsf{PadS} = (\mathrm{PadH}, \mathrm{PadM}, \overline{\mathrm{PadM}}, \mathrm{PadSuf}, \mathrm{Pad})$. Here $K_{\mathsf{EC}} \in \{0,1\}^d$, and $IV \in \{0,1\}^n$ is a fixed public constant.

the shorter string, and so only the n-bits of message data in each d-bit padded message block is XORed with the n-bit chaining variable; similarly, if message M is such that $|M| \bmod n = r$, then the final ciphertext block produced by this process is truncated to the leftmost r-bits. The properties of the compression function ensure that the chaining variables are pseudorandom, thus yielding the required otROR security. By 'reusing' chaining variables as random pads we can achieve encryptment with no additional overhead over just computing the binding tag, yielding better efficiency (see further discussion below).

Decryption $\mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}})$ begins by padding H into d-bit blocks via $\mathrm{PadH}(H, M)$ and parsing C_{EC} into n-bit blocks. The algorithm computes the initial chaining variable as $V_0 = \mathsf{f}(IV, K_{\mathsf{EC}})$, then hashes the padded header as in encryption. The scheme then recovers the first message block M_1 by XORing the chaining variable into the first ciphertext block C_1. This is padded via the online padding function $\overline{\mathrm{PadM}}(H, M_1, 1)$, and then used to compute the next chaining variable via application of f, and so on. Notice how at most n-bits of message data is recovered in each such step; this is why we must process only n-bits of message data in each compression function call, else the decryptor would be unable to compute the next chaining variable. Finally, DO recomputes and verifies the binding tag, returning the message only if verification succeeds.

The verification algorithm (not shown), on input $(K_{\mathsf{EC}}, H, M, B_{\mathsf{EC}})$, pads the message to $\mathrm{PadH}(H, M) \,\|\, \mathrm{PadM}(H, M) \,\|\, \mathrm{PadSuf}(|H|, |M|, M_m)$, XORs K_{EC} into every block, and hashes the resulting string with f^+ with initial chaining variable $V_0 = \mathsf{f}(IV, K_{\mathsf{EC}})$, checking that the output matches the binding tag B_{EC}.

Efficiency. The efficiency of the scheme (in terms of throughput) depends on the parameters d, n, where recall that $\mathsf{f}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$. As discussed previously, at most n-bits of message data can be processed in each compression function call. As such, the HFC encryptment scheme achieves optimal throughput when $d = n$. In this case no padding is applied to the message blocks, and so computing the full encryptment incurs *no overhead* over simply computing the binding tag.

If $d > n$, then some throughput is lost due to padding. We present an alternative padding scheme for this case, which recovers some throughput by padding message blocks with header data. In more detail, consider the padding scheme $\mathsf{AltPadD} = (\mathrm{AltPadH}, \mathrm{AltPadM}, \overline{\mathrm{AltPadM}}, \mathrm{AltPadSuf}, \mathrm{AltPad})$ shown in Figure 12, where AltPadSuf is defined identically to PadSuf in Figure 10. The scheme parses the message data into n-bit blocks and the header data into $(d-n)$-bits blocks. It then con-

```
AltPadH(H, M):
$(H_1, \ldots, H_h) \leftarrow \text{Parse}_{d-n}(H)$
$(M_1, \ldots, M_m) \leftarrow \text{Parse}_n(M)$
$\beta \leftarrow \min(m, h)$
If $\beta = h$ then
    Return $\varepsilon$
Else $(H_{\beta+1}, \ldots, H_h) \leftarrow \text{Parse}_d(H_{\beta+1} \parallel \ldots \parallel H_h)$
    $H_h \leftarrow H_h \parallel 0^{d-|H| \bmod d}$
Return $(H_{\beta+1}, \ldots, H_h)$
```

```
AltPadM(H, M):
$(H_1, \ldots, H_h) \leftarrow \text{Parse}_{d-n}(H)$
$(M_1, \ldots, M_m) \leftarrow \text{Parse}_n(M)$
$\beta \leftarrow \min(m, h)$
For $i = 1, \ldots, \beta$ do
    $X_i \leftarrow M_i \parallel H_i$
If $\beta = h$ then for $i = \beta + 1, \ldots, m - 1$
    $M'_i \leftarrow M_i \parallel 0^{d-n}$
Return $(X_1, \ldots, X_\beta, M'_{\beta+1}, \ldots, M_m)$
```

```
$\overline{\text{AltPadM}}(H, M_i, i)$:
$(H_1, \ldots, H_h) \leftarrow \text{Parse}_{d-n}(H)$
If $i \leq h$ then
    $M'_i \leftarrow M_i \parallel H_i$
Else $M'_i \leftarrow M_i \parallel 0^{d-n}$
Return $M'_i$
```

Figure 12: Alternative padding scheme AltPadD.

structs d-bit blocks by padding the message blocks with header blocks. Any header / message data remaining after this process is padded unambiguously similarly to the previous scheme (Figure 10). It is straightforward to verify that AltPad is injective.

6.1 Analyzing the HFC Encryptment Scheme

We now analyze the security of the HFC encryptment scheme, relative to the security goals detailed in Section 4.

Strong receiver binding. We begin by proving that the HFC encryptment scheme satisfies strong receiver binding provided the underlying padding function is injective. Observe that the binding tag computation performed by the encryptment algorithm HFCEnc is equivalent to hashing an encoding of the input tuple (K_{EC}, H, M) with f^+. The encoding — which consists of padding the header and message, XORing the key K_{EC} into each block and then prepending K_{EC} — is injective provided the padding function Pad is injective. As such, any tuple breaking the sr-BIND security of HFC is a collision against f^+.

A well-known folklore result (see [3]) gives that f^+ is collision-resistant provided the underlying compression function is both collision-resistant and that it is hard to find an input which hashes to the IV. Standard compression functions satisfy both properties. The full proof of the following is given in Appendix H. The conditions on d, n in the theorem can be relaxed; the conditions arise from our choice of padding.

Theorem 3 *Let* HFC *be the scheme defined above using compression function* $\mathsf{f}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$ *for* $d \geq n \geq 128$, *and instantiated with an injective padding scheme* PadS. *Then for any* sr-BIND$_{HFC}$ *adversary* $\mathcal{A}$, *we detail an adversary* $\mathcal{B}$ *such that* $\mathbf{Adv}^{\text{sr-bind}}_{\mathsf{HFC}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{cr}}_{\mathsf{f}^+}(\mathcal{B})$, *where adversary* $\mathcal{B}$ *runs in the same time as* $\mathcal{A}$.

Sender binding and correctness. The s-BIND security of HFC is immediate because decryption verifies the binding tag. Similarly, it is straightforward to verify that the scheme is strongly correct. Therefore Lemma 1 allows us to bound the SCU security of HFC as an immediate consequence of these observations coupled with Theorem 3.

One-time confidentiality. All that remains to prove that HFC is a secure encryptment scheme is to bound its otROR security. We do this in the next theorem, by reducing otROR security of HFC to a the related-key attack (RKA) PRF security [4] of f for a specific class of related-key deriving functions.

Let $F\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$ be a function, and consider the games RKA-PRF0 and RKA-PRF1. In both games the attacker is given access to an oracle to which he may submit queries of the form $(X, Y) \in \{0,1\}^n \times \{0,1\}^d$. In game RKA-PRF0 the oracle returns $F(X, Y \oplus K_{\mathrm{prf}})$ where K_{prf} is a PRF key. In game RKA-PRF1, the oracle returns a random value for each query, answering consistently for repeat queries. The *linear-only RKA-PRF* advantage of an adversary $\mathcal{A}$ is defined as

$$\mathbf{Adv}_F^{\oplus\text{-prf}}(\mathcal{A}) = \left|\Pr\left[\,\text{RKA-PRF0}_F^{\mathcal{A}} \Rightarrow 1\,\right] - \Pr\left[\,\text{RKA-PRF1}^{\mathcal{A}} \Rightarrow 1\,\right]\right|$$

where the probabilities are over the coins used in the games.

We can bound the otROR security of HFC by a game hopping argument, first arguing that we can replace compression function calls with random functions by a reduction to the LKA-PRF security of f, and then replacing the random function outputs with random bit strings using a birthday-bound argument. The full proof is given in Appendix H.

Theorem 4 *Let* HFC *be the shown in Figure 11, built from a compression function* $\mathsf{f}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$ *for* $d \geq n \geq 128$*, and instantiated with an injective padding scheme* PadS*. Let* $\mathcal{A}$ *be an otROR adversary whose encryption query totals at most* ℓ *blocks of* d *bits after padding. Then there exists an adversary* $\mathcal{B}$ *such that* $\mathbf{Adv}_{\mathsf{HFC}}^{\text{ot-ror}}(\mathcal{A}) \leq \mathbf{Adv}_{\mathsf{f}}^{\oplus\text{-prf}}(\mathcal{B}) + \frac{(\ell+1)^2}{2^{n+1}}$*. The adversary* $\mathcal{B}$ *runs in time that of* $\mathcal{A}$ *plus* $\mathcal{O}(\ell)$ *overhead and makes at most* $(\ell+1)$ *queries.*

One-time integrity and ccAEAD security. In the following theorem, we bound the otCTXT secure of HFC under the additional assumption that the padding scheme used is *prefix-free*. As discussed in Section 4, this result combined with the above imply that HFC (reframed in the ccAEAD syntax) is a secure one-time ccAEAD scheme.

The proof first argues that we can replace compression function calls with random functions by a reduction to the LKA-PRF security of f. It then uses the prefix-freeness of the padding scheme to argue that (barring accidental collisions amongst intermediate chaining variables, accounting for the birthday bound term), each binding tag is computed as the result of a fresh query to the random function, and so the probability that $\mathcal{A}$ can guess the value of an unseen binding tag is small (bounding this guessing probability accounts for the final term below). The full proof is given in Appendix H.

We note that without a prefix-free padding scheme, HFC is not otCTXT secure in general. Indeed for the padding scheme of Figure 10, an attacker can construct a pair (H, M) for his enc query, such that a prefix of $\text{Pad}(H, M)$ is equal to $\text{Pad}(H, M')$ for some $(H, M) \neq (H, M')$. By choosing (H, M) such that the binding tag for (H, M') is among the random pads used to encrypt M, $\mathcal{A}$ can then recover the correct binding tag and random pads for (H, M'), from the ciphertext component C_{EC} corresponding to (H, M). This in turn enables $\mathcal{A}$ to construct a valid forgery for (H, M'). It seems likely that small modifications to the HFC construction would achieve otCTXT security while only requiring an injective padding scheme. For example, consider a modified scheme HFC' which takes as input a key $(K_{\mathsf{EC}}, K'_{\mathsf{EC}}) \leftarrow_{\$} \{0,1\}^d \times \{0,1\}^d$. Modified encryptment HFCEnc' on input $((K_{\mathsf{EC}}, K'_{\mathsf{EC}}), H, M)$ computes $\mathsf{HFCEnc}(K_{\mathsf{EC}}, H, M) = (C_{\mathsf{EC}}, B_{\mathsf{EC}})$, and outputs $(C_{\mathsf{EC}}, (B_{\mathsf{EC}}, \mathsf{f}(B_{\mathsf{EC}}, K'_{\mathsf{EC}})))$, where we call the latter value in the tuple the authentication tag. To decrypt $(H, C_{\mathsf{EC}}, (B_{\mathsf{EC}}, B'_{\mathsf{EC}}))$, HFCDec' computes $\mathsf{HFCDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$, and checks that $B'_{\mathsf{EC}} = \mathsf{f}(B_{\mathsf{EC}}, K'_{\mathsf{EC}})$, returning $\perp$ if either of these steps returns an error. Let $(H^*, C^*_{\mathsf{EC}}, (B^*_{\mathsf{EC}}, B^{*'}_{\mathsf{EC}}))$ denote the encryptment arising from the attacker's enc query in game otCTXT against HFC'. Now the attacker will be required to guess a random authentication tag in order to create a successful forgery, *unless* they can find $(H, C_{\mathsf{EC}}) \neq (H^*, C^*_{\mathsf{EC}})$ such that $\mathsf{HFCDec}(H, C_{\mathsf{EC}}, B^*_{\mathsf{EC}}) \neq \perp$, which in turn breaks the SCU security of the underlying HFC scheme. We leave formalizing this intuition and proving the properties of such a modified scheme as an open problem.

Theorem 5 *Let* HFC *be as shown in Figure 11, built from a compression function* $\mathsf{f}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$ *for* $d \geq n \geq 128$*, and instantiated with a prefix-free padding scheme* PadS*. Let* $\mathcal{A}$ *be an otCTXT adversary who makes* q *decryption queries, among which there are* q_1 *distinct header / ciphertext pairs* (H, C_{EC})*. Suppose that the header / message (from the single encryption query), and the* q_1 *distinct header / ciphertexts (from the decryption queries) have a combined length of* ℓ *blocks of* d *bits after padding. Then there exists an adversary* $\mathcal{B}$ *such that* $\mathbf{Adv}^{\mathrm{ot-ctxt}}_{\mathsf{HFC}}(\mathcal{A}) \leq \mathbf{Adv}^{\oplus\text{-prf}}_{\mathsf{f}}(\mathcal{B}) + \frac{(\ell+1)^2}{2^{n+1}} + \frac{q}{2^n-\ell-q}$*. The adversary* $\mathcal{B}$ *runs in time that of* $\mathcal{A}$ *plus an* $\mathcal{O}(\ell)$ *overhead and makes at most* $(\ell+1)$ *queries.*

Instantiations. The obvious (and probably best) choice to instantiate f is the SHA-256 or SHA-512 compression function. These provide good software performance, and there is a shift towards widespread hardware support in the form of the Intel SHA instructions [13, 20, 47]. Extensive cryptanalysis for the CR (e.g., [26, 30, 44]), preimage resistance (e.g., [21, 26]), and RKA-PRP of the associated SHACAL-2 blockcipher (e.g., [23, 27, 28, 32]) gives confidence in its security.

Other options, although in some cases less well-studied cryptanalytically, include SHA-3 finalists. In particular, a variant of the HFC construction using a sponge-based mode such as Keccak, in which the key is fed to the sponge prior to hashing the message blocks, would allow us to avoid the RKA assumption. We could also remove the assumption by using a compression function with a dedicated key input such as LP231 [42].

BLAKE2b [2] is a variant of the BLAKE hash function and was also a SHA-3 finalist. Its compression function is not explicitly blockcipher-based; it is built from a variant of the quarter-round function of the ChaCha20 stream cipher [6]. It is believed to have security comparable to SHA-256, but is notable for efficiency—on some platforms it outperforms even MD5 in software.

Another approach would be to use AES via a PGV compression function [38]; we focus our discussion further on Davies-Meyer (DM), one of the secure PGV constructions [12]. Letting $E\colon \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^n$ be a blockcipher, DM is defined as $\mathsf{DM}(V, M) = E_M(V) \oplus V$. For HFC, an important advantage of DM over other PGV compression functions (e.g., Matyas-Meyer-Oseas) is that the blockcipher keys do not depend on intermediate chaining values, so key scheduling for the entire (encoded) message can be done up front. Another advantage of DM is that the linear-only RKA-PRF security of DM is inherited from a linear-only RKA-PRP security of the underlying cipher. We discuss this further in Appendix G. An obvious choice for E would be AES-128. On systems with AES-NI, HFC instantiated with DM-AES will have very good performance, though not quite as fast as AES-GCM or OCB given the need to rekey every block. Security of AES has been studied extensively, and known attacks do not falsify the assumptions we need [9, 10]. On systems with AES-NI, HFC instantiated with DM-AES will have very good performance. More problematic is that binding can only hold up 2^{64}, which is in general insufficient in practice.

7 Compactly Committing AEAD from Encryptment

In this section we recall the formal notions for compactly committing AEAD schemes (ccAEAD schemes), following the treatment given by GLR [19], and compare these to encryptment. With this in place, we show in Section 7.3 how to build ccAEAD from encryptment with very efficient transforms. In Appendix I we will show how to construct a secure encryptment scheme from a ccAEAD scheme in a way that transfers our negative results from Section 5 to ccAEAD.

7.1 ccAEAD Syntax and Correctness

Encryptment can be viewed as a one-time secure, deterministic variant of ccAEAD. We discuss further the differences between the two primitives later in the section.

ccAEAD schemes. Formally, a ccAEAD scheme is a tuple of algorithms $\mathsf{CE} = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ver})$ with associated key space $\mathcal{K} \subseteq \Sigma^*$, header space $\mathcal{H} \subseteq \Sigma^*$, message space $\mathcal{M} \subseteq \Sigma^*$, ciphertext space $\mathcal{C} \subseteq \Sigma^*$, opening space $\mathcal{K}_f \subseteq \Sigma^*$, and binding tag space $\mathcal{T} \subseteq \Sigma^*$, defined as follows. The randomized key generation algorithm Kg takes no input, and outputs a secret key $K \in \mathcal{K}$. The randomized encryption algorithm Enc takes as input a tuple $(K, H, M) \in \mathcal{K} \times \mathcal{H} \times \mathcal{M}$ and outputs a ciphertext / binding tag pair $(C, C_B) \in \mathcal{C} \times \mathcal{T}$. The deterministic decryption algorithm Dec takes as input a tuple $(K, H, C, C_B) \in \mathcal{H} \times \mathcal{M} \times \mathcal{C} \times \mathcal{T}$, and outputs a message / opening pair $(M, K_f) \in \mathcal{M} \times \mathcal{K}_f$ or the error symbol $\perp$. The deterministic verification algorithm Ver takes as input a tuple $(H, M, K_f, C_B) \in \mathcal{H} \times \mathcal{M} \times \mathcal{K}_f \times \mathcal{T}$, and outputs a bit b. We assume that if Dec and Ver are queried on inputs which do not lie in their defined input spaces, then they return $\perp$ and 0 respectively.

Correctness and compactness. Correctness for ccAEAD schemes is defined identically to the COR correctness notion for encryptment schemes (Figure 5), except in the ccAEAD case the probability is now over the coins of Enc also. We require that the structure of ciphertexts C depend only on the length of the underlying message. Formally, let $M^* = \{i \mid \exists m \in \mathcal{M} \colon |m| = i\}$. Then we require that the ciphertext space $\mathcal{C}$ can be partitioned into disjoint sets $\mathcal{C}(i) \subseteq \mathcal{C}$, $i \in M^*$, such that for all $(H, M) \in \mathcal{H} \times \mathcal{M}$ it holds that $C \in \mathcal{C}(|M|)$ with probability one for the sequence of algorithm executions: $K \leftarrow{\$}\ \mathsf{Kg}$; $(C, C_B) \leftarrow{\$}\ \mathsf{Enc}(K, H, M)$. Finally, we require that the binding tags C_B are *compact*, by which we mean that all C_B returned by a ccAEAD scheme are of constant length blen which is linear in the key size.

Comparison with encryptment. With this in place, we highlight the key differences between encryptment and ccAEAD schemes. The overarching difference is that encryptment schemes are single-use (a key is only ever used to encrypt a single message), whereas ccAEAD schemes are multi-use. To support this, the encryption algorithm for ccAEAD schemes is randomized, whereas for encryptment this algorithm is deterministic. This is necessary for achieving schemes that enjoy security in the face of attackers that can obtain multiple encryptions. Moreover, while encryptment schemes are restricted to use the same key for verification as they use for encryptment, ccAEAD schemes output an explicit opening key K_f during decryption. There is no requirement that this equal the secret key used for encryption. Again, outputting an opening key distinct from the encryption key allows for ccAEAD schemes that maintain confidentiality and integrity even after some ciphertexts produced under a given encryption key have been opened.

AEAD schemes. The definition of an AEAD scheme $\mathsf{AEAD} = (\mathsf{AEAD.kg}, \mathsf{AEAD.enc}, \mathsf{AEAD.dec})$ (see Section 2) can be recovered from the above definition of ccAEAD schemes by noticing that each algorithm can be defined identically to their ccAEAD variants, except we view the ciphertext / binding tag pair as a single ciphertext, and modify decryption to no longer output the opening, in the AEAD case. This framing allows us to define security notions for AEAD schemes as a special case of those notions for ccAEAD schemes for conciseness and ease of comparison. Similarly regular AE schemes are defined to be the same as AEAD schemes but with all references to the header removed.

```
$\underline{\text{MO-REAL}^{\mathcal{A}}_{\mathsf{CE}}:}$
$K \leftarrow\!\!\$\ \mathsf{Kg}$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\textbf{Enc},\textbf{Dec},\textbf{ChalEnc}}$
Return $b'$

$\underline{\textbf{Enc}(H, M)}$
$(C, C_B) \leftarrow\!\!\$\ \mathsf{Enc}(K, H, M)$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, C_B)\}$
Return $(C, C_B)$

$\underline{\textbf{Dec}(H, C, C_B)}$
If $(H, C, C_B) \notin \mathcal{Y}$ then
    Return $\perp$
$(M, K_f) \leftarrow \mathsf{Dec}(K, H, C, C_B)$
Return $(M, K_f)$

$\underline{\textbf{ChalEnc}(H, M)}$
$(C, C_B) \leftarrow\!\!\$\ \mathsf{Enc}(K, H, M)$
Return $(C, C_B)$
```

```
$\underline{\text{MO-RAND}^{\mathcal{A}}_{\mathsf{CE}}:}$
$K \leftarrow\!\!\$\ \mathsf{Kg}$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\textbf{Enc},\textbf{Dec},\textbf{ChalEnc}}$
Return $b'$

$\underline{\textbf{Enc}(H, M)}$
$(C, C_B) \leftarrow\!\!\$\ \mathsf{Enc}(K, H, M)$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, C_B)\}$
Return $(C, C_B)$

$\underline{\textbf{Dec}(H, C, C_B)}$
If $(H, C, C_B) \notin \mathcal{Y}$ then
    Return $\perp$
$(M, K_f) \leftarrow \mathsf{Dec}(K, H, C, C_B)$
Return $(M, K_f)$

$\underline{\textbf{ChalEnc}(H, M)}$
$(C, C_B) \leftarrow\!\!\$\ \mathcal{C}(|M|) \times \{0,1\}^{\mathsf{blen}}$
Return $(C, C_B)$
```

```
$\underline{\text{MO-CTXT}^{\mathcal{A}}_{\mathsf{CE}}:}$
$K \leftarrow\!\!\$\ \mathsf{Kg}$ ; $\mathsf{win} \leftarrow \mathsf{false}$
$\mathcal{A}^{\textbf{Enc},\textbf{Dec},\textbf{ChalDec}}$
Return $\mathsf{win}$

$\underline{\textbf{Enc}(H, M)}$
$(C, C_B) \leftarrow\!\!\$\ \mathsf{Enc}(K, H, M)$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, C_B)\}$
Return $(C, C_B)$

$\underline{\textbf{Dec}(H, C, C_B)}$
Return $\mathsf{Dec}(K, H, C, C_B)$

$\underline{\textbf{ChalDec}(H, C, C_B)}$
If $(H, C, C_B) \in \mathcal{Y}$ then
    Return $\perp$
$(M, K_f) \leftarrow \mathsf{Dec}(K, H, C, C_B)$
If $M \neq \perp$ then
    $\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_f)$
```

Figure 13: Confidentiality (left two games) and ciphertext integrity (rightmost) games for ccAEAD.

7.2 Security Notions for Compactly Committing AEAD

We now define the security notions for ccAEAD schemes, following GLR. They adapt the familiar security notions of real-or-random (ROR) ciphertext indistinguishability [41], and ciphertext integrity (CTXT) [5] for AE schemes to the ccAEAD setting. We focus on GLR's *multi-opening* (MO) security notions. MO-ROR (resp. MO-CTXT) requires that if multiple messages are encrypted under the same key, then learning the message / opening pair (M, K_f) for some of the resulting ciphertexts does not compromise the ROR (resp. CTXT) security of the remaining unopened ciphertexts. This precludes schemes which for example have the opening key K_f equal to the secret encryption key K.

Confidentiality. Games MO-REAL and MO-RAND are shown in Figure 13. In both variants, the attacker is given access to an oracle **ChalEnc** to which he may submit message / header pairs. This oracle returns real (resp. random) ciphertext / binding tag pairs in game MO-REAL (resp. MO-RAND). The attacker is then challenged to distinguish between the two games. To model multi-opening security, the attacker is also given a pair of encryption / decryption oracles, **Enc** and **Dec**, and may submit the (real) ciphertexts generated via a query to the former to the latter, learning the openings of these ciphertexts in the process. The challenge decryption oracle will return $\perp$ for any ciphertext not generated via the encryption oracle, to prevent the attacker trivially winning by decrypting a ciphertext returned by **ChalEnc**. We define the advantage of an attacker $\mathcal{A}$ in game MO-ROR against a ccAEAD scheme CE as

$$\mathbf{Adv}^{\text{mo-ror}}_{\mathsf{CE}}(\mathcal{A}) = \left|\Pr\left[\,\text{MO-REAL}^{\mathcal{A}}_{\mathsf{CE}} \Rightarrow 1\,\right] - \Pr\left[\,\text{MO-RAND}^{\mathcal{A}}_{\mathsf{CE}} \Rightarrow 1\,\right]\right| \ .$$

Ciphertext integrity. Ciphertext integrity guarantees that an attacker cannot produce a fresh ciphertext which will decrypt correctly. The multi-opening adaptation to the ccAEAD setting MO-CTXT is shown in Figure 13. The attacker $\mathcal{A}$ is given access to encryption oracle **Enc** and a challenge decryption oracle **ChalDec**. The attacker wins if he submits a ciphertext to **ChalDec** which decrypts correctly and which wasn't the result of a previous query to the encryption oracle.

To model multi-opening security, the attacker is given access to a further oracle **Dec** via which he may decrypt ciphertexts and learn the corresponding openings. The advantage of an attacker $\mathcal{A}$ in game MO-CTXT against a ccAEAD scheme CE is then defined

$$\mathbf{Adv}^{\text{mo-ctxt}}_{\mathsf{CE}}(\mathcal{A}) = \Pr\left[\text{ MO-CTXT}^{\mathcal{A}}_{\mathsf{CE}} \Rightarrow \mathsf{true} \right] .$$

Security for standard AEAD. We note that the familiar ROR and CTXT notions for AEAD schemes can be recovered from the corresponding ccAEAD games in Figure 13 by reframing the ccAEAD scheme as an AEAD scheme as described previously, removing access to oracle **Dec** in all games, and removing **Enc** in MO-REAL and MO-RAND. Advantage functions are defined analogously. Since here we are removing attacker capabilities, it follows that security for a ccAEAD scheme with respect to these notions implies security for the derived AEAD scheme also.

Receiver and sender binding. Strong receiver binding for ccAEAD schemes is the same as for encryptment (Figure 6), except the attacker outputs openings K_f, K'_f rather than secret keys K, K' as part of his guess. The sender binding game for a ccAEAD scheme challenges an attacker $\mathcal{A}$ to output a tuple (K, H, C, C_B) such that $(K_f, M) \leftarrow \mathsf{Dec}(K, H, C, C_B)$ does not equal $\perp$ but $\mathsf{Ver}(H, M, K_f, C_B) = 0$. This is the same as the associated game for encryptment, except that the opening K_f recovered during decryption is used for verification rather than the key output by $\mathcal{A}$. Given the similarities, we abuse notation by using the same names for ccAEAD binding notion games and advantage terms as in the encryptment case; which version will be clear from the context.

Given that both target certain binding notions, a natural question is whether an sr-BIND secure ccAEAD scheme is also robust [18], and vice versa. In Appendix D, we show that neither notion implies the other in generality. We also discuss the conditions under which the ccAEAD schemes we build from secure encryptment are robust.

7.3 Encryptment to ccAEAD Transforms

We now turn to building ccAEAD from encryptment. Fix an encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EC}, \mathsf{DO}, \mathsf{EVer})$ and a standard AEAD scheme $\mathsf{AEAD} = (\mathsf{AEAD.Kg}, \mathsf{AEAD.enc}, \mathsf{AEAD.dec})$. Let $\mathsf{CE}[\mathsf{EC}, \mathsf{AEAD}] = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ver})$ be the ccAEAD scheme whose encryption, decryption, and verification algorithms are shown in Figure 14. Key generation Kg runs $K \leftarrow\$ \mathsf{AEAD.Kg}$ and outputs K.

To encrypt a header / message (H, M), Enc uses the key generation algorithm of the encryptment scheme to generate a one-time encryptment key $K_{\mathsf{EC}} \leftarrow\$ \mathsf{EKg}$, and computes the encryptment of the header and message via $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$. The scheme then uses the encryption algorithm of the AEAD scheme to encrypt the one-time key K_{EC} with header B_{EC}, producing $C_{\mathsf{AE}} \leftarrow\$ \mathsf{AEAD.enc}(K, B_{\mathsf{EC}}, K_{\mathsf{EC}})$, and outputs $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$. On input $(K, (C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$, Dec computes $K_{\mathsf{EC}} \leftarrow \mathsf{AEAD.dec}(K, B_{\mathsf{EC}}, C_{\mathsf{AE}})$ and if $K_{\mathsf{EC}} = \perp$ returns $\perp$ since this clearly indicates that C_{AE} is invalid. The recovered key K_{EC} is in turn used to recover the message via $M \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$. If $M = \perp$, the scheme returns $\perp$; otherwise, EC returns (M, K_{EC}) as the message / opening pair. Ver simply applies the verification algorithm EVer of the underlying encryptment scheme to the input tuple and returns the result.

Notice that by including the binding tag B_{EC} as the header in the authenticated encryption, this ensures the integrity of B_{EC}. If we did not authenticate B_{EC} then an attacker could trivially break the MO-CTXT-security of the scheme by using an **Enc** query to obtain ciphertext $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$ for a pair (H, M), submitting that ciphertext to **Dec** to recover the opening / key K_{EC}, with which

```
Enc(K, H, M):
$K_{\mathsf{EC}} \leftarrow\!\!\$\ \mathsf{EKg}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow\!\!\$\ \mathsf{AEAD.enc}(K, B_{\mathsf{EC}}, K_{\mathsf{EC}})$
Return $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$

Dec(K, H, (C, C_B)):
$(C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
$K_{\mathsf{EC}} \leftarrow \mathsf{AEAD.dec}(K, B_{\mathsf{EC}}, C_{\mathsf{AE}})$
If $K_{\mathsf{EC}} = \bot$ then Return $\bot$
$M \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \bot$ then Return $\bot$
Return $(M, K_{\mathsf{EC}})$
```

Figure 14: A generic transform from an encryptment scheme EC and a standard authenticated encryption scheme AEAD to a multi-opening ccAEAD scheme $\mathsf{CE}[\mathsf{EC}, \mathsf{AEAD}]$. Verification simply runs EVer.

he can easily create a valid forgery by computing $(C'_{\mathsf{EC}}, B'_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H', M')$ for some distinct header / message pair and outputting $((C'_{\mathsf{EC}}, C_{\mathsf{AE}}), B'_{\mathsf{EC}})$. Including the binding tag as the header in the AEAD ciphertext means that an attacker trying to replicate the above mix-and-match attack must create a forgery for an encryptment binding tag and key already returned as the result of an **Enc** query, thus violating the SCU security of the underlying encryptment scheme.

Security of the transform. Next, we analyze the security of the ccAEAD scheme $\mathsf{CE}[\mathsf{EC}, \mathsf{AEAD}]$ shown in Figure 14. We begin with confidentiality. The proof of the following theorem follows from reductions to the ROR security of the underlying encryptment and AEAD schemes, and is given in Appendix J.

Theorem 6 *Let EC be an encryptment scheme, AEAD be an authenticated encryption scheme, and let $\mathsf{CE}[\mathsf{EC}, \mathsf{AEAD}]$ be the ccAEAD scheme built from EC according to Figure 14. Then for any adversary $\mathcal{A}$ in the* MO-ROR *game against CE making a total of q queries, of which q_c are to* **ChalEnc** *and q_e are to* **Enc**, *there exists adversaries $\mathcal{B}$ and $\mathcal{C}$ such that*

$$\mathbf{Adv}^{\text{mo-ror}}_{\mathsf{CE}}(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}^{\text{ror}}_{\mathsf{AEAD}}(\mathcal{B}) + q_c \cdot \mathbf{Adv}^{\text{ot-ror}}_{\mathsf{EC}}(\mathcal{C}) \ .$$

Adversaries $\mathcal{B}$ and $\mathcal{C}$ run in the same time as $\mathcal{A}$ with an $\mathcal{O}(q)$ overhead, and adversary $\mathcal{B}$ makes at most $q_c + q_e$ encryption oracle queries.

Next we bound the MO-CTXT advantage of any adversary against $\mathsf{CE}[\mathsf{EC}, \mathsf{AEAD}]$, via a reduction to the CTXT security of the underlying AEAD scheme, and the SCU security of the encryptment scheme. The proof is given in Appendix J.

Theorem 7 *Let EC be an encryptment scheme, AEAD be an authenticated encryption scheme, and let $\mathsf{CE}[\mathsf{EC}, \mathsf{AEAD}]$ be the ccAEAD scheme built from EC according to Figure 14. Then for any adversary $\mathcal{A}$ in the* MO-CTXT *game against CE making a total of q queries, of which q_e are to* **Enc**, *there exists adversaries $\mathcal{B}$ and $\mathcal{C}$ such that*

$$\mathbf{Adv}^{\text{mo-ctxt}}_{\mathsf{CE}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{ctxt}}_{\mathsf{AEAD}}(\mathcal{B}) + q_e \cdot \mathbf{Adv}^{\text{scu}}_{\mathsf{EC}}(\mathcal{C}) \ .$$

Adversaries $\mathcal{B}$ and $\mathcal{C}$ run in the same time as $\mathcal{A}$ with an $\mathcal{O}(q)$ overhead, and adversary $\mathcal{B}$ makes at most as many queries as $\mathcal{A}$.

We omit bounding the s-BIND and sr-BIND security of $\mathsf{CE}[\mathsf{EC}, \mathsf{AEAD}]$, since CE inherits these properties directly from EC. By reframing CE as a regular AEAD scheme, our transform yields a

ROR and CTXT secure single-pass AEAD scheme. To implement the transform, the fixed-input-length AE scheme must be instantiated. One can use, for example, AES-GCM or OCB.

An alternate transform. We provide one other approach for building ccAEAD from encryptment. It uses a PRF, most simply re-using the compression function f used in HFC. It uses a long-term key K. Instead of generating the encryptment key K_{EC} randomly, this instantiation will compute $K_{\mathsf{EC}} \leftarrow \mathsf{f}(N, K \oplus \text{fpad})$ for some randomly-chosen nonce $N \leftarrow_{\$} \{0,1\}^n$ and a fixed constant string fpad. After EC outputs $C_{\mathsf{EC}}, B_{\mathsf{EC}}$, the compression function will again be used to compute $C_{\mathsf{AE}} \leftarrow \mathsf{f}(B_{\mathsf{EC}}, K \oplus \text{spad})$ for a fixed constant string spad (distinct from fpad). Decryption is implemented in the obvious way. In Appendix K we will prove that as long as f is a good RKA-PRF, this transform results in a secure MO-ccAEAD scheme. This transform is also conceptually elegant, relying only on a single cryptographic primitive.

8 Other Applications of Encryptment

The applications of encryptment extend beyond ccAEAD, with our single-pass constructions offering new and efficient instantiations of a variety of primitives. In this section we describe several of these primitives and their applications.

Concealment schemes. In [14], Dodis et al. introduce the notion of *concealment* schemes, a primitive which has a number of applications in the context of authenticated encryption. A concealment scheme is defined to be a pair of algorithms $\mathcal{C} = (\mathsf{conceal}, \mathsf{open})$. The randomized concealment algorithm $\mathsf{conceal}$ takes as input a message M, and outputs a pair $(h, b) \leftarrow_{\$} \mathsf{conceal}(M)$. The deterministic opening algorithm open takes as input a hider and binder pair, and outputs either the underlying message M or $\perp$, depending on whether (h, b) is in the range of $\mathsf{conceal}(M)$.

One-pass concealment schemes and applications. In [14], the authors show how to construct concealment schemes using a one-time secure symmetric encryption scheme $\mathsf{E} = (\mathsf{kg}, \mathsf{enc}, \mathsf{dec})$ for which the keys $K \leftarrow_{\$} \mathsf{kg}$ are short, and a collision resistant hash function CRHF. To $\mathsf{conceal}$ a message M, the scheme chooses a random key K, sets $h = \mathsf{enc}(K, M)$, and $b = K || \mathsf{CRHF}(h)$. However this requires two-passes for both the encryption and the hashing step. In contrast, given a otROR, r-BIND and s-BIND secure encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EC}, \mathsf{DO}, \mathsf{EVer})$ with strong correctness, one can construct a secure concealment scheme which achieves the desired security goals with just a single pass over the data. We assume the header $H = \varepsilon$ in the subsequent discussion, and so omit this input from algorithms. The concealment scheme $\mathcal{C}[\mathsf{EC}] = (\mathsf{conceal}.[\mathsf{EC}], \mathsf{open}.[\mathsf{EC}])$ is defined as follows. To conceal a message M, $\mathsf{conceal}.[\mathsf{EC}]$ generates a key $K_{\mathsf{EC}} \leftarrow_{\$} \mathsf{EKg}$, computes $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, M)$. The scheme outputs (h, b) where $h = C_{\mathsf{EC}}$ and $b = K_{\mathsf{EC}} || B_{\mathsf{EC}}$.

With this encryptment-based concealment we can give one-pass instantiations of domain extension for AE and remotely-keyed AE. Domain extension for AE takes an AE for which the message space consists of only 'short' messages and allows it to encrypt much longer messages. The modified encryption algorithm on input (K, M) is defined to first compute the concealment of M via $(h, b) \leftarrow_{\$} \mathsf{conceal}(M)$, and then output ciphertext $(h, \mathsf{AE.enc}(K, b))$. In remotely-keyed AE, a secure but computationally-limited device holding a long-term AE key offloads most of the computational work of encrypting and decrypting to an untrusted but more powerful device. In [14] the authors show a "canonical" RKAE from any concealment scheme, so we can get RKAE from encryptment via $\mathcal{C}[\mathsf{EC}]$. An interesting question for future work is defining and constructing remotely-keyed ccAEAD schemes, and if applying their RKAE transform to $\mathcal{C}[\mathsf{EC}]$ gives a remotely-keyed ccAEAD scheme.

Verifiable outsourced storage from encryptment. In the majority of this work we have viewed encryptment as a combined encryption and commitment scheme, but an alternate way to view it is as a kind of three-party secret sharing scheme for the header and message pair. If EC runs EKg internally instead of accepting a key as input, the (now randomized) EC algorithm will output $(C_{\mathsf{EC}}, B_{\mathsf{EC}}, K_{\mathsf{EC}})$ on input H, M. None of these three values in isolation reveals any information about the message M. We will denote as $\mathsf{SShare}(H, M)$ the randomized algorithm which on input (H, M) first computes $K_{\mathsf{EC}} \leftarrow\!\!\$\ \mathsf{EKg}$ then outputs $(\mathsf{EC}(K_{\mathsf{EC}}, H, M), K_{\mathsf{EC}})$.

This secret-sharing viewpoint of encryptment gives an optimally-efficient and verifiable way to store large files on untrusted cloud storage. There are three parties: an untrusted storage provider, a public ledger providing integrity but not confidentiality (such as a blockchain), and the user. When the user wants to store file M with header H, it runs $\mathsf{SShare}(H, M)$. It stores C_{EC} with the cloud provider, posts B_{EC} on the public ledger, and retains K_{EC} in its own trusted storage. Since K_{EC} is small the user can store it in hardened local storage like a TPM. Likewise, since B_{EC} is small the cost of storing it on the public ledger is minimized. In addition to minimizing local storage overhead for the user, storing B_{EC} in a public ledger gives the user the ability to prove misbehavior on the part of the cloud provider, such as deleting part of the file or trying to modify it.

Acknowledgments

The authors thank Jon Millican for his help on understanding Facebook's message franking systems. Dodis is partially supported by gifts from VMware Labs and Google, and NSF grants 1619158, 1319051, 1314568. Grubbs is supported by an NSF Graduate Research Fellowship (DGE-1650441). A portion of this work was completed while Grubbs visited Royal Holloway University, and he thanks Kenny Paterson for generously hosting him. Ristenpart is supported in part by NSF grants 1704527 and 1514163, as well as a gift from Microsoft. Woodage is supported by the EPSRC and the UK government as part of the Centre for Doctoral Training in Cyber Security at Royal Holloway, University of London (EP/K035584/1).

References

[1] Michel Abdalla, Mihir Bellare, and Gregory Neven. Robust encryption. In *TCC*, 2010.

[2] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. Blake2: simpler, smaller, fast as MD5. In *ACNS*, 2013.

[3] Mihir Bellare, Joseph Jaeger, and Julia Len. Better than advertised: Improved collision-resistance guarantees for MD-based hash functions. In *ACM CCS*, 2017.

[4] Mihir Bellare and Tadayoshi Kohno. A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In *EUROCRYPT*, 2003.

[5] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of generic composition. In *ASIACRYPT*, 2000.

[6] Daniel J. Bernstein. ChaCha, a variant of Salsa20. `https://cr.yp.to/chacha/chacha-20080128.pdf`.

[7] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak sponge function family main document. *Submission to NIST SHA3*, 2009.

[8] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: single-pass authenticated encryption. In *SAC*, 2011.

[9] Alex Biryukov and Dmitry Khovratovich. Related-key cryptanalysis of the full AES-192 and AES-256. In *ASIACRYPT*, 2009.

[10] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolić. Distinguisher and related-key attack on the full AES-256. In *CRYPTO*. 2009.

[11] John Black, Martin Cochran, and Thomas Shrimpton. On the impossibility of highly-efficient blockcipher-based hash functions. In *EUROCRYPT*, 2005.

[12] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from PGV. In *CRYPTO*, 2002.

[13] Advanced Micro Devices. The ZEN microarchitecture, 2016. `https://www.amd.com/en/technologies/zen-core`.

[14] Yevgeniy Dodis and Jee Hea An. Concealment and its applications to authenticated encryption. In *EUROCRYPT*, 2003.

[15] Facebook. Facebook Messenger app. `https://www.messenger.com/`, 2016.

[16] Facebook. Messenger Secret Conversations technical whitepaper, 2016.

[17] Pooya Farshim, Benoît Libert, Kenneth G Paterson, and Elizabeth A Quaglia. Robust encryption, revisited. In *PKC*. 2013.

[18] Pooya Farshim, Claudio Orlandi, and Razvan Rosie. Security of symmetric primitives under incorrect usage of keys. *FSE*, 2017.

[19] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. Message franking via committing authenticated encryption. In *CRYPTO*, 2017.

[20] Sean Gulley, Vinodh Gopal, Kirk Yap, Wajdi Feghali, and Jim Guilford. Intel SHA extensions, 2013. `https://software.intel.com/en-us/articles/intel-sha-extensions`.

[21] Jian Guo, San Ling, Christian Rechberger, and Huaxiong Wang. Advanced meet-in-the-middle preimage attacks: First results on full Tiger, and improved results on MD4 and SHA-2. In *ASIACRYPT*, 2010.

[22] Shai Halevi and Hugo Krawczyk. Strengthening digital signatures via randomized hashing. In *CRYPTO*, 2006.

[23] Seokhie Hong, Jongsung Kim, Sangjin Lee, and Bart Preneel. Related-key rectangle attacks on reduced versions of SHACAL-1 and AES-192. In *FSE*, 2005.

[24] Philipp Jovanovic, Atul Luykx, and Bart Mennink. Beyond $2^{c/2}$ security in sponge-based authenticated encryption modes. In *ASIACRYPT*, 2014.

[25] Charanjit S Jutla. Encryption modes with almost free message integrity. In *EUROCRYPT*, 2001.

[26] Dmitry Khovratovich, Christian Rechberger, and Alexandra Savelieva. Bicliques for preimages: attacks on Skein-512 and the SHA-2 family. In *FSE*, 2012.

[27] Jongsung Kim, Guil Kim, Seokhie Hong, Sangjin Lee, and Dowon Hong. The related-key rectangle attack–application to SHACAL-1. In *ACISP*, 2004.

[28] Jongsung Kim, Guil Kim, Sangjin Lee, Jongin Lim, and Junghwan Song. Related-key attacks on reduced rounds of SHACAL-2. In *INDOCRYPT*, 2004.

[29] Ted Krovetz and Phillip Rogaway. The software performance of authenticated-encryption modes. In *FSE*, 2011.

[30] Mario Lamberger and Florian Mendel. Higher-order differential attack on reduced SHA-256. *IACR ePrint, Report 2011/037*, 2011.

[31] Moses Liskov, Ronald L Rivest, and David Wagner. Tweakable block ciphers. In *CRYPTO*. Springer, 2002.

[32] Jiqiang Lu, Jongsung Kim, Nathan Keller, and Orr Dunkelman. Related-key rectangle attack on 42-round SHACAL-2. In *ICIS*, 2006.

[33] David McGrew and John Viega. The Galois/counter mode of operation (GCM). *NIST Modes of Operation*, 2004.

[34] David McGrew and John Viega. The security and performance of the galois/Counter mode of operation. In *INDOCRYPT*, 2004.

[35] Jon Millican. Personal communication.

[36] Jon Millican. Challenges of E2E Encryption in Facebook Messenger. RWC, 2017.

[37] Payman Mohassel. A closer look at anonymity and robustness in encryption schemes. In *ASIACRYPT*, 2010.

[38] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash functions based on block ciphers: A synthetic approach. In *CRYPTO*, 1993.

[39] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In *ASIACRYPT*, 2004.

[40] Phillip Rogaway, Mihir Bellare, and John Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM TISSEC*, 2003.

[41] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In *EUROCRYPT*, 2006.

[42] Phillip Rogaway and John Steinberger. Constructing cryptographic hash functions from fixed-key blockciphers. In *CRYPTO*, 2008.

[43] Phillip Rogaway and John Steinberger. Security/efficiency tradeoffs for permutation-based hashing. In *EUROCRYPT*, 2008.

[44] Somitra Kumar Sanadhya and Palash Sarkar. New collision attacks against up to 24-step SHA-2. In *INDOCRYPT*, 2008.

[45] Thomas Shrimpton and Martijn Stam. Building a collision-resistant compression function from non-compressing primitives. In *ICALP*, 2008.

[46] Open Whisper Systems. Signal. `https://signal.org/`, 2016.

[47] Wouter van der Linde. Parallel SHA-256 in NEON for use in hash-based signatures, 2016. BSc thesis, Radboud University.

[48] Mark N Wegman and J Lawrence Carter. New hash functions and their use in authentication and set equality. *JCSS*, 1981.

[49] Whatsapp. Whatsapp. `https://www.whatsapp.com/`, 2016.

A Relationship Between Receiver Binding Notions

In the following theorem, we prove that sr-BIND security implies r-BIND security for ccAEAD schemes; our results readily extend to encryptment schemes also. We then show that the converse does not hold by constructing a scheme which has receiver binding but for which strong receiver binding can be trivially broken.

Theorem 8 *Let* $\mathsf{CE} = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ver})$ *be a ccAEAD scheme. Then for any attacker* $\mathcal{A}$ *in game* r-BIND *against* CE*, there exists an attacker* $\mathcal{B}$ *in game* sr-BIND *against* CE *such that*

$$\mathbf{Adv}^{\text{r-bind}}_{\mathsf{CE}}(\mathcal{A}) \le \mathbf{Adv}^{\text{sr-bind}}_{\mathsf{CE}}(\mathcal{B}) ,$$

and moreover, $\mathcal{B}$ *runs in the same time* $\mathcal{A}$*. On the other hand, there exists a CE scheme* $\mathsf{CE}' = (\mathsf{Kg}', \mathsf{Enc}', \mathsf{Dec}', \mathsf{Ver}')$ *which is* r-BIND *secure, but for which there exists an efficient attacker* $\mathcal{C}$ *such that*

$$\mathbf{Adv}^{\text{sr-bind}}_{\mathsf{CE}}(\mathcal{C}) = 1 .$$

Proof: To prove the first claim, let $\mathcal{A}$ be an attacker in game r-BIND against CE. We then define $\mathcal{B}$ to be the attacker in game sr-BIND against CE who simply runs $\mathcal{A}$; eventually $\mathcal{A}$ halts and outputs $((H, M, K_f), (H', M', K'_f), C_B) \leftarrow\!\!\$\; \mathcal{A}$, and $\mathcal{B}$ returns the same tuple to his challenger. Then

$$\begin{aligned}
&\mathbf{Adv}^{\text{r-bind}}_{\mathsf{CE}}(\mathcal{A}) = \Pr\left[\, \text{r-BIND}^{\mathcal{A}}_{\mathsf{CE}} \Rightarrow \mathsf{true} \,\right] \\
&= \Pr\left[\, \mathsf{Ver}(H, M, K_f, C_B) = \mathsf{Ver}(H', M', K'_f, C_B) = 1 \wedge (H, M) \neq (H', M') \,\right] \\
&\le \Pr\left[\, \mathsf{Ver}(H, M, K_f, C_B) = \mathsf{Ver}(H', M', K'_f, C_B) = 1 \wedge (H, M, K_f) \neq (H', M', K'_f) \,\right] \\
&= \Pr\left[\, \text{sr-BIND}^{\mathcal{B}}_{\mathsf{CE}} \Rightarrow \mathsf{true} \,\right] \\
&= \mathbf{Adv}^{\text{sr-bind}}_{\mathsf{CE}}(\mathcal{B}) ,
\end{aligned}$$

where all probabilities are over the coins of $\mathcal{A}$ (recall that Ver is deterministic), proving the claim.

To prove that the converse does not hold, we now define a CE scheme which is r-BIND secure, but not sr-BIND secure. Take any CE scheme $\mathsf{CE} = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ver})$ which is r-BIND secure and for which all valid openings K_f are bit-strings of some fixed length n (for example, one may take the Committing Encrypt-and-PRF scheme from Section 7 of [19]). We define a modified scheme $\mathsf{CE}' = (\mathsf{Kg}', \mathsf{Enc}', \mathsf{Dec}', \mathsf{Ver}')$ as follows. We let $\mathsf{Kg}', \mathsf{Enc}', \mathsf{Dec}'$ be identical to $\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec}$, and define Ver' to be the algorithm which on input (H, M, K_f, C_B) computes $K'_f = [K_f]^n_1$ (where $[x]^n_1$ denotes truncating the binary string x to the n least significant bits), and returns the output of $\mathsf{Ver}(H, M, K'_f, C_B)$.

It is straightforward to see that the modified scheme CE' is r-BIND secure. Indeed for any attacker $\mathcal{D}$ in game r-BIND against CE', we may define an attacker $\mathcal{E}$ in game r-BIND against CE who simply runs $\mathcal{D}$ and outputs $(H, M, [K_f]_1^n), (H', M', [K'_f]_1^n), C_B)$ where $((H, M, K_f), (H', M', K'_f), C_B)$ is the tuple output by $\mathcal{D}$. It is easy to see that any winning output for $\mathcal{D}$ implies a winning output for $\mathcal{E}$ also and so $\mathbf{Adv}^{\text{r-bind}}_{\mathsf{CE}'}(\mathcal{D}) \leq \mathbf{Adv}^{\text{r-bind}}_{\mathsf{CE}}(\mathcal{E})$.

However, an adversary $\mathcal{C}$ may win game sr-BIND with probability 1 by taking any tuple (H, M, K_f, C_B) for which $1 \leftarrow \mathsf{Ver}(H, M, K_f, C_B)$, and submitting $((H, M, 0||K_f), (H, M, 1||K_f), C_B)$ to his challenger. Now $(H, M, 0||K_f) \neq (H, M, 1||K_f)$, but since $\mathsf{Ver}'(H, M, 0||K_f) = \mathsf{Ver}(H, M, 1||K_f) = 1$, it holds that $\mathbf{Adv}^{\text{sr-bind}}_{\mathsf{CE}'}(\mathcal{C}) = 1$, implying the result.

B Proofs from Section 4

Proof of Lemma 1.

We argue by a series of game hops, shown in Figure 15. We first define game G_0 which is identical to game SCU against EC, except we change the specification of oracle dec to perform a number of additional checks, and set flags depending on the results. These additional steps do not affect the outcome of the game, and so it follows that

$$\Pr\left[\,\mathrm{SCU}^{\mathcal{A}}_{\mathsf{EC}} \Rightarrow 1\,\right]\ .$$

Next we define game G_1, which is identical to game G_0 except now if $\mathcal{A}$ makes a query to dec which decrypts correctly to $M' \neq \perp$, but for which verification fails, then the game sets $M' = \perp$ and so the win flag will not be set. These games run identically until the flag bad_1 is set, and so the Fundamental Lemma of Game Playing implies that

$$|\Pr[\,\mathrm{G}_0 \Rightarrow 1\,] - \Pr[\,\mathrm{G}_1 \Rightarrow 1\,] \leq \Pr[\,\mathsf{bad}_1 \leftarrow \mathsf{true}\text{ in }\mathrm{G}_0\,]\ .$$

We bound the probability of this event occurring with a reduction to the s-BIND security of EC. Let $\mathcal{B}$ be an adversary in game s-BIND against EC. $\mathcal{B}$ runs $\mathcal{A}$ as a subroutine as follows. He generates a key $K^*_{\mathsf{EC}} \leftarrow\!\!\$\ \mathsf{EKg}$, and simulates $\mathcal{A}$'s enc query by computing the required encryptment $(C^*_{\mathsf{EC}}, B^*_{\mathsf{EC}})$ and returning this, along with K^*_{EC}, to $\mathcal{A}$. He simulates the dec oracle by applying $\mathsf{DO}(K^*_{\mathsf{EC}}, \cdot, \cdot, B^*_{\mathsf{EC}})$ to the queried header / ciphertext pairs (H', C'_{EC}), and returning the result to $\mathcal{A}$. For queries which decrypt to $M' \neq \perp$, he computes $b' \leftarrow \mathsf{EVer}(H', M', K^*_{\mathsf{EC}}, B^*_{\mathsf{EC}})$. If he ever finds a tuple $(H', C'_{\mathsf{EC}}, B^*_{\mathsf{EC}})$ which decrypts correctly under K^*_{EC} but which does not verify correctly, $\mathcal{B}$ halts and outputs $(K^*_{\mathsf{EC}}, H', C'_{\mathsf{EC}}, B^*_{\mathsf{EC}})$ to his challenger. Notice that such a tuple constitutes a winning query for $\mathcal{B}$, and that the flag bad_1 gets set if and only if such a tuple is found. It follows that

$$\begin{aligned}\Pr[\,\mathsf{bad}_1 \leftarrow \mathsf{true}\text{ in }\mathrm{G}_0\,] &\leq \Pr\left[\,\text{s-BIND}^{\mathcal{B}}_{\mathsf{EC}} \Rightarrow 1\,\right]\\ &= \mathbf{Adv}^{\text{s-bind}}_{\mathsf{EC}}(\mathcal{B})\ .\end{aligned}$$

Next we define game G_2, which is identical to game G_1 except now if $\mathcal{A}$ submits a query (H', C'_{EC}) which decrypts correctly to some message $M' \neq \perp$ and for which verification succeeds, then if (H', M') does not equal the challenge header / message pair (H^*, M^*) then dec sets $M' = \perp$ and win does not get set to true. These games run identically unless flag bad_2 gets set, and so the Fundamental Lemma of Game Playing implies that

$$\Pr[\,\mathrm{G}_1 \Rightarrow 1\,] - \Pr[\,\mathrm{G}_2 \Rightarrow 1\,] \leq \Pr[\,\mathsf{bad}_2 \leftarrow \mathsf{true}\text{ in }\mathrm{G}_1\,]\ .$$

We bound the probability that this event occurs with a reduction to the r-BIND security of EC. Let $\mathcal{C}$ be an adversary in game r-BIND against EC. $\mathcal{C}$ runs $\mathcal{A}$ as a subroutine by choosing a key

$K^*_{\mathsf{EC}} \leftarrow\!\!\$\ \mathsf{EKg}$ and simulating oracles enc and dec as described previously, (except he now rejects any dec queries which fail verification as per G_1). Let $(C^*_{\mathsf{EC}}, B^*_{\mathsf{EC}})$ denote the encryptment generated by the enc query on input (H^*, M^*) with key K^*_{EC}, and notice that the correctness of EC implies that $1 \leftarrow \mathsf{EVer}(H^*, M^*, K^*_{\mathsf{EC}}, B^*_{\mathsf{EC}})$. If at any point the attacker submits a query (H', C'_{EC}) to dec such that $M' \leftarrow \mathsf{DO}(K^*_{\mathsf{EC}}, H', C'_{\mathsf{EC}}, B^*_{\mathsf{EC}})$ and $1 \leftarrow \mathsf{EVer}(H', M', K^*_{\mathsf{EC}}, B^*_{\mathsf{EC}})$ but for which (H', M') does not equal the header / message pair (H^*, M^*) underlying the challenge encryptment, then $\mathcal{C}$ halts and outputs $((H^*, M^*, K^*_{\mathsf{EC}}), (H', M', K^*_{\mathsf{EC}}), B^*_{\mathsf{EC}})$. Such a tuple constitutes a win for $\mathcal{C}$ in his game, and notice that the flag bad_2 is set if and only if such a tuple is found. It follows that

$$\begin{aligned}\Pr[\,\mathsf{bad}_2 \leftarrow \mathsf{true} \text{ in } G_1\,] &\leq \Pr\left[\,\text{r-BIND}^{\mathcal{C}}_{\mathsf{EC}} \Rightarrow 1\,\right]\\ &= \mathbf{Adv}^{\text{r-bind}}_{\mathsf{EC}}(\mathcal{C})\ .\end{aligned}$$

Finally we define game G_3, which is identical to G_2 except that now if the attacker submits a query (H', C'_{EC}) to oracle dec which decrypts correctly to message M' under challenge key K^*_{EC}, verifies correctly, and for which $(H', M') = (H^*, M^*)$, then if it is the case that $C'_{\mathsf{EC}} \neq C^*_{\mathsf{EC}}$ where C^*_{EC} is the ciphertext derived in the challenge query to enc, the game sets $M' \leftarrow \perp$ and the win flag remains false. Notice that this game is impossible to win, since the only queries which will not be rejected by the added checks are those for which $(H', C'_{\mathsf{EC}}) = (H^*, C^*_{\mathsf{EC}})$, but then these themselves are rejected by the first line of pseudocode in dec, and so will never result in win being set. It follows that

$$\Pr[\,G_3 \Rightarrow 1\,] = 0\ .$$

Notice that these games run identically unless the flag bad_3 is set, and so it holds that

$$|\Pr[\,G_2 \Rightarrow 1\,] - \Pr[\,G_3 \Rightarrow 1\,]| \leq \Pr[\,\mathsf{bad}_3 \leftarrow \mathsf{true} \text{ in } G_2\,]\ .$$

We claim that $\Pr[\,\mathsf{bad}_3 \leftarrow \mathsf{true} \text{ in } G_2\,] = 0$. To see this, notice that bad_3 being set implies the existence of a tuple $(K^*_{\mathsf{EC}}, H^*, C'_{\mathsf{EC}}, B^*_{\mathsf{EC}})$ such that $M^* \leftarrow \mathsf{DO}(K^*_{\mathsf{EC}}, H^*, C'_{\mathsf{EC}}, B^*_{\mathsf{EC}})$ but for which $C'_{\mathsf{EC}} \neq C^*_{\mathsf{EC}}$ where we know that $(C^*_{\mathsf{EC}}, B^*_{\mathsf{EC}}) = \mathsf{EC}(K^*_{\mathsf{EC}}, H^*, M^*)$ from the challenge query to enc. Moreover, notice that it must be the case that $(K^*_{\mathsf{EC}}, H^*, C'_{\mathsf{EC}}, B^*_{\mathsf{EC}}) \in \mathcal{K}_{\mathsf{EC}} \times \mathcal{H}_{\mathsf{EC}} \times \mathcal{C}_{\mathsf{EC}} \times \mathcal{T}_{\mathsf{EC}}$, otherwise by definition DO would return $\perp$. However, since EC is deterministic, this then violates the assumed strong correctness of the scheme, and proving the claim.

Putting this all together, we find that

$$\mathbf{Adv}^{\text{scu}}_{\mathsf{EC}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{s-bind}}_{\mathsf{EC}}(\mathcal{B}) + \mathbf{Adv}^{\text{r-bind}}_{\mathsf{EC}}(\mathcal{C})\ .$$

▮

C Encryptment via Generic Composition

Given an encryption scheme $\mathsf{E} = (\mathsf{kg}, \mathsf{enc}, \mathsf{dec})$, and a commitment scheme with verification $\mathsf{CS} = (\mathsf{Com}, \mathsf{VerC})$, it is straightforward to construct an encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EC}, \mathsf{DO}, \mathsf{EVer})$ by composing the two. The resulting scheme is essentially an adaptation of the CtE2 ccAEAD scheme from [19] rephrased in the encryptment syntax, except requires weaker one-time security properties of the underlying primitives.

Key generation EKg outputs keys of the form (K, R_c, R_e) where $K \leftarrow\!\!\$\ \mathsf{Kg}$ is a key for the underlying encryption scheme, and R_c, R_e are drawn randomly from the coin spaces of the commitment and encryption scheme respectively. On input $((K, R_c, R_e), H, M)$, encryptment algorithm EC first computes a commitment / opening pair for $H||M$ via $(c, d) \leftarrow \mathsf{Com}(H||M; R_c)$, and sets $B_{\mathsf{EC}} = c$. It then encrypts the opening along with the message to give $C_{\mathsf{EC}} \leftarrow \mathsf{enc}(K, d||M; R_e)$, and outputs encryptment $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$. Decryption DO works by decrypting C_{EC} under key K to recover message

```
$\mathrm{G}_0, \boxed{\mathrm{G}_1}$:
$K \leftarrow\!\!\$\ \mathsf{EKg}$
$\mathsf{win} \leftarrow \mathsf{false}$
$\mathsf{query\text{-}made} \leftarrow \mathsf{false}$
$\varepsilon \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{enc}(\cdot,\cdot),\mathsf{dec}(\cdot,\cdot)}$
Return $\mathsf{win}$

$\mathsf{enc}(H, M)$
If $\mathsf{query\text{-}made} = \mathsf{true}$ then Return $\perp$
$\mathsf{query\text{-}made} \leftarrow \mathsf{true}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$K^*_{\mathsf{EC}} \leftarrow K_{\mathsf{EC}}$ ; $C^*_{\mathsf{EC}} \leftarrow C_{\mathsf{EC}}$
$B^*_{\mathsf{EC}} \leftarrow B_{\mathsf{EC}}$ ; $H^* \leftarrow H$
Return $((C_{\mathsf{EC}}, B_{\mathsf{EC}}), K_{\mathsf{EC}})$

$\mathsf{dec}(H', C'_{\mathsf{EC}})$
If $\mathsf{query\text{-}made} = \mathsf{false}$ then Return $\perp$
If $(H', C'_{\mathsf{EC}}) = (H^*, C^*_{\mathsf{EC}})$
    Return $\perp$
$M' \leftarrow \mathsf{DO}(K^*_{\mathsf{EC}}, H', C'_{\mathsf{EC}}, B^*_{\mathsf{EC}})$
If $M' = \perp$ Return $\perp$
$b' \leftarrow \mathsf{EVer}(H', M', K^*_{\mathsf{EC}}, B^*_{\mathsf{EC}})$
If $b' = 0$ then
    $\mathsf{bad}_1 \leftarrow \mathsf{true}$ ; $\boxed{M' \leftarrow \perp}$
If $(H', M') \neq (H^*, M^*)$ then
    $\mathsf{bad}_2 \leftarrow \mathsf{true}$
If $(C'_{\mathsf{EC}} \neq C^*_{\mathsf{EC}})$ then
    $\mathsf{bad}_3 \leftarrow \mathsf{true}$
If $M' \neq \perp$ then $\mathsf{win} \leftarrow \mathsf{true}$
Return $M'$
```

```
$\mathrm{G}_2, \boxed{\mathrm{G}_3}$:
$K \leftarrow\!\!\$\ \mathsf{EKg}$
$\mathsf{win} \leftarrow \mathsf{false}$
$\mathsf{query\text{-}made} \leftarrow \mathsf{false}$
$\varepsilon \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{enc}(\cdot,\cdot),\mathsf{dec}(\cdot,\cdot)}$
Return $\mathsf{win}$

$\mathsf{enc}(H, M)$
If $\mathsf{query\text{-}made} = \mathsf{true}$ then Return $\perp$
$\mathsf{query\text{-}made} \leftarrow \mathsf{true}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$K^*_{\mathsf{EC}} \leftarrow K_{\mathsf{EC}}$ ; $C^*_{\mathsf{EC}} \leftarrow C_{\mathsf{EC}}$
$B^*_{\mathsf{EC}} \leftarrow B_{\mathsf{EC}}$ ; $H^* \leftarrow H$
Return $((C_{\mathsf{EC}}, B_{\mathsf{EC}}), K_{\mathsf{EC}})$

$\mathsf{dec}(H', C'_{\mathsf{EC}})$
If $\mathsf{query\text{-}made} = \mathsf{false}$ then Return $\perp$
If $(H', C'_{\mathsf{EC}}) = (H^*, C^*_{\mathsf{EC}})$
    Return $\perp$
$M' \leftarrow \mathsf{DO}(K^*_{\mathsf{EC}}, H', C'_{\mathsf{EC}}, B^*_{\mathsf{EC}})$
If $M' = \perp$ Return $\perp$
$b' \leftarrow \mathsf{EVer}(H', M', K^*_{\mathsf{EC}}, B^*_{\mathsf{EC}})$
If $b' = 0$ then
    $\mathsf{bad}_1 \leftarrow \mathsf{true}$ ; $M' \leftarrow \perp$
If $(H', M') \neq (H^*, M^*)$ then
    $\mathsf{bad}_2 \leftarrow \mathsf{true}$ ; $M' \leftarrow \perp$
If $(C'_{\mathsf{EC}} \neq C^*_{\mathsf{EC}})$ then
    $\mathsf{bad}_3 \leftarrow \mathsf{true}$ ; $\boxed{M' \leftarrow \perp}$
If $M' \neq \perp$ then $\mathsf{win} \leftarrow \mathsf{true}$
Return $M'$
```

Figure 15: Games for proof of Lemma 1.

```
$\underline{\mathrm{FROB}^{\mathcal{A}}_{\mathsf{CE}}}$
$((K,H),(K',H'),(C,C_B)) \leftarrow\!\!\$\ \mathcal{A}$
If $K = K'$ return 0
$M_1 \leftarrow \mathsf{Dec}(K,H,(C,C_B))$
$M_2 \leftarrow \mathsf{Dec}(K',H',(C,C_B))$
Return $(M_1 \neq\perp \wedge M_2 \neq\perp)$
```

Figure 16: The full robustness (FROB) security game for a ccAEAD scheme $\mathsf{CE} = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ver})$.

M and opening d, and verifying the commitment via $b \leftarrow \mathsf{VerC}(c, d, H||M)$, returning $\perp$ if verification fails. Provided verification succeeds, then DO uses the key (K, R_c, R_e) to re-run EC for the revealed message, to verify that the resulting encryptment matches that which was decrypted. The message is returned only if this latter check returns true. On input $(H, M, d, B_{\mathsf{EC}})$, encryptment verification EVer simply computes $\mathsf{VerC}(B_{\mathsf{EC}}, d, H||M)$ and returns the result.

It's easy to see that if E and CS satisfy one-time notions of real-or-random security, then the encryptment scheme satisfies our notion of otROR security. If the commitment scheme is binding, this in turn implies the encryptment scheme is r-BIND secure, since any winning tuple in the r-BIND game also breaks the binding of the commitment scheme. s-BIND security and strong correctness follow from the fact that the commitment is verified, and the encryptment recomputed, during decryption. Together these properties imply the scheme is SCU secure by Lemma 1.

D Stronger Receiver Binding and Robust Encryption

An encryption scheme is said to be *robust* if it is infeasible to find two distinct keys which decrypt the same ciphertext to a valid message. Robust encryption was first formalized in the public-key setting by Abdalla et al. in [1] and later extended in [17, 37]. Farshim et al. provide the first treatment of robust authenticated encryption in [18].

Full robustness. In Figure 16 we adapt the definition of *full robustness* (FROB) for symmetric encryption from Farshim et al. [18] to the ccAEAD setting. The original FROB definition of [18] can be recovered from the one shown in Figure 16 by replacing the ciphertext / binding tag pair (C, C_B) with a single ciphertext C, and removing all references to headers. The attacker $\mathcal{A}$ is challenged to output a tuple $((K,H),(K',H'),(C,C_B))$ such that $K \neq K'$, but the ciphertext / binding tag pair (C, C_B) decrypts correctly under both keys and the corresponding headers. The FROB advantage term for an attacker $\mathcal{A}$ against a ccAEAD scheme CE is defined

$$\mathbf{Adv}^{\mathrm{frob}}_{\mathsf{CE}}(\mathcal{A}) = \Pr\left[\, \mathrm{FROB}^{\mathcal{A}}_{\mathsf{CE}} \Rightarrow 1 \,\right] \ ,$$

where the probability is over the coins of Enc and $\mathcal{A}$.

Separations between FROB **and** sr-BIND **security.** Intuitively, both robust encryption and ccAEAD target some notion of binding security. Thus, a natural question is whether an sr-BIND-secure ccAEAD scheme is also FROB secure, and vice versa. It turns out that the two notions are orthogonal and neither implies the other in generality.

The intuition for this is that breaking robustness requires finding distinct keys which both *decrypt* the same ciphertext correctly, whereas breaking sr-BIND security requires finding distinct tuples $(H, M, K_f) \neq (H', M', K'_f)$ which both *verify* the same binding tag correctly. Since there is no requirement that decrypting with distinct keys will result in distinct openings being recovered during decryption, breaking robustness does not necessarily translate into a win for an attacker in game sr-BIND. However, we note that an attacker can only break FROB security without breaking sr-BIND security if the winning tuple uses the same headers for both decryptions, and both keys decrypt the ciphertext to the *same* underlying message; such an attack would seem to be of little concern in verifiable abuse reporting (and perhaps other settings as well).

Likewise, since any verification performed during decryption uses the opening output by Dec as opposed to one specified by an attacker, one may modify the verification algorithm of a robust ccAEAD scheme in such a way that it becomes easy to produce two distinct openings which verify the same binding tag (thereby breaking the sr-BIND security of the scheme), but in such a way that these 'bad' openings will never be recovered during normal decryption, leaving the robustness of the scheme unaffected.

The case is different if the ccAEAD scheme is such that it always outputs its key as the opening (as is the case for many single-opening ccAEAD schemes). If such a scheme is sender binding — so ciphertexts which decrypt correctly must also verify correctly — then sr-BIND security implies FROB security. It is straightforward to verify that FROB security does not imply sr-BIND security for such schemes in general (for a separating example, modify an FROB-secure ccAEAD scheme which outputs its encryption key as the opening such that it becomes easy to find distinct headers which verify with the same key / message / binding tag tuple, but decryption is left unchanged). We formalize this intuition and provide separating examples in the following theorem.

Theorem 9 ***(1)*** *There exists a ccAEAD scheme* $\mathsf{CE}_1 = (\mathsf{Kg}_1, \mathsf{Enc}_1, \mathsf{Dec}_1, \mathsf{Ver}_1)$ *which is* FROB-*secure, but for which there exists an efficient attacker* $\mathcal{B}$ *such that*

$$\mathbf{Adv}^{\text{sr-bind}}_{\mathsf{CE}_1}(\mathcal{B}) = 1 \; .$$

(2) *There exists a ccAEAD scheme* $\mathsf{CE}_2 = (\mathsf{Kg}_2, \mathsf{Enc}_2, \mathsf{Dec}_2, \mathsf{Ver}_2)$ *which is* sr-BIND-*secure, but for which there exists an efficient attacker* $\mathcal{C}$ *such that*

$$\mathbf{Adv}^{\text{frob}}_{\mathsf{CE}_2}(\mathcal{C}) = 1 \; .$$

(3) *Let* $\mathsf{CE} = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ver})$ *be a ccAEAD scheme which outputs its encryption key as the opening. Then for any attacker* $\mathcal{A}$ *in game* FROB *against* CE, *there exists adversaries* $\mathcal{D}, \mathcal{E}$ *such that*

$$\mathbf{Adv}^{\text{frob}}_{\mathsf{CE}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{s-bind}}_{\mathsf{CE}}(\mathcal{D}) + \mathbf{Adv}^{\text{sr-bind}}_{\mathsf{CE}}(\mathcal{E}) \; .$$

and moreover $\mathcal{D}, \mathcal{E}$ *run in the same time as* $\mathcal{A}$.

Proof: We start with **(1)**. Let $\mathsf{CE} = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ver})$ be an FROB-secure ccAEAD scheme, for which all valid openings K_f are of some fixed length of n-bits. We then construct a modified scheme $\mathsf{CE}_1 = (\mathsf{Kg}_1, \mathsf{Enc}_1, \mathsf{Dec}_1, \mathsf{Ver}_1)$ which is identical to CE except we define Ver_1 to be the algorithm which on input (H, M, K_f, C_B) sets $K'_f = \mathrm{Trunc}_n(K_f)$, and returns the result of $\mathsf{Ver}(H, M, K'_f, C_B)$. Notice that decryption is identical in both schemes; even if Dec' runs the modified Ver' as a subroutine, then since Dec' only outputs openings K_f such that $|K_f| = n$ the modifications to verification have no effect on decryption. Therefore any winning tuple in game FROB against CE' is a winning tuple for CE also, violating the assumed robustness of CE. However, an attacker $\mathcal{B}$ can win the sr-BIND security game with probability one by outputting $((H, M, K_f||0), (H, M, K_f||1), C_B)$ where (H, M, K_f, C_B) is any tuple such that $\mathsf{Ver}(H, M, K_f, C_B) = 1$, thereby proving the first claim.

To prove **(2)**, let $\mathsf{CE} = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ver})$ be an sr-BIND-secure ccAEAD scheme for which all keys output by Kg are bit-strings of length κ. We define a modified scheme $\mathsf{CE}_2 = (\mathsf{Kg}_2, \mathsf{Enc}_2, \mathsf{Dec}_2, \mathsf{Ver}_2)$ which is identical to CE except we define Dec_2 to be the algorithm which on input a tuple (K, H, C, C_B), computes $K' = \mathrm{Trunc}_\kappa(K)$, and returns the output of $\mathsf{Dec}(K', H, C, C_B)$. Since verification in CE_2 is unchanged, the modified scheme CE_2 inherits the sr-BIND security of CE. However, an attacker $\mathcal{C}$ can win the FROB game against CE_2 with probability one by choosing a key K and message M, computing $(C, C_B) \leftarrow\$ \mathsf{Enc}(H, K, M)$, and outputting the tuple $((K||0, H), (K||1, H), (C, C_B))$.

For **(3)**, consider an attacker $\mathcal{A}$ in game FROB against CE. Then any winning tuple $((K, H), (K', H'), (C, C_B))$ for $\mathcal{A}$ must be such that $K \neq K'$ and $\mathsf{Dec}(K, H, (C, C_B)) = (M, K_f) \neq \perp$ and $\mathsf{Dec}(K', H', (C, C_B)) = (M', K'_f) \neq \perp$. Since by definition Dec outputs its key as the opening, it follows that $K_f \neq K'_f$ also. If $\mathcal{A}$ has found a tuple $(K, H, (C, C_B))$ which decrypts successfully to (M, K) but for which $\mathsf{Ver}(H, M, K, C_B) = 0$, then he has broken the s-BIND security of CE. If this is not the case, then $\mathcal{A}$ has found $(K, H, M) \neq (K', H', M')$ which both verify correctly with binding tag C_B, breaking the sr-BIND security of CE. Reductions to the appropriate adversaries then imply the claim.

Robustness of ccAEAD schemes from Section 7.3. In Section 7.3 we present two transforms which allow the construction of a secure ccAEAD from a secure encryptment scheme.

It is straightforward to verify that the ccAEAD scheme $\mathsf{CE} = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ver})$ built from the first transform using a secure AEAD scheme is robust provided the underlying AEAD scheme is itself robust. In fact, provided the underlying encryptment scheme is sr-BIND and s-BIND secure, then the AEAD scheme need only satisfy a weaker property that it is infeasible for an attacker to find a tuple $((K, H), (K', H'), C_{\mathsf{AE}})$ such that $K \neq K'$ and $\mathsf{AEAD.dec}(K, H, C_{\mathsf{AE}}) = \mathsf{AEAD.dec}(K', H', C_{\mathsf{AE}})$. The intuition for this is that if this property is not satisfied then different encryptment keys will be recovered during decryption; therefore since the encryptment scheme is sender binding (and so ciphertexts which decrypt correctly must also verify correctly), this results in a winning tuple for an attacker in game sr-BIND against the underlying encryptment scheme.

The second transform which uses a PRF results in a robust ccAEAD scheme provided the PRF is collision-resistant and the underlying encryptment scheme is both sr-BIND and s-BIND-secure. We formalize this and provide a proof sketch in the following lemma.

Lemma 2 *Let* $\mathsf{CE} = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ver})$ *be the ccAEAD scheme obtained from the second transform using a PRF* PRF*, and an encryptment scheme* $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EC}, \mathsf{DO}, \mathsf{EVer})$*. Then for any attacker* $\mathcal{A}$ *in game* FROB *against* CE*, there exists attackers* $\mathcal{B}$*,* $\mathcal{C}$*, and* $\mathcal{D}$ *such that*

$$\mathbf{Adv}^{\text{frob}}_{\mathsf{CE}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{cr}}_{\mathsf{PRF}}(\mathcal{B}) + \mathbf{Adv}^{\text{s-bind}}_{\mathsf{EC}}(\mathcal{C}) + \mathbf{Adv}^{\text{sr-bind}}_{\mathsf{EC}}(\mathcal{D}) \ ,$$

and moreover, all adversaries run in the same time as $\mathcal{A}$*.*

Proof: (Sketch) Recall that for this transform, encryption on input (K, H, M) computes $K_{\mathsf{EC}} = \mathsf{PRF}(N, K \oplus \text{fpad})$ for random N and a fixed constant fpad, then $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$, finally outputting ciphertext / binding tag pair

$$(C, C_B) = ((N, C_{\mathsf{EC}}, \mathsf{PRF}(B_{\mathsf{EC}}, K)), B_{\mathsf{EC}}) \ .$$

Let $\mathcal{A}$ be an attacker in the FROB game against the resulting ccAEAD scheme, and suppose $\mathcal{A}$ outputs a winning tuple

$$((K, H), (K', H'), ((N, C_{\mathsf{EC}}, \mathsf{PRF}(B_{\mathsf{EC}}, K)), B_{\mathsf{EC}})) \ .$$

This implies that $K \neq K'$, and that the ciphertext decrypts correctly under both keys. Suppose further that K, K' are such that $K_{\mathsf{EC}} = \mathsf{PRF}(N, K \oplus \text{fpad}) = \mathsf{PRF}(N, K' \oplus \text{fpad})$; then this translates into a winning pair for an attacker $\mathcal{B}$ in the CR-game against PRF. On the other hand, suppose that $K_{\mathsf{EC}} \neq K'_{\mathsf{EC}}$ (where $K_{\mathsf{EC}} = \mathsf{PRF}(N, K \oplus \text{fpad})$ and $K'_{\mathsf{EC}} = \mathsf{PRF}(N, K' \oplus \text{fpad})$), but nonetheless $\mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}}) = M \neq \perp$ and $\mathsf{DO}(K'_{\mathsf{EC}}, H', C_{\mathsf{EC}}, B_{\mathsf{EC}}) = M' \neq \perp$. Suppose one (or both) of

```
$\underline{\text{OCB-Enc}_K^N(M)}$:
$\text{Partition } M \text{ into } M_1, \ldots, M_m$
$\text{For } i = 1 \text{ to } m-1 \text{ do } C_i \leftarrow \tilde{E}_K^{1,N,i,0}(M_i)$
$Pad \leftarrow \tilde{E}^{0,N,i,0}(len(M_m))$
$C_m \leftarrow M_m \oplus Pad$
$C \leftarrow C_1 \parallel \cdots \parallel C_m$
$\Sigma \leftarrow M_1 \oplus \cdots \oplus M_{m-1} \oplus C_m 0^* \oplus Pad$
$C_B \leftarrow \tilde{E}_K^{0,N,i,1}(\Sigma)$
$\text{Return } (C, C_B)$
```

```
$\underline{\text{OCB-Dec}_K^N(C, C_B)}$:
$\text{Partition } C \text{ into } C_1, \ldots, C_m$
$\text{For } i = 1 \text{ to } m-1 \text{ do } M_i \leftarrow \tilde{D}_K^{1,N,i,0}(C_i)$
$Pad \leftarrow \tilde{E}^{0,N,i,0}(len(C_m))$
$M_m \leftarrow C_m \oplus Pad$
$M \leftarrow M_1 \parallel \cdots \parallel M_m$
$\Sigma \leftarrow M_1 \oplus \cdots \oplus M_{m-1} \oplus C_m 0^* \oplus Pad$
$C'_B \leftarrow \tilde{E}_K^{0,N,i,1}(\Sigma)$
$\text{If } (C_B = C'_B) \text{ then Return } (M, (K, N))$
$\text{Return } \bot$

$\underline{\text{OCB-Ver}(M, (K, N), C_B)}$:
$\text{Partition } M \text{ into } M_1, \ldots, M_m$
$Pad \leftarrow \tilde{E}_K^{0,N,i,0}(len(M_m))$
$C_m \leftarrow M_m \oplus Pad$
$\Sigma \leftarrow M_1 \oplus \cdots \oplus M_{m-1} \oplus C_m 0^* \oplus Pad$
$C'_B \leftarrow \tilde{E}_K^{0,N,i,1}(\Sigma)$
$\text{If } C'_B \neq C_B \text{ then Return } 0$
$\text{Return } 1$
```

Figure 17: OCB as a nonce-based ccAEAD scheme. By C_m0^* we mean pad C_m with enough zeros to make an n-bit string. By "Partition" we mean parsing a string into n-bit blocks and one (possibly shorter) block. The function len returns the length of a string in bits.

the input tuples do not verify despite de-encrypting correctly; then $\mathcal{A}$ has found a ciphertext and header / key pair which violates the s-BIND security of the encryptment scheme EC; we may bound the probability of this event occurring via a reduction to an attacker $\mathcal{C}$ in game s-BIND against EC. If such an event does not occur, then $\mathcal{A}$ has found tuples $(K_{\mathsf{EC}}, H, M) \neq (K'_{\mathsf{EC}}, H', M')$ which both verify correctly with binding tag B_{EC}. This corresponds to a winning tuple for an attacker $\mathcal{D}$ in game sr-BIND against CE. Putting this together proves the claim.

E OCB is Not Binding

The offset codebook mode (OCB) was first introduced by Rogaway, Bellare, Black, and Krovetz [40] and later refined by Rogaway [39] and Rogaway and Krovetz [29]. We follow the formulation given in [39]. Recall that OCB relies on a tweakable blockcipher $\tilde{E}$ with tweak space $\mathcal{T} = \{0,1\} \times \{0,1\}^n \times [1 \mathinner{..} 2n/2] \times \{0,1\}$ and message space $\{0,1\}^n$ for some block length n (e.g., $n = 128$).

There are multiple ways to reconceptualize OCB as a ccAEAD scheme, in particular because one needs to decide whether to include the IV as part of the opening or the committing portion of the ciphertext. Neither approach provides (any form of) receiver binding; we demonstrate for brevity only the former conceptualization. (Readers unfamiliar with nonce-based ccAEAD should see [19].) Figure 17 details the pseudocode for a nonce-based ccAEAD scheme OCB = (OCB-Kg, OCB-Enc, OCB-Dec, OCB-Ver) based on OCB; we omit associated data from both the construction and the discussion below, for simplicity. Key generation simply picks a key for the underlying tweakable blockcipher $\tilde{E}$.

OCB has excellent performance properties, with rate-1 encryption and decryption (roughly, one blockcipher call per block of message / ciphertext; see Section 5 for a formal definition) and even faster verification. But unfortunately it is clearly not receiver binding as, intuitively, verification does not provide CR. In more detail, consider the following r-BIND adversary $\mathcal{A}$ against OCB. It computes $(C, C_B) \leftarrow \text{OCB-Enc}_K^N(M)$ for arbitrary K, N and message $M = M_1 || M_2 || \ldots || M_m$ such that $m \geq 3$. Attacker $\mathcal{A}$ then sets $M' = \overline{M_1}||\overline{M_2}||M_3||\ldots||M_m$, where $\overline{M_i}$ denotes message block M_i with its least significant bit flipped. Attacker $\mathcal{A}$ then returns $(((K,N),M),((K,N),M'),C_B)$ to its challenger.

Since the first two message blocks in M, M' have different least significant bits, it is clearly the case that $((K,N),M) \neq ((K,N),M')$. However, since $M_m = M'_m$, and $M_1 \oplus M_2 \oplus \cdots \oplus M_{m-1} = M'_1 \oplus M'_2 \oplus \cdots \oplus M'_{m-1}$, it is easy to verify that both messages will result in the same tag C_B when encrypted under (K,N), thereby allowing $\mathcal{A}$ to win game r-BIND with probability one.

F The SHA-3 Duplex Construction

Our HFC construction has a number of similarities to the AE scheme SpongeWrap built using the SHA-3 winner Keccak [7,8]. The SpongeWrap scheme uses a large permutation $\pi\colon \{0,1\}^{r+c} \to \{0,1\}^{r+c}$ for some $r, c \in \mathbb{N}$. This permutation is used to iteratively hash the message, with the resulting output forming the binding tag. Portions of the intermediate chaining variables are simultaneously used as pads to mask the message. The key difference between this scheme and HFC is that the large state size offered by Keccak means that there is no need to key each call to π — this is done indirectly via the extra c bits of state that are not used for outputs.

We now describe our SpongeWrap-like encryptment scheme SPE. It works as follows. Encryptment of a sequence of message blocks $M_1, \ldots, M_m$ each of length r bits with a key $K \in \{0,1\}^r$ first sets $Y_0 = \pi(K \parallel 0^c)$. The scheme then iteratively compute $C_i = Y_{i-1} \oplus M_i$ where $Y_i = \pi(Y_{i-1} \oplus (M_i \parallel 0^c))$ for $1 \leq i \leq m$. (Note that the XOR operation $Y_{i-1} \oplus M_i$ returns the XOR of the leftmost r bits of Y_{i-1} with M_i.) The binding tag B_{EC} is then set to be the leftmost r bits of $\pi(Y_m)$, and the scheme outputs encryptment $(C_1 \parallel \cdots \parallel C_m, B_{\mathsf{EC}})$. Decryption and verification work in the natural way. One can of course include associated data and suitable padding to achieve an encryptment scheme which can handle associated data and arbitrary length messages.

The security analyses from [8] imply that SpongeWrap as an encryptment scheme achieves otROR and SCU security. Tighter bounds can likely be obtained using techniques from [24]. The construction is receiver binding by a lifting of the proof of collision-resistance for Keccak. Moreover, it is straightforward to verify that the scheme is also sender binding and strongly correct. SpongeWrap-based encryptment yields one-pass ccAEAD via the transforms of Section 7.3, and may be a good choice should SHA-3 be implemented widely.

G Davies-Meyer as an RKA-PRF

We analyze the linear-only RKA-PRF security of Davies-Meyer (DM). First we need to define an additional RKA security notion for pseudorandom permutations (PRPs). For a blockcipher $E\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ (where for concreteness we define $\mathcal{K} = \{0,1\}^n$), define game RKA-PRP0 to be identical to game RKA-PRF0 except an oracle query for (X,Y) is answered via $E_{K \oplus Y}(X)$. Define game RKA-PRP1 to be identical to game RKA-PRF1 except permutivity is enforced on the random samples returned for all queries with the same $K \oplus Y$. For a blockcipher E and adversary $\mathcal{A}$, define the *linear-only RKA-PRP* advantage of $\mathcal{A}$ against E as

$$\mathbf{Adv}_{E}^{\oplus\text{-prp}}(\mathcal{A}) = \left| \Pr\left[\,\text{RKA-PRP0}_E^{\mathcal{A}} \Rightarrow 1\,\right] - \Pr\left[\,\text{RKA-PRP1}^{\mathcal{A}} \Rightarrow 1\,\right] \right| .$$

Next, we state the theorem bounding the RKA-PRF advantage of an adversary $\mathcal{A}$ against the DM construction with a blockcipher E by the RKA-PRP advantage of an adversary $\mathcal{B}$ against the blockcipher E. The proof of the following theorem is straightforward.

Theorem 10 *Let $E\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be a blockcipher. Let $\mathsf{DM}[E]$ be the function defined as $\mathsf{DM}[E](K, V_1, V_2) = E_{V_2 \oplus K}(V_1) \oplus V_1$. Let $\mathcal{A}$ be an RKA-PRF adversary against $\mathsf{DM}[E]$ making at most q queries to its oracle. Then there exists an adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\oplus\text{-prf}}_{\mathsf{DM}[E]}(\mathcal{A}) \leq \mathbf{Adv}^{\oplus\text{-prp}}_{E}(\mathcal{B}) + \frac{q^2}{2^n} \ .$$

The adversary $\mathcal{B}$ runs in the same amount of time as $\mathcal{A}$ with an $\mathcal{O}(q)$ overhead and makes q queries.

H Proofs from Section 6

Proof of Theorem 3.

We first introduce some notation to simplify the subsequent discussion. Recall that our HFC padding is defined $\text{Pad}(H,M) = \text{PadH}(H,M) \,\|\, \text{PadM}(H,M) \,\|\, \text{PadSuf}(|H|,|M|,M_m)$. For a header / message pair (H,M) such that $X_1||\ldots||X_\ell \leftarrow \text{Parse}_d(\text{Pad}(H,M))$, we define

$$\mathcal{P}(K_{\mathsf{EC}}, H, M) := K_{\mathsf{EC}}||X_1 \oplus K_{\mathsf{EC}}||\ldots||X_\ell \oplus K_{\mathsf{EC}} \ ;$$

and notice that HFC computes the binding tag of a triple (K_{EC}, H, M) as $B_{\mathsf{EC}} = \mathsf{f}^+(IV, X_1||\ldots||X_\ell)$, where recall that $IV \in \{0,1\}^n$ is a fixed public constant.

Consider an attacker $\mathcal{B}$ in the CR-game against f^+. $\mathcal{B}$ runs the sr-BIND attacker $\mathcal{A}$ as a subroutine. Suppose that $\mathcal{A}$ wins the sr-BIND against HFC with tuple $((K_{\mathsf{EC}}, H, M), (K'_{\mathsf{EC}}, H', M'), B_{\mathsf{EC}})$. Since verification HFCVer works by recomputing the binding tag and checking for equality, a win for $\mathcal{A}$ corresponds to finding $(K_{\mathsf{EC}}, H, M) \neq (K'_{\mathsf{EC}}, H', M')$ such that $\mathsf{f}^+(IV, \mathcal{P}(K_{\mathsf{EC}}, H, M)) = \mathsf{f}^+(IV, \mathcal{P}(K'_{\mathsf{EC}}, H', M'))$. We claim that such a win for $\mathcal{A}$ allows $\mathcal{B}$ to construct a winning collision in the CR-game against f^+. To see this, notice that $\mathcal{P}$ is injective, which is to say that for all $(K_{\mathsf{EC}}, H, M), (K'_{\mathsf{EC}}, H', M')$, if $\mathcal{P}(K_{\mathsf{EC}}, H, M) = \mathcal{P}(K'_{\mathsf{EC}}, H', M')$, this implies that $(K_{\mathsf{EC}}, H, M) = (K'_{\mathsf{EC}}, H', M')$. To justify this, notice that since K_{EC} is prepended to the output of $\mathcal{P}$, if $K_{\mathsf{EC}} \neq K'_{\mathsf{EC}}$ the padded strings are clearly different. On the other hand if $K_{\mathsf{EC}} = K'_{\mathsf{EC}}$, then $\mathcal{P}(K_{\mathsf{EC}}, H, M) = \mathcal{P}(K'_{\mathsf{EC}}, H', M')$ implies that $\text{Pad}(H,M) = \text{Pad}(H',M')$. This in turns implies that $(H,M) = (H',M')$ by the injectivity of $\text{Pad}(H,M)$ as discussed in Section 6.

As such, any winning tuple $((K_{\mathsf{EC}}, H, M), (K'_{\mathsf{EC}}, H', M'), B_{\mathsf{EC}})$ for $\mathcal{A}$ breaking the sr-BIND security of the HFC scheme, corresponds to a winning pair $\mathcal{P}(K_{\mathsf{EC}}, H, M) \neq \mathcal{P}(K'_{\mathsf{EC}}, H', M')$ for $\mathcal{B}$ in the CR-game against f^+. It follows that

$$\mathbf{Adv}^{\text{sr-bind}}_{\mathsf{HFC}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{cr}}_{\mathsf{f}^+}(\mathcal{B}) \ ,$$

as required. ▮

Proof of Theorem 4.

The games for this proof are shown in Figure 18. We begin with game G_0 of that figure, which is a rewriting of otROR0 which runs the encryption procedure of HFC in $\mathcal{A}$'s encryption oracle. Since this is only a syntactic change,

$$\Pr\left[\, \text{otROR0}^{\mathcal{A}}_{\mathsf{HFC}} \Rightarrow 1 \,\right] = \Pr\left[\, G_0 \Rightarrow 1 \,\right] \ .$$

Next we transition to game G_1, in which each fresh query to the compression function f is answered with an n-bit random bit string as opposed to the output of the function. A look up table is maintained to ensure that responses to repeated queries are consistent. We may bound the difference between G_0 and G_1 via a reduction to the RKA-PRF security of f. Let $\mathcal{B}$ be an attacker in the RKA-PRF game against f. Attacker $\mathcal{B}$ simulates $\mathcal{A}$'s encryption oracle using its own ROR oracle as follows. On input (H,M), attacker $\mathcal{B}$ pads and partitions the message as per the scheme. $\mathcal{B}$

then computes the encryptment following the pseudocode description of encryptment algorithm HFCEnc and submitting a query (V, X) to his ROR oracle every time HFCEnc would compute $\mathsf{f}(V, (K_{\mathsf{EC}} \oplus X))$. $\mathcal{B}$ then returns the resulting ciphertext and binding tag $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$ to $\mathcal{A}$, and outputs whatever bit $\mathcal{A}$ does. Notice that if $\mathcal{B}$'s oracle is returning real compression function outputs then this perfectly simulates game G_0, otherwise it perfectly simulates game G_1. Therefore a reduction to the RKA-PRF security of f implies that

$$|\Pr[G_0 \Rightarrow 1] - \Pr[G_1 \Rightarrow 1]| \leq \mathbf{Adv}_{\mathsf{f}}^{\oplus\text{-prf}}(\mathcal{B}) .$$

Next we define game G_2 (not shown) to be identical to G_1, except we answer each fresh query to f with a uniform random bit string, regardless of whether or not the query is fresh. Games G_2 and G_1 run identically unless f is queried on the same input twice. Notice that this may only occur if one of the randomly sampled n-bit chaining variables collides with either another randomly sampled n-bit chaining variable or with $IV \in \{0,1\}^n$. Supposing $\mathcal{A}$'s query is such that the message / header have a combined length of ℓ d-bit blocks after padding, then a standard argument taking a birthday bound implies that

$$|\Pr[G_1 \Rightarrow 1] - \Pr[G_2 \Rightarrow 1]| \leq \frac{(\ell+1)^2}{2^{n+1}} ,$$

where the extra plus one follows from the extra compression function call required to compute the initial chaining variable V_0. Moreover, notice that in game G_2 both C_{EC} and B_{EC} are identically distributed to random bit strings, and so

$$\Pr[G_2 \Rightarrow 1] = \Pr\left[\text{otROR1}_{\mathsf{HFC}}^{\mathcal{A}} \Rightarrow 1\right] .$$

Combining and rewriting gives the desired result. ▌

Proof of Theorem 5.

Let $\mathcal{A}$ be an attacker in game otCTXT against HFC who makes q queries to oracle dec. We argue by a series of game hops. We begin by defining game G_0 which is equivalent to game otCTXT against HFC. We may assume without loss of generality that $\mathcal{A}$ never repeats a query to dec, nor queries the tuple $(H^*, C^*_{\mathsf{EC}}, B^*_{\mathsf{EC}})$ where $(C^*_{\mathsf{EC}}, B^*_{\mathsf{EC}})$ is the encryptment returned in response to $\mathcal{A}$'s enc query (H^*, M^*), since such a query will always result in $\perp$ being returned. More generally, we may assume that $\mathcal{A}$ never makes a dec query of the form $(H^*, C^*_{\mathsf{EC}}, B_{\mathsf{EC}})$ for some $B_{\mathsf{EC}} \neq B^*_{\mathsf{EC}}$, where again starred values correspond to the challenge enc query, since due to the strong correctness of the scheme, the binding tag will be incorrect, and so decryptment will always return an error. Finally, we may assume that $\mathcal{A}$ never makes a dec query such that the 'first phase' of decryptment — by which we mean the steps which recover the message underlying the ciphertext — returns an error, since this cannot possibly correspond to a winning query. As such, to each dec query $(H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$ made by $\mathcal{A}$, we may associate a pair (H, M) which will be recovered in the first decryptment phase from (H, C_{EC}), and moreover due to the strong correctness of the scheme, it follows that if $(H, C_{\mathsf{EC}}) \neq (H', C'_{\mathsf{EC}})$ then the associated header / message pairs $(H, M), (H', M')$ will be distinct also.

With this in place, we now define G_1, which is identical to G_0 except each fresh query to the compression function f is answered with an n-bit random bit string as opposed to the output of the function. A look up table is maintained to ensure that responses to repeated queries are consistent. An analogous argument to that made in the proof of Theorem 4 implies that exists an attacker $\mathcal{B}$ in the RKA-PRF game against f with the claimed query budget such that

$$|\Pr[G_0 \Rightarrow 1] - \Pr[G_1 \Rightarrow 1]| \leq \mathbf{Adv}_{\mathsf{f}}^{\oplus\text{-prf}}(\mathcal{B}) .$$

Next we define game G_2, which is identical to G_1 except we modify oracle dec to decrypt queries for

```
$\underline{G_0}$:
$K_{\mathsf{EC}} \leftarrow\!\!\$ \{0,1\}^d$
$\mathsf{query\text{-}made} \leftarrow \mathsf{false}$
$b \leftarrow\!\!\$ \mathcal{A}^{\mathsf{enc}(\cdot,\cdot)}$
Return $b$

$\underline{\mathsf{enc}(H,M)}$:
If $\mathsf{query\text{-}made} = \mathsf{true}$ then Return $\perp$
$\mathsf{query\text{-}made} \leftarrow \mathsf{true}$
$H_1,\ldots,H_h \leftarrow \mathrm{PadH}(H,M)$
$M_1,\ldots,M_m \leftarrow \mathrm{PadM}(H,M)$
$V_0 \leftarrow \mathsf{f}(IV, K_{\mathsf{EC}})$
For $i = 1,\ldots,h$ do $V_i \leftarrow \mathsf{f}(V_{i-1},(K_{\mathsf{EC}} \oplus H_i))$
$C_{\mathsf{EC}} \leftarrow \varepsilon$
For $i = 1,\ldots,m-1$ do
    $C_{\mathsf{EC}} \leftarrow C_{\mathsf{EC}} \,\|\, (V_{h+i-1} \oplus M_i)$
    $V_{h+i} \leftarrow \mathsf{f}(V_{h+i-1},(K_{\mathsf{EC}} \oplus M_i))$
$C_{\mathsf{EC}} \leftarrow C_{\mathsf{EC}} \,\|\, (V_{h+m-1} \oplus M_m)$
$M'_m, M'_{m+1} \leftarrow \mathrm{Parse}_d(\mathrm{PadSuf}(|H|,|M|,M_m))$
$\alpha \leftarrow \max\{i \;:\; |M'_{m+i}| > 0\}$
For $i = 0,\alpha$
    $V_{h+m+i} \leftarrow \mathsf{f}(V_{h+m+i-1},(K_{\mathsf{EC}} \oplus M_{m+i}))$
$B_{\mathsf{EC}} \leftarrow V_{h+m+\alpha}$
Return $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$
```

```
$\underline{G_1}$:
$K_{\mathsf{EC}} \leftarrow\!\!\$ \{0,1\}^d$
$\mathsf{query\text{-}made} \leftarrow \mathsf{false}$
$b \leftarrow\!\!\$ \mathcal{A}^{\mathsf{enc}(\cdot,\cdot)}$
Return $b$

$\underline{\mathsf{enc}(H,M)}$:
If $\mathsf{query\text{-}made} = \mathsf{true}$ then Return $\perp$
$\mathsf{query\text{-}made} \leftarrow \mathsf{true}$
$H_1,\ldots,H_h \leftarrow \mathrm{PadH}(H,M)$
$M_1,\ldots,M_m \leftarrow \mathrm{PadM}(H,M)$
$G[IV,K_{\mathsf{EC}}] \leftarrow\!\!\$ \{0,1\}^n$ ; $V_0 \leftarrow G[IV,K_{\mathsf{EC}}]$
For $i = 1,\ldots,h$ do
    If $G[V_{i-1},(K_{\mathsf{EC}} \oplus H_i)] = \perp$ then
        $G[V_{i-1},(K_{\mathsf{EC}} \oplus H_i)] \leftarrow\!\!\$ \{0,1\}^n$
    $V_i \leftarrow G[V_{i-1},(K_{\mathsf{EC}} \oplus H_i)]$
$C_{\mathsf{EC}} \leftarrow \varepsilon$
For $i = 1,\ldots,m-1$ do
    $C_{\mathsf{EC}} \leftarrow C_{\mathsf{EC}} \,\|\, (V_{h+i-1} \oplus M_i)$
    If $G[V_{h+i-1},(K_{\mathsf{EC}} \oplus M_i)] = \perp$ then
        $G[V_{h+i-1},(K_{\mathsf{EC}} \oplus M_i)] \leftarrow\!\!\$ \{0,1\}^n$
    $V_{h+i} \leftarrow G[V_{h+i-1},(K_{\mathsf{EC}} \oplus M_i)]$
$C_{\mathsf{EC}} \leftarrow C_{\mathsf{EC}} \,\|\, (V_{h+m-1} \oplus M_m)$
$M'_m, M'_{m+1} \leftarrow \mathrm{Parse}(\mathrm{PadSuf}(|H|,|M|,M_m))$
$\alpha \leftarrow \max\{i \;:\; |M'_{m+i}| > 0\}$
For $i = 0,\alpha$
    If $G[V_{h+m+i-1},(K_{\mathsf{EC}} \oplus M_{m+i})] = \perp$ then
        $G[V_{h+m+i-1},(K_{\mathsf{EC}} \oplus M_{m+i})] \leftarrow\!\!\$ \{0,1\}^n$
    $V_{h+m+i-1} \leftarrow G[V_{h+m+i-1},(K_{\mathsf{EC}} \oplus M_{m+i})]$
$B_{\mathsf{EC}} \leftarrow V_{h+m+\alpha}$
Return $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$
```

Figure 18: Games for the proof of Theorem 4.

which the header / ciphertext pair (H, C_{EC}) have previously been queried to dec by table look-up. In more detail, table D is initialized to $\perp$. Each time dec is queried on a tuple $(H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$ such that $D[H, C_{\mathsf{EC}}] = \perp$, dec runs HFCDec and sets $D[H, C_{\mathsf{EC}}] = (M, B_{\mathsf{EC}})$ where M is the message recovered during decryption, and B_{EC} is the (correct) binding tag for this pair which is computed during decryption. Subsequently, if dec is queried on a tuple $(H, C_{\mathsf{EC}}, B'_{\mathsf{EC}})$, such that $D[H, C_{\mathsf{EC}}] \neq \perp$, it simply checks if $B_{\mathsf{EC}} = B'_{\mathsf{EC}}$, returns M if so, and $\perp$ otherwise. This is a purely syntactic change which does not affect the outcome of the game, and so it follows that

$$\Pr[\,G_1 \Rightarrow 1\,] = \Pr[\,G_2 \Rightarrow 1\,]\ .$$

Next we define game G_3, which is identical to game G_2, except if one of random strings sampled to respond to a fresh query to f collides with a previously sampled string or the initialization vector IV, then we sample again such that this is not the case. Notice that these games run identically unless two of these sampled strings collide, an event we denote coll. The Fundamental Lemma of Game Playing therefore implies that

$$|\Pr[\,\mathrm{G}_2 \Rightarrow 1\,] - \Pr[\,\mathrm{G}_3 \Rightarrow 1\,]| \leq \Pr[\,\mathsf{coll} \text{ in } G_2\,]\ .$$

We now bound this probability. Let (H^0, M^0) denote the query made by $\mathcal{A}$ to enc. Suppose that of $\mathcal{A}$'s q dec queries, these contain q_1 distinct and chronologically ordered pairs $(H^1, C^1_{\mathsf{EC}}), \ldots, (H^{q_1}, C^{q_1}_{\mathsf{EC}})$, and let $(H^1, M^1), \ldots, (H^{q_1}, M^{q_1})$ denote the pairs of headers and underlying messages corresponding to these queries. Moreover, following from the discussion at the beginning of the proof, it must be the case that $(H^i, M^i) \neq (H^j, M^j)$ for $0 \le i < j \le q_1$. Suppose that the total padded length of these messages in d-bit blocks is equal to ℓ. Notice that at most $(\ell + 1)$ strings will be sampled while processing these queries (the plus one term arising from the initial query of (IV, K_{EC}) made by enc to compute the initial chaining variable V_0), and so a birthday bound implies that

$$\Pr[\,\mathsf{coll} \text{ in } G_2\,] \le \frac{(\ell+1)^2}{2^{n+1}} \ .$$

We now argue that in game G_3, each binding tag $B^0_{\mathsf{EC}}, \ldots, B^{q_1}_{\mathsf{EC}}$ is computed as the result of a fresh query to f, and is chosen randomly from a set of size at least $2^n - \ell$. To see this, consider computing the binding tag for each of the padded strings $\mathrm{Pad}(H^0, M^0), \ldots, \mathrm{Pad}(H^{q_1}, M^{q_1})$ in G_3, which are of length $\ell_0, \ldots, \ell_{q_1}$ in d-bit blocks respectively. To $\mathrm{Pad}(H^i, M^i)$, we let $(V^i_0, \ldots, V^i_{\ell_i})$ denote the set of chaining variables passed through during the binding tag computation, where $V^i_0 = \mathsf{f}(IV, K_{\mathsf{EC}}) = V^j_0$ for all $0 \le i < j \le q_1$, and $B^i_{\mathsf{EC}} = V^i_{\ell_i}$. We let $|\mathrm{LCP}(i,j)|$ denote the the length of the longest common prefix in d-bit blocks of $\mathrm{Pad}(H^i, M^i)$ and $\mathrm{Pad}(H^j, M^j)$. More formally, we let $|\mathrm{LCP}(i,j)| = \max\{k \ : \ \mathrm{Trunc}_{k\cdot d}(\mathrm{Pad}(H^i, M^i)) = \mathrm{Trunc}_{k\cdot d}(\mathrm{Pad}(H^j, M^j))\}$ (recall $\mathrm{Trunc}_{k\cdot d}(X)$ returns X if $|X| < kd$ and the first kd bits of X otherwise). We let $|\mathrm{LCP}(j)| = \max_{i<j} |\mathrm{LCP}(i,j)|$; that is to say, the length of the longest prefix in d-bit blocks that $\mathrm{Pad}(H^j, M^j)$ shares with any previously processed padded message. We write M^j_k to denote the kth d-bit block of $\mathrm{Pad}(H^j, M^j)$.

We claim that the following hold. **(1.)** Let $1 \le j \le q_1$, and $p = |\mathrm{LCP}(j)|$. Then for all $0 \le i < j$ such that $\mathrm{LCP}(i,j) = p$, it holds that $p < \min(\ell_i, \ell_j)$. **(2.)** Consider the binding tag computation for padded messages $\mathrm{Pad}(H^i, M^i)$ and $\mathrm{Pad}(H^j, M^j)$ where $0 \le i < j \le q_1$, and suppose that of the associated series of chaining variables it holds that $V^i_k = V^j_{k'}$ for some $0 \le k \le \ell_i$, $0 \le k' \le \ell_j$. Then it must be the case that $k = k'$ and the first k blocks of $\mathrm{Pad}(H^i, M^i)$ and $\mathrm{Pad}(H^j, M^j)$ are equal.

To see that **(1.)** holds, first recall that the padding scheme Pad is prefix-free. Since by assumption $(H^i, M^i) \neq (H^j, M^j)$ for all $0 \le i < j \le q_1$, it must be the case that $\mathrm{Pad}(H^i, M^i)$ is not a prefix of $\mathrm{Pad}(H^j, M^j)$ or vice versa. As such, they must differ in at least one d-bit block, and so it follows that $|\mathrm{LCP}(i,j)| < \min(\ell_i, \ell_j)$.

For **(2.)**, note that if $k = k' = 0$ then the statement is vacuously true, and so we suppose for the remainder of the proof that at least one of k, k' is non-zero. Suppose for a contradiction there exist $\mathrm{Pad}(H^i, M^i), \mathrm{Pad}(H^j, M^j)$ which share a common chaining variable $V^i_k = V^j_{k'}$, but for which the claim does not hold. Since the strings returned in response to compression function f calls are sampled without replacement, distinct queries will always return distinct responses. Suppose that $k, k' > 0$ (the case in which one of k, k' equals 0 is entirely analogous). Then $V^i_k, V^j_{k'}$ are computed by querying $(V^i_{k-1}, (K_{\mathsf{EC}} \oplus M^i_k))$ and $(V^j_{k'-1}, (K_{\mathsf{EC}} \oplus M^j_{k'}))$ to f respectively. As such, it must be the case that $V^i_{k-1} = V^j_{k'-1}$ and $M^i_k = M^j_k$. Repeatedly applying this argument yields that $V^i_{k-\alpha} = V^j_{k'-\alpha}$ and $M^i_{k-\alpha+1} = M^j_{k'-\alpha+1}$ for all $\alpha = 1, \ldots, \min(k, k')$. Suppose first that $k \neq k'$, and suppose without loss of generality that $k < k'$. This implies that $V^i_0 = V^j_{k'-k}$ where $k' - k \ge 1$. However, since $V^i_0 = \mathsf{f}(IV, K_{\mathsf{EC}})$ this implies that $V^j_{k'-k-1} = IV$, which is impossible since $V^j_{k'-k-1}$ must have been computed via querying the random function and in G_3, IV is removed from the range of the function. Therefore it must be the case that $k = k'$, and so by the previous argument $(M^i_1, \ldots, M^i_k) = (M^j_1, \ldots, M^j_k)$ also, proving the claim.

With this in place, consider computing the binding tag for a message $\mathrm{Pad}(H^j, M^j)$, where $p = |\mathrm{LCP}(j)|$. Let $\mathrm{Prev}_j = \{\mathrm{Pad}(H^k, M^k)\}_{k<j}$ be the messages resulting from the first $j - 1$ queries to dec. Let $\mathcal{X}_j = \{\mathrm{Pad}(H^i, M^i) \in \mathrm{Prev}_j \ : \ \mathrm{LCP}(i,j) = p\}$; in other words, the set of previously processed padded header / message pairs which share a prefix of length p blocks with

$\mathrm{Pad}(H^j, M^j)$. Due to the consistency in responses to repeated queries to the compression function, it is clearly the case that for all $\mathrm{Pad}(H^i, M^i) \in \mathcal{X}_j$, it holds that $(V_0^i, \ldots, V_p^i) = (V_0^j, \ldots, V_p^j)$. Since by **(1.)**, it must be the case that $p < \ell_i, \ell_j$, it follows that $M_{p+1}^i, M_{p+1}^j \neq \varepsilon$, and moreover the next chaining variable for $\mathrm{Pad}(H^j, M^j)$ will be computed via querying $(V_p^j, (K_{\mathsf{EC}} \oplus M_{p+1}^j))$ to f.

We claim this will be a fresh query to f, and so will be answered with a string chosen uniformly from a set of at most $2^n - \ell - 1$ untaken points. To see this, notice that for all $\mathrm{Pad}(H^i, M^i) \in \mathcal{X}_j$, it holds that $M_{p+1}^i \neq M_{p+1}^j$, since otherwise this would contradict the maximality of p. Now **(2.)** implies that V_p^j cannot equal V_k^i for any $\mathrm{Pad}(H^i, M^i)$, $0 \leq i < j$ and $0 \leq k \leq \ell_i$ *unless* $\mathrm{Pad}(H^i, M^i) \in \mathcal{X}_j$ and $k = p$. This implies that V_p^j has not been queried to f at any point during the computation of the first $(j-1)$ binding tags other than during the computation of the $(p+1)^{\mathrm{st}}$ chaining variable for strings in $\mathcal{X}_j$. Since we have already argued that all such queries have a distinct message block and so do not collide, it follows that $(V_p^j, (K_{\mathsf{EC}} \oplus M_{p+1}^j))$ represents a fresh query to f, and so V_{p+1}^j is chosen uniformly from the set of available points. This in turn forces the query made to compute the next chaining variable to be distinct from all points previously queried, and so repeatedly applying the same argument implies the binding tag $B_{\mathsf{EC}}^j = V_{\ell_j}^j$ is the result of a fresh query also. Finally, since at most $(\ell + 1)$ chaining variables are sampled while processing the messages of combined length ℓ d-bit blocks, and the random strings returned in response to fresh f queries are sampled without replacement (and from a set excluding $IV \in \{0,1\}^n$), it follows that each binding tag is chosen from a set of size at most $(2^n - \ell - 1)$, proving the claim.

We conclude by bounding the probability that $\mathcal{A}$ makes a winning query to dec, an event we denote win. For $1 \leq i \leq q_1$, we mark the event that $\mathcal{A}$ makes a query $(H^i, C_{\mathsf{EC}}^i, B'_{\mathsf{EC}})$ to dec such that $B'_{\mathsf{EC}} = B_{\mathsf{EC}}^i$ by setting a flag win_i. Suppose that $\mathcal{A}$ makes a_i queries with header / ciphertext (H^i, C_{EC}^i), and notice that $\sum_{i=1}^{q_1} a_i = q$. It follows that

$$\begin{aligned}
\Pr[\,\mathsf{win} = \mathsf{true}\,] = \Pr[\,\vee_{i=1}^{q_1} \mathsf{win}_i = \mathsf{true}\,] &\leq \sum_{i=1}^{q_1}\sum_{j=1}^{a_i} \Pr\Big[\,\mathsf{win}_i \text{ set by } j^{\mathrm{th}} \text{ query of form } (H^i, C_{\mathsf{EC}}^i, \cdot)\,\Big] \\
&\leq \sum_{i=1}^{q_1}\sum_{j=1}^{a_i} \frac{1}{2^n - \ell - 1 - (j-1)} \leq \sum_{i=1}^{q} \frac{a_i}{2^n - \ell - 1 - (q-1)} \\
&\leq \frac{q}{2^n - \ell - q}\,.
\end{aligned}$$

The first inequality follows from a union bound. The second inequality follows since each binding tag is chosen uniformly from a set of size at least $2^n - \ell - 1$, and each incorrect guess at binding tag B_{EC}^i allows $\mathcal{A}$ to reduce the size of the set of possible values for that binding tag by one. The third inequality follows since $a_i \leq q$ for all $1 \leq i \leq q_1$. The final argument follows since $\sum_{i=1}^{q_1} a_i = q$. ∎

I A Transform from ccAEAD to Encryptment

In this section, we describe a transform which builds a secure encryptment scheme from any secure ccAEAD scheme. Moreover, the rate of encryptment in the transformed construction is exactly that of ccAEAD encryption; as such, the negative results on rate-1 encryptment schemes from Section 5 in turn rule out the existence of rate-1 ccAEAD schemes. In Section 7.3, we prove that the other direction of the implication holds by showing that any secure encryptment scheme can be transformed into a secure ccAEAD scheme using an authenticated encryption scheme, thereby establishing an equivalence between encryptment and ccAEAD.

Let $\mathsf{CE} = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ver})$ be a ccAEAD scheme, with associated coin space $\mathcal{R}$. Then we may construct an encryptment scheme $\mathsf{EC}[\mathsf{CE}] = (\mathsf{EKg}, \mathsf{EC}, \mathsf{DO}, \mathsf{EVer})$ from CE as follows. We define EKg to be the algorithm which generates a ccAEAD key $K \leftarrow\!\!\$\ \mathsf{Kg}$, chooses random coins $R \leftarrow\!\!\$\ \mathcal{R}$ where $\mathcal{R}$ denotes the coin space of the ccAEAD scheme, and outputs $K_{\mathsf{EC}} = K \parallel R$. The deterministic encryptment algorithm on input $(K \| R, H, M)$ uses the encryption algorithm of the ccAEAD scheme with coins fixed to R to compute $(C, C_B) \leftarrow \mathsf{Enc}(K, H, M; R)$, returning $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) = (C, C_B)$. In particular, notice that the rate of EC is *exactly* that of Enc.

We define DO to be the algorithm which, on input $(K \| R, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$ first uses the decryption algorithm of the ccAEAD scheme to compute $(M, K_f) \leftarrow \mathsf{Dec}(K, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$. It then checks if $\mathsf{Enc}(K, H, M; R) = (C_{\mathsf{EC}}, B_{\mathsf{EC}})$ and if so returns M; otherwise it returns $\perp$. Notice that re-encrypting and comparing the encryptment in this way ensures the encryptment scheme has strong correctness.

For EVer there are two cases. If the ccAEAD scheme is such that it outputs its key as the opening $K_f = K$, then we define EVer to be the algorithm which on input $(H, M, K \parallel R, B_{\mathsf{EC}})$ simply computes $\mathsf{Ver}(H, M, K, B_{\mathsf{EC}})$ and returns the result. If ccAEAD does not fall in this class of schemes, then EVer needs to recover the opening key K'_f before verifying the binding tag. EVer can always do this (for both classes of ccAEAD scheme) given $(H, M, K \parallel R, B_{\mathsf{EC}})$ by re-computing $(C', C'_B) \leftarrow \mathsf{Enc}(K, H, M; R)$, followed by $(M', K'_f) \leftarrow \mathsf{Dec}(K, H, C', C'_B)$ to recover the opening K'_f, and finally returning the output of $\mathsf{Ver}(H, M, K'_f, B_{\mathsf{EC}})$, (or 0 if any of these intermediate steps return an error).

We describe the security of the derived encryptment scheme $\mathsf{EC}[\mathsf{CE}]$ in the following theorem.

Theorem 11 *Let* CE= (Kg, Enc, Dec, Ver) *be a ccAEAD scheme. Then there exists a strongly correct encryptment scheme* $\mathsf{EC}[\mathsf{CE}]$= (EKg, EC, DO, EVer) *such that for all adversaries* $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$*, we give adversaries* $\mathcal{B}, \mathcal{C}, \mathcal{D}$ *such that* $\mathbf{Adv}^{\text{ot-ror}}_{\mathsf{EC}[\mathsf{CE}]}(\mathcal{A}_1) \leq \mathbf{Adv}^{\text{mo-ror}}_{\mathsf{CE}}(\mathcal{B})$, $\mathbf{Adv}^{\text{r-bind}}_{\mathsf{EC}[\mathsf{CE}]}(\mathcal{A}_2) \leq \mathbf{Adv}^{\text{r-bind}}_{\mathsf{CE}}(\mathcal{C})$, *and* $\mathbf{Adv}^{\text{s-bind}}_{\mathsf{EC}[\mathsf{CE}]}(\mathcal{A}_3) \leq \mathbf{Adv}^{\text{s-bind}}_{\mathsf{CE}}(\mathcal{D})$*. Moreover, adversaries* $\mathcal{B}, \mathcal{C}, \mathcal{D}$ *run in the time of* $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ *respectively.*

Since the encryptment is recomputed during decryption with DO, it is straightforward to see that the scheme is strongly correct. We sketch the proof of the remainder of the three-part theorem below.

(Part 1) It is easy to see that an attacker $\mathcal{B}$ in game MO-ROR against the ccAEAD scheme CE can perfectly simulate game otROR for an attacker $\mathcal{A}_1$ against the derived encryptment scheme $\mathsf{EC}[\mathsf{CE}]$ by submitting $\mathcal{B}$'s encryption query to his own encryption oracle and returning the result; therefore a reduction to the otROR security of CE implies the first result.

(Part 2) To see that $\mathsf{EC}[\mathsf{CE}]$ is receiver binding, notice that to win game r-BIND against $\mathsf{EC}[\mathsf{CE}]$, attacker $\mathcal{A}_2$ must output a tuple

$$((K \parallel R, H, M), (K' \parallel R', H', M'), C_B)$$

such that $(H, M) \neq (H', M')$ but for which

$$\mathsf{Ver}(H, M, K_f, C_B) = \mathsf{Ver}(H', M', K'_f, C_B) = 1$$

where

$$(M, K_f) = \mathsf{Dec}(K, H, \mathsf{Enc}(K, H, M; R)), \text{ and}$$
$$(M', K'_f) = \mathsf{Dec}(K', H', \mathsf{Enc}(K', H', M'; R'))\ .$$

Thus an adversary $\mathcal{C}$ in game r-BIND against CE can simply run $\mathcal{A}_2$. When $\mathcal{A}_2$ outputs $((K \parallel R, H, M), (K' \parallel R', H', M'), C_B)$, $\mathcal{C}$ runs Enc then Dec on both tuples to obtain each of the

```
$\underline{\mathrm{G}_0:}$
$K \leftarrow\!\!\$\ \mathsf{AEAD.Kg}$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathbf{Enc},\mathbf{Dec},\mathbf{ChalEnc}}$
Return $b'$

$\underline{\mathbf{Enc}(H, M):}$
$K_{\mathsf{EC}} \leftarrow\!\!\$\ \mathsf{EKg}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow\!\!\$\ \mathsf{AEAD.enc}(K, B_{\mathsf{EC}}, K_{\mathsf{EC}})$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}})\}$
Return $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$

$\underline{\mathbf{Dec}(H, C, C_B):}$
If $(H, C, C_B) \notin \mathcal{Y}$ then
    Return $\perp$
$(C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
$K_{\mathsf{EC}} \leftarrow \mathsf{AEAD.dec}(K, B_{\mathsf{EC}}, C_{\mathsf{AE}})$
If $K_{\mathsf{EC}} = \perp$ then Return $\perp$
$(M, K_{\mathsf{EC}}) \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $(M, K_{\mathsf{EC}}) = \perp$ then Return $\perp$
Return $(M, K_{\mathsf{EC}})$

$\underline{\mathbf{ChalEnc}(H, M):}$
$K_{\mathsf{EC}} \leftarrow\!\!\$\ \mathsf{EKg}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow\!\!\$\ \mathsf{AEAD.enc}(K, B_{\mathsf{EC}}, K_{\mathsf{EC}})$
Return $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$
```

```
$\underline{\mathrm{G}_1, \boxed{G_2}:}$
$K \leftarrow\!\!\$\ \mathsf{AEAD.Kg}$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathbf{Enc},\mathbf{Dec},\mathbf{ChalEnc}}$
Return $b'$

$\underline{\mathbf{Enc}(H, M):}$
$K_{\mathsf{EC}} \leftarrow\!\!\$\ \mathsf{EKg}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow\!\!\$\ \mathsf{AEAD.enc}(K, B_{\mathsf{EC}}, K_{\mathsf{EC}})$
$\boxed{C_{\mathsf{AE}} \leftarrow\!\!\$\ \mathcal{C}_{\mathsf{AEAD}}(|K_{\mathsf{EC}}|)}$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}})\}$
$D[H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$

$\underline{\mathbf{Dec}(H, C, C_B):}$
If $(H, C, C_B) \notin \mathcal{Y}$ then
    Return $\perp$
$(C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
$(M, K_{\mathsf{EC}}) \leftarrow D[H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}}]$
Return $(M, K_{\mathsf{EC}})$

$\underline{\mathbf{ChalEnc}(H, M):}$
$K_{\mathsf{EC}} \leftarrow\!\!\$\ \mathsf{EKg}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow\!\!\$\ \mathsf{AEAD.enc}(K, B_{\mathsf{EC}}, K_{\mathsf{EC}})$
$\boxed{C_{\mathsf{AE}} \leftarrow\!\!\$\ \mathcal{C}_{\mathsf{AEAD}}(|K_{\mathsf{EC}}|)}$
Return $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$
```

```
$\underline{\mathrm{G}_3, \boxed{G_4}:}$
$K \leftarrow\!\!\$\ \mathsf{AEAD.Kg}$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathbf{Enc},\mathbf{Dec},\mathbf{ChalEnc}}$
Return $b'$

$\underline{\mathbf{Enc}(H, M):}$
$K_{\mathsf{EC}} \leftarrow\!\!\$\ \mathsf{EKg}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow\!\!\$\ \mathcal{C}_{\mathsf{AEAD}}(|K_{\mathsf{EC}}|)$
$\boxed{C_{\mathsf{AE}} \leftarrow\!\!\$\ \mathsf{AEAD.enc}(K, B_{\mathsf{EC}}, K_{\mathsf{EC}})}$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}})\}$
$D[H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$

$\underline{\mathbf{Dec}(H, C, C_B):}$
If $(H, C, C_B) \notin \mathcal{Y}$ then
    Return $\perp$
$(C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
$(M, K_{\mathsf{EC}}) \leftarrow D[H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}}]$
Return $(M, K_{\mathsf{EC}})$

$\underline{\mathbf{ChalEnc}(H, M):}$
$K_{\mathsf{EC}} \leftarrow\!\!\$\ \mathsf{EKg}$
$C_{\mathsf{EC}} \leftarrow\!\!\$\ \{0,1\}^{\mathsf{clen}(|M|)}$
$B_{\mathsf{EC}} \leftarrow\!\!\$\ \{0,1\}^{\mathsf{btlen}}$
$C_{\mathsf{AE}} \leftarrow\!\!\$\ \mathcal{C}_{\mathsf{AEAD}}(|K_{\mathsf{EC}}|)$
Return $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$
```

Figure 19: Games for proof of Theorem 6.

openings, and return $((K_f, H, M), (K'_f, H', M'), C_B)$. This will be a winning tuple if $\mathcal{A}_2$'s output is a winning tuple.

(Part 3) To see that sender binding holds, suppose an attacker $\mathcal{A}_3$ in game s-BIND against $\mathsf{EC}[\mathsf{CE}]$ outputs a winning tuple $(K \parallel R, H, C, C_B)$. To decrypt correctly under DO, it must be the case that $\mathsf{Dec}(K, H, C, C_B) = (M, K_f) \neq \perp$, and $\mathsf{Enc}(K, H, M; R) = (C, C_B)$. The verification algorithm EVer is then run on input $(H, M, K \parallel R, C_B)$, computing $\mathsf{Enc}(K, H, M; R) = (C, C_B)$, followed by $\mathsf{Dec}(K, H, C, C_B) = (M', K'_f)$. Here, the success of decryption implies that these steps do not return an error, and that $K_f = K'_f$; that is to say that the same opening is recovered during both decryption and verification. Therefore, for verification to fail, it must be the case that $\mathsf{Ver}(H, M, K'_f, C_B) = 0$. As such, any winning tuple implies a winning tuple for an adversary $\mathcal{D}$ in game s-BIND against CE.

Finally, combining Lemma 1, Theorem 11, and that $\mathsf{EC}[\mathsf{CE}]$ is strongly correct implies that $\mathsf{EC}[\mathsf{CE}]$ is, moreover, SCU secure.

J Proofs from Section 7

Proof of Theorem 6.

Let $\mathcal{A}$ be an attacker in game MO-ROR against $\mathsf{CE}[\mathsf{EC}, \mathsf{AEAD}]$. We argue by a series of game hops, as shown in Figure 19. We begin by defining game G_0, which is identical to the MO-REAL game of Figure 13, with the generic Enc and Dec procedures replaced with those of $\mathsf{CE}[\mathsf{EC}, \mathsf{AEAD}]$ from Figure 14.

Next we define game G_1, which is identical to game G_0, except decryption by oracle **Dec** is performed via table lookup. In more detail, we begin with an array D, initially set to $\perp$. When **Enc** computes a ciphertext $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$ under key K_{EC} in response to some query (H, M), it sets $D[H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}}] = (M, K_{\mathsf{EC}})$. For subsequent decryption queries of the form $(H, ((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}}))$, **Dec** now returns the pair (M, K_{EC}) stored at entry $D[H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}}]$. We need not maintain a look up table for ciphertexts generated by oracle **ChalEnc**, since **Dec** only returns decryptions of ciphertexts generated via oracle **Enc**. It is easy to see that this is a purely syntactic change and the two games are functionally equivalent, so it follows that

$$\Pr[\,\mathrm{G}_0 \Rightarrow 1\,] = \Pr[\,\mathrm{G}_1 \Rightarrow 1\,]\ .$$

Next we define game G_2, which is identical to game G_1 except all ciphertexts encrypted under the AEAD scheme $\mathsf{AEAD} = (\mathsf{AEAD.Kg}, \mathsf{AEAD.enc}, \mathsf{AEAD.dec})$ are replaced with random ciphertexts of appropriate length. We claim that there exists an adversary $\mathcal{B}_1$ in the ROR-security game against AEAD such that

$$|\Pr[\,\mathrm{G}_1 \Rightarrow 1\,] - \Pr[\,\mathrm{G}_2 \Rightarrow 1\,]| \leq \mathbf{Adv}^{\mathrm{ror}}_{\mathsf{AEAD}}(\mathcal{B}_1)\ .$$

Adversary $\mathcal{B}_1$ proceeds as follows. $\mathcal{B}_1$ runs $\mathcal{A}$ as a subroutine. In response to **Enc** and **ChalEnc** queries on input (H, M), $\mathcal{B}_1$ generates $K_{\mathsf{EC}} \leftarrow\!\!\$\ \mathsf{EKg}$, computes $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$ and then queries $(B_{\mathsf{EC}}, K_{\mathsf{EC}})$ to his challenge encryption oracle, receiving C_{AE} in return. $\mathcal{B}_1$ returns ciphertext $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$ to $\mathcal{A}$, and for **Enc** queries, additionally sets $D[H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}}] = (M, K_{\mathsf{EC}})$. To simulate the **Dec** oracle, $\mathcal{B}_1$ returns $\perp$ if the header / ciphertext pair queried was not the result of a previous query to **Enc**, and returns the pair (M, K_{EC}) stored at $D[H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}}]$ otherwise. At the end of the game $\mathcal{B}_1$ outputs whatever bit $\mathcal{A}$ does. Notice that if $\mathcal{B}_1$'s encryption oracle is returning real ciphertexts as in game $\mathrm{REAL}^{\mathcal{B}_1}_{\mathsf{AEAD}}$ then this perfectly simulates game G_1, and if the oracle is returning random bit strings as in game $\mathrm{RAND}^{\mathcal{B}_1}_{\mathsf{AEAD}}$ then this perfectly simulates game G_2. It follows that

$$\begin{aligned}|\Pr[\,\mathrm{G}_1 \Rightarrow 1\,] - \Pr[\,\mathrm{G}_2 \Rightarrow 1\,]| &= |\Pr\left[\,\mathrm{REAL}^{\mathcal{B}_1}_{\mathsf{AEAD}} \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{RAND}^{\mathcal{B}_1}_{\mathsf{AEAD}} \Rightarrow 1\,\right]| \\ &= \mathbf{Adv}^{\mathrm{ror}}_{\mathsf{AEAD}}(\mathcal{B}_1)\ ,\end{aligned}$$

proving the claim. Next we define game G_3, which is identical to game G_2 except that during **ChalEnc** queries, the ciphertext / binding tag pairs produced by the encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EC}, \mathsf{DO}, \mathsf{EVer})$ are replaced with random bit strings of appropriate length. We claim that there exists an adversary $\mathcal{C}$ in the otROR game against EC such that

$$|\Pr[\,\mathrm{G}_2 \Rightarrow 1\,] - \Pr[\,\mathrm{G}_3 \Rightarrow 1\,]| \leq q_c \cdot \mathbf{Adv}^{\mathrm{ot\text{-}ror}}_{\mathsf{EC}}(\mathcal{C})\ .$$

where q_c denotes the number of **ChalEnc** queries made by $\mathcal{A}$. To see this, we define a series of hybrid games $H_0, \ldots, H_{q_c}$ where H_0 is identical to game G_2, H_{q_c} is identical to game G_3, and H_i is identical to game H_{i-1} except during the i^{th} **ChalEnc** query, the encryptment outputs are replaced with random bit strings. A standard hybrid argument implies that

$$|\Pr[\,\mathrm{G}_2 \Rightarrow 1\,] - \Pr[\,\mathrm{G}_3 \Rightarrow 1\,]| \leq \sum_{i=0}^{q_c-1} |\Pr[\,H_i \Rightarrow 1\,] - \Pr[\,H_{i+1} \Rightarrow 1\,]\ . \tag{1}$$

We now bound the gap between these game. Fix an index $i \in [0, q_c - 1]$ and let $\mathcal{C}_i$ be an attacker in the otROR game against EC. $\mathcal{C}_i$ runs $\mathcal{A}$ as a subroutine, simulating his oracles as follows. For **Enc** / **Dec** queries, $\mathcal{A}$ simulates the oracles by executing the pseudocode descriptions in game G_2 (or equivalently G_3). In response to **ChalEnc** queries, attacker $\mathcal{C}_i$ responds to the first i queries by choosing random $C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}$ of appropriate length, and returning $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$. For the $(i + 1)^{\text{th}}$ query, $\mathcal{C}_i$ queries his own encryption oracle on the query pair (H, M) to receive back

$(C_{\mathsf{EC}}, B_{\mathsf{EC}})$, and returns $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$ for random C_{AE}. For the remaining **ChalEnc** queries, $\mathcal{C}_i$ generates ciphertexts as per the pseudocode description of the oracle in game G_2. At the end of the game, adversary $\mathcal{C}_i$ outputs whatever bit $\mathcal{A}$ does.

Notice that if $\mathcal{C}_i$'s encryption oracle returns real ciphertexts as in game otROR0, then this perfectly simulates game H_i, whereas if the oracle returns random ciphertexts as in game otROR then this perfectly simulates game H_{i+1}. It follows that

$$\begin{aligned} |\Pr[\,H_i \Rightarrow 1\,] - \Pr[\,H_{i+1} \Rightarrow 1\,]| &= |\Pr\left[\,\mathrm{otROR0}_{\mathsf{EC}}^{\mathcal{C}_i} \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{otROR1}_{\mathsf{EC}}^{\mathcal{C}_i} \Rightarrow 1\,\right]| \\ &= \mathbf{Adv}_{\mathsf{EC}}^{\mathrm{ot\text{-}ror}}(\mathcal{C}_i)\ ; \end{aligned}$$

substituting this into equation 1, and defining $\mathcal{C}$ to be the attacker who chooses $i \leftarrow\!\!\$\ [0, q_c - 1]$ and runs attacker $\mathcal{C}_i$, proves the claim.

Notice that in game G_3, in response to a query (H, M), oracle **ChalEnc** always returns a random ciphertext / binding tag pair of appropriate length. Next we define game G_4 to be the same as game G_3, except we revert oracle **Enc** to generate ciphertexts C_{AE} by running $\mathsf{AEAD.enc}$ rather than by choosing random ciphertexts. A reduction to the ROR-security of AEAD analogous to that described above implies that there exists an adversary $\mathcal{B}_2$ in game ROR against AEAD such that

$$|\Pr[\,\mathrm{G}_3 \Rightarrow 1\,] - \Pr[\,\mathrm{G}_4 \Rightarrow 1\,]| \leq \mathbf{Adv}_{\mathsf{AEAD}}^{\mathrm{ror}}(\mathcal{B}_2)\ .$$

In particular, notice that in game G_4, oracle **Enc** always returns real ciphertexts. We define a final transition to game G_5 (not shown) which does away with the look up table and instead runs $\mathsf{AEAD.dec}$ on the relevant queries to **Dec**; again this is a purely syntactic change, and so

$$\Pr[\,\mathrm{G}_4 \Rightarrow 1\,] = \Pr[\,\mathrm{G}_5 \Rightarrow 1\,]\ .$$

Now game G_5 is identical to game MO-RAND. Putting this all together via a triangle inequality, it follows that there exists adversaries $\mathcal{B}$, $\mathcal{C}$ such that

$$\mathbf{Adv}_{\mathsf{CE}}^{\mathrm{mo\text{-}ror}}(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}_{\mathsf{AEAD}}^{\mathrm{ror}}(\mathcal{B}) + q_c \cdot \mathbf{Adv}_{\mathsf{EC}}^{\mathrm{ot\text{-}ror}}(\mathcal{C})\ ,$$

where $\mathcal{B}$ is the attacker who flips a coin and depending on the outcome runs either $\mathcal{B}_1$ or $\mathcal{B}_2$ as defined above, concluding the proof. ▌

Proof of Theorem 7.

We argue by a series of game hops, shown in Figure 20. We begin by defining game G_0, which is simply the MO-CTXT game of Figure 13 with the generic Enc and Dec procedures replaced with those of $\mathsf{CE}[\mathsf{EC}, \mathsf{AEAD}]$ from Figure 14. It follows that

$$\mathbf{Adv}_{\mathsf{CE}}^{\mathrm{mo\text{-}ctxt}}(\mathcal{A}) = \Pr[\,\mathrm{G}_0 \Rightarrow 1\,]\ .$$

Next we define game G_1, which is identical to game G_0, except we maintain a table D of header / ciphertext pairs which were returned from oracle **Enc**; subsequently, the decryption of such ciphertexts in oracles **Dec** and **ChalDec** is performed via table look up. Entries in the table are of the form $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}]$, and we write e.g., $D[\cdot, \cdot, B_{\mathsf{EC}}, C_{\mathsf{AE}}]$ to denote the set

$$\{(M, K_{\mathsf{EC}}) = D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}]\ :\ H \in \mathcal{H}, C_{\mathsf{EC}} \in \mathcal{C}\}\ .$$

This is a purely syntactic change, which does not affect the outcome of the game. Likewise we set a number of bad flags, but these too do not affect the outcome of the game. We additionally modify oracle **Dec** so that if the attacker $\mathcal{A}$ submits a query which decrypts correctly but which does not correspond to a previous query to **Enc** (and is thus not stored in the look up table), the win flag is set to true. This change can only increase the attacker's chance of success, and so it follows that

$$\Pr[\,\mathrm{G}_0 \Rightarrow 1\,] \leq \Pr[\,\mathrm{G}_1 \Rightarrow 1\,]\ .$$

```
$\underline{\mathrm{G}_0::}$

$K \leftarrow\!\$\ \mathsf{AEAD.Kg}$ ; $\mathsf{win} \leftarrow \mathsf{false}$
$\mathcal{A}^{\mathbf{Enc},\mathbf{Dec},\mathbf{ChalDec}}$
Return $\mathsf{win}$

$\underline{\mathbf{Enc}(H, M):}$

$K_{\mathsf{EC}} \leftarrow\!\$\ \mathsf{EKg}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow\!\$\ \mathsf{AEAD.enc}(K, B_{\mathsf{EC}}, K_{\mathsf{EC}})$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}})\}$
Return $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$

$\underline{\mathbf{Dec}(H, C, C_B):}$

$(C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
$K_{\mathsf{EC}} \leftarrow \mathsf{AEAD.dec}(K, B_{\mathsf{EC}}, C_{\mathsf{AE}})$
If $K_{\mathsf{EC}} = \perp$ then Return $\perp$
$M \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \perp$ then Return $\perp$
Return $(M, K_{\mathsf{EC}})$

$\underline{\mathbf{ChalDec}(H, C, C_B):}$

$(C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
If $(H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}}) \in \mathcal{Y}$ then
    Return $\perp$
$K_{\mathsf{EC}} \leftarrow \mathsf{AEAD.dec}(K, B_{\mathsf{EC}}, C_{\mathsf{AE}})$
If $K_{\mathsf{EC}} = \perp$ then Return $\perp$
$M \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \perp$ then Return $\perp$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$
```

```
$\underline{\mathrm{G}_1, \boxed{G_2}::}$

$K \leftarrow\!\$\ \mathsf{AEAD.Kg}$ ; $\mathsf{win} \leftarrow \mathsf{false}$
$\mathcal{A}^{\mathbf{Enc},\mathbf{Dec},\mathbf{ChalDec}}$
Return $\mathsf{win}$

$\underline{\mathbf{Enc}(H, M):}$

$K_{\mathsf{EC}} \leftarrow\!\$\ \mathsf{EKg}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow\!\$\ \mathsf{AEAD.enc}(K, B_{\mathsf{EC}}, K_{\mathsf{EC}})$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}})\}$
$D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$

$\underline{\mathbf{Dec}(H, C, C_B):}$

$(C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
If $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] \neq \perp$
    Return $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}]$
$K_{\mathsf{EC}} \leftarrow \mathsf{AEAD.dec}(K, B_{\mathsf{EC}}, C_{\mathsf{AE}})$
If $K_{\mathsf{EC}} = \perp$ then Return $\perp$
If $D[\cdot, \cdot, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = \{\perp\}$
    $\mathsf{bad}_1 \leftarrow \mathsf{true}$
    $\boxed{\text{Return } \perp}$
$M \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \perp$ then Return $\perp$
If $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = \{\perp\}$
    $\mathsf{bad}_2 \leftarrow \mathsf{true}$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$

$\underline{\mathbf{ChalDec}(H, C, C_B):}$

$(C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
If $(H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}}) \in \mathcal{Y}$ then
    Return $\perp$
$K_{\mathsf{EC}} \leftarrow \mathsf{AEAD.dec}(K, B_{\mathsf{EC}}, C_{\mathsf{AE}})$
If $K_{\mathsf{EC}} = \perp$ then Return $\perp$
If $D[\cdot, \cdot, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = \{\perp\}$
    $\mathsf{bad}_1 \leftarrow \mathsf{true}$
    $\boxed{\text{Return } \perp}$
$M \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \perp$ then Return $\perp$
If $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = \{\perp\}$
    $\mathsf{bad}_2 \leftarrow \mathsf{true}$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$
```

```
$\underline{\mathrm{G}_3::}$

$K \leftarrow\!\$\ \mathsf{AEAD.Kg}$ ; $\mathsf{win} \leftarrow \mathsf{false}$
$\mathcal{A}^{\mathbf{Enc},\mathbf{Dec},\mathbf{ChalDec}}$
Return $\mathsf{win}$

$\underline{\mathbf{Enc}(H, M):}$

$K_{\mathsf{EC}} \leftarrow\!\$\ \mathsf{EKg}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$D_{\mathsf{EC}}[C_{\mathsf{EC}}, B_{\mathsf{EC}}, K_{\mathsf{EC}}] \leftarrow (H, M)$
$C_{\mathsf{AE}} \leftarrow\!\$\ \mathsf{AEAD.enc}(K, B_{\mathsf{EC}}, K_{\mathsf{EC}})$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}})\}$
$D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$

$\underline{\mathbf{Dec}(H, C, C_B):}$

$(C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
If $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] \neq \perp$
    Return $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}]$
$K_{\mathsf{EC}} \leftarrow \mathsf{AEAD.dec}(K, B_{\mathsf{EC}}, C_{\mathsf{AE}})$
If $K_{\mathsf{EC}} = \perp$ then Return $\perp$
If $D[\cdot, \cdot, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = \{\perp\}$
    $\mathsf{bad}_1 \leftarrow \mathsf{true}$
    Return $\perp$
$M \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \perp$ then Return $\perp$
If $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = \{\perp\}$
    $\mathsf{bad}_2 \leftarrow \mathsf{true}$
    Return $\perp$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$

$\underline{\mathbf{ChalDec}(H, C, C_B):}$

$(C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
If $(H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}}) \in \mathcal{Y}$ then
    Return $\perp$
$K_{\mathsf{EC}} \leftarrow \mathsf{AEAD.dec}(K, B_{\mathsf{EC}}, C_{\mathsf{AE}})$
If $K_{\mathsf{EC}} = \perp$ then Return $\perp$
If $D[\cdot, \cdot, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = \{\perp\}$
    $\mathsf{bad}_1 \leftarrow \mathsf{true}$
    Return $\perp$
$M \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \perp$ then Return $\perp$
If $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = \{\perp\}$
    $\mathsf{bad}_2 \leftarrow \mathsf{true}$
    Return $\perp$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$
```

Figure 20: Games for proof of Theorem 7.

Next we define game G_2, which is identical to game G_1 except we change the way in which oracles **Dec** and **ChalDec** respond to queries. Namely now if the attacker submits a query to **Dec** or **ChalDec** of the form $(H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}})$ such that $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = \perp$ and $\mathsf{AEAD.dec}(K, B_{\mathsf{EC}}, C_{\mathsf{AE}}) \neq \perp$, it checks if $D[\cdot, \cdot, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = \{\perp\}$, and if so returns $\perp$. These games run identically unless the bad_1 flag is set, and so the Fundamental Lemma of Game Playing implies that

$$|\Pr[\, \mathrm{G}_1 \Rightarrow 1\,] - \Pr[\, \mathrm{G}_2 \Rightarrow 1\,]| \leq \Pr[\, \mathsf{bad}_1 \leftarrow \mathsf{true} \text{ in } \mathrm{G}_1\,]\ .$$

We bound this probability with a reduction to the CTXT security of AEAD. Let $\mathcal{B}$ be an attacker in the CTXT game against AEAD who runs $\mathcal{A}$ as a subroutine as follows. To simulate oracle **Enc** on query (H, M), $\mathcal{B}$ computes generates an encryptment key $K_{\mathsf{EC}} \leftarrow\!\!{\scriptstyle\$}\ \mathsf{EKg}$, computes $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$, queries $(B_{\mathsf{EC}}, K_{\mathsf{EC}})$ to his own encryption oracle receiving C_{AE} in return, and returns $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$ to $\mathcal{A}$. $\mathcal{B}$ maintains a look up table of queries to **Enc**. To simulate **Dec** and **ChalDec** queries for $(H, (C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$, $\mathcal{B}$ checks if $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] \neq \perp$; if so he returns the stored (M, K_{EC}) if the query was to the **Dec** oracle, and $\perp$ if the query was to **ChalDec**. If no such entry is stored, $\mathcal{B}$ submits the pair $(B_{\mathsf{EC}}, C_{\mathsf{AE}})$ to his own challenge decryption oracle. If $\perp$ is returned, he returns $\perp$ to $\mathcal{A}$, otherwise he simulates the rest of the query as per the pseudocode description. Notice that flag bad_1 being set corresponds to $\mathcal{A}$ making a query $(H, (C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$ where $(B_{\mathsf{EC}}, C_{\mathsf{AE}})$ does not correspond to an encryption query made by $\mathcal{B}$ in his game, but which nonetheless decrypts correctly. As such, this corresponds to a winning query for $\mathcal{B}$. It follows that

$$\begin{aligned} \Pr[\, \mathsf{bad}_1 \leftarrow \mathsf{true} \text{ in } \mathrm{G}_1\,] &= \Pr\left[\, \mathrm{CTXT}^{\mathcal{B}}_{\mathsf{AEAD}} \Rightarrow 1\,\right] \\ &\leq \mathbf{Adv}^{\mathrm{ctxt}}_{\mathsf{AEAD}}(\mathcal{B})\ . \end{aligned}$$

Notice that in game G_2, all encryptment keys K_{EC} recovered and input to DO during decryption oracle queries correspond to keys generated in response to **Enc** queries. As such to each query $(H, (C_{\mathsf{EC}}, B_{\mathsf{EC}}), C_{\mathsf{AE}})$ which may successfully decrypt in game G_2, there must be an entry of the form $D[\cdot, \cdot, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = (\cdot, K_{\mathsf{EC}})$. By the correctness of the encryption scheme, all table entries of that form must share the same key K_{EC}, and this will be the key which is recovered during decryption.

Next, we define a game G_3 which is identical to game G_2 except we again change the way in which **Dec**, **ChalDec** respond to queries. Now if the attacker makes a query of the form $(H, (C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$ such that $D[\cdot, \cdot, B_{\mathsf{EC}}, C_{\mathsf{AE}}] \neq \{\perp\}$, and $\mathsf{DO}(K_{\mathsf{EC}}, C_{\mathsf{EC}}, B_{\mathsf{EC}})$ does not return an error (where K_{EC} is the key underlying $(B_{\mathsf{EC}}, C_{\mathsf{AE}})$), but for which $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{EC}}] = \perp$, we return $\perp$. Notice that this restriction makes the game impossible to win, since **Dec** and **ChalDec** will reject any ciphertext not previously returned by **Enc**. As such it follows that

$$\Pr[\, \mathrm{G}_3 \Rightarrow 1\,] = 0\ .$$

These two games run identically unless the flag bad_2 is set, and so the fundamental lemma of game playing implies that

$$|\Pr[\, \mathrm{G}_2 \Rightarrow 1\,] - \Pr[\, \mathrm{G}_3 \Rightarrow 1\,]| \leq \Pr[\, \mathsf{bad}_2 = \mathsf{true} \text{ in } \mathrm{G}_2\,]\ ;$$

we now bound this probability with a reduction to the SCU security of EC.

Suppose that attacker $\mathcal{A}$ makes q **Enc** queries, and let $\mathcal{C}$ be an attacker in the SCU game against EC who proceeds as follows. $\mathcal{C}$ chooses a key $K \leftarrow\!\!{\scriptstyle\$}\ \mathsf{Kg}$, an index $i \leftarrow\!\!{\scriptstyle\$}\ [1, q]$, and runs $\mathcal{A}$ as a subroutine. $\mathcal{C}$ simulates all apart from the i^{th} **Enc** query by choosing a key $K_{\mathsf{EC}} \leftarrow\!\!{\scriptstyle\$}\ \mathsf{EKg}$, computing $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$ on the given input, setting $C_{\mathsf{AE}} \leftarrow\!\!{\scriptstyle\$}\ \mathsf{AEAD.dec}(K, B_{\mathsf{EC}}, K_{\mathsf{EC}})$ and returning ciphertext $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$. For the i^{th} query, $\mathcal{C}$ submits $\mathcal{A}$'s query (H^*, M^*) to his encryptment oracle, receiving $((C^*_{\mathsf{EC}}, B^*_{\mathsf{EC}}), K^*_{\mathsf{EC}})$ in return. $\mathcal{C}$ computes $C^*_{\mathsf{AE}} \leftarrow \mathsf{AEAD.dec}(K, B^*_{\mathsf{EC}}, K^*_{\mathsf{EC}})$

and returns $((C^*_{\mathsf{EC}}, C^*_{\mathsf{AE}}), B^*_{\mathsf{EC}})$ to $\mathcal{A}$. $\mathcal{C}$ maintains a look up table of the ciphertexts generated in response to **Enc** queries as described previously.

$\mathcal{C}$ simulates oracles **Dec** and **ChalDec** for queries $(H, (C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$ by checking if there is an entry in the look up table of the form $D[\cdot, \cdot, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = (M, K_{\mathsf{EC}})$. If not he returns $\perp$, but if so he computes $\mathcal{M} \cup \{\perp\} \ni X \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$ and returns the output to $\mathcal{A}$. If $\mathcal{A}$ ever makes a decryption query $(H, (C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$ such that $D[\cdot, \cdot, B_{\mathsf{EC}}, C_{\mathsf{AE}}] \neq \{\perp\}$ and which decrypts correctly but for which $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = \perp$, the bad_2 flag will be set. If additionally such a query has $(B_{\mathsf{EC}}, C_{\mathsf{AE}}) = (B^*_{\mathsf{EC}}, C^*_{\mathsf{AE}})$ and so corresponds to the i^{th} **Enc** query in which $\mathcal{C}$ inserted his challenge, then $\mathcal{C}$ submits the tuple (H, C_{EC}) to his challenge decryption oracle.

If such an event occurs, then by definition $\mathsf{DO}(K^*_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B^*_{\mathsf{EC}})$ does not return $\perp$, where K^*_{EC} is the key associated to $(B^*_{\mathsf{EC}}, C^*_{\mathsf{AE}})$, and that which was used in $\mathcal{C}$'s challenge. At the same time, we know that $D[H, C_{\mathsf{EC}}, B^*_{\mathsf{EC}}, C^*_{\mathsf{AE}}] = \perp$, so $(H, C_{\mathsf{EC}}) \neq (H^*, C^*_{\mathsf{EC}})$. Therefore, this constitutes a winning query for $\mathcal{C}$ in the SCU game.

Letting $(B^1_{\mathsf{EC}}, C^1_{\mathsf{AE}}), \ldots, (B^q_{\mathsf{EC}}, C^q_{\mathsf{AE}})$ denote the set of ciphertexts returned in response to the q **Enc** queries made by $\mathcal{A}$, and $\mathsf{bad}(B_{\mathsf{EC}}, C_{\mathsf{AE}})$ denote the ciphertext in the query which resulted in bad_2 being set (by a previous transition, the only ciphertexts which could cause bad_2 to be set must correspond to an **Enc** query). We write $\mathcal{C}^j$ for the attacker who inserts his challenge at the j^{th} **Enc** query. Then taking a union bound, it follows that

$$\begin{aligned}
\Pr\left[\, \mathsf{bad}_2 \leftarrow \mathsf{true} \text{ in } \mathrm{G}_2 \,\right] &\leq \sum_{j=1}^{q} \Pr\left[\, \mathsf{bad}_2 \leftarrow \mathsf{true} \text{ in } \mathrm{G}_2 \wedge \mathsf{bad}(B_{\mathsf{EC}}, C_{\mathsf{AE}}) = (B^j_{\mathsf{EC}}, C^j_{\mathsf{AE}}) \,\right] \\
&\leq \sum_{j=1}^{q} \Pr\left[\, \mathrm{SCU}^{\mathcal{C}^j}_{\mathsf{EC}} \Rightarrow 1 \,\right] \\
&= q \cdot \mathbf{Adv}^{\mathrm{scu}}_{\mathsf{EC}}(\mathcal{C}) \,.
\end{aligned}$$

Putting this altogether, it follows that

$$\mathbf{Adv}^{\text{mo-ctxt}}_{\mathsf{CE[EC,AEAD]}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{ctxt}}_{\mathsf{AEAD}}(\mathcal{B}) + q \cdot \mathbf{Adv}^{\mathrm{scu}}_{\mathsf{EC}}(\mathcal{C})$$

which concludes the proof. ∎

K An Alternate Encryptment-to-ccAEAD Transform

In this section we give a detailed description of the alternate transform for building ccAEAD from encryptment given in Section 7.3. The encryption and decryption algorithms of the transform (denoted as $\mathsf{CE[EC, f]}$) are depicted in Figure 21. The scheme uses a compression function $\mathsf{f}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$ to derive a one-time encryptment key K_{EC} from the long-term key K and a randomly-generated N. It computes the ciphertext and binding tag via $\mathsf{EC}(K_{\mathsf{EC}}, \cdot, \cdot)$, then computes an additional tag using another call to the compression function. The two uses of the compression function are domain-separated using two distinct d-bit constants (fpad and spad) XORed into the long-term key K.

Security of the compression function transform. We will begin by proving that $\mathsf{CE[EC, f]}$ achieves multi-opening real-or-random security if the encryptment scheme EC is one-time real-or-random secure and the compression function f is an RKA-PRF when keyed on its second input.

Theorem 12 *Let* EC *be an encryptment scheme,* $\mathsf{f}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$ *be a compression function and let* $\mathsf{CE[EC, f]}$ *be the ccAEAD scheme built from* EC *and* f *according to Figure 21. Let*

```
CE[EC, f].Enc(K, H, M):
    $N \leftarrow\!\!\$ \{0,1\}^n$
    $K_{\mathsf{EC}} \leftarrow \mathsf{f}(N, K \oplus \text{fpad})$
    $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
    $C_{\mathsf{AE}} \leftarrow \mathsf{f}(B_{\mathsf{EC}}, K \oplus \text{spad})$
    Return $((N, C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$

CE[EC, f].Dec(K, H, (C, C_B)):
    $(N, C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
    $C'_{\mathsf{AE}} \leftarrow \mathsf{f}(B_{\mathsf{EC}}, K \oplus \text{fpad})$
    If $C_{\mathsf{AE}} \neq C'_{\mathsf{AE}}$ then Return $\bot$
    $K_{\mathsf{EC}} \leftarrow \mathsf{f}(N, K \oplus \text{spad})$
    $M \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
    If $M = \bot$ then Return $\bot$
    Return $(M, K_{\mathsf{EC}})$
```

Figure 21: A transform $\mathsf{CE}[\mathsf{EC}, \mathsf{f}]$ for building a ccAEAD scheme from an encryptment scheme EC and a compression function $\mathsf{f}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$. The strings fpad and spad are fixed and distinct.

fpad and spad be fixed distinct d-bit strings. Then for any adversary $\mathcal{A}$ in the MO-ROR *game against* CE *making a total of q queries, of which q_c are to* **ChalEnc** *and q_e are to* **Enc***, there exists adversaries $\mathcal{B}$ and $\mathcal{C}$ such that*

$$\mathbf{Adv}^{\text{mo-ror}}_{\mathsf{CE}}(\mathcal{A}) \le 2 \cdot \mathbf{Adv}^{\oplus\text{-prf}}_{\mathsf{f}}(\mathcal{B}) + q_c \cdot \mathbf{Adv}^{\text{ot-ror}}_{\mathsf{EC}}(\mathcal{C}) + \frac{(q_e+q_c)^2}{2^n} + \frac{q^2}{2^{\mathit{blen}}} .$$

Adversaries $\mathcal{B}$ and $\mathcal{C}$ run in the same amount of time as $\mathcal{A}$. Adversary $\mathcal{B}$ makes at most $2q$ oracle queries.

Proof: This proof will use a sequence of game hops depicted in Figure 22 and Figure 23. We begin with game G_0, which is exactly $\text{MO-REAL}_{\mathsf{CE}[\mathsf{EC},\mathsf{f}]}$ except decryption oracle queries are answered via table lookup. Concretely, the **Enc** oracle keeps a table D indexed by H, C, B_{EC} triples and where the value in D for each triple is M, K_{EC}. On a decryption query the appropriate value in the table is returned. This value is guaranteed to exist because the table $\mathcal{Y}$ is checked before decryption is performed. This is only a syntactic change, so there is no change in advantage.

Game G_1 is identical to G_0 except all calls to f are replaced by calls to a random function. We use $RF(\cdot,\cdot)$ for calls to the random function, where the first input is either a zero or one. We use zero or one instead of fpad and spad to make the domain separation more explicit in the pseudocode. A standard argument gives us a reduction $\mathcal{B}_1$ to the RKA-PRF security of f, which works as follows. When $\mathcal{A}$ queries **Enc** or **ChalEnc** on (H, M), $\mathcal{B}$ samples N then queries (N, fpad) to its oracle. It uses the output of its first oracle call as the K_{EC} value to run $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$. Then it queries its oracle again with $(B_{\mathsf{EC}}, \text{spad})$ and uses the output as C_{AE}. Because fpad and spad are distinct, we see that

$$|\Pr[\, G_1 \Rightarrow 1\,] - \Pr[\, G_0 \Rightarrow 1\,]| \le \mathbf{Adv}^{\oplus\text{-prf}}_{\mathsf{f}}(\mathcal{B}_1) .$$

Game G_2 is the same as G_1 except a flag bad is set if any of the randomly sampled Ns collide. Game G_3 is the same as G_2 except we include the boxed code. Now, in the event of such a collision, the game resamples the N in question such that this is not the case. At most $(q_e + q_c)$ nonces are sampled, so by the Fundamental Lemma of code-based games,

$$|\Pr[\, G_3 \Rightarrow 1\,] - \Pr[\, G_2 \Rightarrow 1\,]| \le \Pr[\, \mathsf{bad} = \mathsf{true} \text{ in } G_3\,] \le \frac{(q_e+q_c)^2}{2^{n+1}} .$$

Game G_4 is a syntactic change to sample the K_{EC} values in **ChalEnc** as uniformly random bit strings.

Games G_5 is the same as G_4 except the outputs of EC in **ChalEnc** are replaced by uniformly random bits.

We can construct a reduction $\mathcal{C}$ to the one-time real-or-random security of EC (via a standard hybrid argument over the q_c queries made to **ChalEnc**) to get that

$$|\Pr[\,G_5 \Rightarrow 1\,] - \Pr[\,G_4 \Rightarrow 1\,]| \le q_c \cdot \mathbf{Adv}^{\text{ot-ror}}_{\mathsf{EC}}(\mathcal{C})\ .$$

In game G_6 we set a bad_1 flag to true if the B_{EC} generated in **ChalEnc** collides with any previous one (including those output by **Enc**). Game G_7 is the same as G_6 except it includes the boxed code, which re-draws a new B_{EC} value, distinct from all previous B_{EC} values, if the bad_1 flag is set. This step is necessary because in a subsequent step C_{AE} values need to be uniformly random bit strings (not outputs of RF). By the fundamental lemma of code-based games, we get

$$|\Pr[\,G_6 \Rightarrow 1\,] - \Pr[\,G_5 \Rightarrow 1\,]| \le \Pr[\,\mathsf{bad} = \mathsf{true} \text{ in } G_6\,] \le \frac{q^2}{2^{\mathsf{blen}+1}}\ .$$

In game G_8, C_{AE} values are sampled uniformly at random instead of being generated via RF. This change is syntactic. Note that in game G_8, RF is never called in **ChalEnc** and all outputs are generated uniformly at random except for N and B_{EC}, which are sampled without replacement. Thus, once we change back to sampling N and B_{EC} with replacement the distribution of **ChalEnc** outputs in game G_8 will be the same as in MO-RAND$_{\mathsf{CE[EC,f]}}$. When we replace the calls to RF in **Enc** with calls to f and undo table decryption, both oracles will be the same as in MO-RAND$_{\mathsf{CE[EC,f]}}$. Games G_9 through G_{12} do this. Since they only undo previous transitions, we will not depict them.

Game G_9 is the same as G_8 except binding tags are sampled with replacement. This undoes the transition from G_5 to G_6, so

$$|\Pr[\,G_9 \Rightarrow 1\,] - \Pr[\,G_8 \Rightarrow 1\,]| \le \frac{q^2}{2^{\mathsf{blen}+1}}\ .$$

Game G_{10} is the same as G_9, except N values are sampled with replacement. This undoes the transition from G_2 to G_3, so

$$|\Pr[\,G_{10} \Rightarrow 1\,] - \Pr[\,G_9 \Rightarrow 1\,]| \le \frac{(q_e + q_c)^2}{2^{n+1}}\ .$$

Game G_{11} replaces calls to RF in **Enc** with keyed calls to f. We can build an RKA-PRF adversary $\mathcal{B}_2$ so that

$$|\Pr[\,G_{11} \Rightarrow 1\,] - \Pr[\,G_{10} \Rightarrow 1\,]| \le \mathbf{Adv}^{\oplus\text{-prf}}_{\mathsf{f}}(\mathcal{B}_2)\ .$$

Game G_{12} undoes table decryption. This change is syntactic, so

$$\Pr[\,G_{12} \Rightarrow 1\,] = \Pr[\,G_{11} \Rightarrow 1\,]\ .$$

We can construct an adversary $\mathcal{B}$, which is an elementary wrapper around $\mathcal{B}_1$ and $\mathcal{B}_2$ and flips a random bit to decide which of the two to run, to replace the PRF advantage terms for $\mathcal{B}_1$ and $\mathcal{B}_2$ with a single term $\mathbf{Adv}^{\oplus\text{-prf}}_{\mathsf{f}}(\mathcal{B})$. Summing the upper bounds on the individual game transitions yields the bound. ▮

```
$\underline{\mathrm{G}_0}$:
$K \leftarrow\!\$ \{0,1\}^d$
Return $\mathcal{A}^{\mathbf{Enc},\mathbf{Dec},\mathbf{ChalDec}}$

$\underline{\mathbf{Enc}(H,M)}$:
$N \leftarrow\!\$ \{0,1\}^n$
$K_{\mathsf{EC}} \leftarrow \mathsf{f}(N, K \oplus \mathrm{fpad})$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow \mathsf{f}(B_{\mathsf{EC}}, K \oplus \mathrm{spad})$
$C \leftarrow N \,\|\, C_{\mathsf{EC}} \,\|\, C_{\mathsf{AE}}$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, B_{\mathsf{EC}})\}$
$D[H, C, B_{\mathsf{EC}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $(C, B_{\mathsf{EC}})$

$\underline{\mathbf{Dec}(H,C,C_B)}$:
If $(H, C, C_B) \notin \mathcal{Y}$ then Return $\bot$
Return $D[H, C, B_{\mathsf{EC}}]$

$\underline{\mathbf{ChalEnc}(H,M)}$:
$N \leftarrow\!\$ \{0,1\}^n$
$K_{\mathsf{EC}} \leftarrow \mathsf{f}(N, K \oplus \mathrm{fpad})$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow \mathsf{f}(B_{\mathsf{EC}}, K \oplus \mathrm{spad})$
$C \leftarrow N \,\|\, C_{\mathsf{EC}} \,\|\, C_{\mathsf{AE}}$
Return $(C, B_{\mathsf{EC}})$
```

```
$\underline{\mathrm{G}_1}$:
Return $\mathcal{A}^{\mathbf{Enc},\mathbf{Dec},\mathbf{ChalDec}}$

$\underline{\mathbf{Enc}(H,M)}$:
$N \leftarrow\!\$ \{0,1\}^n$
$K_{\mathsf{EC}} \leftarrow RF(0, N)$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
$C \leftarrow N \,\|\, C_{\mathsf{EC}} \,\|\, C_{\mathsf{AE}}$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, B_{\mathsf{EC}})\}$
$D[H, C, B_{\mathsf{EC}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $(C, B_{\mathsf{EC}})$

$\underline{\mathbf{Dec}(H,C,C_B)}$:
If $(H, C, C_B) \notin \mathcal{Y}$ then Return $\bot$
Return $D[H, C, B_{\mathsf{EC}}]$

$\underline{\mathbf{ChalEnc}(H,M)}$:
$N \leftarrow\!\$ \{0,1\}^n$
$K_{\mathsf{EC}} \leftarrow RF(0, N)$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
$C \leftarrow N \,\|\, C_{\mathsf{EC}} \,\|\, C_{\mathsf{AE}}$
Return $(C, B_{\mathsf{EC}})$

$\underline{RF(b,X)}$:
If $R_b[X] \neq \bot$ then Return $R_b[X]$
$R_b[X] \leftarrow\!\$ \{0,1\}^n$
Return $R_b[X]$
```

```
$\underline{\mathrm{G}_2, \boxed{G_3}}$:
Return $\mathcal{A}^{\mathbf{Enc},\mathbf{Dec},\mathbf{ChalDec}}$

$\underline{\mathbf{Enc}(H,M)}$:
$N \leftarrow\!\$ \{0,1\}^n$
If $N \in \mathcal{IV}$ then
    $\mathsf{bad} \leftarrow \mathsf{true}$
    $\boxed{N \leftarrow\!\$ \{0,1\}^n \setminus \mathcal{IV}}$
$\mathcal{IV} \leftarrow \mathcal{IV} \cup N$
$K_{\mathsf{EC}} \leftarrow RF(0, N)$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
$C \leftarrow N \,\|\, C_{\mathsf{EC}} \,\|\, C_{\mathsf{AE}}$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, B_{\mathsf{EC}})\}$
$D[H, C, B_{\mathsf{EC}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $(C, B_{\mathsf{EC}})$

$\underline{\mathbf{Dec}(H,C,C_B)}$:
If $(H, C, C_B) \notin \mathcal{Y}$ then Return $\bot$
Return $D[H, C, B_{\mathsf{EC}}]$

$\underline{\mathbf{ChalEnc}(H,M)}$:
$N \leftarrow\!\$ \{0,1\}^n$
If $N \in \mathcal{IV}$ then
    $\mathsf{bad} \leftarrow \mathsf{true}$
    $\boxed{N \leftarrow\!\$ \{0,1\}^n \setminus \mathcal{IV}}$
$\mathcal{IV} \leftarrow \mathcal{IV} \cup N$
$K_{\mathsf{EC}} \leftarrow RF(0, N)$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
$C \leftarrow N \,\|\, C_{\mathsf{EC}} \,\|\, C_{\mathsf{AE}}$
Return $(C, B_{\mathsf{EC}})$

$\underline{RF(b,X)}$:
If $R_b[X] \neq \bot$ then Return $R_b[X]$
$R_b[X] \leftarrow\!\$ \{0,1\}^n$
Return $R_b[X]$
```

```
$\underline{\mathrm{G}_4}$:
Return $\mathcal{A}^{\mathbf{Enc},\mathbf{Dec},\mathbf{ChalDec}}$

$\underline{\mathbf{Enc}(H,M)}$:
$N \leftarrow\!\$ \{0,1\}^n$
If $N \in \mathcal{IV}$ then
    $\mathsf{bad} \leftarrow \mathsf{true}$
    $N \leftarrow\!\$ \{0,1\}^n \setminus \mathcal{IV}$
$\mathcal{IV} \leftarrow \mathcal{IV} \cup N$
$K_{\mathsf{EC}} \leftarrow RF(0, N)$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
$C \leftarrow N \,\|\, C_{\mathsf{EC}} \,\|\, C_{\mathsf{AE}}$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, B_{\mathsf{EC}})\}$
$D[H, C, B_{\mathsf{EC}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $(C, B_{\mathsf{EC}})$

$\underline{\mathbf{Dec}(H,C,C_B)}$:
If $(H, C, C_B) \notin \mathcal{Y}$ then Return $\bot$
Return $D[H, C, B_{\mathsf{EC}}]$

$\underline{\mathbf{ChalEnc}(H,M)}$:
$N \leftarrow\!\$ \{0,1\}^n$
If $N \in \mathcal{IV}$ then
    $\mathsf{bad} \leftarrow \mathsf{true}$
    $\boxed{N \leftarrow\!\$ \{0,1\}^n \setminus \mathcal{IV}}$
$\mathcal{IV} \leftarrow \mathcal{IV} \cup N$
$K_{\mathsf{EC}} \leftarrow\!\$ \mathcal{K}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
$C \leftarrow N \,\|\, C_{\mathsf{EC}} \,\|\, C_{\mathsf{AE}}$
Return $(C, B_{\mathsf{EC}})$

$\underline{RF(b,X)}$:
If $R_b[X] \neq \bot$ then Return $R_b[X]$
$R_b[X] \leftarrow\!\$ \{0,1\}^n$
Return $R_b[X]$
```

Figure 22: Games for the proof of Theorem 12.

```
$\underline{G_5}$:
Return $\mathcal{A}^{\mathbf{Enc},\mathbf{Dec},\mathbf{ChalDec}}$

$\underline{\mathbf{Enc}(H, M)}$:
$N \leftarrow\!\$ \{0,1\}^n$
If $N \in \mathcal{IV}$ then
    $\mathsf{bad} \leftarrow \mathsf{true}$
    $N \leftarrow\!\$ \{0,1\}^n \setminus \mathcal{IV}$
$\mathcal{IV} \leftarrow \mathcal{IV} \cup N$
$K_{\mathsf{EC}} \leftarrow RF(0, N)$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
$C \leftarrow N \,\|\, C_{\mathsf{EC}} \,\|\, C_{\mathsf{AE}}$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, B_{\mathsf{EC}})\}$
$D[H, C, B_{\mathsf{EC}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $(C, B_{\mathsf{EC}})$

$\underline{\mathbf{Dec}(H, C, C_B)}$:
If $(H, C, C_B) \notin \mathcal{Y}$ then Return $\bot$
Return $D[H, C, B_{\mathsf{EC}}]$

$\underline{\mathbf{ChalEnc}(H, M)}$:
$N \leftarrow\!\$ \{0,1\}^n$
If $N \in \mathcal{IV}$ then
    $\mathsf{bad} \leftarrow \mathsf{true}$
    $N \leftarrow\!\$ \{0,1\}^n \setminus \mathcal{IV}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow\!\$ \mathcal{C}(|M|) \times \{0,1\}^{\mathsf{blen}}$
$C_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
$C \leftarrow N \,\|\, C_{\mathsf{EC}} \,\|\, C_{\mathsf{AE}}$
Return $(C, B_{\mathsf{EC}})$

$\underline{RF(b, X)}$:
If $R_b[X] \neq \bot$ then Return $R_b[X]$
$R_b[X] \leftarrow\!\$ \{0,1\}^n$
Return $R_b[X]$
```

```
$\underline{G_6, , \boxed{G_7}}$:
Return $\mathcal{A}^{\mathbf{Enc},\mathbf{Dec},\mathbf{ChalDec}}$

$\underline{\mathbf{Enc}(H, M)}$:
$N \leftarrow\!\$ \{0,1\}^n$
If $N \in \mathcal{IV}$ then
    $\mathsf{bad} \leftarrow \mathsf{true}$
    $N \leftarrow\!\$ \{0,1\}^n \setminus \mathcal{IV}$
$\mathcal{IV} \leftarrow \mathcal{IV} \cup N$
$K_{\mathsf{EC}} \leftarrow RF(0, N)$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$\mathcal{B} \leftarrow \mathcal{B} \cup B_{\mathsf{EC}}$
$C_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
$C \leftarrow N \,\|\, C_{\mathsf{EC}} \,\|\, C_{\mathsf{AE}}$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, B_{\mathsf{EC}})\}$
$D[H, C, B_{\mathsf{EC}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $(C, B_{\mathsf{EC}})$

$\underline{\mathbf{Dec}(H, C, C_B)}$:
If $(H, C, C_B) \notin \mathcal{Y}$ then Return $\bot$
Return $D[H, C, B_{\mathsf{EC}}]$

$\underline{\mathbf{ChalEnc}(H, M)}$:
$N \leftarrow\!\$ \{0,1\}^n$
If $N \in \mathcal{IV}$ then
    $\mathsf{bad} \leftarrow \mathsf{true}$
    $N \leftarrow\!\$ \{0,1\}^n \setminus \mathcal{IV}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow\!\$ \mathcal{C}(|M|) \times \{0,1\}^{\mathsf{blen}}$
If $B_{\mathsf{EC}} \in \mathcal{B}$ then
    $\mathsf{bad}_1 \leftarrow \mathsf{true}$
    $\boxed{B_{\mathsf{EC}} \leftarrow\!\$ \{0,1\}^{\mathsf{blen}} \setminus \mathcal{B}}$
$\mathcal{B} \leftarrow \mathcal{B} \cup B_{\mathsf{EC}}$
$C_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
$C \leftarrow N \,\|\, C_{\mathsf{EC}} \,\|\, C_{\mathsf{AE}}$
Return $(C, B_{\mathsf{EC}})$

$\underline{RF(b, X)}$:
If $R_b[X] \neq \bot$ then Return $R_b[X]$
$R_b[X] \leftarrow\!\$ \{0,1\}^n$
Return $R_b[X]$
```

```
$\underline{G_8}$:
Return $\mathcal{A}^{\mathbf{Enc},\mathbf{Dec},\mathbf{ChalDec}}$

$\underline{\mathbf{Enc}(H, M)}$:
$N \leftarrow\!\$ \{0,1\}^n$
If $N \in \mathcal{IV}$ then
    $\mathsf{bad} \leftarrow \mathsf{true}$
    $N \leftarrow\!\$ \{0,1\}^n \setminus \mathcal{IV}$
$\mathcal{IV} \leftarrow \mathcal{IV} \cup N$
$K_{\mathsf{EC}} \leftarrow RF(0, N)$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$\mathcal{B} \leftarrow \mathcal{B} \cup B_{\mathsf{EC}}$
$C_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
$C \leftarrow N \,\|\, C_{\mathsf{EC}} \,\|\, C_{\mathsf{AE}}$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, B_{\mathsf{EC}})\}$
$D[H, C, B_{\mathsf{EC}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $(C, B_{\mathsf{EC}})$

$\underline{\mathbf{Dec}(H, C, C_B)}$:
If $(H, C, C_B) \notin \mathcal{Y}$ then Return $\bot$
Return $D[H, C, B_{\mathsf{EC}}]$

$\underline{\mathbf{ChalEnc}(H, M)}$:
$N \leftarrow\!\$ \{0,1\}^n$
If $N \in \mathcal{IV}$ then
    $\mathsf{bad} \leftarrow \mathsf{true}$
    $N \leftarrow\!\$ \{0,1\}^n \setminus \mathcal{IV}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow\!\$ \mathcal{C}(|M|) \times \{0,1\}^{\mathsf{blen}}$
If $B_{\mathsf{EC}} \in \mathcal{B}$ then
    $\mathsf{bad}_1 \leftarrow \mathsf{true}$
    $B_{\mathsf{EC}} \leftarrow\!\$ \{0,1\}^{\mathsf{blen}} \setminus \mathcal{B}$
$\mathcal{B} \leftarrow \mathcal{B} \cup B_{\mathsf{EC}}$
$C_{\mathsf{AE}} \leftarrow\!\$ \{0,1\}^n$
$C \leftarrow N \,\|\, C_{\mathsf{EC}} \,\|\, C_{\mathsf{AE}}$
Return $(C, B_{\mathsf{EC}})$

$\underline{RF(b, X)}$:
If $R_b[X] \neq \bot$ then Return $R_b[X]$
$R_b[X] \leftarrow\!\$ \{0,1\}^n$
Return $R_b[X]$
```

Figure 23: Further games for the proof of Theorem 12.

Next we bound the MO-CTXT advantage of any adversary against $\mathsf{CE}[\mathsf{EC},\mathsf{f}]$, via a reduction to the RKA-PRF security of the compression function, and the SCU security of the encryptment scheme.

Theorem 13 *Let* EC *be an encryptment scheme,* $\mathsf{f}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$ *be a compression function and let* $\mathsf{CE}[\mathsf{EC},\mathsf{f}]$ *be the ccAEAD scheme built from* EC *and* f *according to Figure 21. Let fpad and spad be fixed distinct d-bit strings. Then for any adversary* $\mathcal{A}$ *in the* MO-CTXT *game against* CE *making a total of q queries, of which* q_e *are to* **Enc***, there exists adversaries* $\mathcal{B}$ *and* $\mathcal{C}$ *such that*

$$\mathbf{Adv}^{\text{mo-ctxt}}_{\mathsf{CE}}(\mathcal{A}) \leq \mathbf{Adv}^{\oplus\text{-prf}}_{\mathsf{f}}(\mathcal{B}) + q_e \cdot \mathbf{Adv}^{\text{scu}}_{\mathsf{EC}}(\mathcal{C}) + \mathbf{Adv}^{\text{s-bind}}_{\mathsf{EC}}(\mathcal{D}) + \mathbf{Adv}^{\text{sr-bind}}_{\mathsf{EC}}(\mathcal{E}) + \frac{2q^2+q}{2^n} \ .$$

Adversaries $\mathcal{B}$ *and* $\mathcal{C}$ *run in the same amount of time as* $\mathcal{A}$ *plus a* $\mathcal{O}(q)$ *overhead, and adversary* $\mathcal{B}$ *makes at most q queries.*

Proof: This proof will use a sequence of game hops. We begin with game G_0, which is a rewriting of game MO-CTXT$_{\mathsf{CE}[\mathsf{EC},\mathsf{f}]}$ except with the explicit encryption and decryption procedures of $\mathsf{CE}[\mathsf{EC},\mathsf{f}]$. Next we move to game G_1, in which decryption is done via table lookup in **Dec** and **ChalDec**. The table is indexed with $H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}$ tuples and the value of each tuple is the M, K_{EC} pair obtained during the query to **Enc**. Additionally, the win flag can be set in **Dec** if a ciphertext not in the table decrypts correctly. This increases the adversary's advantage, so $\Pr[\,G_1 \Rightarrow \mathsf{true}\,] \geq \Pr[\,G_0 \Rightarrow \mathsf{true}\,]$.

Game G_2 is the same as G_1 except all calls to f have been replaced by a lazy-sampled random function R. A standard argument lets us build a reduction $\mathcal{B}$ so that

$$|\Pr[\,G_2 \Rightarrow \mathsf{true}\,] - \Pr[\,G_1 \Rightarrow \mathsf{true}\,]| \leq \mathbf{Adv}^{\oplus\text{-prf}}_{\mathsf{f}}(\mathcal{B}) \ .$$

The reduction $\mathcal{B}$ is nearly identical to the reduction $\mathcal{B}_1$ described in the proof of Theorem 12, so we elide the details. Here, as in the previous proof, we use zero and one for the first input to the random function to make the domain separation explicit.

Game G_3 is the same as G_2 except Ns are sampled without replacement. By the fundamental lemma of code-based games,

$$|\Pr[\,G_3 \Rightarrow \mathsf{true}\,] - \Pr[\,G_2 \Rightarrow \mathsf{true}\,]| \leq \frac{q_e^2}{2^n} \ .$$

In game G_4, winning queries having the same N and B_{EC} as a previous output of **Enc** are disallowed. Concretely, on a query to **Dec** or **ChalDec** having nonce N and binding tag B_{EC}, the game looks to see if any entries in the table D are of the form $D[\cdot, N, \cdot, B_{\mathsf{EC}}, \cdot]$. Since a winning query of this form will also have the same K_{EC} as the one generated during the corresponding **Enc** call, any such query will also be a valid forgery against EC in game SCU. Note that disallowing repeat nonces, as ensured by the previous transition, is necessary here because the distribution of K_{EC} values in game G_4 must be the same as in SCU. Thus, we can build an adversary $\mathcal{C}$ using a hybrid argument over the q_e queries made to **Enc** to get that

$$|\Pr[\,G_4 \Rightarrow \mathsf{true}\,] - \Pr[\,G_3 \Rightarrow \mathsf{true}\,]| \leq q_e \cdot \mathbf{Adv}^{\text{scu}}_{\mathsf{EC}}(\mathcal{C}) \ .$$

In game G_5 winning queries having B_{EC} values not output by **Enc** are disallowed. Here is where the domain separation of calls to f are important—without it, an adversary may try to forge by using a previously seen C_{AE} value as a key for encryptment, but domain separation prevents this. The only queries which win in G_5 and not G_6 are those for which the adversary has guessed the correct C_{AE} value for a new B_{EC}. Each query only has a $1/2^n$ probability of guessing correctly, so by a union bound

$$|\Pr[\, G_5 \Rightarrow \mathsf{true} \,] - \Pr[\, G_4 \Rightarrow \mathsf{true} \,]| \le \frac{q}{2^n} \; .$$

(Note that this step is essentially identical to the information-theoretic part of the standard security bound for MACs built from PRFs.)

In game G_6, collisions in the output of $RF(0,\cdot)$ are disallowed: if the value sampled in $RF(0,\cdot)$ has been seen previously, a bad flag is set and the output is resampled from the set of unseen values for the bit 0. Here we prevent two different nonces from resulting in the same key. To see why this is important, take two nonces $N_0 \neq N_1$ so that $RF(0, N_0) = RF(0, N_1)$. Then if $(H, N^0, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}})$ is a valid ciphertext, $(H, N^1, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}})$ will be as well, so the adversary can forge. By the fundamental lemma of code-based games,

$$|\Pr[\, G_6 \Rightarrow \mathsf{true} \,] - \Pr[\, G_5 \Rightarrow \mathsf{true} \,]| \le \frac{q^2}{2^n} \; .$$

By a previous transition, the only way to win in game G_6 is to use a previously-seen B_{EC} value with a different N. A different N results in a different key in this game. By s-BIND security, any winning query must also verify correctly, so in G_6 any winning query must also break sr-BIND (as different (K_{EC}, H, M) tuples result in the same binding tag). Thus,

$$\Pr[\, G_6 \Rightarrow \mathsf{true} \,] \le \mathbf{Adv}^{\text{s-bind}}_{\mathsf{EC}}(\mathcal{D}) + \mathbf{Adv}^{\text{sr-bind}}_{\mathsf{EC}}(\mathcal{E}) \; .$$

Summing the upper-bounds for all the game hops (and noting that $q_e \le q \Rightarrow q_e^2 \le q^2$, so we can combine the G_3 and G_6 terms) yields the bound. ▮

We omit proofs for the sr-BIND and s-BIND security of $\mathsf{CE}[\mathsf{EC}, \mathsf{f}]$; the transform inherits these properties directly from EC.

```
$\underline{\mathrm{G}_0}$:
$K \leftarrow\!\!\$\ \{0,1\}^d$ ; $\mathsf{win} \leftarrow \mathsf{false}$
$\mathcal{A}^{\mathbf{Enc},\mathbf{Dec},\mathbf{ChalDec}}$
Return $\mathsf{win}$

$\underline{\mathbf{Enc}(H, M)}$:
$N \leftarrow\!\!\$\ \{0,1\}^n$
$K_{\mathsf{EC}} \leftarrow \mathsf{f}(N, K \oplus \mathrm{fpad})$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow \mathsf{f}(B_{\mathsf{EC}}, K \oplus \mathrm{spad})$
$C \leftarrow N \,\|\, C_{\mathsf{EC}} \,\|\, C_{\mathsf{AE}}$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, B_{\mathsf{EC}})\}$
Return $(C, B_{\mathsf{EC}})$

$\underline{\mathbf{Dec}(H, C, C_B)}$:
$(N, C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
$C'_{\mathsf{AE}} \leftarrow \mathsf{f}(B_{\mathsf{EC}}, K \oplus \mathrm{spad})$
If $C'_{\mathsf{AE}} \neq C_{\mathsf{AE}}$ then Return $\bot$
$K_{\mathsf{EC}} \leftarrow \mathsf{f}(N, K \oplus \mathrm{fpad})$
$M \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \bot$ then Return $\bot$
Return $(M, K_{\mathsf{EC}})$

$\underline{\mathbf{ChalDec}(H, C, C_B)}$:
$(N, C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
If $(H, C, B_{\mathsf{EC}}) \in \mathcal{Y}$ then
    Return $\bot$
$C'_{\mathsf{AE}} \leftarrow \mathsf{f}(B_{\mathsf{EC}}, K \oplus \mathrm{spad})$
If $C'_{\mathsf{AE}} \neq C_{\mathsf{AE}}$ then Return $\bot$
$K_{\mathsf{EC}} \leftarrow \mathsf{f}(N, K \oplus \mathrm{fpad})$
$M \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \bot$ then Return $\bot$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$
```

```
$\underline{\mathrm{G}_1}$:
$K \leftarrow\!\!\$\ \{0,1\}^d$ ; $\mathsf{win} \leftarrow \mathsf{false}$
$\mathcal{A}^{\mathbf{Enc},\mathbf{Dec},\mathbf{ChalDec}}$
Return $\mathsf{win}$

$\underline{\mathbf{Enc}(H, M)}$:
$N \leftarrow\!\!\$\ \{0,1\}^n$
$K_{\mathsf{EC}} \leftarrow \mathsf{f}(N, K \oplus \mathrm{fpad})$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow \mathsf{f}(B_{\mathsf{EC}}, K \oplus \mathrm{spad})$
$C \leftarrow N \,\|\, C_{\mathsf{EC}} \,\|\, C_{\mathsf{AE}}$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, B_{\mathsf{EC}})\}$
$D[H, N, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $(C, B_{\mathsf{EC}})$

$\underline{\mathbf{Dec}(H, C, C_B)}$:
$(N, C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
$V \leftarrow D[H, N, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}]$
If $V \neq \bot$:
    $(M, K_{\mathsf{EC}}) \leftarrow V$
    Return $(M, K_{\mathsf{EC}})$
$C'_{\mathsf{AE}} \leftarrow \mathsf{f}(B_{\mathsf{EC}}, K \oplus \mathrm{spad})$
If $C'_{\mathsf{AE}} \neq C_{\mathsf{AE}}$ then Return $\bot$
$K_{\mathsf{EC}} \leftarrow \mathsf{f}(N, K \oplus \mathrm{fpad})$
$M \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \bot$ then Return $\bot$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$

$\underline{\mathbf{ChalDec}(H, C, C_B)}$:
$(N, C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
If $(H, C, B_{\mathsf{EC}}) \in \mathcal{Y}$ then
    Return $\bot$
$C'_{\mathsf{AE}} \leftarrow \mathsf{f}(B_{\mathsf{EC}}, K \oplus \mathrm{spad})$
If $C'_{\mathsf{AE}} \neq C_{\mathsf{AE}}$ then Return $\bot$
$K_{\mathsf{EC}} \leftarrow \mathsf{f}(N, K \oplus \mathrm{fpad})$
$M \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \bot$ then Return $\bot$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$
```

```
$\underline{\mathrm{G}_2}$:
$\mathsf{win} \leftarrow \mathsf{false}$
$\mathcal{A}^{\mathbf{Enc},\mathbf{Dec},\mathbf{ChalDec}}$
Return $\mathsf{win}$

$\underline{\mathbf{Enc}(H, M)}$:
$N \leftarrow\!\!\$\ \{0,1\}^n$
$K_{\mathsf{EC}} \leftarrow RF(0, N)$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
$C \leftarrow N \,\|\, C_{\mathsf{EC}} \,\|\, C_{\mathsf{AE}}$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, B_{\mathsf{EC}})\}$
$D[H, N, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $(C, B_{\mathsf{EC}})$

$\underline{\mathbf{Dec}(H, C, C_B)}$:
$(N, C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
$V \leftarrow D[H, N, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}]$
If $V \neq \bot$:
    $(M, K_{\mathsf{EC}}) \leftarrow V$
    Return $(M, K_{\mathsf{EC}})$
$C'_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
If $C'_{\mathsf{AE}} \neq C_{\mathsf{AE}}$ then Return $\bot$
$K_{\mathsf{EC}} \leftarrow RF(0, N)$
$M \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \bot$ then Return $\bot$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$

$\underline{\mathbf{ChalDec}(H, C, C_B)}$:
$(N, C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
If $(H, C, B_{\mathsf{EC}}) \in \mathcal{Y}$ then
    Return $\bot$
$C'_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
If $C'_{\mathsf{AE}} \neq C_{\mathsf{AE}}$ then Return $\bot$
$K_{\mathsf{EC}} \leftarrow RF(0, N)$
$M \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \bot$ then Return $\bot$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$

$\underline{RF(b, X)}$:
If $R_b[X] \neq \bot$:
    Return $R_b[X]$
$Y \leftarrow\!\!\$\ \{0,1\}^n$
$R_b[X] \leftarrow Y$
```

```
$\underline{\mathrm{G}_3}$:
$\mathsf{win} \leftarrow \mathsf{false}$
$\mathcal{A}^{\mathbf{Enc},\mathbf{Dec},\mathbf{ChalDec}}$
Return $\mathsf{win}$

$\underline{\mathbf{Enc}(H, M)}$:
$N \leftarrow\!\!\$\ \{0,1\}^n$
If $N \in \mathcal{IV}$ then
    $\mathsf{bad} \leftarrow \mathsf{true}$
    $N \leftarrow\!\!\$\ \{0,1\}^n \setminus \mathcal{IV}$
$\mathcal{IV} \leftarrow \mathcal{IV} \cup N$
$K_{\mathsf{EC}} \leftarrow RF(0, N)$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
$C \leftarrow N \,\|\, C_{\mathsf{EC}} \,\|\, C_{\mathsf{AE}}$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, B_{\mathsf{EC}})\}$
$D[H, N, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $(C, B_{\mathsf{EC}})$

$\underline{\mathbf{Dec}(H, C, C_B)}$:
$(N, C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
$V \leftarrow D[H, N, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}]$
If $V \neq \bot$:
    $(M, K_{\mathsf{EC}}) \leftarrow V$
    Return $(M, K_{\mathsf{EC}})$
If $D[\cdot, N, \cdot, B_{\mathsf{EC}}, \cdot] \neq \bot$ then
    Return $\bot$
$C'_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
If $C'_{\mathsf{AE}} \neq C_{\mathsf{AE}}$ then Return $\bot$
$K_{\mathsf{EC}} \leftarrow RF(0, N)$
$M \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \bot$ then Return $\bot$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$

$\underline{\mathbf{ChalDec}(H, C, C_B)}$:
$(N, C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
If $(H, C, B_{\mathsf{EC}}) \in \mathcal{Y}$ then
    Return $\bot$
$C'_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
If $C'_{\mathsf{AE}} \neq C_{\mathsf{AE}}$ then Return $\bot$
$K_{\mathsf{EC}} \leftarrow RF(0, N)$
$M \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \bot$ then Return $\bot$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$

$\underline{RF(b, X)}$:
If $R_b[X] \neq \bot$:
    Return $R_b[X]$
$Y \leftarrow\!\!\$\ \{0,1\}^n$
$R_b[X] \leftarrow Y$
```

Figure 24: Games for proof of Theorem 13.

```
$\underline{\mathrm{G}_4:}$

win ← false
$\mathcal{A}^{\mathbf{Enc},\mathbf{Dec},\mathbf{ChalDec}}$
Return win

$\underline{\mathbf{Enc}(H, M):}$

$N \leftarrow\$ \{0,1\}^n$
If $N \in \mathcal{IV}$ then
    bad ← true
    $N \leftarrow\$ \{0,1\}^n \setminus \mathcal{IV}$
$\mathcal{IV} \leftarrow \mathcal{IV} \cup N$
$K_{\mathsf{EC}} \leftarrow RF(0, N)$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
$C \leftarrow N \parallel C_{\mathsf{EC}} \parallel C_{\mathsf{AE}}$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, B_{\mathsf{EC}})\}$
$D[H, N, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $(C, B_{\mathsf{EC}})$

$\underline{\mathbf{Dec}(H, C, C_B):}$

$(N, C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
$V \leftarrow D[H, N, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}]$
If $V \neq \perp$:
    $(M, K_{\mathsf{EC}}) \leftarrow V$
    Return $(M, K_{\mathsf{EC}})$
If $D[\cdot, N, \cdot, B_{\mathsf{EC}}, \cdot] \neq \perp$ then
    Return $\perp$
$C'_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
If $C'_{\mathsf{AE}} \neq C_{\mathsf{AE}}$ then Return $\perp$
$K_{\mathsf{EC}} \leftarrow RF(0, N)$
$M \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \perp$ then Return $\perp$
win ← true
Return $(M, K_{\mathsf{EC}})$

$\underline{\mathbf{ChalDec}(H, C, C_B):}$

$(N, C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
If $(H, C, B_{\mathsf{EC}}) \in \mathcal{Y}$ then
    Return $\perp$
If $D[\cdot, N, \cdot, B_{\mathsf{EC}}, \cdot] \neq \perp$ then
    Return $\perp$
$C'_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
If $C'_{\mathsf{AE}} \neq C_{\mathsf{AE}}$ then Return $\perp$
$K_{\mathsf{EC}} \leftarrow RF(0, N)$
$M \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \perp$ then Return $\perp$
win ← true
Return $(M, K_{\mathsf{EC}})$

$\underline{RF(b, X):}$

If $R_b[X] \neq \perp$:
    Return $R_b[X]$
$Y \leftarrow\$ \{0,1\}^n$
$R_b[X] \leftarrow Y$
```

```
$\underline{\mathrm{G}_5:}$

win ← false
$\mathcal{A}^{\mathbf{Enc},\mathbf{Dec},\mathbf{ChalDec}}$
Return win

$\underline{\mathbf{Enc}(H, M):}$

$N \leftarrow\$ \{0,1\}^n$
If $N \in \mathcal{IV}$ then
    bad ← true
    $N \leftarrow\$ \{0,1\}^n \setminus \mathcal{IV}$
$\mathcal{IV} \leftarrow \mathcal{IV} \cup N$
$K_{\mathsf{EC}} \leftarrow RF(0, N)$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
$C \leftarrow N \parallel C_{\mathsf{EC}} \parallel C_{\mathsf{AE}}$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, B_{\mathsf{EC}})\}$
$D[H, N, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $(C, B_{\mathsf{EC}})$

$\underline{\mathbf{Dec}(H, C, C_B):}$

$(N, C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
$V \leftarrow D[H, N, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}]$
If $V \neq \perp$:
    $(M, K_{\mathsf{EC}}) \leftarrow V$
    Return $(M, K_{\mathsf{EC}})$
If $D[\cdot, N, \cdot, B_{\mathsf{EC}}, \cdot] \neq \perp$ then
    Return $\perp$
If $D[\cdot, \cdot, \cdot, B_{\mathsf{EC}}, \cdot] = \perp$ then
    Return $\perp$
$C'_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
If $C'_{\mathsf{AE}} \neq C_{\mathsf{AE}}$ then Return $\perp$
$K_{\mathsf{EC}} \leftarrow RF(0, N)$
$M \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \perp$ then Return $\perp$
win ← true
Return $(M, K_{\mathsf{EC}})$

$\underline{\mathbf{ChalDec}(H, C, C_B):}$

$(N, C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
If $(H, C, B_{\mathsf{EC}}) \in \mathcal{Y}$ then
    Return $\perp$
If $D[\cdot, N, \cdot, B_{\mathsf{EC}}, \cdot] \neq \perp$ then
    Return $\perp$
If $D[\cdot, \cdot, \cdot, B_{\mathsf{EC}}, \cdot] = \perp$ then
    Return $\perp$
$C'_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
If $C'_{\mathsf{AE}} \neq C_{\mathsf{AE}}$ then Return $\perp$
$K_{\mathsf{EC}} \leftarrow RF(0, N)$
$M \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \perp$ then Return $\perp$
win ← true
Return $(M, K_{\mathsf{EC}})$

$\underline{RF(b, X):}$

If $R_b[X] \neq \perp$:
    Return $R_b[X]$
$Y \leftarrow\$ \{0,1\}^n$
$R_b[X] \leftarrow Y$
```

```
$\underline{\mathrm{G}_6:}$

win ← false
$\mathcal{A}^{\mathbf{Enc},\mathbf{Dec},\mathbf{ChalDec}}$
Return win

$\underline{\mathbf{Enc}(H, M):}$

$N \leftarrow\$ \{0,1\}^n$
If $N \in \mathcal{IV}$ then
    bad ← true
    $N \leftarrow\$ \{0,1\}^n \setminus \mathcal{IV}$
$\mathcal{IV} \leftarrow \mathcal{IV} \cup N$
$K_{\mathsf{EC}} \leftarrow RF(0, N)$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
$C \leftarrow N \parallel C_{\mathsf{EC}} \parallel C_{\mathsf{AE}}$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, B_{\mathsf{EC}})\}$
$D[H, N, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $(C, B_{\mathsf{EC}})$

$\underline{\mathbf{Dec}(H, C, C_B):}$

$(N, C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
$V \leftarrow D[H, N, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}]$
If $V \neq \perp$:
    $(M, K_{\mathsf{EC}}) \leftarrow V$
    Return $(M, K_{\mathsf{EC}})$
If $D[\cdot, N, \cdot, B_{\mathsf{EC}}, \cdot] \neq \perp$ then
    Return $\perp$
$C'_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
If $C'_{\mathsf{AE}} \neq C_{\mathsf{AE}}$ then Return $\perp$
$K_{\mathsf{EC}} \leftarrow RF(0, N)$
$M \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \perp$ then Return $\perp$
win ← true
Return $(M, K_{\mathsf{EC}})$

$\underline{\mathbf{ChalDec}(H, C, C_B):}$

$(N, C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
If $(H, C, B_{\mathsf{EC}}) \in \mathcal{Y}$ then
    Return $\perp$
If $D[\cdot, N, \cdot, B_{\mathsf{EC}}, \cdot] \neq \perp$ then
    Return $\perp$
$C'_{\mathsf{AE}} \leftarrow RF(1, B_{\mathsf{EC}})$
If $C'_{\mathsf{AE}} \neq C_{\mathsf{AE}}$ then Return $\perp$
$K_{\mathsf{EC}} \leftarrow RF(0, N)$
$M \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \perp$ then Return $\perp$
win ← true
Return $(M, K_{\mathsf{EC}})$

$\underline{RF(b, X):}$

If $R_b[X] \neq \perp$:
    Return $R_b[X]$
$Y \leftarrow\$ \{0,1\}^n$
If $b = 0 \wedge Y \in R_0[\cdot]$ then
    $\mathsf{bad}_1$ ← true
    $Y \leftarrow\$ \{0,1\}^n \setminus R_0[\cdot]$
$R_b[X] \leftarrow Y$
```

Figure 25: Further Games for proof of Theorem 13.