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 messages 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 analyze 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 [40], Signal [38], and Facebook Messenger [13] 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 provide abuse reporting: should one user send another a harmful message, image, or video, the recipient should be able to report the content to the service 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 [14, 30]. 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) [17] provided the first formal treatment of the problem, and introduced compactly committing authenticated encryption 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). They detailed appropriate security notions and security proofs that provide validation of the main Facebook message franking approach and 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 [28] and OCB [32]. 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 [30]. 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 (preventing a malicious recipient from framing a user as having sent a message they didn't send). Instead it 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 [1,15,16].

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 [22, 28, 32], 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 [1,15,16], concealments [12], remotely keyed authenticated encryption [12], 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 [34, 35, 37]. A simple corollary of [35, 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 8. 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) [3] 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 [6] using Keccak (SHA-3) [5]. In fact the Duplex mode gives rise to a secure encryptment scheme as well. See the full version 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 [20], but the keying serves a different role here and their analysis techniques are not applicable.

From encryptment to ccAEAD. We show several 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. Another transform uses a single tweakable blockcipher call in addition to the encryptment. See the full version for details.

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 [18] 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 the full version of this work, we show how it suffices for building concealments [12] (a conceptually similar, but distinct, primitive) which, in turn, can be used to build remotely keyed AE [12]. 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 [15] 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 the string x.

We let ε denote the empty string, and $\perp$ denote the distinguished error symbol. We write $x \leftarrow\!\!\$\, \mathcal{X}$ to denote choosing an element at random 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 the internal 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 associated 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 some 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.

For space reasons, we direct the reader to [33] for syntax and correctness notions for AEAD. We require that AEAD schemes offer both real-or-random confidentiality and ciphertext integrity. These will be formalized in Section 7.

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. The attack creates a "colliding" GCM ciphertext which decrypts to an abusive attachment via one key and an innocuous attachment via the other. 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 CtE2, Facebook's ccAEAD scheme for chat messages described in [14, 30] and analyzed in [17], as a subroutine. Some encryption and HMAC keys, as well as some other details like headers and associated data not important to the presentation of the protocol, have been removed for simplicity in the diagram and prose

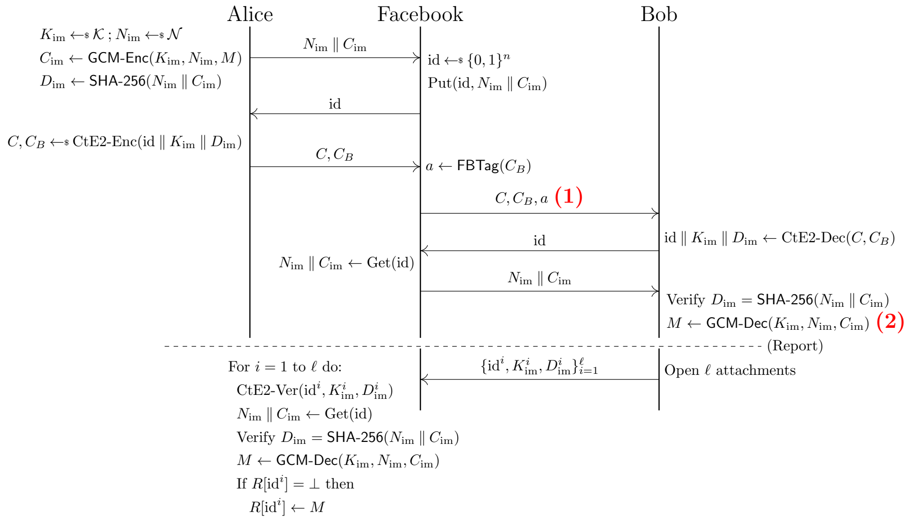
Alice
Facebook
Bob
$K_{\rm im} \leftarrow\!\!\$\ \mathcal{K}\,;\, N_{\rm im} \leftarrow\!\!\$\ \mathcal{N}$
$C_{\rm im} \leftarrow \textsf{GCM-Enc}(K_{\rm im}, N_{\rm im}, M)$
$D_{\rm im} \leftarrow \textsf{SHA-256}(N_{\rm im} \| C_{\rm im})$
$N_{\rm im} \| C_{\rm im}$
$\mathrm{id} \leftarrow\!\!\$\ \{0,1\}^n$
$\mathrm{Put}(\mathrm{id}, N_{\rm im} \| C_{\rm im})$
id
$C, C_B \leftarrow\!\!\$\ \mathrm{CtE2\text{-}Enc}(\mathrm{id} \| K_{\rm im} \| D_{\rm im})$
C, C_B
$a \leftarrow \textsf{FBTag}(C_B)$
C, C_B, a (1)
id
$\mathrm{id} \| K_{\rm im} \| D_{\rm im} \leftarrow \mathrm{CtE2\text{-}Dec}(C, C_B)$
$N_{\rm im} \| C_{\rm im} \leftarrow \mathrm{Get}(\mathrm{id})$
$N_{\rm im} \| C_{\rm im}$
Verify $D_{\rm im} = \textsf{SHA-256}(N_{\rm im} \| C_{\rm im})$
$M \leftarrow \textsf{GCM-Dec}(K_{\rm im}, N_{\rm im}, C_{\rm im})$ (2)
(Report)
For $i = 1$ to ℓ do:
$\mathrm{CtE2\text{-}Ver}(\mathrm{id}^i, K^i_{\rm im}, D^i_{\rm im})$
$N_{\rm im} \| C_{\rm im} \leftarrow \mathrm{Get}(\mathrm{id})$
Verify $D_{\rm im} = \textsf{SHA-256}(N_{\rm im} \| C_{\rm im})$
$M \leftarrow \textsf{GCM-Dec}(K_{\rm im}, N_{\rm im}, C_{\rm im})$
If $R[\mathrm{id}^i] = \perp$ then
$R[\mathrm{id}^i] \leftarrow M$
$\{\mathrm{id}^i, K^i_{\rm im}, D^i_{\rm im}\}_{i=1}^{\ell}$
Open ℓ attachments

Fig. 1: Facebook's attachment franking protocol [29, 30]. 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.

below. Consult [14, 17] 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: In the first part of the sending phase, Alice generates a key $K_{\rm im}$ and nonce $N_{\rm im}$ and encrypts $M_{\rm a}$ using AES-GCM (described in pseudocode in Figure 2) to obtain a ciphertext $C_{\rm im}$. The sender computes the SHA-256 digest $D_{\rm im}$ of $N_{\rm im} \| C_{\rm im}$ and sends Facebook $N_{\rm im} \| C_{\rm im}$ for storage. Facebook generates a random identifier id and puts $N_{\rm im} \| C_{\rm im}$ 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 id $\| K_{\rm im} \| D_{\rm im}$ 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 $\textsf{FBTag}$ on C_B (this amounts to HMAC-SHA256 with an internal Facebook key and some metadata) as in the standard message franking protocol to obtain a. Facebook sends C, C_B, a to the receiver.

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

Reporting phase: Bob sends all recent messages to Facebook along with their commitment openings and a values (not pictured in the diagram). For each message, Facebook verifies the commitment using CtE2-Ver and the authentication tag a using its internal HMAC key. Then, if the commitment verifies correctly and the message contains attachment metadata, Facebook gets the attachment ciphertext and nonce $N_{\mathrm{im}} \,\|\, C_{\mathrm{im}}$ from its key-value store using its identifier id. Facebook verifies that $D_{\mathrm{im}} = \mathsf{SHA\text{-}256}(N_{\mathrm{im}} \,\|\, C_{\mathrm{im}})$ and decrypts C_{im} with K_{im} and N_{im} 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.

Generating the colliding ciphertext—simple variant. 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_{im}, then computes a ciphertext C_{a} via $\text{CTR-Enc}(K_1, N_{\mathrm{im}} + 2, M_{\mathrm{a}}^{\mathrm{ab}})$, where CTR-Enc denotes CTR-mode encryption with the given key and nonce. The nonce is $N_{\mathrm{im}} + 2$ 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 $\mathsf{Collide\text{-}GCM}(K_1, K_2, N_{\mathrm{im}}, C_{\mathrm{a}})$ (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

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

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

Fig. 2: **(Left)** The Galois/Counter block cipher mode. **(Right)** The Collide-GCM algorithm. 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 is in GF(2^{128}). The function Collide-GCM can take arbitrary headers, but we elide them for simplicity.

$N_{\mathrm{im}} \,\|\, C_{\mathrm{im}} \,\|\, T$ 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.

Sending the colliding ciphertext. Alice continues the sending phase with Facebook, obtaining an identifier id for the ciphertext $N_{\mathrm{im}} \,\|\, C_{\mathrm{im}}$. Alice then creates two attachment metadata messages: $\mathrm{MD}_1 = \mathrm{id} \,\|\, K_2 \,\|\, D_{\mathrm{im}}$ and $\mathrm{MD}_2 = \mathrm{id} \,\|\, K_1 \,\|\, D_{\mathrm{im}}$. 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_{im} with K_2, giving M_{j}. After finishing the receiving phase with MD_2, Bob will decrypt C_{im} 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 are violated.

When Bob reports the recent messages, Facebook will verify both MD_1 and MD_2 and check the digest D_{im} matches the value $N_{\mathrm{im}} \| C_{\mathrm{im}}$ 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 human viewing the report will have no idea Bob ever saw the message $M_{\mathrm{a}}^{\mathrm{ab}}$.

3.1 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 $M_{\mathrm{a}}^{\mathrm{ab}}$ is the image of an Axolotl salamander

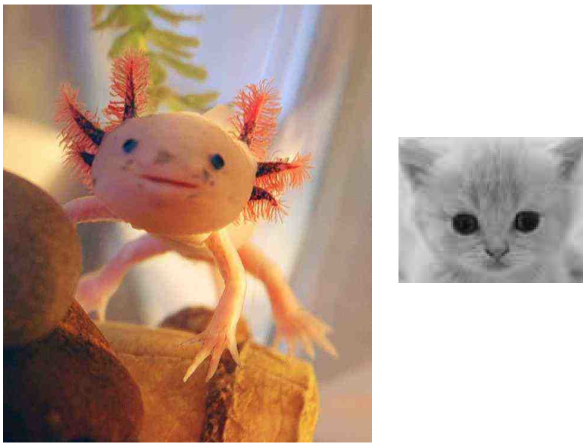

Fig. 3: Two images with the same GCM ciphertext $C_{\mathrm{im}} \parallel T$ when encrypted using 16-byte key $K_1 = (\mathtt{03})^{16}$ or $K_2 = (\mathtt{02})^{16}$, nonce $N_{\mathrm{im}} = 10606665379$, and associated data $H = (\mathtt{ad})^{32}$ (all given in hex where exponentiation indicates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

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 $M_{\mathrm{a}}^{\mathrm{ab}}$, Alice first merges $M_{\mathrm{a}}^{\mathrm{ab}}$ and another innocuous attachment M_j using a function $\text{Att-Merge}(K_1, K_2, M_{\mathrm{a}}^{\mathrm{ab}}, \mathsf{M}_\mathsf{j})$ which takes the two keys and attachments and outputs a nonce N_{im} and C_{a} so that $\text{CTR-Dec}(K_1, N_{\mathrm{im}} + 2, C_{\mathrm{a}})$ displays $M_{\mathrm{a}}^{\mathrm{ab}}$ and $\text{CTR-Dec}(K_2, N_{\mathrm{im}} + 2, C_{\mathrm{a}})$ 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 gives a collision on some region of the ciphertext, 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 *must* begin with the two-byte sequence `ffd8` and end with `ffd9`. JPEGs can have comments. They are indicated with the two-byte sequence `fffe` followed by a big-endian two-byte encoding of the comment length. 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.

Nonce search. Since file formats generally have some internal structure (like having a fixed byte sequence at the beginning or end) Att-Merge must choose a nonce so that the keystreams for the two keys respect this structure. JPEG and

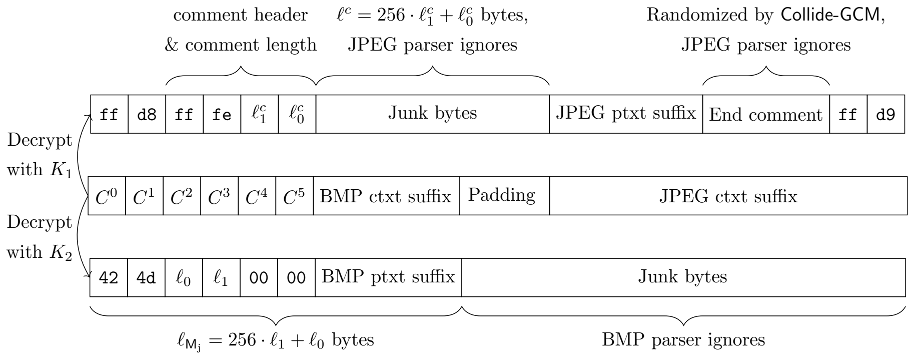

Fig. 4: Diagram of the JPEG M^{ab}_{a} **(top)** and BMP M_j **(bottom)** plaintexts output by the plaintext restructuring step, and their ciphertext **(middle)**. The leftmost block of each file is the first byte. The "BMP ptxt suffix" is the suffix of the original BMP starting at byte 6. The "JPEG ptxt suffix" is the bytes of the original JPEG starting at byte 2 and ending before the final two bytes. The 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.

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 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 a collision in the first four bytes of the ciphertext (marked C^0 through C^4 in Figure 4), which 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.

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.

One important subtlety is that JPEG comments are at most 2^{16} bytes in length, so the BMP image must be smaller than 2^{16} bytes. In fact, it is advantageous for the BMP to be as small as possible because the comment length bytes in the JPEG are not fixed by the nonce search. A more detailed explanation of this issue and plaintext restructuring in general will be given in the full version of this work.

Implementing Collide-GCM. We implemented Collide-GCM in Python 2.7 and verified that arbitrary colliding ciphertexts can be generated in roughly 45 seconds using an unoptimized implementation of 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.

3.2 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 but it is possible. We did not try to implement Att-Merge for video file formats. Such formats are substantially more complex than image formats, but we conjecture it is possible to extend the attack to video files.

Relation to GLR In [17] GLR proved CtE2 is a ccAEAD scheme, and one may wonder whether this attack shows their proof is incorrect. 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. This attack points to a gap between GLR's analysis and what Facebook actually uses, but it does not mean GLR's proof is incorrect. Indeed, the fact that the attack works without breaking CtE2's binding highlights the surprising subtlety of security notions for this setting.

The Collide-GCM algorithm in Figure 2 is related to the r-BIND attack against GCM given by GLR [17]. 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.

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, primarily because it did not require patching the Messenger clients (an expensive and time-consuming process). 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[4], a message. 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 [1,15,16], cryptographic concealments [12], and domain extension for authenticated encryption and remotely keyed AE [12], facilitating the construction of very efficient instantiations of these primitives. In Section 7.3 we show how to build ccAEAD from encryptment. The other primitives are deferred to the full version of this work.

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 $\perp \leftarrow \mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$.

[4] A secure *commitment* allows a user to commit to a message without revealing its content; see [10] for further discussion.

- 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: $K_{\mathsf{EC}} \leftarrow\!\!\$\ \mathsf{EKg}$; $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EC}(K_{\mathsf{EC}}, H, M)$.

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 $\Pr[\,\mathrm{COR}_{\mathsf{EC}}(H, M) \Rightarrow 1\,] = 1$, where the probability is over the coins of EKg.

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

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

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

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 $(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}}) \in \mathcal{K}_{\mathsf{EC}} \times \mathcal{H}_{\mathsf{EC}} \times \mathcal{M}_{\mathsf{EC}} \times \mathcal{C}_{\mathsf{EC}} \times \mathcal{T}_{\mathsf{EC}}$, it holds that $\Pr[\,\mathrm{S\text{-}COR}_{\mathsf{EC}}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}}) \Rightarrow 1\,] = 1$. 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 a ciphertext 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.

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

$\underline{\mathsf{enc}(K_{\mathsf{EC}}, H, M):}$
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 $b = 1$ then
    Return $\mathsf{true}$
Return $\mathsf{false}$
```

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

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 [17], 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. We defer the details to the full version. 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 $\mathbf{Adv}^{\text{s-bind}}_{\mathsf{EC}}(\mathcal{A}) = \Pr\left[\,\text{s-BIND}^{\mathcal{A}}_{\mathsf{EC}} \Rightarrow \mathsf{true}\,\right]$.

Binding notions and the Facebook attack. Looking ahead, the analogous strong receiver binding notion for ccAEAD schemes is the property that would have prevented the Facebook attack, had they used a scheme that enjoyed it. This is because receiver binding implies that it is computationally intractable for an attacker to find two distinct keys that verify the same binding tag. In the Facebook attack, the sender was able to exploit this weakness to violate a security property similar to GLR's sender binding notion [17], which ensures decryption can only succeed if the binding tag commits to the underlying plaintext. Canonically, however, receiver binding is modeling the ability of a malicious receiver to frame the sender as having sent a message they did not, in fact, send. Such an attack doesn't work against Facebook's attachment franking scheme because the encryption of the AES-GCM key enjoys receiver binding, and prevents the recipient from forging an abuse report for an image that wasn't sent.

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 ciphertext integrity. 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. One useful tool towards this is that we can show 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 the 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* $\mathbf{Adv}^{\mathrm{scu}}_{\mathsf{EC}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{s\text{-}bind}}_{\mathsf{EC}}(\mathcal{B}) + \mathbf{Adv}^{\mathrm{r\text{-}bind}}_{\mathsf{EC}}(\mathcal{C})$*, and moreover* $\mathcal{B}$ *and* $\mathcal{C}$ *both run in the same time as* $\mathcal{A}$*.*

We give a proof sketch and defer details to the full version. 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 $\mathcal{A}$ subsequently 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 [17]. We defer the details to the full version. 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 [32], 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 the full version.

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 [9,34,35]. 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 [35], we may write $\mathcal{H}$ as shown in Figure 7, where we let $K_1, \ldots, K_k$ be any *fixed* strings[5], and $f_i\colon \{0,1\}^{(m+(i-1))n} \to \{0,1\}^{n}$ ($i = 1, \ldots, k$), $g\colon \{0,1\}^{(m+k)n} \to \{0,1\}^{rn}$ are 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 [35] (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.

```
$\mathcal{H}(V)$
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 $W$
```

Fig. 7: 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 computing $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) = \mathsf{EC}(K_{\mathsf{EC}}, H, M)$ and outputting B_{EC}. 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 $\mathcal{H}\colon \{0,1\}^{*} \to$

[5] One can modify our definitions so keys can be picked from a set as a function of the current round and messages, what Rogaway and Steinberger refer to as the no-fixed order model, and as first done in [9]. A negative result based on [9, Th. 5] would rule out encryptment using any rate-1 no-fixed order verification algorithm.

$\{0,1\}^n$ 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). The following is simple to prove.

Theorem 1. *Let* EC *be a encryptment scheme with binding codes, and let* $\mathcal{H}$ *be defined as in the previous paragraph. For any collision-resistance adversary* $\mathcal{A}$*, we give an* r-BIND *adversary* $\mathcal{B}$ *so that* $\mathbf{Adv}^{\mathrm{cr}}_{\mathcal{H}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{r-bind}}_{\mathsf{EC}}(\mathcal{B})$*. 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 [35, Th. 1]:

Corollary 1. *Fix* $m > r \geq 1$ *and* $n > 0$ $(m, r, n \in \mathbb{N})$*. Let* $N = 2^n$*. Let* EC *be an encryptment scheme with ideal-cipher-based binding codes of length* rn *and 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* $\mathbf{Adv}^{\text{r-bind}}_{\mathsf{EC}}(\mathcal{A}) = 1$*, where* $k \in \mathbb{N}$ *denotes the number of permutation calls required to compute the binding code 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 [35, Th. 2] under some technical conditions about the verification function and the adversary. The results above were cast in terms of r-BIND security, but extend to sr-BIND security because the latter implies the former.

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 Section 7, we will describe how the existence of an r-BIND-secure ccAEAD scheme of a given rate implies the existence of a given 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.

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 the construction is shown in Figure 8 (padding details are omitted), where f is a compression function. In summary, the scheme hashes the key, associated data and message data (the

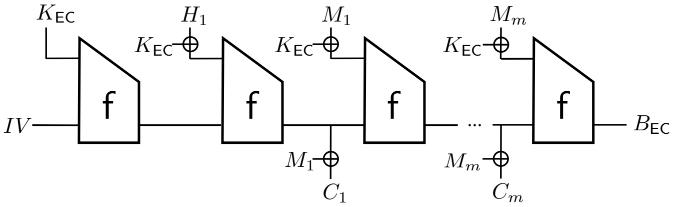

Fig. 8: Encryptment in the HFC scheme for a 1-block header and m-block message. For simplicity the diagram does not show the details of padding.

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 $(X_1, \dots, X_\ell)$ such that $|X_i| = d$ for $1 \leq i \leq \ell - 1$ and $|X_\ell| = |X| \mod d$. For correctness, we require that $X = X_1 \| \dots \| X_\ell$. 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.

Our scheme utilizes a padding scheme $\mathsf{PadS} = (\mathrm{PadH}, \mathrm{PadM}, \mathrm{PadSuf}, \mathrm{Pad})$. The padding scheme is parameterized by a pair of numbers d, n, but we omit these in the notation for simplicity. We assume $d \geq n \geq 128$. The algorithms PadH, PadM, and PadSuf are shown in Figure 9. Notice that for all header and message pairs (H, M), it holds that if $|M| \bmod n = r$, then $r + |\mathrm{PadSuf}(|H|, |M|)|$ will be equal to either d or $2d$. The full padding function is then defined to be $\mathrm{Pad}(H, M) = \mathrm{PadH}(H) \| \mathrm{PadM}(M) \| \mathrm{PadSuf}(|H|, |M|)$. Note that $|\mathrm{Pad}(H, M)|$ is a multiple of d and that the function $\mathrm{Pad}(H, M)$ is injective, i.e., for all pairs $(H, M), (H', M')$, $\mathrm{Pad}(H, M) = \mathrm{Pad}(H', M')$ only if $(H, M) = (H', M')$.

```
PadH(H)
$(H_1, \dots, H_h) \leftarrow \mathrm{Parse}_d(H \| 0^{d-|H| \bmod d})$
Return $(H_1, \dots, H_h)$

PadM(M)
$(M'_1, \dots, M'_m) \leftarrow \mathrm{Parse}_n(M)$
For $i = 1, \dots, m-1$
    $M_i \leftarrow M'_i \| 0^{d-n}$
Return $(M_1, \dots, M_{m-1}, M'_m)$

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

Fig. 9: Padding scheme $\mathsf{PadS} = (\mathrm{PadH}, \mathrm{PadM}, \mathrm{PadSuf}, \mathrm{Pad})$. We require that $\ell_H, \ell_M \in \mathbb{N}^2$.

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

```
$\mathsf{HFCDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$:
$H_1, \ldots, H_h \leftarrow \mathrm{PadH}(H)$
$C_1, \ldots, C_m \leftarrow \mathrm{Parse}_n(C_{\mathsf{EC}})$
$V_0 \leftarrow \mathsf{f}(IV, K_{\mathsf{EC}})$
$V_h \leftarrow \mathsf{f}^+\big(V_0, (K_{\mathsf{EC}} \oplus H_1) \parallel \cdots \parallel (K_{\mathsf{EC}} \oplus H_h)\big)$
For $i = 1, \ldots, m-1$ do
    $M_i \leftarrow V_{h+i-1} \oplus C_i$
    $V_{h+i} \leftarrow \mathsf{f}(V_{h+i-1}, (K_{\mathsf{EC}} \oplus M_i || 0^{d-n}))$
$M_m \leftarrow V_{h+m-1} \oplus C_m$
$M'_m, M'_{m+1} \leftarrow \mathrm{Parse}_d(M_m \parallel \mathrm{PadSuf}(|H|, |C_{\mathsf{EC}}|))$
$B_{\mathsf{EC}} \leftarrow \mathsf{f}^+(V_{h+m-1}, (K_{\mathsf{EC}} \oplus M'_m) || (K_{\mathsf{EC}} \oplus M'_{m+1}))$
If $B'_{\mathsf{EC}} \neq B_{\mathsf{EC}}$ then
    Return $\perp$
Return $M_1 \parallel \cdots \parallel M_m$
```

Fig. 10: 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 $\mathsf{PadS} = (\mathrm{PadH}, \mathrm{PadM}, \mathrm{PadSuf}, \mathrm{Pad})$. Here $K_{\mathsf{EC}} \in \{0,1\}^d$, and $IV \in \{0,1\}^n$ is a fixed public constant.

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 $\mathsf{f}^+(V_0, X_1 \| \cdots \| X_m) = V_m$ 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 10.

Key generation HFCKg simply 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 $\mathrm{PadH}(H) \parallel \mathrm{PadM}(M) \parallel \mathrm{PadSuf}(|H|, |M|)$ 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. 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 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, incurring a significant efficiency saving (see further discussion below).

Decryption $\mathsf{DO}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}})$ begins by padding H into d-bit blocks via $\mathrm{PadH}(H)$ 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 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 $\mathrm{PadH}(H) \parallel \mathrm{PadM}(M) \parallel \mathrm{PadSuf}(|H|, |M|)$, 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}.

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.

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 the padding. In the full version we present an alternative padding scheme for this case, which recovers some throughput by padding message blocks with header data.

6.1 Analyzing the HFC Encryptment Scheme

In this section, we analyze the security of the HFC encryptment scheme, relative to the security goals detailed in Section 4. We also discuss some of the options for instantiating the compression function f.

Strong receiver binding. We begin by proving that the HFC encryptment scheme satisfies strong receiver binding. Observe that the binding tag computation performed by HFCEnc on input tuple (K_{EC}, H, M) is equivalent to XORing K_{EC} into each d-bit block of $0^d \parallel \mathrm{Pad}(H, M)$ (we refer to this as 'encoding' the tuple), and hashing the resulting string with f^+. Moreover, it is straightforward to verify that the injectivity of Pad implies that the encoding map is injective also. So any tuple breaking the sr-BIND security of HFC is a collision against f^+.

A well-known folklore result (see [2]) gives that f^+ is collision-resistant provided the underlying compression function is 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 the full version. The conditions on d, n below are due to the padding scheme and can be relaxed.

Theorem 2. *Let* HFC *be as shown in Figure 10, using compression function* $\mathsf{f}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$ *where* $d \geq n \geq 128$*. Then for any adversary* $\mathcal{A}$ *in the* sr-BIND *game against* HFC*, there exists 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 2.

One-time confidentiality. All that remains is to bound the otROR security of HFC. We do this in the next theorem, by reducing otROR security of HFC to the related-key attack (RKA) PRF security [3] 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 a key $K_{\text{prf}} \leftarrow\!\!\$\; \{0,1\}^d$ is chosen. 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 $F(X, Y \oplus K_{\text{prf}})$. In game RKA-PRF1, the oracle returns a random bit string for each query, answering consistently if $(X, Y \oplus K_{\text{prf}})$ collides with a previous query. The *linear-only* RKA-PRF advantage of an adversary $\mathcal{A}$ is defined as

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

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

The proof of the following theorem then follows from a reduction to the RKA-PRF security of f, coupled with a birthday bound to account for collisions during the challenge ciphertext computation. The proof is given in the full version.

Theorem 3. *Let* HFC *be as shown in Figure 10, using compression function* $\mathsf{f}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$ *where* $d \geq n \geq 128$*. Then for any adversary* $\mathcal{A}$ *in the otROR game against* HFC*, there exists an adversary* $\mathcal{B}$ *such that* $\mathbf{Adv}^{\text{ot-ror}}_{\mathsf{HFC}}(\mathcal{A}) \leq \mathbf{Adv}^{\oplus\text{-prf}}_{\mathsf{f}}(\mathcal{B}) + \frac{\ell^2}{2^n}$*, where* $\ell \cdot d$ *denotes the length of* $\mathcal{A}$*'s encryption query after padding. The adversary* $\mathcal{B}$ *runs in time that of* $\mathcal{A}$ *plus an* $\mathcal{O}(\ell)$ *overhead and makes at most* ℓ *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 [11, 18, 39]. Extensive cryptanalysis for the CR (e.g., [23, 26, 36]), preimage resistance (e.g., [19, 23]), and RKA-PRP of the associated SHACAL-2 blockcipher (e.g., [21, 24, 25, 27]) gives confidence in its security. Another approach would be to use AES via a PGV compression function [31] like Davies-Meyer (DM). Security of AES has been studied extensively, and known attacks do not falsify the assumptions we need [7, 8]. 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. 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 [34]. We discuss both cases, and include a more thorough discussion of instantiations, in the full version.

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 [17], 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 the full version, 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; this result does not appear here for space reasons.

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 usual definition of AEAD schemes (see Section 2) can be recovered from the above definition of ccAEAD schemes by noticing that the tuple of AEAD algorithms $\mathsf{AEAD} = (\mathsf{AEAD.kg}, \mathsf{AEAD.enc}, \mathsf{AEAD.dec})$ 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.

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 [33], and ciphertext integrity (CTXT) [4] 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 11. In both variants, the attacker is given access to an oracle **ChalEnc** to which he may

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

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

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

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

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

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

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

$\mathbf{ChalEnc}(H, M)$
$(C, C_B) \leftarrow\!\!\$\ \mathcal{C}(|M|) \times \mathcal{T}$
Return $(C, C_B)$
```

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

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

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

$\mathbf{ChalDec}(H, C, C_B)$
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 $(M, K_f)$
```

Fig. 11: Confidentiality (left two games) and ciphertext integrity (rightmost) games for ccAEAD.

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

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

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 11. 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

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

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 11 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 [16], and vice versa. In the full version, 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 12. 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 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 12. 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 the full version.

Theorem 4. *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 12. 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.*

```
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 $(M, K_{\mathsf{EC}})$
```

Fig. 12: 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.

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; we defer the proof to the full version.

Theorem 5. *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 12. 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. In the full version of the paper, we provide two other approaches for building ccAEAD from encryptment, which use a PRF and a tweakable block cipher respectively.

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. A portion of this work was completed while Grubbs visited Royal Holloway University, and he thanks Kenny Patterson 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 Greg Neven. Robust encryption. In *TCC*, 2010.
2. Mihir Bellare, Joseph Jaeger, and Julia Len. Better than advertised: Improved collision-resistance guarantees for MD-based hash functions. In *ACM CCS*, 2017.
3. Mihir Bellare and Tadayoshi Kohno. A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In *EUROCRYPT*, 2003.
4. Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of generic composition. In *ASIACRYPT*, 2000.
5. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak sponge function family main document. *Submission to NIST SHA3*, 2009.
6. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: single-pass authenticated encryption. In *SAC*, 2011.
7. Alex Biryukov and Dmitry Khovratovich. Related-key cryptanalysis of the full AES-192 and AES-256. In *ASIACRYPT*, 2009.
8. Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolić. Distinguisher and related-key attack on the full AES-256. In *CRYPTO*. 2009.
9. John Black, Martin Cochran, and Thomas Shrimpton. On the impossibility of highly-efficient blockcipher-based hash functions. In *EUROCRYPT*, 2005.
10. Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *JCSS*, 1988.
11. Advanced Micro Devices. The ZEN microarchitecture, 2016. `https://www.amd.com/en/technologies/zen-core`.
12. Yevgeniy Dodis and Jee Hea An. Concealment and its applications to authenticated encryption. In *EUROCRYPT*, 2003.
13. Facebook. Facebook Messenger app. `https://www.messenger.com/`, 2016.
14. Facebook. Messenger Secret Conversations technical whitepaper, 2016.
15. Pooya Farshim, Benoît Libert, Kenneth G Paterson, and Elizabeth A Quaglia. Robust encryption, revisited. In *PKC*. 2013.
16. Pooya Farshim, Claudio Orlandi, and Razvan Rosie. Security of symmetric primitives under incorrect usage of keys. *FSE*, 2017.
17. Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. Message franking via committing authenticated encryption. In *CRYPTO*, 2017.

18. 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`.
19. 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.
20. Shai Halevi and Hugo Krawczyk. Strengthening digital signatures via randomized hashing. In *CRYPTO*, 2006.
21. 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.
22. Charanjit S Jutla. Encryption modes with almost free message integrity. In *EUROCRYPT*, 2001.
23. Dmitry Khovratovich, Christian Rechberger, and Alexandra Savelieva. Bicliques for preimages: attacks on Skein-512 and the SHA-2 family. In *FSE*, 2012.
24. Jongsung Kim, Guil Kim, Seokhie Hong, Sangjin Lee, and Dowon Hong. The related-key rectangle attack–application to SHACAL-1. In *ACISP*, 2004.
25. Jongsung Kim, Guil Kim, Sangjin Lee, Jongin Lim, and Junghwan Song. Related-key attacks on reduced rounds of SHACAL-2. In *INDOCRYPT*, 2004.
26. Mario Lamberger and Florian Mendel. Higher-order differential attack on reduced SHA-256. *IACR ePrint, Report 2011/037*, 2011.
27. Jiqiang Lu, Jongsung Kim, Nathan Keller, and Orr Dunkelman. Related-key rectangle attack on 42-round SHACAL-2. In *ICIS*, 2006.
28. David McGrew and John Viega. The Galois/counter mode of operation (GCM). *NIST Modes of Operation*, 2004.
29. Jon Millican. Personal communication.
30. Jon Millican. Challenges of E2E Encryption in Facebook Messenger. RWC, 2017.
31. Bart Preneel, René Govaerts, and Joos Vandewalle. Hash functions based on block ciphers: A synthetic approach. In *CRYPTO*, 1993.
32. Phillip Rogaway, Mihir Bellare, and John Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM TISSEC*, 2003.
33. Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In *EUROCRYPT*, 2006.
34. Phillip Rogaway and John Steinberger. Constructing cryptographic hash functions from fixed-key blockciphers. In *CRYPTO*, 2008.
35. Phillip Rogaway and John Steinberger. Security/efficiency tradeoffs for permutation-based hashing. In *EUROCRYPT*, 2008.
36. Somitra Kumar Sanadhya and Palash Sarkar. New collision attacks against up to 24-step SHA-2. In *INDOCRYPT*, 2008.
37. Thomas Shrimpton and Martijn Stam. Building a collision-resistant compression function from non-compressing primitives. In *ICALP*, 2008.
38. Open Whisper Systems. Signal. `https://signal.org/`, 2016.
39. Wouter van der Linde. Parallel SHA-256 in NEON for use in hash-based signatures, 2016. BSc thesis, Radboud University.
40. Whatsapp. Whatsapp. `https://www.whatsapp.com/`, 2016.