Automated Analysis of Protocols that use Authenticated Encryption: How Subtle AEAD Differences can impact Protocol Security

Cas Cremers, Alexander Dax, Charlie Jacomme, Mang Zhao

▶ **To cite this version:**

Cas Cremers, Alexander Dax, Charlie Jacomme, Mang Zhao. Automated Analysis of Protocols that use Authenticated Encryption: How Subtle AEAD Differences can impact Protocol Security. 32nd USENIX Security Symposium 2023, USENIX, Aug 2023, Anaheim, United States. hal-04126116v3

HAL Id: hal-04126116

https://inria.hal.science/hal-04126116v3

Submitted on 17 Aug 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Automated Analysis of Protocols that use Authenticated Encryption: How Subtle AEAD Differences can impact Protocol Security

Cas Cremers[1], Alexander Dax[1,3], Charlie Jacomme[2], and Mang Zhao[1,3]

[1]CISPA Helmholtz Center for Information Security, Germany
[2]Inria Paris, France
[3]Saarland University

August 17, 2023– v1.1*

Abstract

Many modern security protocols such as TLS, WPA2, WireGuard, and Signal use a cryptographic primitive called Authenticated Encryption (optionally with Authenticated Data), also known as an AEAD scheme. AEAD is a variant of symmetric encryption that additionally provides authentication. While authentication may seem to be a straightforward additional requirement, it has in fact turned out to be complex: many different security notions for AEADs are still being proposed, and several recent protocol-level attacks exploit subtle behaviors that differ among real-world AEAD schemes.

We provide the first automated analysis method for protocols that use AEADs that can systematically find attacks that exploit the subtleties of the specific type of AEAD used. This can then be used to analyze specific protocols with a fixed AEAD choice, or to provide guidance on which AEADs might be (in)sufficient to make a protocol design secure. We develop generic symbolic AEAD models, which we instantiate for the Tamarin prover. Our approach can automatically and efficiently discover protocol attacks that could previously only be found using manual inspection, such as the Salamander attack on Facebook's message franking, and attacks on SFrame and YubiHSM. Furthermore, our analysis reveals undesirable behaviors of several other protocols.

1 Introduction

Authenticated Encryption (AE) and Authenticated Encryption with Associated Data (AEAD) are some of the most commonly used cryptographic building blocks. AEAD primitives are built from symmetric encryption primitives and augmented with authentication mechanisms. Their applications include the vast majority of encrypted internet data, such as in TLS, WPA2 from IEEE 802.11 (WiFi), WireGuard, and by messaging apps such as Signal or WhatsApp. For example, in TLS, TLSCiphertext is constructed from an AEAD applied to a header and payload: both are authenticated, but only the payload is encrypted, and the plaintext header includes the content type and the ciphertext length.

While AEADs are ubiquitous in modern secure communications, there is no commonly agreed "strong" security notion that they should satisfy. In fact, the current landscape of security notions for AEADs is rather chaotic: there are many proposed frameworks and security notion variants [1, 2, 3, 7, 8, 9, 15, 17, 25, 30, 31, 42, 50, 54]. For some of these notions, their implication relations are known [7], but many of them are hard to compare for technical reasons.

In reality, there are good examples of recent protocol attacks that exploit subtle properties of concrete AEAD schemes, such as [25, 32, 41]. These have all been found through manual inspection of the protocol and knowledge of the particular AEAD scheme, such as exploiting the reuse of a nonce, a number meant to be used only once. We would like to formally prove the absence of such attacks: i.e., that a protocol (e.g., TLS, WhatsApp, WPA2), when instantiated with a specific AEAD (e.g., AES-GCM), satisfies a desired security notion. At a methodological level, these attacks can be hard to model because they require a methodology that is not only precise enough to capture AEAD details (e.g. impact of nonce reuse) but also scales well enough to allow for modeling the often complex possible executions of a protocol that determine whether such attack requirements can be met.

*An extended abstract of this paper appears at USENIX Security'23; this is the full version.

In this work, we develop the first systematic methodology for the automated analysis of security protocols that can find attacks that leverage subtle behaviors of concrete AEAD schemes, or show their absence. For our methodology, we leverage the Tamarin prover [44], a symbolic protocol analysis tool, which we augment with novel fine-grained models of AEAD primitives. This tool choice enables us to analyze several non-trivial protocols. We also considered using tools in the computational model (e.g., [5, 6, 12]), but these currently do not yet scale to the complexity of the protocols we are interested in, and cannot find attacks.

One of the challenges in developing our models is that they require finding middle ground between theoretical security notions, weaknesseses exploited in practice, and suitability for automated analysis. We identify the core theoretical and practical concerns of AEADs, and use these to develop our generic symbolic AEAD models. We notably identify how the *collision resistance* of AEAD is a central issue of their design: we illustrate how a lack of it leads to multiple attack classes, and how satisfying collision resistance implies that many existing security notions are met. In our protocol case studies, we rediscover previously reported attack classes, such as *accountability* or *authentication*, but also identify a new attack class concerning *content agreement* in group messaging scenarios: can a dishonest group member send a single message that will be interpreted differently by multiple parties?

Contributions Our main contributions are the following:

- We develop the first systematic automated methodology for analyzing security protocols that takes the subtle properties of specific AEAD instantiations into account.
- In case studies, we show our methodology effectively rediscovers known attacks on several protocols, including YubiHSM [41], Facebook's Message Franking [25], and SFrame [32]. We also rediscover a theoretical attack variant on Facebook's Message Franking first mentioned in [30]. Moreover, our analysis uncovers unexpected behavior in WebPush [55], Whatsapp Group Messaging [58], and Scuttlebutt [53].
- We formally prove the missing or conjectured relations between existing AEAD security notions w.r.t. collision resistance, completing the picture in the domain.

We provide the full formal Tamarin models and analysis scripts at [19].

Overview We first give background on AEADs in Section 2. We build the foundations for our methodology by revisiting the AEAD landscape and real-world attack patterns in Section 3, and prove some missing relations between AEAD notions. We then develop our symbolic modeling and analysis approach in Section 4. We evaluate our approach on several protocol case studies and present details on selected case studies in Section 5. We discuss limitations in Section 6 and describe further related work in Section 7. We conclude in Section 8.

In the supplementary appendix we provide full theorems and proofs, and illustrate the collision resistance of existing AEAD schemes.

2 Background on AEADs and protocol attacks

$\underline{\mathrm{Expr}^{\mathsf{IND\$\text{-}CPA}}_{\mathsf{AEAD}}}$:
1 $\mathsf{b} \leftarrow\$ \{0,1\}$
2 $\mathcal{L}_c \leftarrow \emptyset$
3 $k \leftarrow\$ \mathsf{KGen}()$
4 $\mathsf{b}' \leftarrow\$ \mathcal{A}^{\mathrm{Enc}}()$
5 **return** $[\![\mathsf{b} = \mathsf{b}']\!]$

$\underline{\mathrm{Expr}^{\mathsf{IND\$\text{-}CCA}}_{\mathsf{AEAD}}}$:
1 $\mathsf{b} \leftarrow\$ \{0,1\}$
2 $\mathcal{L}_c \leftarrow \emptyset$
3 $k \leftarrow\$ \mathsf{KGen}()$
4 $\mathsf{b}' \leftarrow\$ \mathcal{A}^{\mathrm{Enc},\mathrm{Dec}}()$
5 **return** $[\![\mathsf{b} = \mathsf{b}']\!]$

$\underline{\mathrm{Enc}(N, H, m)}$:
6 **if** $(N, H, m, _) \in \mathcal{L}_c$
7 **return** $\perp$
8 **if** $\mathsf{b} = 0$
9 $c \leftarrow \mathsf{Enc}(k, N, H, m)$
10 **else** $c \leftarrow\$ \{0,1\}^{\ell(|m|)}$
11 $\mathcal{L}_c \leftarrow \mathcal{L}_c \cup \{(N, H, m, c)\}$
12 **return** c

$\underline{\mathrm{Dec}(N, H, c)}$:
13 **if** $(N, H, _, c) \in \mathcal{L}_c$
14 **return** $\perp$
15 $m \leftarrow \mathsf{Dec}(k, N, H, c)$
16 **if** $m \neq \perp$
17 $\mathcal{L}_c \leftarrow \mathcal{L}_c \cup \{(N, H, m, c)\}$
18 **return** m

Figure 1: IND$-CPA and IND$-CCA security for an AEAD = (KGen, Enc, Dec) scheme.

The modern approach to protecting privacy and authenticity of messages is to use authenticated encryption. This primitive evolved as a variant of symmetric encryption that highly efficiently offers two additional properties that are useful in real-world applications: authentication of the encrypted data, and concurrent authentication of some plaintext data. One could in theory achieve this by using a combination of symmetric encryption and MACs, but AEADs are much more efficient and ensure the authenticated part is strictly bound to the encrypted-and-authenticated part.

$\underline{\text{Expr}^{\text{CTI-CPA}}_{\text{AEAD}}}$:
1 $\mathcal{L}_c \leftarrow \emptyset$
2 $k \leftarrow\!\$\ \mathsf{KGen}()$
3 $(N, H, c) \leftarrow\!\$\ \mathcal{A}^{\text{Enc}}()$
4 **if** $c \in \mathcal{L}_c$
5 **return** 0
6 **return** $[\![\mathsf{Dec}(k, N, H, c) \neq \perp]\!]$

$\underline{\text{Expr}^{\text{CTI-CCA}}_{\text{AEAD}}}$:
1 $\mathcal{L}_c \leftarrow \emptyset$
2 $k \leftarrow\!\$\ \mathsf{KGen}()$
3 $(N, H, c) \leftarrow\!\$\ \mathcal{A}^{\text{Enc},\text{Dec}}()$
4 **if** $c \in \mathcal{L}_c$
5 **return** 0
6 **return** $[\![\mathsf{Dec}(k, N, H, c) \neq \perp]\!]$

$\underline{\text{Enc}(N, H, m)}$:
7 $c \leftarrow \mathsf{Enc}(k, N, H, m)$
8 $\mathcal{L}_c \leftarrow \mathcal{L}_c \cup \{c\}$
9 **return** c
$\underline{\text{Dec}(N, H, c)}$:
10 **return** $\mathsf{Dec}(N, H, c)$

Figure 2: CTI-CPA and CTI-CCA security for an AEAD = (KGen, Enc, Dec) scheme.

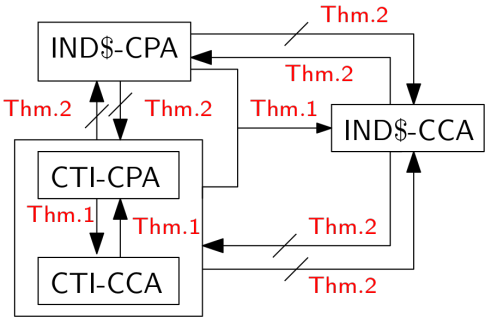


Figure 3: The relation between integrity and privacy for AEAD.

The efficient combination of these constructions has proven to be surprisingly intricate. In standard encryption it is desirable that if Alice encrypts the same message twice, the attacker cannot tell this from the ciphertexts. This property is typically achieved by ensuring that when encrypting the next message, some data x is integrated that was not used before, ensuring uniqueness of the ciphertext. As we will see later, this sometimes fails for various reasons, causing some x to be re-used. Intuitively, we would hope that while the attacker is now able to detect message re-encryption, there should not be any further negative consequences. For some provably secure AEADs, it turns out that the situation is worse.

Furthermore, much like symmetric encryption, AEADs in real-world deployments typically construct and send ciphertext incrementally. To start decrypting partial ciphertexts as soon as possible, the syntax or API of some AEADs allows to decouple decryption from the verification of authentication, which can be helpful but also cause problems elsewhere.

Historically, Authenticated Encryption with Associated Data (AEAD) was introduced by Rogaway [50] to entwine privacy and authenticity for both messages and headers in a single and compact mode. The definition of AEAD is given in the nonce-based pattern, where the nonce is named after the ***n***umber that are supposed to be used only ***once***. The nonce-based AEADs are expected to relax the security requirements of the randomized or counter-based pattern – ensuring no reuse of the nonce during the encryption is sufficient for privacy and authenticity. Moreover, Bellare and Hoang [8] initialized the study on binding keys and other optional inputs to the ciphertexts.

In this section, we first recall the formal syntax of AEADs and the canonical privacy and integrity definitions in Section 2.1. In Section 2.2, we review the main attacks on protocols based on subtle AEADs behaviors and weaknesses. In Section 2.3, we summarize and classify the main AEAD frameworks in the literature.

2.1 Formal AEAD syntax and requirements

Notations. We consider that all algorithms defined in this paper are parameterized implicitly by the security parameter. Let $s \leftarrow\!\$\ S$ denote sampling a variable s uniformly at random from a set S. Let $x \leftarrow\!\$\ X$ denote the execution of a probabilistic algorithm X followed by assigning the output to a variable x. We write $x \leftarrow X$ if the algorithm X is deterministic. Let $\perp$ denote an special error symbol that is not included in any set in this paper. We use _ to denote a variable that is irrelevant.

In the presentation of this work, we focus on the definition of nonce-based AEADs. The main reason, besides reducing complexity, is that randomness- and counter-based AEADs can be cast as instances of nonce-based AEADs. Looking ahead, all our symbolic models will cover nonce-based AEADs and can then trivially be used to model randomized or nonce-based AEADs.

Definition 1 ([50]). *Let* Key, Nonce, Header, Message, Ciphertext *respectively denote the space of keys,*

nonces, headers (aka. associated data), messages, and ciphertexts. An authenticated encryption with associated data scheme $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *is a tuple of algorithms where*

- KGen *the key generation algorithm outputs a symmetric key* $k \in \mathsf{Key}$*, i.e.,* $k \leftarrow\!\!\$\ \mathsf{KGen}()$*.*
- Enc *the encryption algorithm inputs a key* $k \in \mathsf{Key}$*, a nonce* $N \in \mathsf{Nonce}$*, a header* $H \in \mathsf{Header}$*, and a message* m *and (deterministically) outputs a ciphertext* c*, i.e.,* $c \leftarrow \mathsf{Enc}(k, N, H, m)$*.*
- Dec *the decryption algorithm inputs a key* $k \in \mathsf{Key}$*, a nonce* $N \in \mathsf{Nonce}$*, a header* $H \in \mathsf{Header}$*, and a ciphertext* $c \in \mathsf{Ciphertext}$ *and deterministically outputs a message* $m \in \mathsf{Message} \cup \{\perp\}$*, i.e.,* $m \leftarrow \mathsf{Dec}(k, N, H, c)$*.*

Over such schemes, the N, H and ciphertext c need to be sent over the network[1], and the correctness of the scheme requires that the decryption of a ciphertext with the same parameters N, H, k indeed returns the plaintext. We assume that the decryption with inputs outside the corresponding spaces must output $\perp$.

The two core security guarantees are integrity and privacy.

Definition 2 (Privacy [50])**.** *We say an* $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *is* ϵ-$\mathsf{IND\$\text{-}CPA}$ *secure, if the below defined advantage of any attacker* $\mathcal{A}$ *against* $\mathrm{Expr}^{\mathsf{IND\$\text{-}CPA}}_{\mathsf{AEAD}}$ *experiment in Fig. 1 is bounded by:*

$$\mathsf{Adv}^{\mathsf{IND\$\text{-}CPA}}_{\mathsf{AEAD}} := |\Pr[\mathrm{Expr}^{\mathsf{IND\$\text{-}CPA}}_{\mathsf{AEAD}}(\mathcal{A}) = 1] - \frac{1}{2}| \leq \epsilon$$

Definition 3 (Integrity [50])**.** *We say an* $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *is* ϵ-$\mathsf{CTI\text{-}CPA}$ *secure, if the below defined advantage of any attacker* $\mathcal{A}$ *against* $\mathrm{Expr}^{\mathsf{CTI\text{-}CPA}}_{\mathsf{AEAD}}$ *experiment in Fig. 2 is bounded by:*

$$\mathsf{Adv}^{\mathsf{CTI\text{-}CPA}}_{\mathsf{AEAD}} := \Pr[\mathrm{Expr}^{\mathsf{CTI\text{-}CPA}}_{\mathsf{AEAD}}(\mathcal{A}) = 1] \leq \epsilon$$

Both for integrity and privacy, we can define two security variants, $\mathsf{CTI\text{-}CCA}$ and $\mathsf{IND\$\text{-}CCA}$, based on whether the attacker also has access to a decryption oracle during the experiment, see e.g., the definition of the experiment for $\mathsf{CTI\text{-}CCA}$ in Fig. 2. We summarize the well-known relations in Fig. 3, and give the corresponding theorems in Appendix B .

2.2 Historical real-world protocol attacks exploiting AEADs

Beside the well-studied privacy and integrity, some recent attacks against practical application protocols suggest that the underlying AEADs need an evolution to meet stronger security guarantees. We identify six main classes of these protocol attacks that have occurred in the wild and briefly describe their high-level requirements.

A1 Nonce reuse attacks - While nonces are expected to be used only once, this can fail in practice for three main reasons. First, protocol designs might aim to establish nonces, but their complex state machines may hide edge cases in which they are in fact reused, as in e.g., WPA2 [56]. Second, the generation of nonces might involve external sources, which may be unreliable, e.g., YubiKey [41] or Trustzone [54]. Third, implementations may be flawed. For example, the Zerologon attack[61] notably exploited the fact that the nonce underlying the AES-CBF8 mode in Microsoft Netlogon protocol is a constant string of zero bits. The encryption of a block of zero bits equals to 0^{16} with probability $1/256$ for any key k, breaking the authentication of windows servers.

A2 Padding oracle attacks [57] - Many AEADs and symmetric encryption schemes are constructed from block ciphers and require the length of input messages to be multiple of a fixed value. Messages whose length is not a multiple are extended before encryption using a so-called *padding scheme*. These can enable plaintext recovery attacks if the attacker has a way to determine if a ciphertext is correctly padded or not, e.g., through timing leaks or error messages. Padding oracle attacks have found on many protocols, including SSL [16], IPSec [24], and GPG [46].

A3 SSH fragmentation attacks [4] - SSH was designed for securing Internet traffic over the unstable channel, where ciphertext blocks in a packet might get lost. The length of a SSH packet is encrypted in its first block. If the number of delivered blocks is less than the length decrypted from the first ciphertext block, no ciphertext integrity is executed.

If an attacker can inject the first ciphertext block and observe the error message reported by the SSH connection, then the plaintext of the transmitted ciphertext can be recovered.

[1] We stress that AEAD schemes can be used offline in practice, where nonces and headers both are hidden from attackers' view. However, this paper focuses on a more common case where the attackers might have access to the nonce and headers, e.g., which are transmitted over network.

A4 Partitioning oracle attacks [42] - Some real-world applications do not sample the AEAD symmetric keys randomly but simply pick users' passwords. Thus, attackers might know a set of possible password candidates and perform brute-force attacks. Even worse, if attackers have access to a partitioning oracle, which tells whether the password of a ciphertext belongs to some known sets, then the password can be recovered exponentially faster.

In practice, attackers sometimes can obtain the partitioning oracle by observing the reply messages responding to a selected ciphertext. This causes the vulnerability of applications in the real world, such as Shadowsocks [42].

A5 Salamander attack [25] - The end-to-end secure messaging provides high security against the surveillance of the server but potentially prevents the server from blocking the abusive messages. To mitigate this, Facebook invents a abuse report mechanism that allows each user to report the received abusive messages from a claimed sender.

However, this mechanism turns out to be broken because a malicious sender could send a single encrypted attachment that would decrypt to both an abusive message and an innocent message under two distinct keys.

A6 Sframe attack [32] - An AEAD scheme authenticates the owners of a symmetric key of a ciphertext rather than the sender's identity. This is especially relevant for group communication, where an AEAD cannot use the shared group key to authenticate the specific sender. To provide sender authenticity in groups, while keeping low bandwidth cost, the IETF SFrame protocol v01 [48] requires senders to sign a portion of the AEAD ciphertext using digital signatures.

Unfortunately, the sender identity authenticity of SFrame mechanism turns out to be broken, since the underlying AEAD schemes, AES-CM-HMAC and AES-GCM, do not provide collision resistance for the unsigned portion. This means, a malicious group member holding the symmetric key can forge the unsigned portion of other group members' ciphertexts.

2.3 Theoretical AEAD frameworks

Apart from the previously outlined classes of real-word attacks linked to AEADs, on the theoretical side, many variants of AEADs have been designed in the past twenty years following the seminal work from [50]. Each of those variants come with their own flavors of properties like, e.g., integrity, confidentiality, nonce-misuse resistance, or robustness, leading to dozens of distinct security definitions. Furthermore, these AEAD variants differ in functionality, with some enabling e.g. ciphertext fragmentation or nonce-hiding. We categorize the main differences between distinct AEAD variants as follows:

F1 does each ciphertext (or a part of it) bind to a set of its encryption inputs? This question motivates the study of a novel (compactly) committing AEAD (ccAEAD , Definition 6) regime as well as various security properties, such as collision resistance (Definition 9) , commitment (Definition 10) , sender binding (Definition 14) , and receiver binding (Definition 15) [2, 8, 25, 30]. Roughly speaking, the collision resistance prevents the collisions between AEAD encryption with different inputs. The commitment ensures that each valid AEAD decryption indicates the agreement on a subset of its encryption/decryption inputs. The sender- and receiver binding properties are relevant in the abuse-reporting scenarios. While the sender binding allows every ccAEAD receiver to report abusive messages, the receiver binding prevents malicious receivers from framing honest ccAEAD senders. We give their full definitions in Appendix B - motivated by **A5**.

F2 can we find collisions on valid decryption inputs for the same ciphertext? This question motivates the study of a novel property called robustness (Definition 11) [1, 42]. Briefly speaking, robustness prevents attackers from having a single ciphertext decrypt to multiple distinct valid messages on different inputs. - motivated by **A4**, **A6**.

F3 is the AEAD supporting fragmentation of the ciphertexts? That is, can we start decrypting chunks of data before having verified the whole ciphertext?

F4 is the decryption atomic, or split into a decryption and an integrity check? [7] - motivated by **A2**.

F5 is the AEAD nonce-hiding? That is, is the nonce explicitly needed for the decryption, or is it included and hidden inside the ciphertexts? [9, 17]

F6 is the AEAD nonce-misuse resistant? [31] Must a nonce be used once strictly, or can repeat? - motivated by **A1**.

Concrete AEAD	Integrity and Privacy	Full Collision Resistance	Nonce Misuse Resistance
XSalsa20-Poly1305	●	✗ [2]	✗ Xor of plaintexts
AES-GCM	✓ [33, 43]	✗ [25]	✗ Forgeability + xor of plaintexts
ChaCha20-Poly1305	✓ [49]	✗ [2]	✗ Xor of plaintexts
OCB3	✓ [11, 40]	✗ [2]	✗ Forgeability + equality of blocks
EtM (unrelated keys)	✓ [50]	✗ [30][4]	✗ Encryption dependent
AES-CCM	✓ [29, 36]	●	✗ Xor of plaintexts
AES-EAX	✓ [10, 45]	●	✗ Xor of plaintexts
EtM (related keys)	✓ [50]	✓ [30]	✗ Encryption dependent
CAU-C4	✓ [8]	✓ [8]	✗ Forgeability + Xor of plaintexts
AES-GCM-SIV	✓ [31, 34]	✗ [2]	✓ [31]
CAU-SIV-C4	✓ [8]	✓ [8]	✓ [8]

✓ : proven in the cited work(s). ● : we conjecture that this holds, but do not know of a proof.
✗ : does not hold, with reference or explanation of counterexample.

Table 1: AEADs (in)-security guarantees: *Integrity and Privacy* refers to IND$-CPA and CTI-CPA. *Full Collision Resistance* refers to Definition 4. For *Nonce Misuse Resistance* we indicate the potential impact of reusing nonces if the AEAD scheme does not have this property.

3 Generalizing real-world AEAD (in)security for systematic analysis

In this section we develop systematic generalizations of AEAD security and weaknesses, which form the foundation of the symbolic models that we will design in Section 4.

In the previous section we recalled attack classes and security frameworks from the literature. However, these do not exist within a single systematic framework, and basing our symbolic modeling on them would lead to incomparable and ad-hoc models. For our more systematic approach, we first identify the core properties of concrete real life AEAD schemes that lead to the attacks: privacy and integrity, collision-resistance, and nonce-misuse resistance. We show the relevance of these four points by:

- providing in Table 1 the security and weaknesses of many widely deployed AEADs with respect to those core properties;
- illustrating how collision resistance theoretically allows covering notions from **F1** and **F2** in Section 3.2; and
- summarizing the concrete existing collision capabilities for deployed AEADs.

In particular, we identify the collision resistance property as a central concern, which we then investigate first from the theoretical and then the practical point of view.

3.1 Core properties

Stepping back from the many theoretical definitions, we identify three main causes for the protocol attacks:

- **A1** comes from a *misuse of nonces*.
- **A2** and **A3** from a *decryption misuse*, where the decryption is not atomic but performed in two steps, in which case we lose the integrity and privacy guarantees.
- **A4**, **A5**, and **A6** actually all stem from a lack of *collision-resistance*.

This leads us to summarizing the concrete security guarantees for AEADs in three categories:

- **privacy** and **integrity** - the core guarantees that we defined previously, and are expected to be met by all AEADs. This is what is lost under decryption misuse.
- **collision-resistance** - this guarantee hinders attackers from coming up with collisions over the output of Enc, i.e. find two distinct sets of inputs $\vec{i}_1$ and $\vec{i}_2$ such that $\mathsf{Enc}(\vec{i}_1) = \mathsf{Enc}(\vec{i}_2)$.
- **nonce-misuse resistance** - this guarantees that using a weak nonce twice or the same nonce for distinct message does not lead to a compromise.

With respect to those core properties, we provide in Table 1 the security and weaknesses of many widely deployed AEADs. In addition to the concrete constructions, we also provide in this table the generic constructions of AEAD such as Encrypt then Mac (EtM), whose security guarantees depend on the concrete encryption and MAC algorithm instantiations. For the generic construction EtM, we distinguish two cases based on whether the encryption and mac keys are related, e.g. derived from k with a key derivation function, or unrelated, e.g. simply the first and the second half of the input k.

Notably, while all of AEADs in the table do provide integrity and privacy (otherwise they would not be used), only some of them tolerate that a single nonce is reused twice for different messages. Moreover,

we can also observe that the picture for collision-resistance is very disparate and many deployed schemes do not meet it.

The nonce-misuse, privacy, and integrity properties are now well-understood in the community. In contrast, the collision part is more nascent: there are multiple variants for it on the security notions side, and in practice the concrete weaknesses have not been systematized. In this section, we carry on clarifying the theoretical and practical implications of collision-resistance of AEADs.

3.2 Generalizing AEAD collision resistance and relations

We consider the CMT-4 definition in [8] as a natural definition for full collision resistance and recall it below. Roughly speaking, full collision resistance means that each AEAD ciphertext can only be computed by unique input. While this appears as a strong property, its absence may introduce surprising behaviors for some protocols. Looking forward, this is illustrated by some known attacks or our case-studies, which seem to indicate that despite its strength, full collision resistance is a meaningful and desirable notion.

Definition 4 (Full Collision Resistance)**.** *We say an* $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *has* ϵ*-full collision resistance (or* ϵ-full-CR *), if the below defined advantage of any attacker* $\mathcal{A}$ *against the* $\mathrm{Expr}^{\mathsf{full\text{-}CR}}_{\mathsf{AEAD}}$ *experiment in Fig. 4 is bounded by*

$$\mathsf{Adv}^{\mathsf{full\text{-}CR}}_{\mathsf{AEAD}} := \Pr[\mathrm{Expr}^{\mathsf{full\text{-}CR}}_{\mathsf{AEAD}}(\mathcal{A}) = 1] \leq \epsilon$$

$\mathrm{Expr}^{\mathsf{full\text{-}CR}}_{\mathsf{AEAD}}$:

1 $\Big((k_1, N_1, H_1, m_1), (k_2, N_2, H_2, m_2)\Big) \leftarrow\$ \mathcal{A}()$
2 **if** $\perp \in \{k_1, N_1, H_1, m_1, k_2, N_2, H_2, m_2\}$ **or** $(k_1, N_1, H_1, m_1) = (k_2, N_2, H_2, m_2)$
3 **return** 0
4 $c_1 \leftarrow \mathsf{Enc}(k_1, N_1, H_1, m_1)$, $c_2 \leftarrow \mathsf{Enc}(k_2, N_2, H_2, m_2)$
5 **return** $[\![c_1 = c_2]\!]$

Figure 4: full-CR security for an $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$.

Relationship with existing frameworks It turns out that this notion of collision resistance, while straightforward, is enough to cover in practice multiple notions of the literature from [2, 8, 28, 30, 42]. Informally, these notions are:

- *tidyness* - for a fixed key, is the encryption function the inverse of the decryption one? It implies that collisions over encryptions or decryptions are equivalent.
- *commitment* (CMT-l and CMTD-l for $l \in \{1, 3, 4\}$ [8]) - can we find collisions either over the encryption or the decryption, with different parts of the inputs being allowed to stay fixed based on l? In order to capture more variants in this property class that are not included in [8], in this paper we rename CMT-l to *collision resistance* (X-CR) and CMTD-l to *input bound ciphertexts* (X-IBC), where $\mathsf{X} \subseteq (k, N, H, m)$[2] denotes the inputs that a AEAD scheme commits to. We recall the definitions of X-CR and X-IBC respectively in Definition 9 and Definition 10 and their relations in Theorem 3 . In particular, the full-CR in Definition 4 is identical to (k, N, H, m)-CR in Definition 9 [3].
- *full robustness* (FROB [28]) and *even fuller robustness* (eFROB [30]) - is any attacker able to compute a ciphertext that decrypts correctly under two distinct inputs? This notion was initially defined for randomized AEADs. In this paper, we extend the robustness notions FROB and eFROB for randomized AEADs to a unified notion X-FROB for nonce-based AEADs in Definition 11 , where $\mathsf{X} \subseteq (k, N, H, m)$ denotes the degree of robustness. Moreover, we prove that X-FROB is equivalent to X-IBC in Theorem 5 .
- *key committing* KC security [2] - is any attacker able to compute a ciphertext that decrypts correctly under different keys but same nonce? In this paper, we recall the KC definition in Definition 12 and show that X-FROB with $k \in \mathsf{X}$ implies KC, while the reverse does not hold, in Theorem 6 .

[2]Here, we slightly abuse notation and use $(\cdot)$ to denote a set. Thus, by $\mathsf{X} \subseteq (k, N, H, m)$ we mean that X is a subset of the set (k, N, H, m). For a single element set, we sometimes also omit the parenthesis and regard it as a single element. For instance, we write $k \in \mathsf{X} \Leftrightarrow (k) \subseteq \mathsf{X}$.

[3]In Theorem 4 we will show that (k, N, H, m)-CR implies all variants, which motivated our choice to abbreviate (k, N, H, m)-CR to full-CR.

- *multi-key collision resistance* (MKCR) [42] - is any attacker able to compute a ciphertext that decrypts correctly under multiple keys but same nonce and header? The MKCR is parameterized by the number of distinct keys $\kappa \geq 2$. In this paper, we focus on the simplified case where $\kappa = 2$. We recall the MKCR definition in Definition 13 and show that KC implies the simplified MKCR, while the reverse does not hold, in Theorem 7 .
- *receiver binding* (r-BIND) [30] - is any attacker able to compute a ciphertext that can be verified under the different header and message? This notion was initially defined for a variant of compactly committing AEAD (ccAEAD), and showed how it can be instantiated for instance with an Encrypt then Mac construction[4]. Note that [30] also introduces how to transform any AEAD to ccAEAD by a "*traditionally committing encryption*" approach (ccAEAD[AEAD]). In this paper, we recall the r-BIND definition in Definition 15 and show its relations with other security notions (in this list) in Theorem 8 and Theorem 9 .

We provide the full relations between the above notions in the theorem below, which is illustrated in Fig. 5. We give the detailed proofs in Appendix B . While some of the relations were conjectured before ([8]), we are the first to provide the full proofs, as well as provide generalizations of some notions to enable a comparison.

Theorem (Informal)**.** *For any* AEAD *scheme, we have that*

1. X-FROB *implies* X-CR *for any* $\mathsf{X} \subseteq (k, N, H, m)$. *If* AEAD *is tidy, the reverse also holds. See Theorem 3 .*
2. X-FROB/X-CR/X-IBC *resp. implies* X′-FROB/X′-CR/ X′-IBC *for any* $\mathsf{X}' \subseteq \mathsf{X} \subseteq (k, N, H, m)$. *See Theorem 4 .*
3. X-FROB *and* X-IBC *are equivalent for any* $\mathsf{X} \subseteq (k, N, H, m)$. *See Theorem 5 .*
4. k-FROB *implies* KC *but not in reverse. See Theorem 6 .*
5. KC *implies* MKCR *but not in reverse. See Theorem 7 .*
6. (H, m)-FROB *of* AEAD *implies* r-BIND *of* ccAEAD[AEAD]. r-BIND *of* ccAEAD[AEAD] *implies* X-FROB *of* AEAD *for any* $\mathsf{X} \subseteq (H, m)$. *See Theorem 8 .*
7. *Neither* KC *nor* MKCR *of* AEAD *implies* r-BIND *of* ccAEAD[AEAD]. *The reserve is same. See Theorem 9 .*

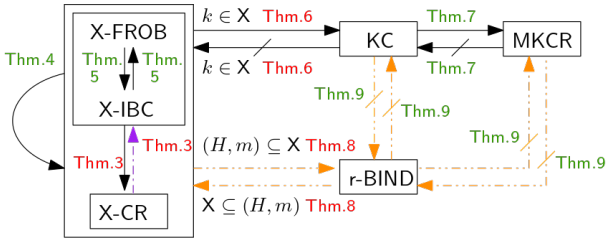


Figure 5: The relation between collision related properties for AEAD with key space Key. The black arrow $\rightarrow$ indicates the general implication. The purple dash-dotted arrow -·-▸ indicates the implication for tidy AEAD. The orange dash-dot-dotted arrow -··-▸ indicates the implication for ccAEAD[AEAD]. The X in the figure is a subset of (k, N, H, m), i.e., $\mathsf{X} \subseteq (k, N, H, m)$. The theorems highlighted with red color are claimed or proven in other papers. The theorems highlighted with green color are part of our third contribution.

Note that Theorem 3 was proven in [8]. Theorem 6 and Theorem 8 were respectively claimed in [8] and [30] without giving any proofs. Proofs for Theorem 4, Theorem 5, Theorem 7 and Theorem 9 are part of our third contribution. Recall that we have (k, N, H, m)-CR = full-CR in this figure. This theorem indicates that the full collision resistance implies all other existing notions in this figure under the tidyness assumption, which is in fact met by all classical constructions.

3.3 Collision attacks on deployed AEADs

In general, any kind of collision between two ciphertexts can lead to a security issue, and we will advocate that general use AEADs should be fully resistant to collisions. However, many popular deployed AEADs do not meet the full collision resistance, as shown in Table 1. Below, we recall the known attacks against various kinds of collision resistances of different AEAD schemes in the literature.

[4] [30] proposes to use HMAC-SHA256 to instantiate a keyed random oracle, which is technically false without an additional assumption, as $HMAC(k, m) = HMAC(H(k), m)$ whenever k is bigger than 256 bits.

AEAD	nColl	KeysColl
AES-GCM	[25]	[2, 25, 30, 42]
AES-GCM-SIV	[25]	[2, 25, 30, 42]
ChaCha20-Poly1305	[25]	[2, 25, 30, 42]
Encrypt-then-MAC (EtM)	[25]	[25, 30]

Table 2: Effective attacks against collision resistance of several AEADs in the literature. nColl describe collisions where, for given keys and a header, the attacker uses brute-force over the nonce to produce colliding ciphertexts. In KeysColl, the attacker brute-forces, given a nonce and header, over the keys.
For the generic Encrypt-then-MAC paradigm we give concrete attacks for CTR, OFB, CBC, and CFB modes in Appendix A .

1. r-BIND: [30] shows a generic attack against any EtM construction with unrelated keys by finding the second key that causes collision by . This attack also applies to real-world modes using Carter-Wegman MACs, e.g., GCM and ChaCha20-Poly1305. [25] shows a concrete attack against AES-GCM and OCB by finding the nonce that causes collision and sketches an faster attack by doing birthday attack on keys. Moreover, at the hand of a corollary of Theorem 1 in [51], [25] claims that this attack also applies to any so-called *rate-1* AEAD, that is, "one blockcipher call per block of message" [25]. This potentially indicates the vulnerability of AES-GCM-SIV and ChaCha20-Poly1305 and any EtM constructions.
2. KC: [2] extends the known attack in [25] against AES-GCM to new proof-of-concept attacks against several commonly used AEAD, including AES-GCM, ChaCha20-Poly1305, AES-GCM-SIV, and OCB3. This attack shows how to create ciphertext collision on two distinct keys. Then, [2] also shows that their new attacks also make impacts in some real-world scenarios, such as the binary polyglots setting.
3. MKCR: [42] shows a novel partitioning oracle attack that feasibly breaks the MKCR security with parameter $\kappa \geq 2$ of widely used AEAD schemes, including AES-GCM, AES-GCM-SIV, ChaCha20-Poly1305, and XSalsa20-Poly1305.
4. X-CR and X-IBC: [8] finds that all above attacks also break the k-CR and -IBC security of respective AEAD schemes. Thus, AES-GCM, AES-GCM-SIV, XSalsa20-Poly1305, and ChaCha20-Poly1305 and OCB are all k-CR insecure, i.e., CMT-1-insecure in [8].

4 Symbolic models for automated verification

We next describe how to, using the generalizations we developed in the previous section, develop symbolic models for AEADs that encompass many of the essential weaknesses from Section 2.2. For each essential weakness, we will develop a specific model that gives the attacker the capability to use the given weakness. Analyzing a protocol with those attacker capabilities enabled will then allow us to automatically find attacks on protocols that may rely on subtle AEAD weaknesses.

Our models cover:

- *collisions* Coll- covering **A4**, **A5**, **A6**, and definitions from **F1** and **F2**.
- *nonce reuse* NR- covering **A1** and **F6**.
- *decryption misuse* Forge- covering **A2**, **A3**, **F3** and **F4**.

Some modern protocols, like [25] or [48], rely on additional features of AEADs that we cover in a modular fashion:

- *explicit tag* Tag- for most AEADs, one can extract a verification tag from the ciphertext, needed to model protocols like [48] for **A6**.
- *explicit commit* Com- to extract a value from the ciphertext committing to the inputs of the encryption. Needed to model protocols like [25] covering **A5** and **F1**.

Collisions can then be lifted to the tag or the commit in a modular fashion, and are essentially only impacting on the complexity of mounting concrete attacks.

We additionally build a model Leak that provides an explicit capability to reveal the nonce used for encryption to the attacker. Not sending out the nonce by default but using a dedicated functionality allows accounting for nonce hiding AEADs covering **F5**. While we cannot claim completeness of our models w.r.t. to all possible AEAD weaknesses that may arise in the future, we provide a set of models based on our analysis of the real-world security of AEADs Section 3 that covers most practical attacks.

We develop and specify the previously enumerated models of AEADs in the *symbolic model* of cryptography, an abstract model used in the formal methods community to express and automate the

analysis of cryptographic protocols in Section 4.1. We then present symbolic models of the before-mentioned AEAD weaknesses in Section 4.2.

4.1 The symbolic model of cryptography

The symbolic model uses function symbols to denote algorithms, and capture their properties through equations. For instance, an encryption is modeled by two binary function symbols $\mathtt{enc}$ and $\mathtt{dec}$, with the equation:

$$\mathtt{dec}(\mathtt{enc}(k, m), k) = m$$

Note that the randomness or nonce is not explicit in this classical modeling. And crucially, in the symbolic model, only the equations that are explicitly specified imply equalities. This results in the so-called perfect cryptography assumption: in the previous example, the encryption is perfect, in the sense that given $\mathtt{enc}(k, m)$ and not k, the attacker learns absolutely nothing about m or k as it cannot apply the decryption equation. The attacker cannot change the content of the message, and no collisions will exist.

While the previous assumption may seem too restrictive, it allows for highly automated tools which are one of the strengths of the symbolic model. These tools were already successfully used to automatically find attacks on protocols [22, 59] and aid standardization processes to avoid design-level flaws [20, 23, 47].

In recent years, effort was put into improving the symbolic model with better and more fine-grained support for cryptographic primitives. [21] introduced a stronger version of symbolic Diffie-Hellman group models, while [18] and [35] gave more fine-grained models of cryptographic hash functions and digital signatures, respectively.

4.2 Symbolic AEAD models

We first explicitly model all the input parameters, making the $\mathtt{enc}$ and $\mathtt{dec}$ having four inputs, $\mathtt{enc}(k, n, h, m)$. Then, we model the multiple weaknesses previously discussed. While we focus on providing models for nonce-based AEADs, as it is the most fine-grained model of AEADs, it is easy to derive from them models for counter-based or randomized AEADs. They can typically be modelled by removing the explicit nonce as $\mathtt{enc}(k, h, m)$, and all equations or capabilites given in the following and that do not directly relate to the nonce can be transposed to this case.

Practical Collision models Coll We start by adding collision capabilities that match the known real-world collision capabilities. When using these models reports an attack on the protocol in one of the automated tools, we can then investigate its feasibility in practice based on the concrete AEAD used and the message encodings by referring to Section 3, and in particular Table 2.

We start with the capability that can be reasonably computed on many AEADs and was shown to be practical by [25] for Facebook's Message Franking protocol. As an example, consider the scenario where an attacker tries to produce some colliding ciphertexts given two keys. One option would be to brute-force over the nonce for a fixed header, e.g., an empty header. If successful, the attacker would have a ciphertext that could be decrypted to distinct plaintexts under a common nonce and header using the two keys. We model this nonce finding algorithm in the symbolic model as an additional function symbol $\mathtt{c}_n$, modeling the colliding nonce, which will take as input all the given parameters the collision depends on. We then add the collision model nColl:

$$\begin{aligned}&\mathtt{enc}(k_1, \mathtt{c}_n(k_1, k_2, h, m_1, m_2), h, m_1)\\ &= \mathtt{enc}(k_2, \mathtt{c}_n(k_1, k_2, h, m_1, m_2), h, m_2)\end{aligned} \qquad \text{(nColl)}$$

Another widespread collision capability is captured by adding two function symbols $\mathtt{c}_k^1$ and $\mathtt{c}_k^2$ with the collision model KeysColl:

$$\begin{aligned}&\mathtt{enc}(\mathtt{c}_k^1(n, h, m_1, m_2), n, h, m_1)\\ &= \mathtt{enc}(\mathtt{c}_k^2(n, h, m_1, m_2), n, h, m_2)\end{aligned} \qquad \text{(KeysColl)}$$

To check whether a potential attack found using KeysColl might be feasible, refer to Table 2. Whereas for KeysColl the attacker needs to produce a collision on the AEAD for a fixed nonce and header, the same kind of collision appears also to be feasible in the case where one of the keys is already fixed. We model this slight variation of KeysColl as well (KeyColl).

Notice that we, for instance, set that the two colliding encryptions may necessarily use the same nonce and header in those equations. This is caused by many existing protocols implementing that nonces and associated data can be computed independently by the parties, or that they cannot be sent out twice with distinct values.

Generic Collision models FullColl The previous collision models allow to efficiently check for the collisions that are most likely to be practical for existing AEADs and given their use in protocols. While obtaining an attack in those models is instantly interesting, we may miss some future practical attacks. Indeed, as illustrated by Table 1, most AEADs do not meet the full collision resistance property. As we in fact do not know if they meet any kind of collision resistance properties as no proofs exists for many of them, it is possible that in the coming years, new practical ways of building collisions on the existing AEADs are discovered. As such, from a security point of view, for any non collision resistant AEADs, it is prudent to consider that many more collisions are possible than currently practical.

Our approach makes it easy to define such a prudent model, and capture all attacks that may be possible on a protocol if the AEADs is not fully collision resistant. We do this by allowing more collisions and changing which part of the encryption input is fixed on both sides, and which part the attacker is brute-forcing over.

$$\begin{aligned} &\mathtt{enc}(k,n,h,m) \\ &= \mathtt{enc}(\mathsf{gen\text{-}c}_k(n,n_2,h,h_2,m,m_2),n_2,h_2,m_2) \end{aligned} \quad \text{(FullKeyColl)}$$

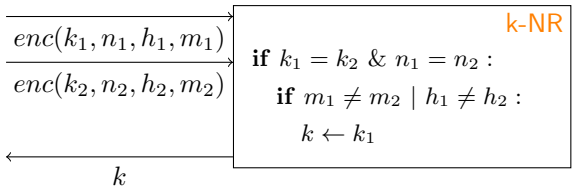
$$\begin{aligned} &\mathtt{enc}(k,n,h,m) \\ &= \mathtt{enc}(k_2,n_2,h_2,\mathsf{gen\text{-}c}_m(k,k_2,n,n_2,h,h_2)) \end{aligned} \quad \text{(Full-mColl)}$$

$$\begin{aligned} &\mathtt{enc}(k,n,h,m) \\ &= \mathtt{enc}(k_2,\mathsf{gen\text{-}c}_n(k,k_2,h,h_2,m,m_2),h_2,m_2) \end{aligned} \quad \text{(Full-nColl)}$$

$$\begin{aligned} &\mathtt{enc}(k,n,h,m) \\ &= \mathtt{enc}(k_2,n_2,\mathsf{gen\text{-}c}_h(k,k_2n,n_2,m,m_2),m_2) \end{aligned} \quad \text{(Full-adColl)}$$

With FullKeyColl, Full-mColl, Full-nColl, and Full-adColl, we capture the capability of an attacker to find collisions by just finding one distinct k, n, h, or m, respectively. These models may cover collisions that are impractical as their main purpose is to check whether the analyzed protocol relies on collision resistance of AEAD schemes. Indeed, if we get an attack in such a model, it implies that a strong collision resistance notion is needed to prove the security of the protocol in the computational model. Further, and as we see later in Section 5, some of these attack may even be practical and could not have been easily discovered in another way.

Nonce-reuse NR The nonce-reuse issue **A1** is slightly more complex to model, as we cannot capture it using an equation. We thus have to use a less classical way to model primitives: we model the attacker capability by providing access to an additional process, or oracle, that does the following:



In this oracle, the attacker can provide two ciphertexts. If those ciphertexts are encrypted under the same key and nonce but differ in either header of message, the attacker learns the secret encryption key. Similar to the collision model, Coll, we included a model of this process into our set of AEAD models and call it k-NR. This process models the strongest possible leak, namely leakage of the secret key. We can also make it more fine-grained by leaking, e.g., $m_1 \oplus m_2$ instead of k. As not all tools in the symbolic model provide support for exclusive-or like equations, we modeled an over-approximation m-NR, which leaks both m_1 and m_2 as an example. Note that with these kinds of oracle-like models, the concrete leaked values can be decided by the capabilities of the chosen tool and the actual weaknesses listed in Table 1.

Decryption Misuses Forge Some protocols, notably SSH, that allow for ciphertext fragmentation, also use AEADs in a non-recommended way by splitting the atomic `dec` operation into verification and decryption. This may also be the case for protocols building their own AEAD based on the EtM construction. In such a case, instead of `dec` that checks integrity, we use a weak decryption function `w-dec` and a verification function `verify`, with the equations:

$$\begin{aligned} \mathtt{w\text{-}dec}(k,n,h,\mathtt{enc}(k,n,h,m)) &= m \\ \mathtt{verify}(\mathtt{enc}(k,n,h,m),k,n,h,m) &= \mathtt{true} \end{aligned}$$

We model the fact that the decryption is weak by making decryption succeed on messages forged by the attacker using the `forge` algorithm:

$$\texttt{w-dec}(k_2, n_2, h_2, \texttt{forge}(\texttt{enc}(k, n, h, m), m_2)) = m_2$$

Remark that a limitation of this Forge model is that the attacker cannot compute a valid ciphertext for some function of the message m, which is sometimes possible. Assume that for a given protocol we know that plaintexts are pairs of elements, denoted by $< x, y >$, we can also add dedicated forgery rules:

$$\texttt{w-dec}(k_2, n_2, h_2, \texttt{forge}_1(\texttt{enc}(k, n, h, < m_1, m_2 >))) = m_1$$
$$\texttt{w-dec}(k_2, n_2, h_2, \texttt{forge}_2(\texttt{enc}(k, n, h, < m_1, m_2 >))) = m_2$$

If the encryption is XOR based, the attacker should also be able to encrypt at this stage any XOR of a value to the ciphertext. This limitation notably implies that we cannot cover in general premature release of ciphertexts or the SSH fragmentation attacks. While we can do this for particular cases as illustrated, lifting this limitation generically in the symbolic model requires advances in the existing tools and symbolic techniques that we consider out of scope for this work.

Explicit Tag Tag Despite the recommendations, some protocols do not use AEADs only through a decryption and encryption API, but actually rely on some more low-level detail. For instance, schemes rely on the fact that the ciphertext is often a pair (encryption, tag), where the encryption is a basic symmetric encryption of the message and the tag is what provides the integrity. Instead of going to such a low-level, which would be AEAD dependent, we capture this possibility modularly by adding a new function symbol `get_tag`, that inputs ciphertext $\texttt{enc}(k, n, h, m)$. We can then model collisions over the tags, by adding a variant of each of the previous collision equations over the tag, with, e.g., nTag being:

$$\begin{aligned}&\texttt{get_tag}(\texttt{enc}(k_1, \texttt{c}_n(k_1, k_2, h, m_1, m_2), h, m_1))\\ =\ &\texttt{get_tag}(\texttt{enc}(k_2, \texttt{c}_n(k_1, k_2, h, m_1, m_2), h, m_2))\end{aligned}$$

Reasoning about explicit tags allows for a top-down approach rather than bottom up. For example, it allows us to ignore implementation details, such as to which side of the encryption the tag is concatenated.

Explicit commitment Com We model compactly committing AEADs by adding a `get-commit` function symbol similar to the `get_tag`. Once again, this allows for a modular model of compactly committing AEADs, where we only specify that a commitment can be extracted, but do not specify how. This extraction can be combined with the collision capabilities to model non-committing AEADs, as a collision on the ciphertexts directly translates to a collision on the commitment. Modeling the fact that the collisions are only on the commitment in a more fine grained way would be possible, but would not yield better attack finding capabilities as they are covered by the ciphertexts collisions. Further, it appears that collisions on the ciphertext or the commitment only differ in the ease of mounting attacks, the commitment being smaller and easier to manipulate than the whole cipher.

Nonce-Leaking Leak Following **F5**, we capture that an AEAD may not hide the nonce by adding a nonce extraction function symbol `get_nonce` along with the needed equation:

$$\texttt{get_nonce}(\texttt{enc}(k, n, h, m)) = n$$

This equation can now be also used instead of sending the nonce to the network explicitly.

Concrete encodings for tools Previously, we presented general equations that can be used to capture multiple AEAD weaknesses in symbolic models. However, symbolic analysis tools often have restrictions on the type of equations that are supported. The most efficient class of equations that are supported by tools satisfy the so called *subterm convergence* property: the right hand side of the equation only contains terms that occur as subterms in the left-hand side. This is not the case for all equations we introduced previously. However, all of them can be expressed in an equivalent fashion using only subterm convergent equations. For instance, for the nColl equation the same attacker capability can be captured with the following equations, where we introduce a new function symbol `ct` as an encoding artifact:

$$\begin{aligned}&\texttt{enc}(k_1, \texttt{c}_n(\texttt{ct}(k_1, k_2, h, m_1, m_2)), h, m_1)\\ &\qquad = \texttt{ct}(k_1, k_2, h, m_1, m_2)\\ &\texttt{enc}(k_2, \texttt{c}_n(\texttt{ct}(k_1, k_2, h, m_1, m_2)), h, m_2)\\ &\qquad = \texttt{ct}(k_1, k_2, h, m_1, m_2)\end{aligned} \qquad \text{(n-subtermColl)}$$

4.3 Automated analysis methodology

We now have a set of models to capture multiple weaknesses of AEADs. To analyze a protocol, the following steps should be followed with the symbolic tool of choice:

1. Verify the protocol in all possible threat models (malicious participants, AEADs weaknesses)
2. If there is an attack based on collisions or nonce misuse, check which AEAD the protocol is using and whether it has the corresponding weakness (Table 1);
3. If an attack is from KeyColl, KeysColl, or nColl, use Section 3.3 to check whether the use AEAD is non collision resistant. If it is not, check Table 2 to evaluate if the attack is practical or not.
4. If an attack is from one of the over-approximated capabilities FullKeyColl,Full-mColl,Full-nColl, Full-adColl, there are two consequences:
   - Collision Resistance is probably needed to prove the protocol computationally secure.
   - The attack may however be impractical, and one needs to check the trace to see if the attacker can have enough control over the ciphertext inputs to create a collision.

False attacks In the previously outlined methodology, we say that an attack found in a symbolic model may not be a true attack. To provide more details: during modeling, we sometimes on purpose overapproximate the possible AEAD weaknesses, both to completely rule out classes of attacks or detect subtle attacks. This may indeed lead to us finding "false attacks" that are possible on the design of the protocol but not on all possible implementations for all concrete primitives. In such cases, when by following the methodology we get an attack trace, we have to carefully inspect it, and identify precisely whether it could lead to an attack on the concrete protocol. This typically depends on details of the plaintext encoding used by the protocol (json, concatenation, lengths, etc).

The Tamarin prover Our methodology is generalized enough to not be bound to a specific tool. The tool of choice needs to support custom equational theories and explicit means to express attacker knowledge. These are criteria fulfilled by various state-of-the-art symbolic tools like [14, 26, 38]. We chose the Tamarin prover [44] to demonstrate our methodology, as it offers a straightforward way to add custom equational theories and oracle-like processes needed for NR.

Automated analysis setup We split the models from Section 4 into two general classes:

1. collisions and nonce misuse (Coll, NR, Leak)
2. explicit functionalities (Forge, Com, Tag)

Class 1) corresponds to a set of weaknesses that we can check on any protocol using an AEAD scheme. We build a library of those models and a script that verifies the security of a given protocol against those models. Class 2) only makes sense on protocols that do rely on some explicit functionality, like an explicit commitment. Hence, we only model them in the relevant cases where the protocol relies on these explicit functionalities. For our set of case studies, we want to explore their security guarantees against all our AEAD models. To do so, we implemented a Python script that for a given protocol, automatically executes Tamarin for all possible combinations of threat models, and provides a summary of the secure or insecure scenarios.

When doing this exhaustively, it would mean running Tamarin 2^{10} times for each case study of class 1) and up to 2^{19} times for class 2). We optimize the script by re-using strict implications of some of our models, e.g. FullKeyColl makes the attacker strictly more powerful than KeyColl. Additionally, some models can be restricted to not be used at the same time with others. This reduces the possible model combinations to 2^9 (and up to 2^{13} for class 2).) As this is still a huge number of calls, we can use the same implications mentioned before to do some dynamic pruning. The number of prunable model combinations can vary a lot depending on the case study. The total Tamarin calls that our script automatically made for our case studies can be found in Table 3. Using these implications is useful, both for dynamic pruning and to optimize the search, but also to provide a more compact view of the final results, only displaying non-redundant secure or insecure scenarios. Our script provides us with a summary of the security of a scheme, that can then be formatted in a table as illustrated in Table 4.

A limiting factor in our analysis is the run-time of the protocol models. As the problem of automatically analyzing protocol models is in general undecidable, running Tamarin could lead to non-termination. We deal with this possibility by introducing timeouts into our experiments such that for each Tamarin call, we either find a proof, a potential attack trace, or we have a timeout.

Using our technique, which automatically modifies the model for each of the AEAD model combinations, can lead to non-termination more easily, especially on fragile models that were manually tailored toward

Protocol	AEAD Scheme	Model	Analysis Results	Time (s)	Novel?	Status
YubiHSM [60]	AES-CCM	NR	Key secrecy attack	2	[41]	Fixed
SFrame [48]	AES-GCM, EtM CTR	Tag	Authentication attack	<1	[32]	Fixed
FB Message Franking [27]	AES-GCM	Coll	Content Agreement attack	8	[25]	Fixed
FB Message Franking [27]	AES-GCM	Coll	Framing attack	3	[25, 30]	Fixed
GPG SED [37]	PGP-CFB	Coll	No Content Agreement	<1	✓	Deprecated
GPG SEIPDv2 [37]	AES-OCB	Coll	No Content Agreement	<1	✓	Infeasible
Saltpack [52]	XSalsa20-Poly1305	Coll	No Content Agreement	8	✓	Infeasible
WebPush [55]	AES-GCM	Coll	Server Accountability	8	✓	Reported
WhatsApp [58]	EtM CBC	Coll	No Content Agreement	3	✓	Reported ‡
Scuttlebutt [53]	XSalsa20-Poly1305	Coll	No Content Agreement	3	✓	Reported *

* = Feasibility depends on the collision resistance of XSalsa20-Poly1305 (not in Table 2.) See discussion in Section 5.3 .
‡ = Reported to WhatsApp. Feasibility heavily relies on implementation details, which are not open source.

Table 3: Summary of the main analysis results from our case-studies, illustrating the generality of our models by rediscovering previous attacks and finding new subtleties. In each case, we give the threat model, the used AEAD scheme, the analysis result, as well as the time it took Tamarin to find it. We also give some additional notes on the status of the observation.

termination. We selected the value of the timeout depending on the run-time of the protocol model with the classic AEAD model in use.

As an exhaustive search might not be feasible for every future case study, in Appendix D we describe how one can correctly choose the right AEAD model for a certain protocol model.

Protocol	Threat Model	Content Agreement
GPG SED	Full-mColl ∧ Full-nColl ∧ Full-adColl ∧ Forge ∧ k-NR ∧ m-NR ∧ Leak	✓
	KeyColl	✗
GPG SEIPDv2	FullKeyColl ∧ Full-nColl ∧ Full-adColl ∧ Forge ∧ k-NR ∧ m-NR ∧ Leak	✓
	Full-mColl	✗
Scuttlebutt	Full-adColl ∧ Forge ∧ k-NR ∧ m-NR ∧ Leak	✓
	KeysColl ∨ Full-mColl ∨ nColl	✗

Table 4: Example of how our methodology can, given a protocol and a security property, automatically establish the minimal requirements on the AEAD guarantees for the property to hold. We achieve this by analyzing all possible AEAD models, here applied to content agreement for GPG SED, GPG SEIPDv2, and Scuttlebutt. For each analyzed protocol, we obtain the strongest combination of AEAD models under which content agreement holds (✓), which directly yields minimal requirements on the AEAD. The weakest combinations of AEAD models under which a potential violation of the target property (here content agreement) is found is marked with (✗).

5 Case studies

We demonstrate our symbolic models for automated verification on a set of eight protocols, classified into four categories depending on the analyzed security property:

- Key Secrecy - rediscovering the attack on *YubiHSM* [60]
- Authentication - rediscovering the attack on *SFrame* [48]
- Accountability - rediscovering the attacks on the accountability of the Facebook Message Franking mechanism [27] and finding that the Web Push [55] standard does not provide server accountability.
- Content Agreement - analysis of multiple group messaging and content delivery protocols, namely SaltPack [52], WhatsApp Groups [58], Scuttlebutt [53], and GPG [37].

We tested our methodology on a computing cluster with Intel® Xeon® Gold 6244 CPUs and 1TB RAM against all possible combinations of the threat models from Section 4. We automate this process using a Python program as described in Section 4.3. For each Tamarin call within our script we limit Tamarin to use 4 threads and set the timeout to 60 seconds per Tamarin call.

For the 8 case studies (plus 3 variants) we had a total evaluation time of 17 hours and 29 minutes with a total of 1404 Tamarin calls. We give an overview of the results in Table 3. We show an excerpt of the detailed results in Table 4, while all results are reproducible and can be found in GitHub [19].

Because of space limitations, we only highlight three case studies: The *Facebook Message Franking mechanism* [27], the *Web Push API* [55], and the *Whatsapp group messaging encryption* in Sections 5.1 to 5.3. We refer the reader to Appendix C for details on the remaining case studies and their detailed attack scenarios.

5.1 Facebook Message Franking

In the setting of End-to-End encryption, reporting the abusive behavior of users seems hard to achieve without weakening security guarantees. In 2016, Facebook introduced *Message Franking* [27] to allow reporting of offensive message attachments. The idea is for a recipient of a malicious message attachment to use a cryptographically sound way to prove that it was sent by a specific sender.

[25] found an attack against Facebook's message franking mechanism in 2018. The practical attack they demonstrated involved finding a collision on the used AEAD's ciphertext. As the sender in this scenario was able to choose the cryptographic keys, messages, and the nonce, they showed how to compute two keys k_1 and k_2, two message attachments (for which one is the malicious one) m_1 and m_2, and a nonce n, such that the encryption of m_1 under n and k_1 leads to the same ciphertext as the encryption of m_2 under k_2 and n.

After reporting this attack to Facebook, Facebook immediately patched it. That attack demonstrates the practicality and the impact of collision attacks on real-world schemes.

To show that such attacks can be found on the design level by our analysis, we modeled Facebook's Message Franking mechanism in Tamarin. In the initial setting, with the attacker being a malicious sender, we could automatically find the reported attack in a few seconds using the KeysColl model.

In addition to analyzing the property violated by the initial attack – can a malicious sender avoid detection? – we also studied the converse property – can a malicious receiver create a fake report? The converse property got first reported as a concern by [30].

We thus additionally model a malicious receiver that tries to report an honest user. In this threat model, we, therefore, look at frameability properties. Being able to frame another party can be severe in practice, for instance, by falsely accusing another person of having sent illegal material. After testing our AEAD models against it, we could re-discover a potential attack [30] on the previously property. However, this attack would require finding a collision on the ciphertext for which one key, the nonce, and the ciphertext itself need to be fixed. Unless further weaknesses of AEADs are found, in this particular case over AES-GCM, this attack is, as of now, impractical.

5.2 Web Push

The Web Push protocol provides means for a server to push notifications to clients by depositing an encrypted notification to the push service that will be fetched by the client when they go online. Web Push is standardized at IETF [55], and, for instance, Apple is planning to integrate it into its ecosystem.

Web Push aims to provide confidential push notifications from a server to its users and to ensure certain privacy properties, like the the unlinkability of unique identifiers through the push notification content. Given the wide array of possible applications and concrete use cases, we consider it interesting to check whether the server is accountable or not: can a client prove to a third party that it received a particular push notification from a given server? In contexts where push notifications trigger important actions from a user, protecting users from malicious servers that would try to make the user act and then be able to claim never having done so is crticial. The importance of this guarantee would depend on the actual deployment and usage of Web Push; we are currently in an ongoing discussion with IETF on this point. To include this case in our threat model, we thus consider a malicious server controlled by the attacker and verify if it is possible to upload one notification that could be interpreted in two different ways, for instance, offensively or benignly.

Our analysis reports that this guarantee does not hold w.r.t. Full-mColl: a single notification can be decrypted validly to two different plaintexts, depending on whether we use the current or deprecated public key of the user. As Full-mColl is a strong over-approximation an attack first seemed impractical. After manual inspection of the counterexample trace given by Tamarin, we could see that this theoretical attack carries over to the real world: WebPush relies on AES-GCM, and we can then reuse similar techniques as for Facebook Message Franking attack: concretely, an attacker can brute-force over the salt used to produce a nonce/key pair to encrypt the message to find a collision over the unauthenticated part of the ciphertext, and then inject at the end of the ciphertext the needed block to create a collision over the tag. The practicality of the attack depends on the encoding of the plaintexts, and the severity depends on whether the server being not accountable is critical given the use case.

Protocol	Content Agreement with CR	Content Agreement without CR	Notes
Whatsapp	✓	✗	Practicality depends on plaintext encodings
Scuttlebutt	✓	✗	Practical
GPG SED (to be deprecated)	✓	✗	Practical
GPG SEIPD v1/v2	✓	✓	Only theoretical attacks
SaltPack	✓	✓	Only theoretical attacks
Signal	✗	✗	Pairwise channels, hence no content agreement

Table 5: Content Agreement summary, with and without Collision Resistance (CR): for a set of group messaging applications and multiple recipients message sending mechanism, we summarize whether a given message can yield to different message for multiple users. In this table, we mention that the Signal application does not meet consistency as a side-remark: as Signal uses pairwise channels to send messages in groups, a different message can be sent to each member of the group.

5.3 Content Agreement

We focus now on analyzing the design of multiple messaging mechanisms. We study them in the multiple-recipient setting, trying to answer the following question: *Can a dishonest member of the group send a single message that will be read differently by some recipients?* This question leads us to analyze Content Agreement in the following contexts:

- end-to-end encrypted group messaging applications, like WhatsApp or Signal, or
- dedicated encrypted message mechanism, like GPG, Saltpack, or Scuttlebutt.

Our study reveals that there is a discrepancy between existing guarantees, which we summarize in Table 5.

We only present the WhatsApp group case study next, and refer the reader to Appendix C for the others.

WhatsApp groups We model the design of WhatsApp's group messaging. Because the code of WhatsApp is not available, we constructed our model based on the available information provided in its whitepaper [58, p. 10]. While it relies on the Signal X3DH protocol to establish pairwise channels between the members of the groups, sending a message is slightly different:

- The sender generates a so called *sender key* (also part of the Signal library), and sends this key to each participant over the corresponding pairwise channel;
- To send a message, there is then a single encrypted payload which is uploaded to the server.

While content agreement is trivially broken in Signal itself because of the pairwise channels, it could intuitively be expected within the setting of group messaging. It is however not guaranteed, as reported by our model. Our model captures a group of three people, where one of them is the attacker. We then aim to verify that a given message uploaded to the server will yield the same plaintext for all group members. Our automated analysis reports an attack on this property when enabling ciphertext collisions under KeyColl.

The group messaging mechanism relies on an AES-CBC encryption which is then signed with an independent key. This is similar to the Encrypt-then-Mac with unrelated keys scenario. It means that the complexity of mounting an attack in practice is equivalent to the complexity of finding meaningful collisions over AES-CBC. We have seen that with the current capabilities, this strongly depends on the concrete encoding of plaintexts, and whether we can find so-called polyglot plaintexts [2]. As WhatsApp is closed source, verifying the practicality of the attack would require to reverse engineer the full message encoding, which we consider out of scope for this paper. However, given the variety of possible message contents (notification, GIFs, media, React, ...) it is likely that the encoding would be loose enough to carry out the attack.

5.4 Disclosure

Using our methodology, we detected several undesirable behaviors in the protocol design of Scuttlebutt, Web Push, and WhatsApp Groups [53, 55, 58]. While the behaviors are possible on the protocol design level, their implementation-level feasibility depends on low-level encoding choices.

The behaviors we found did not violate the main specified goals of the respective protocols, and hence we did not mark them as "attacks". Nevertheless, we contacted the developers of the affected protocols and explained our observations, such that they can assess their implementation-level feasibility, with distinct feedback:

- The developers of the WebPush standard acknowledged the issue, and a discussion is ongoing to determine how to best document the possible behaviors in the standard;
- WhatsApp considered this outside their threat model, and noted that using a different AEAD would still allow a variant of the behavior with the same effect; and
- The Scuttlebutt developers did not respond.

6 Limitations

In an ideal world, we would like to (a) cover all possible AEAD definitions and weaknesses, and (b) have the guarantee that if our method reports an attack, the attack is always feasible in practice. Unfortunately this is not the case yet.

In terms of possible AEAD definitions there are subtle differences that we currently do not capture yet. This includes, for example, properties beyond collisions and nonce-reuse, such as the "s-way commiting security property" [8] that generalizes the CMT notions to the multi-user setting. Our models can also be improved with respect to the Forge capability, as discussed in Section 4. On the positive side, we define general models and capture for instance collisions that given the current knowledge are not practical, but could become so in the future, e.g., with new developments on AES. While we do not claim to cover all possible AEAD attacks in the future, this allows to future proof protocols.

With respect to practical feasibility of attacks, the fundamental problem is that our analysis method and standard cryptographic analyses in fact consider protocols *designs* and not their implementation details. For example, this includes abstracting away from encoding details, i.e., how values and compound structures are exactly mapped to bitstrings. Yet such details are critical to determine whether certain collision attacks are possible or not. As a consequence, when we find an attack on the protocol design, this should intuitively be interpreted as: there exists an encoding scheme for which the protocol implementation is insecure. We argue that security of a protocol design should avoid depending on its encoding scheme, and if not, specify the requirements explicitly. The problems we found here using our framework are therefore real concerns for the protocol designs. Still, manual inspection of the implementation is still needed to check whether the encodings allow for generation of the required collisions; but our analysis indicates the types of collisions needed.

7 Further Related Work

We provided background and summary of the related works on AEADs in the first two sections. Here we discuss additional works in formal analysis of security protocols.

Improving the symbolic models of primitives to enable automated attack finding has recently been explored for several types of basic primitives, like cryptographic hashes [18], Diffie-Hellman groups [21], or digital signatures [35]. Our work is centered on AEAD, for which no systematic approach had been attempted yet.

Ad-hoc approaches include a specific form of nonce-reuse in the Tamarin analysis of WPA [22] and the analysis of Yubikey [41]. In a different approach, [39] modeled the fine-grained block based encryption in the tool ProVerif, but this approach did not scale to protocols of the complexity considered here. Overall, our work is the first to systematically explore weaknesses of concrete algorithms or formal definitions for AEADs and provide models amenable to automation.

ProVerif [13] is, besides Tamarin, the other major tool for automated analysis of protocols in the symbolic model. While we developed our models and case-studies in Tamarin, they could also be used in the ProVerif framework.

As we are focused on automated attack finding due to real life weaknesses of AEADs, our work is orthogonal to tools from the computational model [5, 6, 12] that are all focused on proving security, and cannot find attacks. Our automated models can be useful to establish the assumptions on the AEAD before attempting a computational proof.

With respect to our case studies, based on the absence of collision resistance of AEADs, also referred to as robustness or key-commitment, [25] already reported an attack on the Facebook abuse reporting mechanism, which violates accountability. We are the first to report on behaviors in which is content agreement is not satisfied due to AEAD weaknesses. Other undesirable behaviors have arisen due to collisions, which are linked to oracle partitioning attacks [42], where an attacker can obtain a better than brute force advantage against the OPAQUE protocol. While we can model the relevant AEAD weakness in our framework, modeling the violated security property in the symbolic model is left to future work.

8 Conclusions

We developed the first methodology to analyze the impact of detailed AEAD behaviors on the protocols that use them. Our methodology thus enables detecting protocol weaknesses for a given AEAD, or conversely, determining a protocol's AEAD requirements. The case studies indicate that our methodology is effective and efficient in finding potential weaknesses, notably automatically finding attacks that previously could only found by manual inspection, thus bringing a new class of attacks within the realm of automated detection.

Acknowledgments

This work received funding from the France 2030 program managed by the French National Research Agency under grant agreement No. ANR-22-PECY-0006.

References

[1] Michel Abdalla, Mihir Bellare, and Gregory Neven. *Robust Encryption*. Cryptology ePrint Archive, Report 2008/440. `https://ia.cr/2008/440`. 2008.

[2] Ange Albertini, Thai Duong, Shay Gueron, Stefan Kölbl, Atul Luykx, and Sophie Schmieg. "How to Abuse and Fix Authenticated Encryption Without Key Commitment". In: *31st USENIX Security Symposium*. 2020.

[3] Martin R Albrecht, Jean Paul Degabriele, Torben Brandt Hansen, and Kenneth G Paterson. "A surfeit of SSH cipher suites". In: *ACM Conference on Computer and Communications Security (CCS)*. 2016.

[4] Martin R Albrecht, Kenneth G Paterson, and Gaven J Watson. "Plaintext recovery attacks against SSH". In: *IEEE Symposium on Security and Privacy (S&P)*. 2009.

[5] David Baelde, Stéphanie Delaune, Charlie Jacomme, Adrien Koutsos, and Solène Moreau. "An interactive prover for protocol verification in the computational model". In: *IEEE Symposium on Security and Privacy (S&P)*. IEEE. 2021.

[6] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. "Computer-aided security proofs for the working cryptographer". In: *Annual Cryptology Conference*. Springer. 2011.

[7] Guy Barwell, Daniel Page, and Martijn Stam. "Rogue Decryption Failures: Reconciling AE Robustness Notions". In: *Proceedings of the 15th IMA International Conference on Cryptography and Coding*. 2015.

[8] Mihir Bellare and Viet Tung Hoang. *Efficient Schemes for Committing Authenticated Encryption*. Cryptology ePrint Archive, Report 2022/268. `https://ia.cr/2022/268`. 2022.

[9] Mihir Bellare, Ruth Ng, and Björn Tackmann. *Nonces are Noticed: AEAD Revisited*. Cryptology ePrint Archive, Report 2019/624. `https://ia.cr/2019/624`. 2019.

[10] Mihir Bellare, Phillip Rogaway, and David Wagner. "The EAX mode of operation". In: *International Workshop on Fast Software Encryption*. 2004.

[11] Ritam Bhaumik and Mridul Nandi. "Improved security for OCB3". In: *International Conference on the Theory and Application of Cryptology and Information Security*. 2017.

[12] Bruno Blanchet. "CryptoVerif: Computationally sound mechanized prover for cryptographic protocols". In: *Dagstuhl seminar "Formal Protocol Verification Applied*. Vol. 117. 2007.

[13] Bruno Blanchet, Vincent Cheval, and Véronique Cortier. "Proverif with lemmas, induction, fast subsumption, and much more". In: *IEEE Symposium on Security and Privacy (S&P)*. 2022.

[14] Bruno Blanchet, Ben Smyth, Vincent Cheval, and Marc Sylvestre. *ProVerif 2.00: automatic cryptographic protocol verifier, user manual and tutorial*. 2018.

[15] Alexandra Boldyreva, Jean Paul Degabriele, Kenneth G Paterson, and Martijn Stam. "Security of symmetric encryption in the presence of ciphertext fragmentation". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 2012.

[16] Brice Canvel, Alain Hiltgen, Serge Vaudenay, and Martin Vuagnoux. "Password interception in a SSL/TLS channel". In: *Annual International Cryptology Conference*. 2003.

[17] John Chan and Phillip Rogaway. *Anonymous AE*. Cryptology ePrint Archive, Report 2019/1033. https://ia.cr/2019/1033. 2019.

[18] Vincent Cheval, Cas Cremers, Alexander Dax, Lucca Hirschi, Charlie Jacomme, and Steve Kremer. "Hash Gone Bad: Automated discovery of protocol attacks that exploit hash function weaknesses". In: *32nd USENIX Security Symposium*. 2023.

[19] Cas Cremers, Alexander Dax, Charlie Jacomme, and Mang Zhao. *Tamarin models and analysis scripts to reproduce the results in this paper*. https://github.com/AutomatedAnalysisOf/AEADProtocols. 2023.

[20] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. "A comprehensive symbolic analysis of TLS 1.3". In: *ACM Conference on Computer and Communications Security (CCS)*. 2017.

[21] Cas Cremers and Dennis Jackson. "Prime, Order Please! Revisiting Small Subgroup and Invalid Curve Attacks on Protocols using Diffie-Hellman". In: *IEEE 32nd Computer Security Foundations Symposium (CSF)*. 2019.

[22] Cas Cremers, Benjamin Kiesl, and Niklas Medinger. "A Formal Analysis of IEEE 802.11's WPA2: Countering the Kracks Caused by Cracking the Counters". In: *29th USENIX Security Symposium*. 2020.

[23] Alexander Dax, Robert Künnemann, Sven Tangermann, and Michael Backes. "How to wrap it up-a formally verified proposal for the use of authenticated wrapping in PKCS# 11". In: *IEEE 32nd Computer Security Foundations Symposium (CSF)*. 2019.

[24] Jean Paul Degabriele and Kenneth G Paterson. "On the (in) security of IPsec in MAC-then-encrypt configurations". In: *ACM Conference on Computer and Communications Security (CCS)*. 2010.

[25] Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. *Fast Message Franking: From Invisible Salamanders to Encryptment*. Cryptology ePrint Archive, Report 2019/016. https://ia.cr/2019/016. 2019.

[26] Santiago Escobar, Catherine Meadows, and José Meseguer. "Maude-NPA: Cryptographic protocol analysis modulo equational properties". In: *Foundations of Security Analysis and Design*. Springer, 2009.

[27] *Facebook - Messenger Secret Conversations Technical Whitepaper*. https://about.fb.com/wp-content/uploads/2016/07/messenger-secret-conversations-technical-whitepaper.pdf. accessed: 2022-08-08. 2017.

[28] Pooya Farshim, Claudio Orlandi, and Răzvan Roşie. *Security of Symmetric Primitives under Incorrect Usage of Keys*. Cryptology ePrint Archive, Report 2017/288. https://ia.cr/2017/288. 2017.

[29] Pierre-Alain Fouque, Gwenaëlle Martinet, Frédéric Valette, and Sébastien Zimmer. "On the Security of the CCM Encryption Mode and of a Slight Variant". In: *International Conference on Applied Cryptography and Network Security*. 2008.

[30] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. *Message Franking via Committing Authenticated Encryption*. Cryptology ePrint Archive, Report 2017/664. https://ia.cr/2017/664. 2017.

[31] Shay Gueron and Yehuda Lindell. "GCM-SIV: full nonce misuse-resistant authenticated encryption at under one cycle per byte". In: *ACM Conference on Computer and Communications Security (CCS)*. 2015.

[32] Takanori Isobe, Ryoma Ito, and Kazuhiko Minematsu. "Security Analysis of SFrame". In: *European Symposium on Research in Computer Security*. 2021.

[33] Tetsu Iwata, Keisuke Ohashi, and Kazuhiko Minematsu. "Breaking and repairing GCM security proofs". In: *Annual Cryptology Conference*. 2012.

[34] Tetsu Iwata and Yannick Seurin. "Reconsidering the Security Bound of AES-GCM-SIV". In: *IACR Transactions on Symmetric Cryptology* (2017).

[35] Dennis Jackson, Cas Cremers, Katriel Cohn-Gordon, and Ralf Sasse. "Seems Legit: Automated Analysis of Subtle Attacks on Protocols that Use Signatures". In: *ACM Conference on Computer and Communications Security (CCS)*. 2019.

[36] Jakob Jonsson. "On the security of CTR+ CBC-MAC". In: *International Workshop on Selected Areas in Cryptography*. 2002.

[37] Werner Koch, Paul Wouters, Daniel Huigens, and Justus Winter. *OpenPGP Message Format*. Internet-Draft draft-ietf-openpgp-crypto-refresh-06. Work in Progress. Internet Engineering Task Force, June 2022. 163 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-openpgp-crypto-refresh/06/.

[38] Steve Kremer and Robert Künnemann. "Automated analysis of security protocols with global state". In: *Journal of Computer Security* 24.5 (2016).

[39] Steve Kremer and Mark D. Ryan. "Analysing the Vulnerability of Protocols to Produce Known-pair and Chosen-text Attacks". In: *Proceedings of the 2nd International Workshop on Security Issues in Coordination Models, Languages and Systems (SecCo'04)*. Ed. by Riccardo Focardi and Gianluigi Zavattaro. May 2005.

[40] Ted Krovetz and Phillip Rogaway. "The software performance of authenticated-encryption modes". In: *International Workshop on Fast Software Encryption*. 2011.

[41] Robert Künnemann and Graham Steel. "YubiSecure? Formal security analysis results for the Yubikey and YubiHSM". In: *International Workshop on Security and Trust Management*. Springer. 2012.

[42] Julia Len, Paul Grubbs, and Thomas Ristenpart. "Partitioning Oracle Attacks". In: *30th USENIX Security Symposium*. 2021.

[43] David A McGrew and John Viega. "The security and performance of the Galois/Counter Mode (GCM) of operation". In: *International Conference on Cryptology in India*. 2004.

[44] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. "The TAMARIN prover for the symbolic analysis of security protocols". In: *International conference on computer aided verification*. 2013.

[45] Kazuhiko Minematsu, Stefan Lucks, and Tetsu Iwata. "Improved authenticity bound of EAX, and refinements". In: *International Conference on Provable Security*. 2013.

[46] Serge Mister and Robert Zuccherato. "An attack on CFB mode encryption as used by OpenPGP". In: *International Workshop on Selected Areas in Cryptography*. 2005.

[47] Karl Norrman, Vaishnavi Sundararajan, and Alessandro Bruni. "Formal Analysis of EDHOC Key Establishment for Constrained IoT Devices". In: *CoRR* abs/2007.11427 (2020). arXiv: 2007.11427.

[48] E. Omara, J. Uberti, A. GOUAILLARD, and S. Murillo. *Secure Frame (SFrame) v01*. https://datatracker.ietf.org/doc/html/draft-omara-sframe-01. accessed: 2022-08-08. 2020.

[49] Gordon Procter. "A Security Analysis of the Composition of ChaCha20 and Poly1305". In: *Cryptology ePrint Archive, Paper 2014/613*. https://eprint.iacr.org/2014/613. 2014.

[50] Phillip Rogaway. "Authenticated-Encryption with Associated-Data". In: *ACM Conference on Computer and Communications Security (CCS)*. CCS '02. 2002.

[51] Phillip Rogaway and John Steinberger. "Security/Efficiency Tradeoffs for Permutation-Based Hashing". In: *EUROCRYPT 2008*. 2008.

[52] *Saltpack v2*. https://saltpack.org/encryption-format-v2. accessed: 2022-08-08. 2017.

[53] *Scuttlebot Private Box v0.3.1*. https://scuttlebot.io/more/protocols/private-box.html. accessed: 2022-08-08. 2019.

[54] Alon Shakevsky, Eyal Ronen, and Avishai Wool. "Trust Dies in Darkness: Shedding Light on Samsung's TrustZone Keymaster Design". In: *Cryptology ePrint Archive, Paper 2022/208*. https://eprint.iacr.org/2022/208. 2022.

[55] Martin Thomson. *Message Encryption for Web Push*. RFC 8291. Nov. 2017. URL: https://www.rfc-editor.org/info/rfc8291.

[56] Mathy Vanhoef and Frank Piessens. "Key reinstallation attacks: Forcing nonce reuse in WPA2". In: *ACM Conference on Computer and Communications Security (CCS)*. 2017.

[57] Serge Vaudenay. "Security flaws induced by CBC padding—applications to SSL, IPSEC, WTLS..." In: *International Conference on the Theory and Applications of Cryptographic Techniques*. 2002.

[58] *WhatsApp Security Whitepaper*. https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf. accessed: 2022-08-08. 2021.

[59] Jianliang Wu, Ruoyu Wu, Dongyan Xu, Dave Jing Tian, and Antonio Bianchi. "Formal Model-Driven Discovery of Bluetooth Protocol Design Vulnerabilities". In: *IEEE Symposium on Security and Privacy (S&P)*. 2022.

[60] *YubiHSM*. https://www.yubico.com/resource/hsm-security-for-manufacturing/. accessed: 2022-08-08. 2021.

[61] *Zerologon – hacking Windows servers with a bunch of zeros*. https://nakedsecurity.sophos.com/2020/09/17/zerologon-hacking-windows-servers-with-a-bunch-of-zeros/. accessed: 2023-02-01. 2020.

A Collision resistance of existing schemes

A.1 Encrypt-then-MAC modes

CTR mode [25]

This is one of the simplest block-cipher modes to create colliding ciphertexts of the previous form for. Indeed, to encrypt a plaintext $\mathsf{plaintext} := \mathsf{plaintext}_1 \parallel \cdots \parallel \mathsf{plaintext}_n$ with initial vector IV and key k, we have that $c_i = \mathrm{Block}_{\mathsf{CTR}}(k, i \| IV, \mathsf{plaintext}_i)$ for $i = 1, ..., n$, where $\mathrm{Block}_{\mathsf{CTR}}(k, N, m) := m \oplus AES_k(N)$. Then, once the first colliding block has been brute forced, adding the rest of the data is simply done by using the encrypted part under the desired key.

OFB mode

Here, we have $K_1 = AES_k(IV)$, $K_i = AES_k(K_{i-1})$, and $c_i = \mathsf{plaintext}_i \oplus K_i$. Thus, finding a collision for OFB mode is strictly similar to CTR mode and as easy.

CBC and CFB

Here, it is slightly more complex, as there is a ciphertext propagation. Indeed, we have $c_1 = \mathrm{Block}_{\mathsf{X}}(k, IV, \mathsf{plaintext}_1)$, and then $c_i = \mathrm{Block}_{\mathsf{X}}(k, c_{i-1}, \mathsf{plaintext}_i)$ for $\mathsf{X} \in \{\mathsf{CBC}, \mathsf{CFB}\}$, where for CBC mode $\mathrm{Block}_{\mathsf{CBC}}(k, N, m) := AES_k(m \oplus N)$, for CFB mode $\mathrm{Block}_{\mathsf{CFB}}(k, N, m) := AES_k(N) \oplus m$.

It is still quite easy to build a valid colliding ciphertext under two keys, by taking care of the propagation. Assume a fixed first ciphertext c_1, two keys k_1 and k_2 and a long plaintext $\mathsf{plaintext} = \mathsf{plaintext}_1 \parallel \cdots \parallel \mathsf{plaintext}_n \parallel \mathsf{plaintext}_{n+1} \parallel \cdots \parallel \mathsf{plaintext}_{2n}$ for some n and $\mathsf{X} \in \{\mathsf{CBC}, \mathsf{CFB}\}$:

- compute $c_i = \mathrm{Block}_{\mathsf{X}}(k_1, c_{i-1}, \mathsf{plaintext}_{i-1})$ for $i = 2, ..., n+1$.
- compute $c_i = \mathrm{Block}_{\mathsf{X}}(k_2, c_{i-1}, \mathsf{plaintext}_{i-1})$ for $i = n+2, ..., 2n+1$.
- output $c = c_1 \parallel c_2 \parallel \cdots \parallel c_{2n+1}$.

Then, c will be decrypted to different messages respectively under k_1 and k_2.

A.2 GPG encryption - Symmetrically Encrypted Data packet

GPG relies on a variant of the CFB mode to encrypt data, where it essentially initializes the property by encrypting a random plaintext R concatenated with a repetition of the last two bytes R. Here, we use $R_{[i_1:i_2]}$ to denote the substring of a string R from i_1-th bytes to i_2-th bytes and l denote the block length. Thus, the last two bytes of R can be denoted by $R_{[l-1:l]}$. This duplication is actually meant to enable the decryption process to verify if it has the correct key. This integrity check leads to a CPA attack [46], and we can use it here to efficiently find collisions, as it can be seen as an extra junk data field always available.

$$\begin{aligned} c_1 &= E_k(0) \oplus R \\ c_2 &= E_k(c_1)_{[1:2]} \oplus R_{[l-1:l]} \\ c_3 &= E_k(c_{1\,[3:l]} \| c_2) \oplus \mathsf{plaintext} \\ &\cdots \end{aligned}$$

where E denotes the underlying encryption scheme, such as AES.

The first way to brute force is perfectly similar to the salamander attack, where we fix the keys and look for a until we find one such that we have the collisions over the b bytes of the message, plus the two bytes for c_2. Hence, the cost here is 2^{48}, or 2^{25} for a fixed nonce and a birthday search over the keys.

The second way is to simply fix a given value of T, pick a key k and first reverse the computation:

$$(C_1)_{[3:l]} \| C_2 = E_k^{-1}(T)$$

And then brute force on two bytes b_0, b_1 to let $c_1 = (b_0 \| b_1 \| (C_1)_{[3:l]})$ until the equation is satisfied:

$$\Big(c_1 \oplus E_k(0)\Big)_{[b-1:b]} = c_2 \oplus E_k(c_1)_{[1:2]}$$

We are essentially looking over bitstrings of length $l+2$ with two bytes equal to two other one. Assuming a uniform distribution, this should occur with probability $\frac{1}{2^{16}}$, which makes for a very efficient brute force attack. For instance with $K = T$ equals a string of bit 1, there are three such collisions out of the 2^{16} possibilities, which are found under a second.

Now, if we have this for a given T, we can build infinitely many valid ciphertexts for plaintext $\mathsf{plaintext}$ as $c_1 \| c_2 \| (\mathsf{plaintext} \oplus T)$. And if we have two fixed T_1, T_2, we have collisions for plaintexts such that $\mathsf{plaintext}_1 \oplus T_1 = \mathsf{plaintext}_2 \oplus T_2$.

An interesting point here is that the complexity of finding a collision has become independent from the number of targeted collision bytes, and we can aim for a full block collision.

It must be noted that such packets are not the default ones for PGP, but are not deprecated. Softwares usually rely on Sym. Encrypted Integrity Protected Data Packet, openPGP is undergoing a standardization process for a crypto refresh.

B Formal definitions and proofs

B.1 Additional Preliminaries

We recall a primitive called committing PRF and its simplified binding security, which was first defined in [8].

Definition 5. *Let* $\mathsf{cPRF} : \mathsf{Key} \times \mathsf{Message} \to \mathsf{Rand} \times \mathsf{Tag}$ *denote a deterministic function inputs a key* $k \in \mathsf{Key}$ *and a message* $m \in \mathsf{Message}$ *and outputs* $r \in \mathsf{Rand}$ *and* $t \in \mathsf{Tag}$*. We say* cPRF *is* ϵ*-binding (or* ϵ-bind*) secure, if the below defined advantage of any attacker* $\mathcal{A}$ *against* $\mathrm{Expr}^{\mathsf{bind}}_{\mathsf{cPRF}}$ *experiment in Fig. 6 is bounded by:*

$$\mathsf{Adv}^{\mathsf{bind}}_{\mathsf{cPRF}} := \Pr[\mathrm{Expr}^{\mathsf{bind}}_{\mathsf{cPRF}}(\mathcal{A}) = 1] \leq \epsilon$$

$\underline{\mathrm{Expr}^{\mathsf{bind}}_{\mathsf{cPRF}}}$:
1 $(k_1, m_1, k_2, m_2) \leftarrow\$ \mathcal{A}()$
2 $(r_1, t_1) \leftarrow \mathsf{cPRF}(k_1, m_1)$, $(r_2, t_2) \leftarrow \mathsf{cPRF}(k_2, m_2)$
3 **return** $[\![t_1 = t_2]\!]$

Figure 6: bind security for a cPRF function.

B.2 Additional Definitions

Definition 6. *An* AEAD *can be extended to (compactly) commiting* AEAD *(*ccAEAD*) if two additional algorithms are defined. Let* $\mathsf{VrfyKey}$ *denote the space of verification key.*

- $\mathsf{openCommit}$ *the open commitment algorithm inputs a key* $k \in \mathsf{Key}$*, a nonce* $N \in \mathsf{Nonce}$*, a header* $H \in \mathsf{Header}$*, and a ciphertext* $c \in \mathsf{Ciphertext}$ *and (deterministically) outputs a verification key* $k_f \in \mathsf{VrfyKey} \cup \{\perp\}$*, i.e.,* $k_f \leftarrow \mathsf{openCommit}(k, N, H, c)$
- $\mathsf{vrfyCommit}$ *the commitment verification algorithm inputs a verification key* $k_f \in \mathsf{VrfyKey}$ *, a nonce* $N \in \mathsf{Nonce}$*, a header* $H \in \mathsf{Header}$*, a message* m*, and a ciphertext* $c \in \mathsf{Ciphertext}$ *and (deterministically) outputs a boolean value* $v \in \{\mathsf{true}, \mathsf{false}\}$*, i.e.,* $v \leftarrow \mathsf{vrfyCommit}(k_f, N, H, m, c)$*.*

Each AEAD scheme is assumed to be defined with a length function ℓ such that $|\mathsf{Enc}(k, N, H, m)| = \ell(|m|)$ for all $(k, N, H, m) \in \mathsf{Key} \times \mathsf{Nonce} \times \mathsf{Header} \times \mathsf{Message}$.

We say an AEAD scheme is ϵ-correct if for all $(N, H, m) \in \mathsf{Nonce} \times \mathsf{Header} \times \mathsf{Message}$ and $k \leftarrow\$ \mathsf{KGen}()$ it holds that

$$\Pr[m' \leftarrow \mathsf{Dec}(k, N, H, \mathsf{Enc}(k, N, H, m)) : m \neq m'] \leq \epsilon$$

In particular, we say AEAD is perfect correct if $\epsilon = 0$.

We say an AEAD scheme is *tidy* if for each $(k, N, H, c) \in \mathsf{Key} \times \mathsf{Nonce} \times \mathsf{Header} \times \mathsf{Ciphertext}$ it holds that

$$\perp \neq m \leftarrow \mathsf{Dec}(k, N, H, c) \implies c \leftarrow \mathsf{Enc}(k, N, H, m)$$

B.3 Privacy and Integrity

We start with the confidentiality notion IND$-CPA and extend it to IND$-CCA in a natural way.

Definition 7. *We say an* $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *is* ϵ-IND$-XXX *for* $\mathsf{XXX} \in \{\mathsf{CPA}, \mathsf{CCA}\}$ *secure, if the below defined advantage of any attacker* $\mathcal{A}$ *against* $\mathrm{Expr}^{\mathsf{IND\$\text{-}XXX}}_{\mathsf{AEAD}}$ *experiment in Fig. 1 is bounded by:*

$$\mathsf{Adv}^{\mathsf{IND\$\text{-}XXX}}_{\mathsf{AEAD}} := |\Pr[\mathrm{Expr}^{\mathsf{IND\$\text{-}XXX}}_{\mathsf{AEAD}}(\mathcal{A}) = 1] - \frac{1}{2}| \leq \epsilon$$

Definition 8. *We say an* $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *is* ϵ-CTI-XXX *for* $\mathsf{XXX} \in \{\mathsf{CPA}, \mathsf{CCA}\}$ *secure, if the below defined advantage of any attacker* $\mathcal{A}$ *against* $\mathrm{Expr}^{\mathsf{CTI\text{-}XXX}}_{\mathsf{AEAD}}$ *experiment in Fig. 2 is bounded by:*

$$\mathsf{Adv}^{\mathsf{CTI\text{-}XXX}}_{\mathsf{AEAD}} := \Pr[\mathrm{Expr}^{\mathsf{CTI\text{-}XXX}}_{\mathsf{AEAD}}(\mathcal{A}) = 1] \leq \epsilon$$

The above security notions captures the standard privacy and authenticity requirements. The relation among them has been explored in [7]. Below, we recall the results.

Theorem 1 ([7])**.** *Let* $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *be an authenticated encryption with associated data. It holds that:*

1. *[7, Proposition 1]: If* AEAD *is* ϵ-CTI-CPA *secure, then* AEAD *is also* ϵ-CTI-CCA *secure, and vice versa.*
2. *[7, Theorem 6]: If* AEAD *is* $\epsilon^{\mathsf{IND\$\text{-}CPA}}_{\mathsf{AEAD}}$-IND$-CPA *secure and* $\epsilon^{\mathsf{CTI\text{-}CPA}}_{\mathsf{AEAD}}$-CTI-CPA *secure, then* AEAD *is also* $\epsilon^{\mathsf{IND\$\text{-}CCA}}_{\mathsf{AEAD}}$-IND$-CCA *secure such that*

$$\epsilon^{\mathsf{IND\$\text{-}CCA}}_{\mathsf{AEAD}} \leq 2(\epsilon^{\mathsf{IND\$\text{-}CPA}}_{\mathsf{AEAD}} + \epsilon^{\mathsf{CTI\text{-}CPA}}_{\mathsf{AEAD}})$$

Moreover, we also have some trivial results for the relations between the privacy and authenticity notions.

Theorem 2. *Let* $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *be an authenticated encryption with associated data. Then, it holds that:*

1. *If* AEAD *is* ϵ-IND$-CCA *secure, then* AEAD *is also* ϵ-IND$-CPA *secure.*
2. *[7]: If* AEAD *is* ϵ-IND$-CPA *secure, then* AEAD *might not be* ϵ'-IND$-CCA *secure for any negligible* ϵ'*.*
3. *If* AEAD *is* ϵ-IND$-CPA *secure, then* AEAD *might not be* ϵ'-CTI-CPA *secure for any negligible* ϵ'*.*
4. *If* AEAD *is* ϵ-CTI-CPA *secure, then* AEAD *might not be* ϵ'-IND$-CPA *secure for any negligible* ϵ'*.*
5. *If* AEAD *is* ϵ-IND$-CCA *secure, then* AEAD *might not be* ϵ'-CTI-CPA *secure for any negligible* ϵ'*.*
6. *If* AEAD *is* ϵ-CTI-CPA *secure, then* AEAD *might not be* ϵ'-IND$-CCA *secure for any negligible* ϵ'*.*

Proof. We prove each of these statements in turn.

1. Statement 1: This statement can be proven by a trivial reduction. If there exists an attacker $\mathcal{A}$ that breaks IND$-CPA security of AEAD, then we can construct an attacker $\mathcal{B}$ that breaks IND$-CCA security of AEAD by invoking AEAD. $\mathcal{B}$ simply invokes $\mathcal{A}$, forwards all $\mathcal{A}$'s queries to its challenger and returns the responses to $\mathcal{A}$, and finally outputs $\mathcal{A}$'s decision. We observe that $\mathcal{B}$ wins if and only if $\mathcal{A}$ wins, which concludes the proof.
2. Statement 2: See [7, Lemma 2].
3. Statement 3: We prove this statement by counter example. Let $\mathsf{AEAD}_1 = (\mathsf{KGen}_1, \mathsf{Enc}_1, \mathsf{Dec}_1)$ and $\mathsf{AEAD}_2 = (\mathsf{KGen}_2, \mathsf{Enc}_2, \mathsf{Dec}_2)$ denote two independent ϵ-IND$-CPA secure authenticated encryption with associated data schemes with the same message space Message. We then construct $\mathsf{AEAD}' = (\mathsf{KGen}', \mathsf{Enc}', \mathsf{Dec}')$ from AEAD_1 and AEAD_2 as follows:
   - $\mathsf{KGen}'()$: runs $k_1 \leftarrow\!\!\$\ \mathsf{KGen}_1()$ and $k_2 \leftarrow\!\!\$\ \mathsf{KGen}_2()$ followed by outputting $k' := k_1 \parallel k_2$.
   - $\mathsf{Enc}'(k', N, H, m)$: first parses $k_1 \parallel k_2 \leftarrow k'$ and then runs $c_1 \leftarrow \mathsf{Enc}(k_1, N, H, m)$ and $c_2 \leftarrow \mathsf{Enc}(k_2, N, H, m)$, followed by outputting $c' := c_1 \parallel c_2$.
   - $\mathsf{Dec}'(k', N, H, c')$: parses $k_1 \parallel k_2 \leftarrow k'$ and $c_1 \parallel c_2 \leftarrow c'$, followed by outputting $\mathsf{Dec}(k_1, N, H, c_1)$.

   It is straightforward to prove that AEAD' is 2ϵ-IND$-CPA secure by reduction. If there exists an attacker $\mathcal{A}$ that breaks the IND$-CPA security of AEAD', then we can construct an attacker $\mathcal{B}$ that breaks the IND$-CPA security of AEAD_1 or AEAD_2.
   However, AEAD' is not CTI-CPA secure. An attacker can queries $\textsc{Enc}(N, H, m)$ for any N, H, m for a ciphertext $c' = c_1 \parallel c_2$ such that $c_1 \neq c_2$, followed by outputting $c'' = c_1 \parallel c_1$. It is easy to see that $c'' \notin \mathcal{L}_c$ since $c' \neq c''$ and $\mathsf{Dec}'(k, N, H, c'') = \mathsf{Dec}'(k, N, H, c') \neq \perp$. Thus, this attacker always win the CTI-CPA experiment.

4. Statement 4: We prove this statement by counter example. Let $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ denote an ϵ-CTI-CPA secure authenticated encryption with associated data scheme with the message space $\mathsf{Message}$. We then construct $\mathsf{AEAD}' = (\mathsf{KGen}', \mathsf{Enc}', \mathsf{Dec}')$ from AEAD as follows:
   - $\mathsf{KGen}'()$: is identical to $k \leftarrow\!\$\ \mathsf{KGen}()$.
   - $\mathsf{Enc}'(k, N, H, m)$: runs $c \leftarrow \mathsf{Enc}(k, N, H, m)$ followed by outputting $c' := c \parallel c$.
   - $\mathsf{Dec}'(k, N, H, c')$: first parses $c_1 \parallel c_2 \leftarrow c'$ and outputs $\perp$ if $c_1 \neq c_2$. Otherwise, outputs $\mathsf{Dec}(k, N, H, c_1)$.

   It is straightforward to prove that AEAD' is ϵ-CTI-CPA secure by reduction. If there exists an attacker $\mathcal{A}$ that breaks the CTI-CPA security of AEAD', then we can construct an attacker $\mathcal{B}$ that breaks the CTI-CPA security of AEAD.
   However, AEAD' is not IND$-CPA secure since an attacker can easily distinguish any ciphertext $c' = c'_1 \parallel c'_2$ of AEAD' from a random string of the same length by checking whether $c'_1 = c'_2$.
5. Statement 5: We prove this statement by counter example. Let $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ denote an ϵ-IND$-CCA secure authenticated encryption with associated data scheme with message space $\mathsf{Message}$, nonce space Nonce, and header space Header, and ciphertext space $\mathsf{Ciphertext}$. In particular, let $\tilde{N} \in \mathsf{Nonce}$ and $\tilde{H} \in \mathsf{Header}$ denote two arbitrary strings in the respective domains. We then construct $\mathsf{AEAD}' = (\mathsf{KGen}', \mathsf{Enc}', \mathsf{Dec}')$ from AEAD as follows:
   - $\mathsf{KGen}'()$: runs $k \leftarrow\!\$\ \mathsf{KGen}_1()$ and samples $r \leftarrow\!\$\ \mathsf{Message}$ followed by outputting $k' := k \parallel r$.
   - $\mathsf{Enc}'(k', N, H, m)$: first parses $k \parallel r \leftarrow k'$ and then outputs c as a string of $l(|m|)$ zero bits if $r = m$, $N = \tilde{N}$ and $H = \tilde{H}$. Otherwise, outputs $c \leftarrow \mathsf{Enc}(k, N, H, m)$.
   - $\mathsf{Dec}'(k', N, H, c)$: first parses $k \parallel r \leftarrow k'$, followed by outputting r if c is a string of $l(|m|)$ zero bits, $N = \tilde{N}$ and $H = \tilde{H}$. Otherwise, outputs $\mathsf{Dec}(k, N, H, c)$.

   It is straightforward to prove that AEAD' is $(\epsilon + \frac{q}{|\mathsf{Message}|})$-IND$-CCA secure by reduction, where q denotes the number of queries that $\mathcal{A}$ can make in polynomial time. If there exists an attacker $\mathcal{A}$ that breaks the IND$-CCA security of AEAD', then we can construct an attacker $\mathcal{B}$ that breaks the IND$-CCA security of AEAD.
   However, AEAD' is not CTI-CPA secure since a string of $l(|m|)$ zero bits is always a ciphertext, which can be decrypted to a message $r \in \mathsf{Message}$ for any $k \in \mathsf{Key}$, $N = \tilde{N}$, and $H \in \tilde{H}$.
6. Statement 6: This statement is implied by Statements 1 and 4.

□

B.4 Collision Resistance

For any $\mathsf{X} \subseteq (k, N, H, m)$, we define a class of projection functions $f_{\mathsf{X}} : \mathsf{Key} \times \mathsf{Nonce} \times \mathsf{Header} \times \mathsf{Message} \rightarrow \mathsf{dom}(\mathsf{X})$, where $\mathsf{dom}(\mathsf{X})$ denotes the domain of X. The function inputs a tuple (k, N, H, m) and outputs the values that X projects to. For instance, $f_k(k, N, H, m) = k$ and $f_{(k,N,H,m)}(k, N, H, m) = (k, N, H, m)$.

Definition 9. *We say an* $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *is* ϵ-X *collision resistant (or* ϵ-X-CR *) for* $\mathsf{X} \subseteq (k, N, H, m)$*, if the below defined advantage of any attacker* $\mathcal{A}$ *against* $\mathrm{Expr}^{\mathsf{X\text{-}CR}}_{\mathsf{AEAD}}$ *experiment in Fig. 7 is bounded by:*

$$\mathsf{Adv}^{\mathsf{X\text{-}CR}}_{\mathsf{AEAD}} := \Pr[\mathrm{Expr}^{\mathsf{X\text{-}CR}}_{\mathsf{AEAD}}(\mathcal{A}) = 1] \leq \epsilon$$

Definition 10. *We say an* $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *has* ϵ-X *input bound ciphertext (or* ϵ-X-IBC *) for* $\mathsf{X} \subseteq (k, N, H, m)$*, if the below defined advantage of any attacker* $\mathcal{A}$ *against* $\mathrm{Expr}^{\mathsf{X\text{-}IBC}}_{\mathsf{AEAD}}$ *experiment in Fig. 8 is bounded by:*

$$\mathsf{Adv}^{\mathsf{X\text{-}IBC}}_{\mathsf{AEAD}} := \Pr[\mathrm{Expr}^{\mathsf{X\text{-}IBC}}_{\mathsf{AEAD}}(\mathcal{A}) = 1] \leq \epsilon$$

$\underline{\mathrm{Expr}^{\mathsf{X\text{-}CR}}_{\mathsf{AEAD}}}$:

1 $\Big((k_1, N_1, H_1, m_1), (k_2, N_2, H_2, m_2)\Big) \leftarrow\!\$\ \mathcal{A}()$
2 **if** $\perp \in \{k_1, N_1, H_1, m_1, k_2, N_2, H_2, m_2\}$
3 **return** 0
4 **if** $f_{\mathsf{X}}(k_1, N_1, H_1, m_1) = f_{\mathsf{X}}(k_2, N_2, H_2, m_2)$
5 **return** 0
6 $c_1 \leftarrow \mathsf{Enc}(k_1, N_1, H_1, m_1)$
7 $c_2 \leftarrow \mathsf{Enc}(k_2, N_2, H_2, m_2)$
8 **return** $[\![c_1 = c_2]\!]$

Figure 7: X-CR security for an AEAD scheme. f_{X} is a projection function maps inputs to the subset indicated by X.

$\underline{\text{Expr}^{\mathsf{X\text{-}IBC}}_{\mathsf{AEAD}}}$:

$\left(c, (k_1, N_1, H_1, m_1), (k_2, N_2, H_2, m_2)\right) \leftarrow\$ \mathcal{A}()$
if $\perp \in \{k_1, N_1, H_1, m_1, k_2, N_2, H_2, m_2\}$
  return 0
if $f_{\mathsf{X}}(k_1, N_1, H_1, m_1) = f_{\mathsf{X}}(k_2, N_2, H_2, m_2)$
  return 0
$m'_1 \leftarrow \mathsf{Dec}(k_1, N_1, H_1, c)$
$m'_2 \leftarrow \mathsf{Dec}(k_2, N_2, H_2, c)$
return $[\![m_1 = m'_1]\!]$ **and** $[\![m_2 = m'_2]\!]$

Figure 8: X-IBC security for an AEAD scheme. f_{X} is a projection function maps inputs to the subset indicated by X.

The above X-CR and X-IBC security notions for $\mathsf{X} \in \{k, (k, N, H), (k, N, H, m)\}$ are respectively identical to the security notions CMT-l and CMTD-l for $l \in \{1, 3, 4\}$ in [8]. More precisely, we have that

1. k-CR = CMT-1
2. (k, N, H)-CR = CMT-3
3. (k, N, H, m)-CR = CMT-4
4. k-IBC = CMTD-1
5. (k, N, H)-IBC = CMTD-3
6. (k, N, H, m)-IBC = CMTD-4

Thus, from the conclusion in [8] we have that X-IBC implies X-CR. Moreover, these two notions are equivalent if the AEAD is tidy.

Theorem 3 ([8, Appendix A])**.** *Let* $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *be an authenticated encryption with associated data scheme. Then, it holds that:*

1. *If* AEAD *is* ϵ-X-IBC *secure, then* AEAD *is also* ϵ-X-CR *secure.* [5]
2. *If* AEAD *is* ϵ-X-CR *secure and tidy, then* AEAD *is also* ϵ-X-IBC *secure.*[6]
3. *If* AEAD *is* ϵ-(k, N, H, m)-IBC *(resp.* CR*) secure, then* AEAD *is* ϵ-(k, N, H)-IBC *(resp.* CR*) secure, and vice versa.*
4. *If* AEAD *is* ϵ-(k, N, H)-IBC *(resp.* CR*) secure, then* AEAD *is* ϵ-k-IBC *(resp.* CR*) secure.*

Other interesting notions are the full robustness FROB and its extension eFROB, which were first defined for the randomized AEAD. In this paper, we define a generalized FROB for nonce-based AEAD.

Definition 11. *We say an* $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *has* ϵ-X *full robustness (or* ϵ-X-FROB *) for* $\mathsf{X} \subseteq (k, N, H, m)$*, if the below defined advantage of any attacker* $\mathcal{A}$ *against* $\text{Expr}^{\mathsf{X\text{-}FROB}}_{\mathsf{AEAD}}$ *experiment in Fig. 9 is bounded by:*

$$\mathsf{Adv}^{\mathsf{X\text{-}FROB}}_{\mathsf{AEAD}} := \Pr[\text{Expr}^{\mathsf{X\text{-}FROB}}_{\mathsf{AEAD}}(\mathcal{A}) = 1] \leq \epsilon$$

$\underline{\text{Expr}^{\mathsf{X\text{-}FROB}}_{\mathsf{AEAD}}}$:

$\left(c, (k_1, N_1, H_1), (k_2, N_2, H_2)\right) \leftarrow\$ \mathcal{A}()$
if $\perp \in \{k_1, N_1, H_1, k_2, N_2, H_2\}$
  return 0
$m_1 \leftarrow \mathsf{Dec}(k_1, N_1, H_1, c)$
$m_2 \leftarrow \mathsf{Dec}(k_2, N_2, H_2, c)$
if $f_{\mathsf{X}}(k_1, N_1, H_1, m_1) = f_{\mathsf{X}}(k_2, N_2, H_2, m_2)$
  return 0
return $[\![m_1 \neq \perp]\!]$ **and** $[\![m_2 \neq \perp]\!]$

Figure 9: X-FROB security for an AEAD scheme. f_{X} is a projection function maps inputs to the subset indicated by X.

The above (k, N)-FROB for nonce-based AEAD is defined in a similar way as the FROB definition for the randomized AEAD in [28]. The above (k, N, H, m)-FROB for nonce-based AEAD is defined in a similar way as the eFROB definition for the randomized AEAD in [30]. For X-CR, X-IBC, and X-FROB, we can have following trivial conclusion that generalizes Theorem 3.

[5] We stress that although [8, Appendix A] only proves the theorem for $\mathsf{X} \in \{k, (k, N, H), (k, N, H, m)\}$, the proof for all other $\mathsf{X} \subseteq (k, N, H, m)$ can be easily given in a similar way.

[6] Similar to above, the proof for all other $\mathsf{X} \subseteq (k, N, H, m)$ can be easily given in a similar way as in [8, Appendix A].

Theorem 4. *Let* $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *be an authenticated encryption with associated data. If* AEAD *is* ϵ-X-CR *(resp.* X-IBC *or* X-FROB*), then it is also* ϵ-X′-CR *(resp.* X′-IBC *or* X′-FROB*) for any* $\mathsf{X}' \subseteq \mathsf{X}$.

Proof. This theorem can by proven by three trivial reductions. Here, we only give the trivial reductions for CR security, the reductions for IBC and FROB can be given in a similar way.

Let $\mathcal{A}$ denotes an attacker that breaks X′-CR security of AEAD. We define an attacker $\mathcal{B}$ that invokes $\mathcal{A}$ and outputs same as $\mathcal{A}$. Note that f_{X} is a projection function that maps inputs to the subset indicated by X. By $\mathsf{X}' \subseteq \mathsf{X}$, we have that $f_{\mathsf{X}'}(k, N, H, m)$ is a subset of $f_{\mathsf{X}}(k, N, H, m)$. This indicates that $f_{\mathsf{X}'}(k_1, N_1, H_1, m_1) \neq f_{\mathsf{X}'}(k_2, N_2, H_2, m_2) \implies f_{\mathsf{X}}(k_1, N_1, H_1, m_1) \neq f_{\mathsf{X}}(k_2, N_2, H_2, m_2)$. Thus, $\mathcal{B}$ wins X-CR security experiment of AEAD whenever $\mathcal{A}$ wins. □

Notably, we find that X-FROB and X-IBC are identical.

Theorem 5. *Let* $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *be an authenticated encryption with associated data. If* AEAD *is* ϵ-X-FROB *secure for* $\mathsf{X} \subseteq (k, N, H, m)$*, then* AEAD *is also* ϵ-X-IBC *secure, and vice versa.*

Proof. Suppose that $\mathcal{A}$ can break the ϵ-X-IBC security of AEAD. Then we can construct an attacker $\mathcal{B}$ that breaks the ϵ-X-FROB security of AEAD. When $\mathcal{A}$ outputs $\Big(c, (k_1, N_1, H_1, m_1), (k_2, N_2, H_2, m_2)\Big)$, $\mathcal{B}$ simply outputs $\Big(c, (k_1, N_1, H_1), (k_2, N_2, H_2)\Big)$. If $\mathcal{A}$ wins, then it must hold that

1. $\perp \notin \{k_1, N_1, H_1, m_1, k_2, N_2, H_2, m_2\}$. In particular, this implies that $\perp \notin \{k_1, N_1, H_1, k_2, N_2, H_2\}$, $m_1 \neq \perp$, and $m_2 \neq \perp$
2. $f_{\mathsf{X}}(k_1, N_1, H_1, m_1) = f_{\mathsf{X}}(k_2, N_2, H_2, m_2)$
3. $\mathsf{Dec}(k_1, N_1, H_1, c) = m_1$. In particular, this implies that $\mathsf{Dec}(k_1, N_1, H_1, c) \neq \perp$
4. $\mathsf{Dec}(k_2, N_2, H_2, c) = m_2$. In particular, this implies that $\mathsf{Dec}(k_2, N_2, H_2, c) \neq \perp$

This implies that $\mathcal{B}$ also wins.

In reverse, suppose that $\mathcal{A}$ can break the ϵ-X-FROB security of AEAD. Then we can construct an attacker $\mathcal{B}$ that breaks the ϵ-X-IBC security of AEAD. When $\mathcal{A}$ outputs $\Big(c, (k_1, N_1, H_1), (k_2, N_2, H_2)\Big)$, $\mathcal{B}$ simply computes $m_1 \leftarrow \mathsf{Dec}(k_1, N_1, H_1, c)$ and $m_2 \leftarrow \mathsf{Dec}(k_2, N_2, H_2, c)$ and outputs

$$\Big(c, (k_1, N_1, H_1, m_1), (k_2, N_2, H_2, m_2)\Big).$$

If $\mathcal{A}$ wins, then it must hold that

1. $\perp \notin \{k_1, N_1, H_1, k_2, N_2, H_2\}$
2. $f_{\mathsf{X}}(k_1, N_1, H_1, m_1) = f_{\mathsf{X}}(k_2, N_2, H_2, m_2)$
3. $m_1 \neq \perp$
4. $m_2 \neq \perp$

This implies that $\perp \notin \{k_1, N_1, H_1, m_1, k_2, N_2, H_2, m_2\}$. Then, $\mathcal{B}$ always wins, which concludes the proof. □

In the literature, there are a number of other related strong security notions. The *key committing* KC security [2] says that different keys but same nonce indicate the different ciphertexts. The multi-key collision resistance (MKCR) [42] says that no attacker can forge any nonce-header-ciphertext tuple (N, H, c) that can be decrypted to a valid message under any key from a attacker-chosen key space with certain minimal cardinality.

Definition 12. *We say an* $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *has* (q, ϵ)*-key commitment (or is* (q, ϵ)-KC *secure), if the below defined advantage of any attacker* $\mathcal{A}$ *against* $\mathrm{Expr}^{\mathsf{KC}}_{\mathsf{AEAD},q}$ *experiment in Fig. 10 is bounded by:*

$$\mathsf{Adv}^{\mathsf{KC}}_{\mathsf{AEAD},q} := \Pr[\mathrm{Expr}^{\mathsf{KC}}_{\mathsf{AEAD},q}(\mathcal{A}) = 1] \leq \epsilon$$

In particular, we say AEAD *is* ϵ-KC *for short, if* AEAD *is* (q, ϵ)-KC *secure for any* $q \geq 2$.

Definition 13. *We say an* $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *with key space* Key *has* (κ, ϵ)*-multi-key collision resistance (or is* (κ, ϵ)-MKCR *secure) for some parameter* $\kappa > 1$*, if the below defined advantage of any attacker* $\mathcal{A}$ *against* $\mathrm{Expr}^{\mathsf{MKCR}}_{\mathsf{AEAD},\kappa}$ *experiment in Fig. 10 is bounded by:*

$$\mathsf{Adv}^{\mathsf{MKCR}}_{\mathsf{AEAD},\kappa} := \Pr[\mathrm{Expr}^{\mathsf{MKCR}}_{\mathsf{AEAD},\kappa}(\mathcal{A}) = 1] \leq \epsilon$$

In particular, we say AEAD *is* ϵ-MKCR *for short, if* AEAD *is* (κ, ϵ)-MKCR *secure for* $\kappa = 2$.

$\underline{\text{Expr}^{\mathsf{KC}}_{\mathsf{AEAD},q}}$:
1 $\mathcal{L} \leftarrow \emptyset$, $n \leftarrow 0$
2 $() \leftarrow \mathcal{A}^{\text{Enc},\text{Dec}}()$
3 **foreach** $(k_1, N_1, H_1, m_1, c_1), (k_2, N_2, H_2, m_2, c_2) \in \mathcal{L}$
4 **if** $k_1 \neq k_2$ **and** $N_1 = N_2$ **and** $c_1 = c_2 \neq \perp$ **and** $m_1 \neq \perp$ **and** $m_2 \neq \perp$
5 **return** 1
6 **return** 0

$\underline{\text{Enc}(k, N, H, m)}$:
7 $c \leftarrow \mathsf{Enc}(k, N, H, m)$
8 **if** $n < q$
9 $\mathcal{L} \leftarrow \mathcal{L} \cup (k, N, H, m, c)$
10 $n \leftarrow n + 1$
11 **return** c

$\underline{\text{Dec}(k, N, H, c)}$:
12 $m \leftarrow \mathsf{Dec}(k, N, H, c)$
13 **if** $n < q$
14 $\mathcal{L} \leftarrow \mathcal{L} \cup (k, N, H, m, c)$
15 $n \leftarrow n + 1$
16 **return** m

$\underline{\text{Expr}^{\mathsf{MKCR}}_{\mathsf{AEAD},\kappa}}$:
1 $(\mathsf{Key}^\star, N^\star, H^\star, c^\star) \leftarrow\!\!\$\ \mathcal{A}()$
2 **if** $|\mathsf{Key}^\star| < \kappa$
3 **return** 0
4 **foreach** $k \in \mathsf{Key}^\star$
5 **if** $\mathsf{Dec}(k, N^\star, H^\star, c^\star) = \perp$
6 **return** 0
7 **return** 1

Figure 10: KC and MKCR security for an AEAD scheme.

Interestingly, [8] has the following observations without giving formal proof. We hereby provide the proof below.

Theorem 6. *Let* $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *be an authenticated encryption with associated data with key space* Key. *Then, it holds that:*

1. *If* AEAD *is* ϵ-k-FROB *secure, then* AEAD *is* ϵ-KC *secure.*
2. *If* AEAD *is* ϵ-KC *secure, then* AEAD *might not be* ϵ'-k-FROB *secure for any negligible* ϵ'.

Proof. We prove each of these statements in turn.

1. Statement 1: Suppose that $\mathcal{A}$ can break the (q, ϵ)-KC security of AEAD for any $q \geq 2$. Then we can construct an attacker $\mathcal{B}$ that breaks the ϵ-k-FROB security of AEAD. $\mathcal{B}$ initializes an empty list $\mathcal{L}$ and simulates experiment $\text{Expr}^{\mathsf{KC}}_{\mathsf{AEAD},q}$ to $\mathcal{A}$. When $\mathcal{A}$ terminates, $\mathcal{B}$ checks there exist entries $(k_1, N_1, H_1, m_1, c_1), (k_2, N_2, H_2, m_2, c_2) \in \mathcal{L}$ such that
   (a) $k_1 \neq k_2$
   (b) $c_1 = c_2 \neq \perp$
   (c) $m_1 \neq \perp$
   (d) $m_2 \neq \perp$

   If such entries do not exist, $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ outputs $\Big(c_1, (k_1, N_1, H_1), (k_2, N_2, H_2)\Big)$.
   If $\mathcal{A}$ wins, then such entries must exist, which further implies that $\mathcal{B}$ wins. The proof is concluded.
2. Statement 2: We prove this statement by giving a counter-example. Let $\mathsf{SKE} = (\mathsf{KGen}', \mathsf{Enc}', \mathsf{Dec}')$ be an one-time pad with spaces $\mathsf{Key}' = \mathsf{Message}' = \{0,1\}^t$ for some $t > 0$. Let $\mathsf{cPRF} : \mathsf{Key}' \times \mathsf{Header} \to \mathsf{Key}' \times \mathsf{Tag}$ denote a $\epsilon^{\mathsf{bind}}_{\mathsf{cPRF}}$-bind secure function for some spaces Header and Tag. We then construct an $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ with spaces $\mathsf{Key} = \mathsf{Nonce} = \mathsf{Key}' = \mathsf{Message}' = \{0,1\}^t$ from SKE and cPRF as follows:
   (a) $\mathsf{KGen}()$: is identical to $\mathsf{KGen}'()$
   (b) $\mathsf{Enc}(k, N, H, m)$: computes $(y, y') \leftarrow \mathsf{cPRF}(k \oplus N, H)$ and $c' \leftarrow \mathsf{Enc}'(y, m \oplus N)$, followed by outputting $c = (c', y')$.
   (c) $\mathsf{Dec}(k, N, H, c)$: parses $(c', y') \leftarrow c$ and verifies whether $(y, y') = \mathsf{cPRF}(k \oplus N, H)$ for some y. If the verification fails, then outputs $\perp$. Otherwise, outputs $\mathsf{Dec}'(y, c') \oplus N$.

   We first prove that AEAD is $\epsilon^{\mathsf{KC}}_{\mathsf{AEAD}}$-KC secure for any $q \geq 2$, where $\epsilon^{\mathsf{KC}}_{\mathsf{AEAD}} \leq \epsilon^{\mathsf{bind}}_{\mathsf{cPRF}}$. Suppose an attacker $\mathcal{A}$ that breaks the KC security of AEAD, then we can construct an attacker $\mathcal{B}$ that breaks bind security of the underlying cPRF. $\mathcal{B}$ simply invokes $\mathcal{A}$ and honestly simulates the KC experiment to $\mathcal{A}$. If $\mathcal{A}$ wins, then there must exist $(k_1, N_1, H_1, m_1, c_1), (k_2, N_2, H_2, m_2, c_2) \in \mathcal{L}$ such that
   (a) $k_1 \neq k_2$
   (b) $N_1 = N_2$
   (c) $c_1 = c_2 \neq \perp$
   (d) $m_1 \neq \perp$ and $m_2 \neq \perp$

This implies that $(k_1 \oplus N_1, H_1) \neq (k_2 \oplus N_2, H_2)$. Moreover, for $c_1 = (c'_1, y'_1)$ and $c_2 = (c'_2, y'_2)$, the condition $c_1 = c_2$ implies that $y'_1 = y'_2$. Then, $\mathcal{B}$ can simply checks all elements in the list $\mathcal{L}$ for such $k_1, k_2, N_1, N_2, y'_1, y'_2$ and outputs $(k_1 \oplus N_1, y'_1, k_2 \oplus N_2, y'_2)$. After that, $\mathcal{B}$ wins whenever $\mathcal{A}$ wins.
We then prove that AEAD is not ϵ'-k-FROB for any negligible ϵ'. An attacker $\mathcal{A}$ can simply execute following steps:

(a) samples $k_1, k_2 \leftarrow\$ \mathsf{Key}$ such that $k_1 \neq k_2$, $N_1 \leftarrow\$ \mathsf{Nonce}$, $H_1 = H_2 \leftarrow\$ \mathsf{Header}$
(b) computes $N_2 = k_1 \oplus k_2 \oplus N_1$
(c) picks any message $m \in \mathsf{Message}$ and computes $c \leftarrow \mathsf{AEAD}(k_1, N_1, H_1, m)$
(d) outputs $\Big(c, (k_1, N_1, H_1), (k_2, N_2, H_2)\Big)$

It is straightforward that for $k_1 \neq k_2$, $m_1 = m \neq \perp$, and $m_2 = m_1 \oplus N_1 \oplus N_2 \neq \perp$ for $m_1 \leftarrow \mathsf{Dec}(k_1, N_1, H_1, c)$ and $m_2 \leftarrow \mathsf{Dec}(k_2, N_2, H_2, c)$. Thus, $\mathcal{A}$ always wins.

□

Moreover, we also find that KC security implies MKCR security, while the reverse direction does not hold.

Theorem 7. *Let* $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *be an authenticated encryption with associated data with key space* Key. *Then, it holds that:*

1. *If* AEAD *is* ϵ-KC *secure, then* AEAD *is* ϵ-MKCR *secure.*
2. *If* AEAD *is* ϵ-MKCR *secure, then* AEAD *might not be* ϵ'-KC *secure for any negligible* ϵ'.

Proof. We prove each of these statements in turn.

1. Statement 1: We prove this statement by reduction. If there exists an attacker $\mathcal{A}$ that breaks the MKCR security of AEAD with probability ϵ, then we can construct an attacker $\mathcal{B}$ that breaks the KC security of AEAD also with probability ϵ. The attacker $\mathcal{B}$ simply invokes $\mathcal{A}$. When $\mathcal{A}$ outputs $(\mathsf{Key}^\star, N^\star, H^\star, c^\star)$, $\mathcal{B}$ picks two arbitrary $k_1, k_2 \in \mathsf{Key}^\star$ with $k_1 \neq k_2$. Then, $\mathcal{B}$ queries DEC oracle twice, respectively with inputs $(k_1, N^\star, H^\star, c^\star)$ and $(k_2, N^\star, H^\star, c^\star)$. If $\mathcal{A}$ wins, then it must hold that $\perp \neq m_1 = \mathsf{Dec}(k_1, N^\star, H^\star, c^\star)$ and $\perp \neq m_2 = \mathsf{Dec}(k_2, N^\star, H^\star, c^\star)$. Thus, $\mathcal{B}$ always wins.
2. Statement 2: We prove this statement by giving a counter-example. Let $\mathsf{SKE} = (\mathsf{KGen}', \mathsf{Enc}', \mathsf{Dec}')$ be an one-time pad with spaces $\mathsf{Key}' = \mathsf{Message}' = \{0,1\}^t$ for some $t > 0$. Let $\mathsf{cPRF} : \mathsf{Key}' \times \mathsf{Nonce} \rightarrow \mathsf{Key}' \times \mathsf{Tag}$ denote a $\epsilon^{\mathsf{bind}}_{\mathsf{cPRF}}$-cPRF secure function for some spaces Nonce and Tag. We then construct an $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ with spaces $\mathsf{Key} = H$ from SKE and F as follows:
   (a) $\mathsf{KGen}()$: is identical to $\mathsf{KGen}'()$
   (b) $\mathsf{Enc}(k, N, H, m)$: computes $(y, y') \leftarrow \mathsf{cPRF}(k \oplus H, N)$ and $c' \leftarrow \mathsf{Enc}'(y, m)$, followed by outputting $c = (c', y')$.
   (c) $\mathsf{Dec}(k, N, H, c)$: parses $(c', y') \leftarrow c$ and verifies whether $(y, y') = \mathsf{cPRF}(k \oplus H, N)$ for some y. If the verification fails, then outputs $\perp$. Otherwise, outputs $\mathsf{Dec}'(y, c')$.

   We first prove that AEAD is ϵ-MKCR secure, where $\epsilon = \epsilon^{\mathsf{bind}}_{\mathsf{cPRF}}$. Suppose an attacker $\mathcal{A}$ that breaks the MKCR security of AEAD, we then construct an attacker $\mathcal{B}$ that breaks the bind security of the underlying cPRF. $\mathcal{B}$ simply invokes $\mathcal{A}$ and honestly simulates the MKCR experiment for $\kappa = 2$. The attacker $\mathcal{A}$ wins if it can output $(\mathsf{Key}^\star, N^\star, H^\star, c^\star)$ such that
   (a) $|\mathsf{Key}^\star| \geq 2$, and
   (b) $\mathsf{Dec}(k, N^\star, H^\star, c^\star) \neq \perp$ for all $k \in \mathsf{Key}^\star$

   Then, $\mathcal{B}$ simply picks any $k_1 \neq k_2$ from space $\mathsf{Key}^\star$. It must hold that $(k_1 \oplus H^\star, N^\star) \neq (k_2 \oplus H^\star, N^\star)$. The decryption $\mathsf{Dec}(k_1, N^\star, H^\star, c^\star) \neq \perp$ and $\mathsf{Dec}(k_2, N^\star, H^\star, c^\star) \neq \perp$ indicates that the verification inside the decryption never fails. This means, for $(y_1, y'_1) \leftarrow \mathsf{cPRF}(k_1 \oplus H^\star, N^\star)$ and $(y_2, y'_2) \leftarrow \mathsf{cPRF}(k_2 \oplus H^\star, N^\star)$ it must hold that $y'_1 = y'_2$. Thus, $\mathcal{B}$ always wins if it outputs $(k_1 \oplus H^\star, N^\star, k_2 \oplus H^\star, N^\star)$.

   Then, we prove that AEAD is not ϵ'-KC secure for any non-negligible ϵ'. An attacker $\mathcal{A}$ can simply pick arbitrary $(k_1, N_1, H_1, m) \in \mathsf{Key} \times \mathsf{Nonce} \times \mathsf{Header} \times \mathsf{Message}$ and invokes ENC oracle with the input (k_1, N_1, H_1, m) for a ciphertext c_1. Then, $\mathcal{A}$ invokes DEC oracle with input (k_2, N_2, H_2, c_2) for m_2 such that $k_1 \neq k_2$, $k_2 \oplus H_2 = k_1 \oplus H_1$, $N_1 = N_2$, and $c_1 = c_2$. It is straightforward that $m_1 = m_2 \neq \perp$ and $\mathcal{A}$ always wins, which concludes the proof.

□

In the realm of ccAEAD, there are two important notions: sender binding s-BIND and receiver binding r-BIND [30]. While the s-BIND property ensures that the attacker cannot forge any (k, H, c) tuple such that the decrypted message can be verified using the opened verification key. The r-BIND ensures that each ciphertext is bound to the same nonce-header-message tuple.

Definition 14. *We say an* ccAEAD = (KGen, Enc, Dec, openCommit, vrfyCommit) *has ϵ-sender binding (or is ϵ-s-BIND secure), if the below defined advantage of any attacker $\mathcal{A}$ against* $\mathrm{Expr}^{\text{s-BIND}}_{\text{AEAD}}$ *experiment in Fig. 11 is bounded by:*

$$\mathsf{Adv}^{\text{s-BIND}}_{\text{ccAEAD}} := \Pr[\mathrm{Expr}^{\text{s-BIND}}_{\text{ccAEAD}}(\mathcal{A}) = 1] \leq \epsilon$$

Definition 15. *We say an* ccAEAD = (KGen, Enc, Dec, openCommit, vrfyCommit) *has ϵ-receiver binding (or is ϵ-r-BIND secure), if the below defined advantage of any attacker $\mathcal{A}$ against* $\mathrm{Expr}^{\text{r-BIND}}_{\text{AEAD}}$ *experiment in Fig. 12 is bounded by:*

$$\mathsf{Adv}^{\text{r-BIND}}_{\text{ccAEAD}} := \Pr[\mathrm{Expr}^{\text{r-BIND}}_{\text{ccAEAD}}(\mathcal{A}) = 1] \leq \epsilon$$

$\underline{\mathrm{Expr}^{\text{s-BIND}}_{\text{ccAEAD}}}$:

$(k, N, H, c) \leftarrow\!\$\ \mathcal{A}()$
$m \leftarrow \mathsf{Dec}(k, N, H, c)$
$k_f \leftarrow \mathsf{openCommit}(k, N, H, c)$
if $m = \perp$
  return 0
if $\mathsf{vrfyCommit}(k_f, N, H, m, c)$
  return 0
return 1

Figure 11: s-BIND security for an ccAEAD = (KGen, Enc, Dec, openCommit, vrfyCommit) scheme.

$\underline{\mathrm{Expr}^{\text{r-BIND}}_{\text{ccAEAD}}}$:

$\Big(c, (k_{f_1}, N_1, H_1, m_1), (k_{f_2}, N_2, H_2, m_2)\Big) \leftarrow\!\$\ \mathcal{A}()$
if $\perp \in \{k_1, N_1, H_1, m_1, k_2, N_2, H_2, m_2\}$
  return 0
if $(H_1, m_1) = (H_2, m_2)$
  return 0
if $\mathsf{vrfyCommit}(k_{f_1}, N_1, H_1, m_1, c)$ **and** $\mathsf{vrfyCommit}(k_{f_2}, N_2, H_2, m_2, c)$
  return 1
return 0

Figure 12: r-BIND security for an ccAEAD = (KGen, Enc, Dec, openCommit, vrfyCommit) scheme.

From an AEAD = (KGen, Enc, Dec), we can easily extend to ccAEAD[AEAD] = (KGen, Enc, Dec, openCommit, vrfyCommit) using "traditionally committing encryption" approach [30], such that

$$\mathsf{openCommit}(k, N, H, c) := k$$
$$\mathsf{vrfyCommit}(k, N, H, m, c) := [\![m = \mathsf{Dec}(k, N, H, c)]\!]$$

It is interesting to observe that any ccAEAD[AEAD] is s-BIND secure. Moreover, it is stated in [30] that the r-BIND security for the randomized "traditionally committing encryption" AEAD can be implied by eFROB but cannot be implied by the standard FROB security without including headers. In terms of our syntax, we have similar conclusions.

Theorem 8. *Let* AEAD = (KGen, Enc, Dec) *be an authenticated encryption with associated data. Let* ccAEAD[AEAD] *denote the compactly committing* AEAD *derived from* AEAD *using "traditionally committing encryption" approach. Then, it holds that:*

1. ccAEAD[AEAD] *is* 0-s-BIND *secure.*
2. *If* AEAD *is ϵ-*X-FROB *secure for $(H, m) \subseteq \mathsf{X}$, then* ccAEAD[AEAD] *is also ϵ-*r-BIND *secure.*
3. *If* ccAEAD[AEAD] *is ϵ-*r-BIND *secure, then* ccAEAD[AEAD] *is also ϵ-*X-FROB *secure for $\mathsf{X} \subseteq (H, m)$.*

Proof. We prove each of these statements in turn.

1. Statement 1: For any (k, N, H, c) output by $\mathcal{A}$ and $m \leftarrow \mathsf{Dec}(k, N, H, c)$, $\mathcal{A}$ can win only when
   (a) $m \neq \perp$, and
   (b) $m \neq \mathsf{Dec}(k, N, H, c)$.
   The second condition contracts to the fact that $m \leftarrow \mathsf{Dec}(k, N, H, c)$. Thus, $\mathcal{A}$ always loses.
2. Statement 2: We prove this claim by reduction. If there exists an attacker $\mathcal{A}$ that breaks the ϵ-r-BIND security of ccAEAD[AEAD], then we can construct an attacker $\mathcal{B}$ that breaks the ϵ'-X-FROB security of AEAD for $(H, m) \subseteq \mathsf{X}$ and $\epsilon' = \epsilon$. When $\mathcal{A}$ outputs $\Big(c, (k_1, N_1, H_1, m_1), (k_2, N_2, H_2, m_2)\Big)$, $\mathcal{A}$ wins if

(a) $\perp \notin \{k_1, N_1, H_1, m_1, k_2, N_2, H_2, m_2\}$
(b) $(H_1, m_1) \neq (H_2, m_2)$
(c) $m_1 = \mathsf{Dec}(k_1, N_1, H_1, c)$
(d) $m_2 = \mathsf{Dec}(k_2, N_2, H_2, c)$

This in particular indicates that
(a) $\perp \notin \{k_1, N_1, H_1, k_2, N_2, H_2\}$,
(b) $m_1 = \mathsf{Dec}(k_1, N_1, H_1, c)$
(c) $m_2 = \mathsf{Dec}(k_2, N_2, H_2, c)$
(d) $f_{\mathsf{X}}(k_1, N_1, H_1, m_1) \neq f_{\mathsf{X}}(k_2, N_2, H_2, m_2)$ for any $(H, m) \subseteq \mathsf{X}$
(e) $m_1 \neq \perp$ and $m_2 \neq \perp$

Thus, $\mathcal{B}$ can simply output $\Big(c, (k_1, N_1, H_1), (k_2, N_2, H_2)\Big)$ and win whenever $\mathcal{A}$ wins.

3. Statement 3: We prove this claim by reduction. If there exists an attacker $\mathcal{A}$ that breaks the ϵ-X-FROB security of ccAEAD[AEAD] for $\mathsf{X} \subseteq (H, m)$, then we can construct an attacker $\mathcal{B}$ that breaks the ϵ'-r-BIND security of ccAEAD[AEAD] and $\epsilon' = \epsilon$. When $\mathcal{A}$ outputs $\Big(c, (k_1, N_1, H_1), (k_2, N_2, H_2)\Big)$, $\mathcal{A}$ wins if
(a) $\perp \notin \{k_1, N_1, H_1, k_2, N_2, H_2\}$,
(b) for $m_1 := \mathsf{Dec}(k_1, N_1, H_1, c)$ and $m_2 := \mathsf{Dec}(k_2, N_2, H_2, c)$ it holds that $f_{\mathsf{X}}(k_1, N_1, H_1, m_1) \neq f_{\mathsf{X}}(k_2, N_2, H_2, m_2)$ for any $\mathsf{X} \subseteq (H, m)$
(c) $m_1 \neq \perp$ and $m_2 \neq \perp$

This in particular indicates that
(a) $\perp \notin \{k_1, N_1, H_1, m_1, k_2, N_2, H_2, m_2\}$
(b) $(H_1, m_1) \neq (H_2, m_2)$
(c) $m_1 = \mathsf{Dec}(k_1, N_1, H_1, c)$
(d) $m_2 = \mathsf{Dec}(k_2, N_2, H_2, c)$

Thus, $\mathcal{B}$ can simply output $\Big(c, (k_1, N_1, H_1, m_1), (k_2, N_2, H_2, m_2)\Big)$, where $m_1 := \mathsf{Dec}(k_1, N_1, H_1, c)$ and $m_2 := \mathsf{Dec}(k_2, N_2, H_2, c)$, and win whenever $\mathcal{A}$ wins.

□

Moreover, we also observe that neither KC nor MKCR security of AEAD implies the r-BIND security of ccAEAD[AEAD]. Conversely, the r-BIND security of ccAEAD[AEAD] does not imply KC or MKCR security.

Theorem 9. *Let* $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *be an authenticated encryption with associated data scheme. Let* ccAEAD[AEAD] *denote the compactly committing* AEAD *derived from* AEAD *using "traditionally committing encryption" approach. Then, it holds that:*

1. *If* AEAD *is* ϵ-KC *secure, then* ccAEAD[AEAD] *might not be* ϵ'-r-BIND *secure for any negligible* ϵ'.
2. *If* AEAD *is* ϵ-MKCR *secure, then* ccAEAD[AEAD] *might not be* ϵ'-r-BIND *secure for any negligible* ϵ'.
3. *If* ccAEAD[AEAD] *is* ϵ-r-BIND *secure, then* ccAEAD[AEAD] *might not be* ϵ'-KC *secure for any negligible* ϵ'.
4. *If* ccAEAD[AEAD] *is* ϵ-r-BIND *secure, then* ccAEAD[AEAD] *might not be* (κ, ϵ')-MKCR *secure for any* $\kappa \geq 2$ *and any negligible* ϵ'.

Proof. We prove each of these statements in turn.

1. Statement 1 and 2: We prove these statements by a counter-example AEAD, which is identical to the one in the proof of Statement 2 in Theorem 6. As shown in Theorem 6, we know that AEAD is KC secure. By Theorem 7, we know that AEAD is also MKCR secure. Below, we prove that ccAEAD[AEAD] is not ϵ'-r-BIND secure for any negligible ϵ'.
   An attacker $\mathcal{A}$ can simply pick arbitrary $k_1, k_2 \in \mathsf{Key}$, $N_1, N_2 \in \mathsf{Nonce}$, $H_1, H_2 \in \mathsf{Header}$, $m_1 \in \mathsf{Message}$ such that $k_1 \neq k_2$, $N_2 = N_1 \oplus k_1 \oplus k_2$, $H_1 = H_2$. Then, $\mathcal{A}$ computes $c \leftarrow \mathsf{Enc}(k_1, N_1, H_1, m_1)$ and $m_2 \leftarrow \mathsf{Dec}(k_2, N_2, H_2, c)$. It holds that $m_2 = m_1 \oplus N_1 \oplus N_2$. By $k_1 \neq k_2$, we know that $m_2 = m_1 \oplus N_1 \oplus N_2 = m_1 \oplus k_1 \oplus k_2 \neq m_1$ and therefore $(H_1, m_1) \neq (H_2, m_2)$. Finally, $\mathcal{A}$ outputs $\Big(c, (k_1, N_1, H_1, m_1), (k_2, N_2, H_2, m_2)\Big)$ and always wins.
2. Statement 3: We prove this statement by a counter-example ccAEAD[AEAD]. Let $\mathsf{SKE} = (\mathsf{KGen}', \mathsf{Enc}', \mathsf{Dec}')$ denote one-time pad with spaces $\mathsf{Key}' = \mathsf{Message}' = \{0,1\}^t$ for some $t > 0$. We then define $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ with space $\mathsf{Key} = \{0,1\}^t$ and $\mathsf{Message} = \{0,1\}^{\frac{t}{2}}$ from SKE and a collision-resistant function $\mathsf{F} : \mathsf{Header} \times \mathsf{Message} \to \mathsf{Tag}$ for some space Tag as follows:
   - $\mathsf{KGen}()$: identical to $k \leftarrow\!\!\$\ \mathsf{KGen}'()$.
   - $\mathsf{Enc}(k, N, H, m)$: runs $c' \leftarrow \mathsf{Enc}'(k, m \parallel m)$ and $t \leftarrow \mathsf{F}(H, m)$, followed by outputting $c \leftarrow c' \parallel t$.

- $\mathsf{Dec}'(k, N, H, c)$: parses $c' \parallel t \leftarrow c$ and runs $m \parallel m' \leftarrow \mathsf{Dec}(k, c')$, followed by outputting $\perp$ if $t \neq \mathsf{F}(H, m)$ and m otherwise.

We first prove that ccAEAD[AEAD] is r-BIND: Note that an attacker $\mathcal{A}$ can break the r-BIND security of AEAD only when $\mathcal{A}$ outputs $(H_1, m_1) \neq (H_2, m_2)$. By the collision resistance of the underlying F, we know that $\mathsf{F}(H_1, m_1) \neq \mathsf{F}(H_2, m_2)$ except negligible probability. This further implies that for any $c = c' \parallel t$, at least one of the conditions $t = \mathsf{F}(H_1, m_1)$ and $t = \mathsf{F}(H_2, m_2)$ cannot hold except negligible probability. Thus, $\mathcal{A}$ can never win the r-BIND experiment with non-negligible probability. Below, we prove that ccAEAD[AEAD] is not ϵ'-KC secure for any negligible ϵ'. An attacker $\mathcal{A}$ can simply pick arbitrary $(k_1, N_1, H_1, m_1) \in \mathsf{Key} \times \mathsf{Nonce} \times \mathsf{Header} \times \mathsf{Message}$ and invoke Enc oracle with input (k_1, N_1, H_1, m_1) for a ciphertext c. Then, $\mathcal{A}$ sets k_2 identical to k_1 except flipping the final bit, $N_2 = N_1$, and $H_2 = H_1$, followed by querying Dec oracle with input (k_2, N_2, H_2, c) for a message m_2. It is straightforward that $m_2 = m_1 \neq \perp$ and $\mathcal{A}$ always wins.

3. Statement 4: We prove this statement by a counter-example ccAEAD[AEAD], which is identical to the one in above proof of Statement 3. From the above statement, we know that ccAEAD[AEAD] is r-BIND secure. Below, we prove that ccAEAD[AEAD] is not ϵ'-MKCR secure for any $\kappa \geq 2$ and any negligible ϵ'.
   An attacker $\mathcal{A}$ can easily pick $(k^\star, N^\star, H^\star, m^\star) \in \mathsf{Key} \times \mathsf{Nonce} \times \mathsf{Header} \times \mathsf{Message}$ and compute $c^\star \leftarrow \mathsf{Enc}(k^\star, N^\star, H^\star, m^\star)$. Next, $\mathcal{A}$ sets $\mathsf{Key}^\star = \{k : k \text{ and } k^\star \text{ have the same first half bits}\}$. Finally, $\mathcal{A}$ outputs $(\mathsf{Key}^\star, N^\star, H^\star, c^\star)$. We have that for any $k \in \mathsf{Key}^\star$, $\mathsf{Dec}(k, N^\star, H^\star, c^\star) = m^\star \neq \perp$. Moreover, recall that $\mathsf{Key} = \{0,1\}^t$ and $\mathsf{Message} = \{0,1\}^{\frac{t}{2}}$ for arbitrary $t > 0$. For any $\kappa \geq 2$, $\mathcal{A}$ can always find suitable $t > 0$ such that $|\mathsf{Key}^\star| = 2^{\frac{t}{2}} \geq \kappa$. Thus, $\mathcal{A}$ always wins, which concludes the proof. □

C Further details on the Case Studies

We summarize our attacks and undesired behaviors, and discuss their practicality in Table 3. We also provide a summary of the current content agreement guarantees of multiple messaging mechanisms in Table 5, illustrating a discrepancy between them.

C.1 Key Secrecy

YubiHSM The YubiHSM [60] is hardware security module by Yubico to generate, store and manage cryptographic key material. It implements an API to strictly separate key usage from its applications, to mainly prevent full or partial leakage of secure key material.

In earlier versions of the YubiHSM, [41] found an attack on the YubiHSM API to leak secret keys by exploiting the ability of the user to specify nonces. With the underlying AEAD not being nonce-reuse resistant, they were able to leak secret keys.

By now, this very nonce-reuse issue is known and well-studied throughout the community. However, just recently, Samsungs Trustzone [54] was found to have the same kind of attack, demonstrating that nonce misuse is still worth paying attention towards.

When instantiating the original Tamarin model by [41] with our AEAD library, we efficiently rediscover the attack using k-NR.

C.2 Authentication

SFrame SFrame is a communication protocol developed by CoSMo Software and Google [48] with the goal to be used for online audio and video meeting protocols. It uses end-to-end encryption and is made to support groups of multiple users.

[32] found forgery attacks on the authentication of the SFrame protocol. For a malicious user of the protocol, who is part of a group, it is enough to find collisions on the authentication tags of the used AEAD to break authentication of the messages. This is mainly possible by only explicitly authenticating on the tags of AEADs instead of the full ciphertexts. They reported the attack to be practical on:

1. schemes with short tag length, or
2. schemes that allow to easily find collisions with key knowledge.

We modeled the SFrame protocol with its groups and the sending and receiving of frames. We modeled it against an attacker with the power to join groups and the power to act as a group participant. Using our extended AEAD models, which allows adding explicit tags, Tamarin could quickly find the reported attack by using collisions under the tags.

When exploring all possible scenarios of our AEAD models, TAMARIN also found a potential attack on the same authentication property as in the original attack. Executing it would require to produce a full AEAD ciphertext collision (Full-mColl & Full-nColl) instead of a collision on the tags. However, this attack does not appear to be practical and directly implies collision of tags as a collision on the whole ciphertext is computationally harder.

C.3 Content Agreement

GPG GPG is the golden standard for file and mail encryption and signing. We review its ongoing cryptographic update [37]. It contains three different encryption formats:

- *Symmetrically Encrypted Data* (SED) – the legacy encryption with the dedicated GPG-CFB encryption mode. It is now marked as deprecated, and accepting such messages must raise a warning.
- *Version 1 Symmetric Encrypted Integrity Protected Data* (SEIPD v1) – the legacy authenticated encryption format, not deprecated.
- SEIPD v2 – the current proposal relying on AEADs.

SED is not integrity protected, and is simply a symmetric encryption with a dedicated mode. SEIPD v1 is a variant, still relying on a dedicated encryption mode, and given the plaintext p, returns the encryption of $p \| SHA1(p)$ (abstracting away some message formatting). SEIPD v2 relies on AEADs, with the specificity that the plaintext can be split into chunks, each chunk corresponding to a call with the same AEAD and same key but different nonces. A final specific tag is always appended to the ciphertext by computing a final AEAD over a *null* plaintext with the same key and a nonce depending on the number of chunks.

We modeled all three encryption modes, checking if an attacker can send the same message to different recipients. In all cases, our automated analysis reported attacks, but under different collision capabilities. That gives us the following practical consequences:

- SED is trivially broken, and even more so as we showed that collisions over the dedicated mode can be found in constant time (Appendix A). The severity is however low as it is to be deprecated.
- We find that for a non collision resistant encryption, SEIPD v1 is theoretically broken, but appears to be secure in practice. While the attacker can brute force the keys to try to come up with collisions, the SHA1 value appended to the plaintext puts too many constraints over the collision. This indicates however that if e.g. weak keys were found for AES, this could be attacked.
- The results are similar for SEIPD v2. By appending an additional call to the AEAD (either GCM, EAX or OCB) with the same key, it implies that in practice, one would need to find not one collision, but a pair of collisions, which greatly increases the complexity and makes the attack impractical. Going back to the variants of known collision that are practical (Table 2), finding two collisions at the same time has not been explored, and is for the moment an open question. As such, we consider this to not be possible in practice as of now.

Scuttlebutt Scuttlebutt is a protocol that provides an authenticated append-only feed. The private box feature [53] allows publishing encrypted messages that are uploaded in public and meant for multiple recipients. In this case, Content Agreement appears to be valuable and intuitively expected.

We model the mechanism with a malicious sender, and TAMARIN does report an attack under the collision models KeysColl and nColl. Scuttlebutt uses the XSalsa20-Poly1305 AEAD, a variant of ChaCha20-Poly1305 (which is in Table 2). Hence, this attack appears to be feasible under the condition that the known attacks against ChaCha20-Poly1305 can be translated to XSalsa20-Poly1305.

SaltPack SaltPack is a proposed alternative to GPG. We modeled the surprisingly involved version 2 format [52].

Nonces and integrity packet checks are derived with multiple iterations of MACs and hashes. Notably, after a fresh payload key k has been asymmetrically encrypted for each of the recipients public key, a MAC is computed for each recipient. The payload key k is used both to encrypt the desired plaintext $\mathsf{Enc}(k, N_1, \emptyset, m)$, but also the sender public key spk with $\mathsf{Enc}(k, N_2, \emptyset, spk)$. We use $\emptyset$ to denote empty headers.

On this protocol, Tamarin does report an attack with Full-mColl. Intuitively, it seems that the scheme ensures consistency, as we need once again to come up with a pair of collisions for the desired plaintext

m_1, m_2:

$$\mathsf{Enc}(k_1, N_1, \emptyset, m_2) = \mathsf{Enc}(k_2, N_1, \emptyset, m_1)$$
$$\mathsf{Enc}(k_1, N_2, \emptyset, spk) = \mathsf{Enc}(k_2, N_2, \emptyset, spk)$$

This is a similar kind of collisions that is required for the GPG case, and is once again not possible in practice with respect to the current techniques.

D Choosing the correct AEAD model

Whereas using the fully automated methodology from the previous section covers all AEAD models, it can be out-of-scope for complex and detailed protocol models. As complex protocol models often need manual work to aid automation, it might be more feasible to a priori choose the correct AEAD model for the instantiations actually used in the protocol. We demonstrate a way to choose the right combinations of AEAD models on the example of a toy protocol using AES-GCM. Assume that the protocol explicitly adds the functionality that compares the tag instead of using authenticated decryption of the ciphertext:

- As a first step check whether your protocol specification forbids sending the nonce used for AES-GCM. If no, add Leak to you AEAD model combination.
- Check Table 1 and see if the the AEAD is resistant to nonce-reuse attacks. For AES-GCM we see that an XOR of plaintexts can be leaked and there is the possibility to forge ciphertexts. Here, add k-NR to the AEAD models. As this is an over-approximation of the before-mentioned weakness, you can also decide to instead of leaking the encryption key, to leak the XOR of plaintext (if your tool of choice allows modeling of XOR) or to output a forged ciphertext under the given key.
- When checking Table 1 again, AES-GCM is not collision resistant. Then we check Table 2 and see that AES-GCM is also vulnerable to collisions of type KeysColl (,KeyColl), and nColl. As KeyColl is strictly stronger than KeysColl, we only need to add KeyColl and nColl to the set of combinations. However, if we would like to future proof the protocol (and we know that AES-GCM is not collision-resistant) we could also decide to add the strongest collision models, e.g. FullKeyColl, instead. With this, we could see if the protocol relies on collision resistant AEADs.
- As the described protocol explicitly uses AES-GCM tags we would also add the Tag models. As collisions on tags are as hard or even easier than finding collisions on the AEAD scheme itself, we would recommend to use at least the same kind of collision types for tags as well, for instance FullKeyTag.