Automated Analysis of Protocols that use Authenticated Encryption: Analysing the Impact of the Subtle Differences between AEADs on Protocol Security

***Abstract*—Many modern security protocols such as TLS, WPA2, WireGuard, and Signal use a cryptographic primitive called Authenticated Encryption (optionally with Authenticated Data) that are also known as an AEAD scheme. AEADs are usually built from (symmetric) encryption schemes and additionally provide authentication. While this may seem to be a straightforward and intuitive additional requirement, it has in fact turned out to be complex: many different security notions for AEADs are still being proposed, and many recent protocol-level attacks that exploit subtle AEAD behaviors suggest that protocol designers struggle with the many complex edge cases of AEAD guarantees and assumptions.**

In this work, we provide the first automated security analysis method for protocols that use AEADs and that can account for the subtleties of the specific AEAD that is being 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 show that our approach can automatically and efficiently discover protocol attacks that were previously found using manual inspection, such as the Salamander attack on Facebook's message franking, and attacks on SFrame and YubiHSM. Furthermore, we discover several new attacks.

A major hurdle for our work is the divergent landscape of AEAD definitions and their mismatch to real-world attack scenarios. To address this, we revisit several recent cryptographic AEAD definitions, prove new results about their relations, and extract the core requirements for a generic AEAD model. Our generic AEAD model enables us to develop a family of symbolic AEAD models that can be used with symbolic protocol analysis tools. We instantiate them for the Tamarin prover, which we use for our case studies.

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's so-called record layer, WPA2 from IEEE 802.11 (WiFi), WireGuard, and transmissions by messaging apps such as Signal or WhatsApp.

While they are ubiquitous in modern secure communications, there is no commonly agreed "strong" security notion that AEADs should satisfy. In fact, the current landscape of security notions for AEAD is rather chaotic: there are many proposed frameworks and security notions variants [1, 2, 3, 5, 6, 7, 11, 13, 19, 24, 25, 36, 44, 48]. For some of these, their implication relations are known [5], but many notions are hard to compare for technical reasons.

In reality, there are good examples of recent protocol attacks that exploit weaknesses of some AEAD schemes, such as [19, 27, 35]. These have all been found through time-consuming manual inspection of the protocol and knowledge of the particular AEAD scheme, such as exploiting nonce reuse. Some of these are related to a subset of the proposed security notions, but there is a surprising discrepancy between the set of AEAD-based protocol attacks and the proposed security notions: several new security notions have not been motivated by concrete attack examples, but have instead evolved over modeling technicalities, and this has led to a disconnect between attacks and security notions.

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. However, even for protocols for which we have proofs in any framework, they are not modeled in sufficient detail or with respect to the right property to discover subtle attacks based on the behaviour of the AEAD. At a methodological level, these attacks can be hard to model because they require a methodology that is not only low-level enough to capture AEAD details (e.g. impact of nonce reuse) but also scales well enough to allow for modeling the often complex state machines that determine whether such attack requirements can be met.

Our methodology/framework would have, for instance, been helpful in the analysis of the WPA2 protocol. WPA2 has a very complex state machine: originally only a very abstract security proof existed [26] that neither considered the complex state machine nor any AEAD details. Years later, an AEAD-based attack was manually found and a fix was proposed [51], but without a proof; the same authors showed a year later that their fix was insufficient [52] and proposed a new fix. A protocol-specific symbolic analysis that detected these attacks and could show that the second fix works was only developed much later [16].

In this work, we develop the first systematic methodology for the automated analysis of security protocols that use AEADs primitives, to either find attacks or show their absence. For our methodology, we leverage the TAMARIN prover [38], a protocol analysis tool, which we augment with

novel fine-grained models of AEAD primitives.

To develop meaningful symbolic models, we first have to overcome the hurdle of the many hard to compare theoretical security notions and their relations to real-world attacks. We thus have to carry out a thorough review of both theoretical and practical concerns, such that we can identify the core features and weaknesses that must be captured by our generic AEAD models. In our review of the landscape, we notably highlight how the *collision resistance* of AEAD is a central issue of AEAD's design: we illustrate how a lack of it leads to multiple classes of attacks, and satisfying this security property implies that many existing security notions are met. In our case studies, we rediscover previously reported classes of attacks over either *accountability* or *authentication*, but also notably identify a new class of attacks with respect to *content agreement* in group messaging scenarios: can a dishonest member of a group send a single message that will be interpreted differently by multiple parties?

Our intended audience includes those interested in formal analysis methods, new protocol attacks, as well as those who would like to better understand the AEAD landscape.

Contributions. Our main contributions are the following:

- We develop the first systematic automated methodology for the analysis of security protocols with respect to specific AEAD instantiations.
- In case studies, we show our methodology effectively rediscovers known attacks on YubiHSM [35], Facebook's Message Franking [19], and SFrame [27]. With our methodology we also rediscover a theoretical attack variant on Facebooks Message Franking first mentioned by [24] and several new potential attacks on GPG [32], Saltpack [46], Scuttlebutt [47], Webpush [50], and Whatsapp Group Messaging [54].
- We review the theoretical and practical AEADs landscape, extract the core requirements for our generic AEAD model, and prove new relations between existing AEAD notions.

We provide the full formal Tamarin models and analysis scripts at [49].

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, setting up the symbolic modeling and analysis approach that we develop in Section 4. We evaluate our approach in Section 5. We discuss limitations in Section 6, disclosure concerns in Section 7, and conclude in Section 8.

We give all the theorems and proofs in Appendix C.

2. Background on AEADs and protocol attacks

In this section, we review the main attacks on protocols based on subtle AEADs behaviors and weaknesses in Section 2.1. In Section 2.2, we recall the formal syntax of AEADs and the two core definitions for privacy and integrity. In Section 2.3, we identify the various dimensions that play a role in the variations of definitions for AEAD schemes that have been given in the literature.

2.1. Historical protocol attacks exploiting AEADs

Historically, the security performance of AEAD schemes is often evaluated based solely on some canonical security guarantees, such as privacy and integrity. However, some recent attacks that effectively break the security of several practical applications suggest that the underlying AEAD schemes need an evolution to meet stronger security guarantees. We give six main protocol attack classes that occurred in the wild that relying on AEADs and symmetric encryption schemes, briefly describing their high-level requirements.

A1 Nonce reuse attacks - Many nonce-based symmetric encryption schemes assume the nonces, as the name indicates, are the numbers used only once. If the nonces are wrongly reused, for instance, due to the flawed implementation or the interfere with the generation of the nonces by attackers, then the security of the underlying schemes becomes in doubt.

On the one hand, the generation of nonces is protocol-specific and error-prone. A practical example is the Zerologon attack[1], which 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.

On the other hand, the generation of nonces might involve external data in practice. For instance, the nonces are sometimes produced by using information from external users, as happened for Yubikey [35] or Trustzone [48]. In this case, a malicious user might provide information that causes the collision of nonces. Other times, the nonces maybe implemented as counters inside protocols, as happened for WPA2 [51]. On an insecure hardware environment, an attacker might maliciously trigger the counter reset.

A2 Padding oracle attacks [53] - Many AEADs and symmetric encryption schemes are constructed from block ciphers, which 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 [12], IPSec [18], and GPG [40].

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. In order to ensure whether the whole packet is delivered, the length of the whole packet is encrypted in the first block. In reverse, if the number of delivered blocks is less than the number of whole packet, which is indicated by 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. https://nakedsecurity.sophos.com/2020/09/17/zerologon-hacking-windows-servers-with-a-bunch-of-zeros/

A4 Partitioning oracle attacks [36] - Sometimes, AEAD schemes are deployed by using passwords rather than random bit strings as symmetric keys. Since the passwords are often not sampled uniformly at random, an attacker might know a set of possible password candidates. Even worse, if an attacker has access to a partitioning oracle, which takes a ciphertext as input and outputs whether the password belongs to some known subset of passwords, then the symmetric password can be efficiently recovered, with a exponentially faster speed than brute-force.

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 [36].

A5 The Salamander attack [19] - 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, the Facebook abuse report mechanism turns out to be broken because a malicious sender could send a single encrypted attachment that would decrypt to both an abuse and an innocent message under two distinct keys.

A6 The Sframe attack [27] - 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 [42] 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 possessing the symmetric key can forge the unsigned portion of other group members' ciphertexts.

2.2. Formal AEAD syntax and requirements

Notations. We consider that all algorithms defined in this paper are parameterized implicitly by the security parameter. For a variable s and a set S, let $s \leftarrow\!\$\, S$ denote sampling s uniformly at random from S. For a variable x and a randomized algorithm X, let $x \leftarrow\!\$\, X$ denote the probabilistic execution of X followed by assigning the output to x. For a variable y and a deterministic algorithm Y, let $y \leftarrow Y$ denote the execution of Y followed by assigning the output to y. We use $\perp$ to denote an special error symbol that is not included in any set in this paper. We use _ to denote a variable that is irrelevant. Throughout this paper, for any (negligible) bound ϵ and any security notion sec, we say an AEAD scheme is ϵ-sec if the advantage of any attacker that breaks sec security of AEAD in polynomial time is bounded by ϵ.

There are several different AEAD definitions in the literature, such as randomized-, nonce-based, and counter-based variants. In this paper, we only focus on the nonce-based AEAD out of following considerations: On the one hand, the nonce-based AEAD has many popular instantiations underlying plenty of practical applications, for instance AES-GCM, ChaCha20-Poly1305, and OCB3 in TLS, WhatsApp, WPA2. The analysis of nonce-based AEAD can prove the security or reveal the vulnerability of not only AEAD primitives themselves but also numerous applications in the real life. On the other hand, note that the stateful counter and the encryption randomness can be considered as a nonce with overwhelming probability if the underlying concrete implementation is correct and the randomness quality is good. The security analysis of nonce-based AEAD is so general that also applies to the randomized- and counter-based AEADs.

We introduce here the main formal definitions for AEADs.

Definition 1 ([44]). *Let* Key, Nonce, Header, Message, Ciphertext *respectively denote the space of keys, nonces, headers (also called 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, 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. We reuse the one introduced in the original paper [44]. In addition, we consider by default in the definitions that the attacker is nonce-respecting, i.e. query the Enc oracle with every nonce at most once. This indicates that the nonces are supposed to be transmitted with out-of-band authentication in practice, in particular, for the privacy and the integrity. For each security property, we can then trivially obtain a variant without this assumption to capture nonce-misuse resistance.

Definition 2 (Privacy [44]). *We say an* $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *is* ϵ-IND$-CPA, *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,*

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

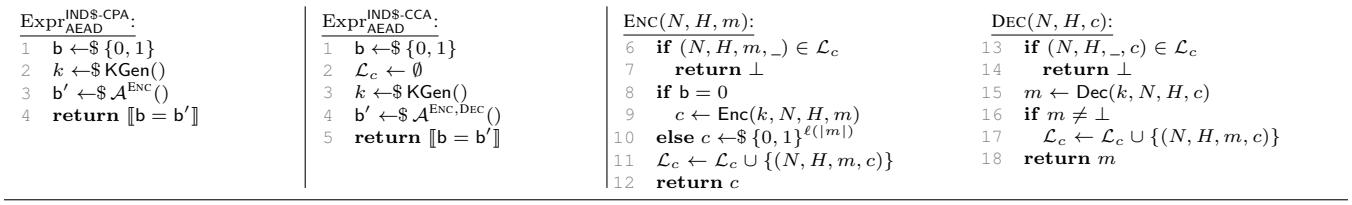


Figure 1: IND$-CPA and IND$-CCA security for an AEAD = (KGen, Enc, Dec) scheme from [44].

Definition 3 (Integrity [44]). *We say an* AEAD $=$ (KGen, Enc, Dec) *is* ϵ-CTI-CPA *secure, if the below defined advantage of any attacker* $\mathcal{A}$ *against* $\text{Expr}^{\text{CTI-CPA}}_{\text{AEAD}}$ *experiment in Fig. 2 is bounded by,*

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

Both for integrity and privacy, we can define two variants of the security definitions, CTI-CCA and IND$-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 CTI-CCA in Fig. 2.

In effect, the conjunction of CTI-CPA and IND$-CPA is enough to imply those variants. While there are theoretical separations at the AEAD level, that is, it is possible to construct an AEAD satisfying one but not the other, we don't know any of such practically deployed constructions. We summarize the well-known relations in Fig. 3, with the corresponding theorems in Appendix C.

2.3. Dimensions underlying AEAD definitions

Many variants of AEADs have been designed in the past twenty years following the seminal work from [44]. Each of those variant come with their own flavour of security definitions, about integrity, confidentiality, commitment, nonce-misuse resistance, robustness, ... - browsing through the different papers, it is easy to find dozens of distinct security definitions. We categorize main 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 (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, 6, 19, 24]. Roughly speaking, the collision resistance prevents the collisions between AEAD encryption with different inputs. The commitment ensures that each AEAD ciphertext must bind to a set of its encryption/decryption inputs - the decryption cannot output any other valid messages, if the decryption inputs do not agree with the encryption inputs on the binding portion. The sender- and receiver binding properties play a great role 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 C - motivated by **A5**.
F2 can we find collisions between encryptions or parts of it for different inputs? This question motivates the study of a novel property called robustness (Definition 11), [1, 36]. Briefly speaking, the robustness ensures that each AEAD ciphertext must bind to a set of its encryption inputs and the decryption with inputs, whose binding portion is unequal to the encryption one, must output $\perp$ - 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? [5] - 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? [7, 13]
F6 is the AEAD nonce-misuse resistant? [25] Must a nonce be used a single time, and must a nonce be fully random? - motivated by **A1**.

3. Real World (in)-Security of AEADs

After having reviewed many real-world attacks and the theoretical frameworks they spawned, we first identify in this section the core AEAD's weaknesses of concrete real life schemes that lead to the attacks: privacy and integrity, collision-resistance, and nonce-misuse. 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. Main attack pattern: collisions

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 in fact met by all AEADs. This is what is lost under decryption misuse.
- **collision-resistance** - it is difficult to come 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 $\mathsf{Enc}(\vec{i}_1) = \mathsf{Enc}(\vec{i}_2)$.

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

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

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

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

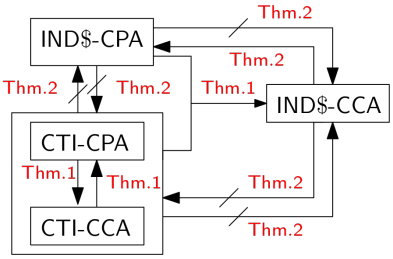


Figure 3: The relation between integrity and privacy properties for AEAD.

- **nonce-misuse** - 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 in [6] as a natural definition for full collision resistance and recall it below. Roughly speaking, the full collision resistance means that each AEAD ciphertext can only be computed by unique input.

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

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

$\underline{\text{Expr}^{\text{full-CR}}_{\text{AEAD}}}$:
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** $(k_1, N_1, H_1, m_1) = (k_2, N_2, H_2, m_2)$
5 **return** 0
6 $c_1 \leftarrow \text{Enc}(k_1, N_1, H_1, m_1)$
7 $c_2 \leftarrow \text{Enc}(k_2, N_2, H_2, m_2)$
8 **return** $[\![c_1 = c_2]\!]$

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

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, 6, 22, 24, 36]. We can list informally those notions as follows:

- *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 $l \in \{1, 3, 4\}$ [6] - 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 this paper, we rename CMT-l as *collision resistance* (X-CR) and CMTD-l as *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.
- *full robustness* FROB [22] and *even fuller robustness* eFROB [24] - is any adversary able to compute a ciphertext that decrypts correctly under two distinct

2. Here, we slightly abuse the notations 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 $k \in \mathsf{X} \Leftrightarrow (k) \subseteq \mathsf{X}$.

AEAD	Integrity and Privacy	Full Collision Resistance	Nonce Misuse Resistance
AES-GCM	✓ [28, 37]	✗[19]	✗- forgeability + xor of plaintexts
ChaCha20-Poly1305	✓ [43]	✗[2]	✗- xor of plaintexts
XSalsa20-Poly1305	✓*	✗[2]	✗- xor of plaintexts
OCB3	✓ [9, 34]	✗[2]	✗- forgeability + equality of blocks
EtM (unrelated keys)	✓[44]	✗[24]	✗- encryption dependent
EtM (related keys)	✓[44]	✓[24]	✗- encryption dependent
AES-CCM	✓ [23, 31]	✓*	✗- xor of plaintexts
AES-EAX	✓ [8, 39]	✓*	✗- xor of plaintexts
CAU-C4	✓[6]	✓[6]	✗- forgeability + xor of plaintexts
AES-GCM-SIV	✓ [25, 29]	✗[2]	✓
CAU-SIV-C4	✓[6]	✓[6]	✓

* = The security is conjectured there, but we could not find a proof in the literature.

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 additionally indicate the potential impact of reusing nonces if the AEAD scheme does not have this property.

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 adversary 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.
- *multi-key collision resistance* (MKCR) [36] - is any adversary 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) [24] - is any adversary 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). Note that [24] 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 present in Fig. 5 the relations between those notions, where some notions depending on a parameter X needed for the generality and being able to cover all notions. Recall that we have (k, N, H, m)-CR = full-CR in this figure. It is easy to observe 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.

We provide all the formal definitions, theorems, and proofs for this figure in Appendix C. While some of the relations were conjectured before ([6]), we are the first to provide the proofs, as well as provide the generalization of some notions in order to enable a comparison.

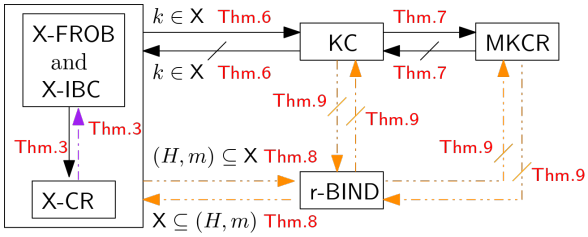


Figure 5: The relation between collision related properties for AEAD with key space Key. The black arrow $\rightarrow$ indicates the general implication. The purple dash-dotted arrow $\dashrightarrow$ indicates the implication for tidy AEAD. The orange dash-dot-dotted arrow $\dashrightarrow$ indicates the implication for ccAEAD[AEAD]. The X in the figure is a subset of (k, N, H, m), i.e., $\mathsf{X} \subseteq (k, N, H, m)$.

A lack of foundations. Even for the few AEADs for which a security proof exists for collisions resistance, they however rely on very strong assumptions. Indeed, most of those proofs rely on the PRF assumption. [6] stressed that one must be aware of the extension attack if the nonce length is not fixed. Similarly, we also find that the PRF assumption is not always met in practice for an attacker chosen key. For instance, if the length of the keys is flexible, HMAC is instantly not a PRF as for any long enough key, the key and the hash of the key lead to the same HMAC instantiation. We illustrate in more details in Appendix A how two existing collision resistance proofs are fragile.

In addition, collision resistance as well as the difficulty of actually coming up with meaningful collisions rely on problems that have not garnered the needed study from cryptographers. For instance, they rely on:

- is it feasible to find two distinct keys k_1, k_2 such that the AES encryption of a string of 126 bits zero collide, i.e., $AES_{k_1}(0^{126}) = AES_{k_2}(0^{126})$? [2]
- given a target bitstring c, is it feasible to find a key k and an input i such that c is the AES encryption of i under key k, i.e., $AES_k(i) = c$?

- more generally, does there exist weak keys such that it then offers strange behaviours?

We illustrate those questions for AES, but they of course lift to any blockcipher, and we hope that those questions will gather more interest from the community in the future. Answering them positively would break some current collision resistant designs and make many more sorts of collisions tractable. Answering them negatively would conversely strengthen the guarantees.

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.

1) r-BIND: [24] 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. [19] 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 [45], [19] claims that this attack also applies to any so-called *rate-1* AEAD, that is, "one blockcipher call per block of message" [19]. This potentially indicates the vulnerability of AES-GCM-SIV and ChaCha20-Poly1305 and any EtM constructions.
2) KC: [2] extends the known attack in [19] 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: [36] 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: [6] 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 [6].

4. Models for automated verification

Equipped with our newly acquired understanding of which concrete AEAD weaknesses lead to which class of potential protocol attacks, we now develop models for AEADs that encompass many of the essential weaknesses from Section 2.1. We thus derive generic models covering:

AEAD	nColl	KeysColl
AES-GCM	[19]	[2, 19, 24, 36]
AES-GCM-SIV	[19]	[2, 19, 24, 36]
ChaCha20-Poly1305	[19]	[2, 19, 24, 36]
EtM	[19]	[19, 24]

TABLE 2: Effective attacks against collision resistance of several AEADs in the literature. nColl describe collisions where, for given keys and a header, the adversary uses brute-force over the nonce to produce colliding ciphertexts. In KeysColl, the attacker brute-forces, given a nonce and header, over the keys.
We give concrete attacks for CTR, OFB, CBC, and CFB modes underlying EtM in Appendix B.

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

There exist modern protocols, like [42] or [19], which also 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 [42] for **A6**.
- *explicit commit* Com- one can extract from the ciphertext a value committing to the inputs of the encryption, needed to model protocols like [19] for **A5** and to cover **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 an encryption to the attacker. Not sending out the nonce by default but using a dedicated functionality allows to account for nonce hiding AEADs covering **F5** .

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 (Section 4.1). We then present symbolic models of the beforementioned 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 `enc` and `dec`, with the equation:

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

Remark that the randomness or nonce is not explicit in this classical modeling. And crucially, in the symbolic model, only the equations that are specified imply equalities. This results in the 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 [16, 55] and aid standardization processes to avoid design-level flaws [14, 17, 41].

In recent years, effort was put into improving the symbolic model with better and more fine-grained support for cryptographic primitives. [15] introduced a stronger version of symbolic Diffie-Hellman group models, while [30] gave more fine-grained models of digital signatures.

4.2. Symbolic AEAD models

We first explicitly model all the input parameters, making the enc and dec having four inputs, $\mathtt{enc}(k, n, h, m)$.

Collision models Coll. 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:

$$\mathtt{enc}(k_1, \mathtt{c}_n(k_1, k_2, h, m_1, m_2), h, m_1) = \mathtt{enc}(k_2, \mathtt{c}_n(k_1, k_2, h, m_1, m_2), h, m_2)$$

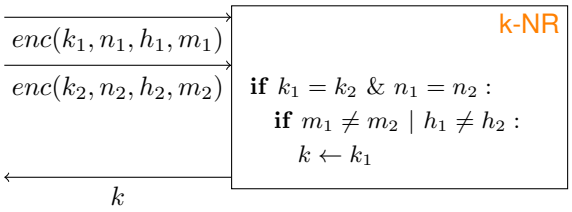
When using this model in one of the automated tools results to report an attack on the protocol, we can then investigate its feasibility in practice based on the concrete AEAD used and the message encodings by referring to Section 3. For this, refer to Table 2. It can be reasonably computed on many AEADs, and was shown to be practical by [19] for Facebook's message franking protocol. Another widespread collision capability is captured by adding two function symbols $\mathtt{c}^1_k$ and $\mathtt{c}^2_k$ with the collision model KeysColl:

$$\mathtt{enc}(\mathtt{c}^1_k(n, h, m_1, m_2), n, h, m_1) = \mathtt{enc}(\mathtt{c}^2_k(n, h, m_1, m_2), n, h, m_2)$$

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).

We can similarly model all possible sorts of collisions, by changing which part of the encryption input is fixed on both sides, and which part the attacker is brute-forcing over. While we are not aware of any other practical collision, we decided to model the strongest collision models that still not collapse to a single point. With FullKeyColl, Full-mColl, Full-nColl, and Full-adColl, we capture the capability of an attacker to find collision by just finding one distinct k, n, h, or m, respectively. These models may cover collisions that are impractical and have the purpose to check whether the analysed protocol relies on collision resistance of AEAD schemes. 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 kind 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 a verify and a 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 verify function verify, with the equations:

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

And we model the fact that the decryption is weak by making it so that decryption may succeed on messages forged by the attacker using the forge algorithm:

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

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, often simply

by an XOR of a value to the ciphertext. Lifting this limitation 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 takes as input a ciphertext $\mathtt{enc}(k, n, h, m)$. We can then model collisions over the tags, by adding for each of the previous collision equations a variant over the tag, with, e.g., nTag being:

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

Explicit commitment Com. We model compactly committing AEADs by adding a `get-commit` function symbol similar to the `get_tag`, but over which there is no equation for collisions. Recall that we assume the commitment schemes to be collision resistant. Hence no new collision are introduced but only propagated by colliding AEADs.

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:

$$\mathtt{get\_nonce}(\mathtt{enc}(k, n, h, m)) = n$$

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

4.3. Automated analysis methodology

As described before, we now have a set of models to capture multiple weaknesses of AEADs. To perform the analysis of a protocol, the following steps should be followed with the symbolic tool of choice:

- Verify the protocol in all possible threat models (malicious participants, AEADs weaknesses)
- 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);
- 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.
- 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.

The TAMARIN prover. Our methodology is generalized enough to not be bound to a specific tool. The tool of choice needs to allow usage of custom equational theories (ET) and explicit means to express attacker knowledge. These are criteria fulfilled by various state-of-the-art symbolic tools like [10, 20, 33]. We chose the TAMARIN prover [38] to demonstrate our methodology, as TAMARIN offers a straightforward way to add custom ETs and oracle-like processes needed for NR.

Analysis setup. We split the models from Section 4 into two general classes:

1) collisions, nonce-reuse, and nonce-reveal (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 can then have 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 the set of our case studies, we want to explore their security guarantees against all of our AEAD models. 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). By re-using strict implications of some of our models, e.g. FullKeyColl makes the adversary strictly more powerful than KeyColl, 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 for our case studies can be found in Table 3. Another factor in our analysis is the run-time of the protocol models. As the problem of automatically analysing protocol models is in general undecidable, running TAMARIN could lead to non-termination. For this, we introduce timeouts into our experiments. For each TAMARIN call with a fixed combination of AEAD models, we either find a proof, a potential attack trace or we have a timeout. Note that in practice models in symbolic tools are often tailored towards the case study to make termination more likely. Using this 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 where manually tailored towards termination. We selected the value of the short timeout depending on run-time of the protocol model with the classic AEAD model in use.

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.

5. Case-studies

We demonstrate our symbolic models for automated verification on a set of 8 protocols [21, 32, 42, 46, 47, 50, 54, 56]. We modeled all 8 in TAMARIN, where one of those models is a variant of an already existing case study [56]. The other protocols were modeled by us and more details can be found in Section 5.2 - 5.5. We then automatically analyse these case studies using the new modeling of AEADs introduced in the previous section. To evaluate the efficiency of our new modeling, we execute the protocol models using the methodology introduced in Section 4.3. The results can be found in Table 3.

After providing an overview of our setup and presenting the overall results in Section 5.1, we discuss the individual protocols and their detailed attack scenarios. We divide our case studies into four categories depending on the analysed security property:

- Key Secrecy - rediscovering the attack on *YubiHSM*
- Authentication - rediscovering the attack on *SFrame*
- Accountability - rediscovering the attacks on the accountability of the Facebook Message Franking mechanism and finding a new attack on the server accountability of the WebPush standard.
- Content Agreement- analysis of multiple group messaging protocols and content delivery protocols, namely WhatsApp Groups, GPG, Scuttlebutt and SaltPack.

We summarize our attacks, both old and new, 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 in Appendix D, illustrating a discrepancy between them.

5.1. Experimental results

We tested our methodology on a computing cluster with Intel® Xeon® Gold 6244 CPUs and 1TB RAM. For each TAMARIN call we used 4 threads. We set the timeout per TAMARIN call to 60 seconds.

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. The overview of the results can be found in Table 3. We show an excerpt of the detailed results in Table 4, while all results to the case studies can be found in GitHub [49] and are reproducible.

5.2. Key Secrecy

YubiHSM. The YubiHSM [56] 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, [35] 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 [48] was found to have the same kind of attack, demonstrating that nonce misuse is still an worth paying attention towards.

When instantiating the original TAMARIN model by [35] with our AEAD library, we efficiently rediscover the attack using k-NR.

5.3. Authentication

SFrame. SFrame is a communication protocol developed by CoSMo Software and Google [42] with the goal to be used for online audio and video meeting protocols. It uses

Protocol	AEAD Scheme	Model	Attacked Property	Time (s)	Discovered	Attack	Notes
YubiHSM [56]	AES-CCM	NR	Key Secrecy	2	[35]	Feasible	Fixed
SFrame [42]	AES-GCM & EtM CTR	Tag	Authentication	<1	[27]	Feasible	Fixed
GPG SED [32]	PGP-CFB	Coll	Content Agreement	<1	This work	Feasible	Deprecated
Scuttlebutt [47]	XSalsa20-Poly1305	Coll	Content Agreement	3	This work	Feasible *	Reported
WhatsApp [54]	EtM CBC	Coll	Content Agreement	3	This work	Feasible ‡	Reported
WebPush [50]	AES-GCM	Coll	Server Deniability	8	This work	Feasible	Reported
FB Message Franking [21]	AES-GCM	Coll	Content Agreement	8	[19]	Feasible	Fixed
FB Message Franking [21]	AES-GCM	Coll	Framability	3	[19, 24]	Theoretical	Fixed
Saltpack [46]	Xsalsa20-Poly1305	Coll	Content Agreement	8	This work	Theoretical	Infeasible
GPG SEIPDv2 [32]	AES-OCB	Coll	Content Agreement	<1	This work	Theoretical	Infeasible

* = Feasibility of this attack depends on the collision resistance of XSalsa20-Poly1305 (not in Table 2.) See discussion in Section 5.5.
‡ = The attack has been reported to WhatsApp and feasibility heavily relies on implementation details which are not open source

TABLE 3: Summary: Case-studies
We summarize here our set of case-studies, illustrating the generality of our models by both rediscovering previous attacks as well as uncovering new ones. For each attack, we give the threat model, the use AEAD scheme, the violated property, as well as the time it took TAMARIN to find it. We also give some additional notes on the status of the attack.

Protocol	Secure	Threat Model
GPG SED	✓	Full-mColl, Full-nColl, Full-adColl, k-NR, m-NR, Leak
GPG SED	✗	KeyColl
GPG SEIPDv2	✓	FullKeyColl, Full-nColl, Full-adColl k-NR, m-NR, Leak
GPG SEIPDv2	✗	Full-mColl
Scuttlebutt	✓	Full-adColl, k-NR, m-NR, Leak
Scuttlebutt	✗	KeysColl
Scuttlebutt	✗	Full-mColl
Scuttlebutt	✗	nColl

TABLE 4: Example - exhaustive content agreement analysis
An excerpt of the results returned by our automated methodology. For each protocol analysed we get a the strongest combination of threat models under which we cannot find an attack (✓). The weakest combinations of AEAD threat models under which a potential attack is found is marked with (✗).

end-to-end encryption and is made to support groups of multiple users.

[27] 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 adversary with the power to join groups and the power to act as a group participant. Using our extended AEAD models, which allows to add 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 requiring 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.

5.4. Accountability

Facebooks Message Franking. In the setting of End-to-End encryption, reporting abusive behaviour of users seems to be hardly achievable without weakening the security guarantees. In 2016, Facebook introduced *message franking* [21] to allow reporting offensive message attachments. The idea is for a recipient of a malicious message attachment to be able to use a cryptographically sound way to prove that it was send by a specific sender.

[19] found an attack against Facebooks message franking mechanism in 2018. While the practical attack they demonstrated was more involved finding a collisions on the used AEADs 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 appeared to directly patch it. That attack demonstrates the practicality and the impact of collision attacks on real world schemes.

To show that such attacks can be found by our analysis already on the design level, we modeled the Facebook message franking report mechanism in TAMARIN. In the initial setting, with the adversary being a malicious sender, we could automatically find the reported attack in a few seconds using the KeysColl model.

In addition to carrying out the analysis of the property violated by the initial Salamander attack – can a malicious sender avoid detection? – we also studied the converse property – can a malicious receiver create a fake report? This was first reported as a concern by [24].

We thus additionally model a malicious receiver that tries to report an honest user. In this threat model, we therefore look at frameability properties. These kind of properties can be as severe in practice, e.g., by falsely accusing another person to have send illegal material. After testing our AEAD models against it, we could find a theoretical attack. However, this attack would require to find 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 impractical.

WebPush. The WebPush protocol is a protocol that provides a way 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. It has been standardized at the IETF [50], and, for instance, Apple is planning to integrate it inside its ecosystem.

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 report a message sent by the server to a third party? In our threat model, we thus consider a malicious server controlled by the attacker, and we verify if it is possible to upload one notification that could be interpreted in two different ways.

Our analysis reports an attack using Full-mColl, where 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 the attack scenario first seemed impractical. After manual inspection of the attack 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 the Salamender attack. Concretely, an attacker can brute-force over the salt used to produce a nonce/key pair to encrypt the message in order 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 of course depends on whether the non-deniability is critical given the use case.

5.5. Content Agreement

We focus here on analysing 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 analyse 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.

WhatsApp groups. We model the design of the WhatsApp group messaging. The code of WhatsApp is not available, we thus made our model based on the available information provided in its whitepaper [54, p. 10]. While it relies on the Signal 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*, 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 a 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. 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, it is likely given the variety of possible message contents (notification, GIFs, media, react, ...) that the encoding would be loose enough to carry out the attack.

GPG. GPG is the golden standard for file and mail encryption and signing. We review its ongoing cryptographic update [32]. 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 encryptions 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 B). 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 contraints 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.

Scuttlebutt. Scuttlebutt is a protocol that provides an authenticated append-only feed. The private box feature [47] allows to publish 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 [46].

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 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 once again not possible in practice.

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 and their differences, there are subtle differences that we currently do not capture yet. This includes, for example, properties beyond collisions and nonce-reuse. Similarly, the APIs of real-world AEADs may not follow the abstract APIs from the literature, such as the separation between decryption and verification, and similarly incremental variants of each.

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 secure. 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. Disclosure

The main three attacks we found are attacks on the protocol design, but not necessarily on the implementation level, since they depend on low-level encoding choices. Additionally, since the violated properties are not the core objectives of the protocols, but rather desirable features, we assessed that there is no immediate significant threat to users. Nevertheless, we contacted the developers of the three affected protocols and explained our design-level attacks, such that they can assess their implementation-level feasibility.

8. Conclusions

We have 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. Our method is not only effective in finding previously reported attacks, but also identified new protocol weaknesses, meaning that similar issues could have been found earlier by using our approach.

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: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016.

[4] Martin R Albrecht, Kenneth G Paterson, and Gaven J Watson. "Plaintext recovery attacks against SSH". In: *30th IEEE Symposium on Security and Privacy*. 2009.

[5] 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.

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

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

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

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

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

[11] 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.

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

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

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

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

[16] 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.

[17] 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.

[18] Jean Paul Degabriele and Kenneth G Paterson. "On the (in) security of IPsec in MAC-then-encrypt configurations". In: *Proceedings of the 17th ACM conference on Computer and communications security*. 2010.

[19] 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.

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

[21] *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.

[22] 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.

[23] 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.

[24] 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.

[25] Shay Gueron and Yehuda Lindell. "GCM-SIV: full nonce misuse-resistant authenticated encryption at under one cycle per byte". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 2015.

[26] Changhua He, Mukund Sundararajan, Anupam Datta, Ante Derek, and John C. Mitchell. "A Modular Correctness Proof of IEEE 802.11i and TLS". In: *Proceedings of the 12th ACM Conference on Computer and Communications Security*. 2005.

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

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

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

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

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

[32] 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/.

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

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

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

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

[37] 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.

[38] 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.

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

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

[41] 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.

[42] 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.

[43] 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.

[44] Phillip Rogaway. "Authenticated-Encryption with Associated-Data". In: *Proceedings of the 9th ACM Conference on Computer and Communications Security*. CCS '02. 2002.

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

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

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

[48] 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.

[49] *Tamarin models and analysis scripts to reproduce the results in this paper*. https://github.com/AEADProtocolSecurity/AEADProtocolSecurity. 2022.

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

[51] Mathy Vanhoef and Frank Piessens. "Key reinstallation attacks: Forcing nonce reuse in WPA2". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017.

[52] Mathy Vanhoef and Frank Piessens. "Release the Kraken: New KRACKs in the 802.11 Standard". In: *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 2018.

[53] 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.

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

[55] 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 (SP)*. 2022.

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

Appendix A.
Fragile security proofs

Example 1 - breaking a proposed instantiation. Consider the proof of Theorem 4 in [24]. It proves that a Mac-Then-Encrypt construction that they detail in Figure 10 satisfies the r-BIND, notably under the assumption that the attacks have access to a keyed random oracle function. They in addition propose to instantiate this keyed random oracle with HMAC-SHA256.

While it is accepted that HMAC is a PRF under honestly generated keys, such a claim in the malicious key setting is dangerous, and actually false. The $HMAC(k, m)$ constructions is defined such that if the length of the key k is bigger than some constant const, it is first hashed. Formally, we have

$$HMAC(k, m) = \begin{cases} HMAC'(k, m) \text{ if } k \leq \text{const} \\ HMAC'(\mathsf{H}(k), m) \text{ otherwise} \end{cases}$$

Remark that we trivially have that $HMAC(k, m) = HMAC(\mathsf{H}(k), m)$ whenever $k >$ const. This implies that for attacker-chosen keys, $HMAC$ is trivially not a keyed random oracle. This also leads to a trivial attack where using the two authentication keys k and $\mathsf{H}(k)$ we obtain an attack over the receiver binding property for the scheme of Figure 10.

Of course, such attacks would easily be mitigated in practice, for instance by forcing the key length to be fixed. But it illustrate how the keyed random oracle assumption can be dangerous.

Example 2 - no resilience to minimal weakness of the hash function. We now consider the encryptment scheme proposed in Figure 11 of [19]. It notably relies on a keyed hash function f and its iteration f^+, and we extract two following lines relevant to our discussions from the Figure 11:

$$V_0 \leftarrow f(IV, K_{EC})$$
$$V_h \leftarrow f^+(V_0, (K_{EC} \oplus H_1) \| \dots \| (K_{EC} \oplus H_h))$$

Here, IV is a public constant, K_{EC} is the encryption key, and $H_1, \dots, H_h$ are paddings of the headers. The security is proven in Theorem 3 under the collision resistance assumption over f^+. As discussed before, this collision resistance assumption is never met, and while it can be reasonable in some cases, let us show how fragile is the corresponding proof.

As IV is a fixed public constant, one could spend a very long computation time to come up with two collisions keys K_1 and K_2 such that $f(IV, K_1) = f(IV, K_2)$. Then, this single collision that can be seen as precomputed leads to an infinite number of violations of the receiver binding, as collisions will be obtained with $\mathsf{Enc}(K_1, (H_1, \dots, H_h), \dots) = \mathsf{Enc}(K_2, (H_1 \oplus K_2 \oplus K_1, \dots, H_h \oplus K_2 \oplus K_1), \dots)$.

Appendix B.
Collision resistance of existing schemes

CTR mode [19]. This is one of the simplest block-cipher mode to create colliding ciphertexts of the previous form. Indeed, to encrypt a plaintext $\mathsf{plaintext} := \mathsf{plaintext}_1 \| \cdots \| \mathsf{plaintext}_n$ with initial vector IV and key k, we have that $c_i = \text{Block}_{\mathsf{CTR}}(k, i \| IV, \mathsf{plaintext}_i)$ for $i = 1, ..., n$, where $\text{Block}_{\mathsf{CTR}}(k, N, m) := m \oplus AES_k(N)$. 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 $c_1 = \text{Block}_{\mathsf{X}}(k, IV, \mathsf{plaintext}_1)$, and then $c_i = \text{Block}_{\mathsf{X}}(k, c_{i-1}, \mathsf{plaintext}_i)$ for $\mathsf{X} \in \{\mathsf{CBC}, \mathsf{CFB}\}$, where for CBC mode $\text{Block}_{\mathsf{CBC}}(k, N, m) := AES_k(m \oplus N)$, for CFB mode $\text{Block}_{\mathsf{CFB}}(k, N, m) := AES_k(N) \oplus m$.

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 $\mathsf{plaintext} = \mathsf{plaintext}_1 \| \cdots \| \mathsf{plaintext}_n \| \mathsf{plaintext}_{n+1} \| \cdots \| \mathsf{plaintext}_{2n}$ for some n and $\mathsf{X} \in \{\mathsf{CBC}, \mathsf{CFB}\}$:

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

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

B.1. 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 [40], and we can use it here to efficiently find collisions, as it can be seen as an extra junk data field always available.

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

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 bruteforce. 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 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 intersting 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 standardadization process for a crypto refresh.

Appendix C. Formal definitions and proofs

C.1. Additional Preliminaries

We recall a primitive called committing PRF and its simplified binding security, which was first defined in [6].

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,*

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

$\mathrm{Expr}^{\mathsf{bind}}_{\mathsf{cPRF}}$:
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.

C.2. Additional Definitions

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

- 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.,* $k_f \leftarrow \mathsf{openCommit}(k, N, H, c)$
- 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

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

C.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 [5]. Below, we recall the results.

Theorem 1 ([5]). *Let* $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *be an authenticated encryption with associated data. It holds that:*

1) *[5, Proposition 1]: If* AEAD *is* ϵ-CTI-CPA *secure, then* AEAD *is also* ϵ-CTI-CCA *secure, and vice versa.*
2) *[5, 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) *[5]: 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 respectively prove the above six statements.

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. It is easy to know that $\mathcal{B}$ wins if and only if $\mathcal{A}$ wins, which concludes the proof.
2) Statement 2: See [5, Lemma 2].
3) Statement 3: We prove this statement by a 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 know 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 examples. 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 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 examples. Let $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ denote an ϵ-IND\$-CCA secure authenticated encryption with associated data scheme with message space Message, nonce space Nonce, and header space Header, and ciphertext space Ciphertext. In particular, let $\tilde{N} \in$ Nonce and $\tilde{H} \in$ 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 $r \in$ Message for any $k \in$ Key, $N = \tilde{N}$, and $H \in \tilde{H}$.

6) Statement 6: This statement is implied by Statement 1 and 4.

□

C.4. Collision Resistance

For any $\mathsf{X} \subseteq (k, N, H, m)$, we define a class of projection functions $f_{\mathsf{X}} : \mathsf{Key} \times \mathsf{Nonce} \times \mathsf{Header} \times \mathsf{Message} \to \mathsf{dom}(\mathsf{X})$, 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,*

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

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

$\big((k_1, N_1, H_1, m_1), (k_2, N_2, H_2, m_2)\big) \leftarrow\$\, \mathcal{A}()$
if $\perp \in \{k_1, N_1, H_1, m_1, k_2, N_2, H_2, m_2\}$
  return 0
if $f_{\mathsf{X}}(k_1, N_1, H_1, m_1) = f_{\mathsf{X}}(k_2, N_2, H_2, m_2)$
  return 0
$c_1 \leftarrow \mathsf{Enc}(k_1, N_1, H_1, m_1)$
$c_2 \leftarrow \mathsf{Enc}(k_2, N_2, H_2, m_2)$
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.

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

$\big(c, (k_1, N_1, H_1, m_1), (k_2, N_2, H_2, m_2)\big) \leftarrow\$\, \mathcal{A}()$
if $\perp \in \{k_1, N_1, H_1, m_1, k_2, N_2, H_2, m_2\}$
  return 0
if $f_{\mathsf{X}}(k_1, N_1, H_1, m_1) = f_{\mathsf{X}}(k_2, N_2, H_2, m_2)$
  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 [6]. 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 [6] we have that X-IBC implies X-CR. Moreover, these two notions are equivalent if the AEAD is tidy.

Theorem 3 ([6, Appendix A])**.** *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-X-IBC *secure, then* AEAD *is also* ϵ-X-CR *secure for* $\mathsf{X} \in \{k, (k, N, H), (k, N, H, m)\}$.
2) *If* AEAD *is* ϵ-X-CR *secure and tidy, then* AEAD *is also* ϵ-X-IBC *secure for* $\mathsf{X} \in \{k, (k, N, H), (k, N, H, m)\}$.
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* $\mathrm{Expr}^{\mathsf{X\text{-}FROB}}_{\mathsf{AEAD}}$ *experiment in Fig. 9 is bounded by,*

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

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

$\big(c, (k_1, N_1, H_1), (k_2, N_2, H_2)\big) \leftarrow\$\, \mathcal{A}()$
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 [22]. 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 [24]. For X-CR, X-IBC, and X-FROB, we can have following trivial conclusion that generalizes Theorem 3.

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) [36] 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* $\kappa = 2$.

$\mathrm{Expr}^{\mathsf{KC}}_{\mathsf{AEAD},q}$:
1 $\mathcal{L} \leftarrow \emptyset, n \leftarrow 0$
2 $() \leftarrow \mathcal{A}^{\mathrm{Enc},\mathrm{Dec}}()$
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

$\mathrm{Enc}(k, N, H, m)$:
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

$\mathrm{Dec}(k, N, H, c)$:
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

$\mathrm{Expr}^{\mathsf{MKCR}}_{\mathsf{AEAD},\kappa}$:
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, [6] 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 respectively prove the above statements.

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 $\mathrm{Expr}^{\mathsf{KC}}_{\mathsf{AEAD},q}$ 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 given 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 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 the KC security implies the 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 respectively prove the above statements.

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 given 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 $\mathsf{cPRF} : \mathsf{Key}' \times \mathsf{Nonce} \to \mathsf{Key}' \times \mathsf{Tag}$ 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 picks 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 [24]. 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* $\text{Expr}^{\text{s-BIND}}_{\text{AEAD}}$ *experiment in Fig. 11 is bounded by,*

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

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* $\text{Expr}^{\text{r-BIND}}_{\text{AEAD}}$ *experiment in Fig. 12 is bounded by,*

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

$\underline{\text{Expr}^{\text{s-BIND}}_{\text{ccAEAD}}}$:
1 $(k, N, H, c) \leftarrow\$\, \mathcal{A}()$
2 $m \leftarrow \text{Dec}(k, N, H, c)$
3 $k_f \leftarrow \text{openCommit}(k, N, H, c)$
4 **if** $m = \perp$
5 **return** 0
6 **if** $\text{vrfyCommit}(k_f, N, H, m, c)$
7 **return** 0
8 **return** 1

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

$\underline{\text{Expr}^{\text{r-BIND}}_{\text{ccAEAD}}}$:
1 $\left(c, (k_{f_1}, N_1, H_1, m_1), (k_{f_2}, N_2, H_2, m_2)\right) \leftarrow\$\, \mathcal{A}()$
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** $(H_1, m_1) = (H_2, m_2)$
5 **return** 0
6 **if** $\text{vrfyCommit}(k_{f_1}, N_1, H_1, m_1, c)$ **and** $\text{vrfyCommit}(k_{f_2}, N_2, H_2, m_2, c)$
7 **return** 1
8 **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 [24], such that

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

It is interesting to observe that any ccAEAD[AEAD] is s-BIND secure. Moreover, it is stated in [24] 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 the 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 respectively prove the above statements.

1) Statement 1: For any (k, N, H, c) output by $\mathcal{A}$ and $m \leftarrow \text{Dec}(k, N, H, c)$, $\mathcal{A}$ can win only when
   a) $m \neq \perp$, and
   b) $m \neq \text{Dec}(k, N, H, c)$.
   The second condition contracts to the fact that $m \leftarrow \text{Dec}(k, N, H, c)$. 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 $\left(c, (k_1, N_1, H_1, m_1), (k_2, N_2, H_2, m_2)\right)$, $\mathcal{A}$ 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) $m_1 = \text{Dec}(k_1, N_1, H_1, c)$
   d) $m_2 = \text{Dec}(k_2, N_2, H_2, c)$
   This in particular indicates that
   a) $\perp \notin \{k_1, N_1, H_1, k_2, N_2, H_2\}$,
   b) $m_1 = \text{Dec}(k_1, N_1, H_1, c)$
   c) $m_2 = \text{Dec}(k_2, N_2, H_2, c)$
   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 $\left(c, (k_1, N_1, H_1), (k_2, N_2, H_2)\right)$ 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 $\left(c, (k_1, N_1, H_1), (k_2, N_2, H_2)\right)$, $\mathcal{A}$ wins if
   a) $\perp \notin \{k_1, N_1, H_1, k_2, N_2, H_2\}$,
   b) for $m_1 := \text{Dec}(k_1, N_1, H_1, c)$ and $m_2 := \text{Dec}(k_2, N_2, H_2, c)$ 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]. In reverse, the r-BIND security of ccAEAD[AEAD] does not imply the KC or MKCR security.

Theorem 9. *Let* $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *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) *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 respectively prove the above statements.

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 picks 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 easy to know 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 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 $(\mathsf{Key}^\star, N^\star, H^\star, c^\star)$. It is easy to know 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.

□

Appendix D. Content Agreement in Messaging

Currently, we witness that there is a discrepancy between existing guarantees, as can be seen from Table 5.

Protocol	Content Agreement		Notes
	with Collision Resistance	without	
Signal	✗	✗	pairwise channels, hence to content agreement
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

TABLE 5: Content Agreement summary

A summary of our finding for Content Agreement: 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.