TEK4500: Authenticated encryption

Håkon Jacobsen

Fall 2023

1 Introduction

So far in the course we've been treating the goals of confidentiality and integrity[1] separately. However, in practice we usually want to have both confidentiality and integrity at the same time. This leads to the notion of *authenticated encryption* which combines both properties into a single primitive.

2 Syntax

As usual we begin with the syntax. An authenticated encryption scheme is syntactically almost identical to a regular encryption scheme, with the only difference being that an authenticated encryption scheme allows for the possibility of failure during decryption. That is, the decryption algorithm can potentially output $\perp$ to indicate that something went wrong during the decryption process. More formally, the syntax of an authenticated encryption is defined as follows.

Definition 1. An *authenticated encryption scheme* is a tuple $\Sigma = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ of algorithms, where:

- KeyGen is a probabilistic key generation algorithm that outputs a key in a *key space* $\mathcal{K}$.
- $\mathsf{Enc} : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ is a probabilistic encryption algorithm which takes as input a key in $\mathcal{K}$ and a message from a *message space* $\mathcal{M}$ and outputs a ciphertext in a *ciphertext space* $\mathcal{C}$.
- $\mathsf{Dec} : \mathcal{K} \times \mathcal{C} \to \mathcal{M} \cup \{\perp\}$ is a deterministic decryption algorithm which takes as input a key in $\mathcal{K}$ and a ciphertext in $\mathcal{C}$ and outputs either a message in $\mathcal{M}$ or a distinct *error symbol* $\perp$.

Σ is required to satisfy the following *correctness requirement*: for all $K \in \mathcal{K}$ and all $M \in \mathcal{M}$ we have

$$\mathsf{Dec}(K, \mathsf{Enc}(K, M)) = M. \tag{1}$$

Comments. Typically we have $\mathcal{K} = \{0,1\}^{128}$ or $\mathcal{K} = \{0,1\}^{256}$, i.e., the key space is the set of all 128 bit strings or the set of all 256 bit strings. Moreover, the key generation algorithm is usually defined to simply draw the key uniformly at random from the key space $\mathcal{K}$. Thus, we'll mostly ignore the key generation algorithm from now on. The message and ciphertext spaces will usually be $\{0,1\}^*$, i.e., plaintexts and ciphertexts are bit strings of arbitrary length.

[1] The terms authentication and integrity will be used interchangeably in this note.

3 Security definition

To define the security of an authenticated encryption scheme, perhaps the most natural approach is simply to model confidentiality and integrity as two separate security requirements and then demand that the scheme satisfies both of these definitions separately. We give this description in Section 3.1.

Alternatively, it's also possible to define authenticated encryption in a single security experiment that covers both confidentiality and integrity at the same time. We give this description Section 3.2.

So given that there are two different ways to define authenticated encryption, which one should we use? Is one better than the other? May one notion be stronger than the other? Fortunately, it turns out they are both equivalent! We prove this in Section 3.3. Thus, it doesn't matter which definition we take as our "real" definition of authenticated encryption.

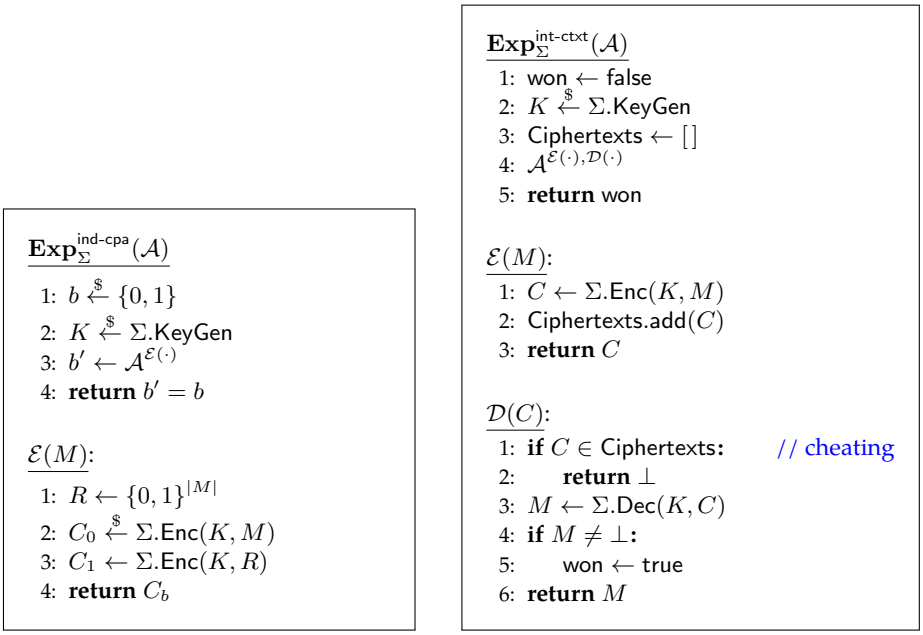
3.1 Separate confidentiality and integrity definitions

Here we define authenticated encryption as the collection of two separate security definitions, one for confidentiality and one for integrity. For an authenticated encryption scheme to be considered secure it needs to satisfy both of these definitions separately. Please note that while we define two different *security notions*, there is still only one *scheme*. That is, there is a single scheme Σ and it simultaneously needs to satisfy both the confidentiality definition and the integrity definition in order to be considered a secure authenticated encryption scheme. Contrast this with the *distinct* schemes we have looked at so for obtaining confidentiality and integrity, namely encryption schemes and MACs. These are nothing alike neither syntactically nor operationally, while for authenticated encryption there is only a single primitive with two associated security definitions. We begin with confidentiality.

Confidentiality. This one's easy because we simply re-use the IND-CPA security definition for (non-authenticated) encryption schemes. The IND-CPA security experiment is repeated in Fig. 1a. As usual we define the *IND-CPA advantage* of an adversary $\mathcal{A}$ as:

$$\mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\Sigma}(\mathcal{A}) = \left| 2 \cdot \Pr[\mathbf{Exp}^{\mathsf{ind\text{-}cpa}}_{\Sigma}(\mathcal{A}) \Rightarrow \mathsf{true}] - 1 \right| . \tag{2}$$

Integrity. For the integrity definition we mimic the UF-CMA notion that we used for MACs. Recall that the essence of the UF-CMA definition was that it should be hard for an adversary to come up with a message and a tag that will be accepted by the verification algorithm of the MAC. In other words, it should be hard to *forge* valid tags. Analogously, for authenticated encryption we demand that it should be hard for the adversary to forge valid *ciphertexts*. That is, it should be hard for the adversary to come up with a ciphertext C^* such that it decrypts to something other than $\perp$. We call this notion *integrity of ciphertexts* (INT-CTXT) and the formal security experiment is shown in Fig. 1b. Having INT-CTXT security means that only the honest users (who know the secret key) will be able to create ciphertexts that decrypt correctly. Even a single bit flip done by the adversary will be detected.



(**a**) IND-CPA (confidentiality) (**b**) INT-CTXT (integrity)

Figure 1: Separate confidentiality and integrity security experiments for an authenticated encryption scheme Σ.

For an adversary $\mathcal{A}$ we define its INT-CTXT *advantage* as:

$$\mathbf{Adv}^{\text{int-ctxt}}_{\Sigma}(\mathcal{A}) = \Pr[\mathbf{Exp}^{\text{int-ctxt}}_{\Sigma}(\mathcal{A}) \Rightarrow \mathsf{true}] \tag{3}$$

Notice that, up to a change of notation, the INT-CTXT experiment is more or less identical to the UF-CMA experiment. That is, the $\mathcal{E}$ oracle in the INT-CTXT corresponds to the Tag oracle in the UF-CMA experiment, while the $\mathcal{D}$ oracle corresponds to the Vrfy oracle. Specifically, the purpose of the $\mathcal{E}$ oracle in the INT-CTXT game is to allow the adversary to see validly created ciphertexts for messages of its own choice. Allowing this is necessary since in real life the adversary is very likely to at least be able to see the ciphertexts sent by the honest users. The purpose of the $\mathcal{D}$ oracle is for the adversary to test its attempts at ciphertext forgeries (just like the Vrfy oracle in the UF-CMA game). Note that we need to do a bit of bookkeeping in order to avoid trivial wins for the adversary. Specifically, the adversary shouldn't be rewarded if it first asks the $\mathcal{E}$ oracle for the encryption of a message and then submits the resulting ciphertext to the $\mathcal{D}$ oracle. By correctness of the authenticated encryption scheme (Definition 1) this will of course yield a valid decryption. Thus we record all the valid ciphertexts the adversary have received from the $\mathcal{E}$ oracle in the list Ciphertexts. In order to win the adversary needs to come up with a ciphertext *not* in Ciphertexts.

Definition 2 (AE security (informal))**.** An authenticated encryption scheme Σ is *AE-secure* if both $\mathbf{Adv}^{\text{ind-cpa}}_{\Sigma}(\mathcal{A})$ and $\mathbf{Adv}^{\text{int-ctxt}}_{\Sigma}(\mathcal{A})$ are "small" for all "resource bounded" $\mathcal{A}$.

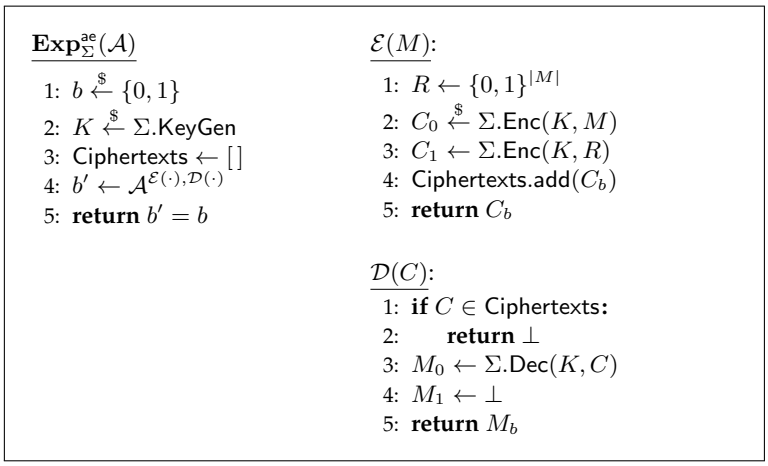


Figure 2: Authenticated encryption (AE) security experiment (all-in-one variant).

3.2 All-in-one definition

Here's a different way of defining authenticated encryption which combines both confidentiality and integrity into a single security experiment. This variant, defined formally in Fig. 2, considers two different "worlds": if $b = 0$ then the adversary is given access to a real encryption oracle and a real decryption oracle, while if $b = 1$ the adversary is given access to an encryption oracle only returns encryptions of random messages and a decryption oracle that *always* returns $\perp$. The adversary is now asked: can it tell which world it's in? If it cannot then we say that the scheme is AE secure. More formally, we define an adversary $\mathcal{A}$'s *AE advantage* as:

$$\mathbf{Adv}^{\mathsf{ae}}_{\Sigma}(\mathcal{A}) = \left|2 \cdot \Pr[\mathbf{Exp}^{\mathsf{ae}}_{\Sigma}(\mathcal{A}) \Rightarrow \mathsf{true}] - 1\right|, \tag{4}$$

and (informally) say that a scheme is AE secure if *all* "resource bounded" adversaries have "tiny" AE advantages. As usual, we leave it unspecified exactly what "resources bounded" and "tiny" are.

Definition 3 (AE security (informal))**.** An authenticated encryption scheme Σ is *AE-secure* if $\mathbf{Adv}^{\mathsf{ae}}_{\Sigma}(\mathcal{A})$ is "small" for all "resource bounded" $\mathcal{A}$.

Why does the all-in-one AE experiment capture both confidentiality and integrity? For confidentiality notice that if you ignore the $\mathcal{D}$ oracle in Fig. 2, then you are basically left with the IND-CPA experiment. For integrity the connection is a bit more subtle. The idea is that if the adversary can forge ciphertexts, then this will allow it to trivially distinguish the two worlds. To see how this works, suppose an adversary is able to forge a valid ciphertext C^*. If the adversary submits C^* to the $\mathcal{D}$ oracle while being in the real world ($b = 0$), then it'll get back a message $M \neq \perp$. But if the adversary submits C^* to the $\mathcal{D}$ oracle while being in the ideal world ($b = 1$), then by definition it will always get back $\perp$. Thus, being able to forge valid ciphertexts gives the adversary a way to distinguish between the real world and the fake world.

Please note that getting back $\perp$ from the $\mathcal{D}$ oracle does not necessarily imply that the adversary is in the fake world (unlike if it gets back something other than $\perp$, in which case it's guaranteed to be in the real world). Indeed, the forgery attempt could simply have been wrong. But notice that this requires the encryption scheme to be able to detect forgeries! In other words, if an adversary is unable to distinguish the two worlds then this must imply that the decryption algorithm more or less always acts identical to the "always-$\perp$" oracle in the fake world.

3.3 All-in-one AE is equivalent to separate definitions AE

This section is optional and can be skipped on first reading.

In this section we prove that the two ways of defining authenticated encryption from Sections 3.1 and 3.2 are equivalent.

IND-CPA + INT-CTXT $\implies$ all-in-one AE. We begin by showing that the individual confidentiality and integrity definitions from Section 3.1 together imply the all-in-one AE security definition from Section 3.2.

Theorem 1.

$$\mathbf{Adv}^{\mathsf{ae}}_{\Sigma}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\Sigma}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{int\text{-}ctxt}}_{\Sigma}(\mathcal{C}) \tag{5}$$

Proof. To simplify notation let's write $\mathcal{A}^{\mathcal{E}(\cdot),\mathcal{D}(\cdot)} \Rightarrow 1$ to denote the event that $\mathcal{A}$ interacts with a real encryption oracle and a real decryption in the all-in-one experiment (i.e. this is the "real world") and finally outputs 1. Similarly, we use $\mathcal{A}^{\mathcal{E}(\$),\perp(\cdot)} \Rightarrow 1$ to denote that $\mathcal{A}$ outputs 1 when interacting with an encryption oracle that returns encryptions of random messages and a decryption oracle that always returns $\perp$ (i.e, this is the "ideal world"). Using this notation we can rewrite the definition of $\mathcal{A}$'s AE advantage into an equivalent form:

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{ae}}_{\Sigma}(\mathcal{A}) &\stackrel{\text{def}}{=} 2 \cdot \Pr[\mathbf{Exp}^{\mathsf{ae}}_{\Sigma}(\mathcal{A}) \Rightarrow \mathsf{true}] - 1 && (6)\\
&= 2 \cdot \left(\Pr[\mathcal{A} \Rightarrow 0 \mid b = 0] \cdot \frac{1}{2} + \Pr[\mathcal{A} \Rightarrow 1 \mid b = 1] \cdot \frac{1}{2} \right) - 1 && (7)\\
&= \Pr[\mathcal{A}^{\mathcal{E}(\cdot),\mathcal{D}(\cdot)} \Rightarrow 0] + \Pr[\mathcal{A}^{\mathcal{E}(\$),\perp(\cdot)} \Rightarrow 1] - 1 && (8)\\
&= \Pr[\mathcal{A}^{\mathcal{E}(\cdot),\mathcal{D}(\cdot)} \Rightarrow 0] + 1 - \Pr[\mathcal{A}^{\mathcal{E}(\$),\perp(\cdot)} \Rightarrow 0] - 1 && (9)\\
&= \Pr[\mathcal{A}^{\mathcal{E}(\cdot),\mathcal{D}(\cdot)} \Rightarrow 0] - \Pr[\mathcal{A}^{\mathcal{E}(\$),\perp(\cdot)} \Rightarrow 0] && (10)
\end{aligned}$$

We can now use a common trick, namely to add "0":

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{ae}}_{\Sigma}(\mathcal{A}) &= \Pr[\mathcal{A}^{\mathcal{E}(\cdot),\mathcal{D}(\cdot)} \Rightarrow 0] - \Pr[\mathcal{A}^{\mathcal{E}(\cdot),\perp(\cdot)} \Rightarrow 0] \\
&\quad + \Pr[\mathcal{A}^{\mathcal{E}(\cdot),\perp(\cdot)} \Rightarrow 0] - \Pr[\mathcal{A}^{\mathcal{E}(\$),\perp(\cdot)} \Rightarrow 0] && (11)\\
&\leq \Pr[\mathcal{A}^{\mathcal{E}(\cdot),\mathcal{D}(\cdot)} \Rightarrow 0] - \Pr[\mathcal{A}^{\mathcal{E}(\cdot),\perp(\cdot)} \Rightarrow 0] + \mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\Sigma}(\mathcal{B}) && (12)
\end{aligned}$$

Specifically, in (11) we added and subtracted the term $\Pr[\mathcal{A}^{\mathcal{E}(\cdot),\perp(\cdot)} \Rightarrow 1]$. This corresponds to a "hybrid" world where $\mathcal{A}$ interacts with a real encryption oracle, but the "always-$\perp$" decryption oracle $\perp(\cdot)$. Then, in (12) we noticed that the term $\Pr[\mathcal{A}^{\mathcal{E}(\cdot),\perp(\cdot)} \Rightarrow 0] - \Pr[\mathcal{A}^{\mathcal{E}(\$),\perp(\cdot)} \Rightarrow 0]$ is identical to the IND-CPA game

since an decryption oracle that always returns $\perp$ is equivalent to not having a decryption oracle at all.

It remains to bound the difference $\Pr[\mathcal{A}^{\mathcal{E}(\cdot),\mathcal{D}(\cdot)} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathcal{E}(\cdot),\perp(\cdot)} \Rightarrow 1]$. This difference can be interpreted as the advantage of $\mathcal{A}$ in distinguishing between the real world and the hybrid world introduced above. Also, note that the real world and the hybrid-fake world are *identical* unless $\mathcal{A}$ forges a ciphertext. Let F denote this event. Then we have:

$$\begin{aligned}
&\Pr[\mathcal{A}^{\mathcal{E}(\cdot),\mathcal{D}(\cdot)} \Rightarrow 0] - \Pr[\mathcal{A}^{\mathcal{E}(\cdot),\perp(\cdot)} \Rightarrow 0] \\
&\qquad = \Pr[\mathcal{A}^{\mathcal{E}(\cdot),\mathcal{D}(\cdot)} \Rightarrow 0 \wedge \overline{F}] + \Pr[\mathcal{A}^{\mathcal{E}(\cdot),\mathcal{D}(\cdot)} \Rightarrow 0 \wedge F] \\
&\qquad\quad - \Pr[\mathcal{A}^{\mathcal{E}(\cdot),\perp(\cdot)} \Rightarrow 0 \wedge \overline{F}] - \Pr[\mathcal{A}^{\mathcal{E}(\cdot),\perp(\cdot)} \Rightarrow 0 \wedge F] && (13) \\
&\qquad = \Pr[\mathcal{A}^{\mathcal{E}(\cdot),\mathcal{D}(\cdot)} \Rightarrow 0 \wedge F] - \Pr[\mathcal{A}^{\mathcal{E}(\cdot),\perp(\cdot)} \Rightarrow 0 \wedge F] && (14) \\
&\qquad = \Big(\Pr[\mathcal{A}^{\mathcal{E}(\cdot),\mathcal{D}(\cdot)} \Rightarrow 0 \mid F] - \Pr[\mathcal{A}^{\mathcal{E}(\cdot),\perp(\cdot)} \Rightarrow 0 \mid F]\Big) \Pr[F] && (15) \\
&\qquad \leq \Pr[F] && (16)
\end{aligned}$$

Explanation for each line above: (13) is just the law of total probability; (14) is cancellation of the terms where event F doesn't happen (because in this case both worlds are identical); (15) is just the law of total probability again (using the formulation based on conditional probability); finally (16) is just using the fact that $\Pr[\mathcal{A}^{\mathcal{E}(\cdot),\mathcal{D}(\cdot)} \Rightarrow 0 \mid F] - \Pr[\mathcal{A}^{\mathcal{E}(\cdot),\perp(\cdot)} \Rightarrow 0 \mid F]$ is a number between -1 and 1, hence always upper bounded by 1.

Thus, we only need to bound $\Pr[F]$, namely the probability that $\mathcal{A}$ manages to forge a ciphertext in the real world or the fake world (doesn't matter which since they are identical until this happens). But this is exactly what the INT-CTXT game is all about! More concretely, let $\mathcal{C}$ be the following INT-CTXT adversary. It starts running $\mathcal{A}$, and when $\mathcal{A}$ makes an $\mathcal{E}(\cdot)$ oracle query it forwards it to the $\mathcal{E}(\cdot)$ oracle in its own INT-CTXT game. When $\mathcal{A}$ makes a decryption oracle query (which either is supposed to be $\mathcal{D}(\cdot)$ or $\perp(\cdot)$) then $\mathcal{C}$ forwards it to its own $\mathcal{D}(\cdot)$ oracle. If the INT-CTXT decryption oracle returns something other than $\perp$ then $\mathcal{C}$ simply stops. Notice that in this case $\mathcal{C}$ wins in its INT-CTXT game, hence

$$\Pr[\mathcal{A}^{\mathcal{E}(\cdot),\mathcal{D}(\cdot)} \Rightarrow 0] - \Pr[\mathcal{A}^{\mathcal{E}(\cdot),\perp(\cdot)} \Rightarrow 0] \leq \Pr[F] \leq \mathbf{Adv}^{\mathsf{int\text{-}ctxt}}_{\Sigma}(\mathcal{C}) \tag{17}$$

Plugging this into (12) gives

$$\mathbf{Adv}^{\mathsf{ae}}_{\Sigma}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\Sigma}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{int\text{-}ctxt}}_{\Sigma}(\mathcal{C}) \tag{18}$$

□

All-in-one AE $\Longrightarrow$ IND-CPA + INT-CTXT. We now show that the all-in-one AE definition implies confidentiality and integrity independently.

Theorem 2.

$$\mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\Sigma}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{ae}}_{\Sigma}(\mathcal{A}) \tag{19}$$

Proof. This is true because the IND-CPA game is simply a more limited version of the AE game where we ignore the decryption oracle $\mathcal{D}(\cdot)$. □

Theorem 3.

$$\mathbf{Adv}^{\mathsf{int\text{-}ctxt}}_{\Sigma}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{ae}}_{\Sigma}(\mathcal{B}) \tag{20}$$

Proof. Let $\mathcal{B}$ be the following adversary. It runs $\mathcal{A}$ and forwards all of its $\mathcal{E}(\cdot)$ and $\mathcal{D}(\cdot)$ queries to its own $\mathcal{E}(\cdot)$ and $\mathcal{D}(\cdot)$ oracles. If $\mathcal{B}$'s decryption oracle $\mathcal{D}(\cdot)$ ever returns something other than $\perp$, then $\mathcal{B}$ stops and outputs $b' = 0$. Else, it outputs $b' = 1$. Using the same simplifying notation as in the proof of Theorem 1, we get

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{ae}}_{\Sigma}(\mathcal{B}) &= \Pr[\mathcal{B}^{\mathcal{E}(\cdot),\mathcal{D}(\cdot)} \Rightarrow 0] - \Pr[\mathcal{B}^{\mathcal{E}(\$),\perp(\cdot)} \Rightarrow 0] && (21)\\
&\geq \Pr[F] - 0 && (22)\\
&= \mathbf{Adv}^{\mathsf{int\text{-}ctxt}}_{\Sigma}(\mathcal{A}) && (23)
\end{aligned}$$

□

4 Achieving AE – generic composition

Having spent much time defining authenticated encryption, how do we actually achieve it? Seeing as authenticated encryption is a combination of confidentiality and integrity, and given that we have previously managed to create primitives achieving both of these goals separately, namely encryption schemes and MACs, it's reasonable to try to combine them in order to create an authenticated encryption scheme. In particular, suppose we have an IND-CPA secure encryption scheme $\Sigma = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ and an UF-CMA secure MAC scheme $\mathsf{MAC} = (\mathsf{KeyGen}, \mathsf{Tag}, \mathsf{Vrfy})$, how can we combine these two to create an AE secure authenticated encryption scheme $\mathsf{AE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$?

There are at least three natural ways to combine Σ and MAC.

- MAC-*then*-Encrypt (MtE): first MAC the message then encrypt both the message and the tag:

  $$C \leftarrow \mathsf{Enc}(K_1, M \| \mathsf{Tag}(K_2, M))$$

  Upon reception of C the receiver first decrypts the ciphertext, then verifies the resulting message with the (decrypted) tag.

  This was the way encryption worked in the TLS protocol up until version 1.2.

- *Encrypt-and-MAC* (E&M): encrypt and MAC the message independently:

  $$C \leftarrow \mathsf{Enc}(K_1, M)$$

  $$T \leftarrow \mathsf{Tag}(K_2, M)$$

  Upon reception of C, T the receiver first decrypts the ciphertext, then verifies the resulting message with the tag.

  This was the way encryption worked in the SSH protocol up until the most recent version.

- *Encrypt-then-MAC* (EtM): first encrypt the message then MAC the ciphertext; send ciphertext and tag:

$$C \leftarrow \mathsf{Enc}(K_1, M)$$

$$T \leftarrow \mathsf{Tag}(K_2, C)$$

Upon reception of C, T the receiver first verifies the ciphertext with the tag, and only decrypts C if the verification passes.

This is the original encryption mode used by the IPsec protocol.

Given that MtE, E&M, and EtM have all been used in real life, are they all achieving AE security? Unfortunately, no.

MtE is not (generically) secure. Suppose Σ is an IND-CPA secure encryption scheme and MAC is a UF-CMA secure MAC. From Σ define a new encryption scheme Σ' that simply prepends a 0 to the front of the ciphertext. Upon decryption Σ' simply ignores the first bit. As you showed in Homework 3, Σ' is still an IND-CPA secure encryption scheme. But what happens if you combine Σ' and MAC in the MtE mode? Is the resulting scheme AE secure? The answer is no, as you are asked to show in the exercises.

E&M is not (generically) secure. Suppose Σ is an IND-CPA secure encryption scheme and MAC is a UF-CMA secure MAC. Since MAC is a deterministic function, what happens if you encrypt the same message twice in E&M? You get the same tag! But this violates the property that IND-CPA secure encryption should hide repeated plaintexts, so E&M is not generically secure.

In fact, it is possible for E&M to violate confidentiality even worse than this. From MAC define a new scheme MAC$'$ which prepends the *message* to the tag, i.e., $\mathsf{MAC}'.\mathsf{Tag}(K, M) = M \| \mathsf{MAC}.\mathsf{Tag}(K, M)$. As you showed in Homework 4, MAC$'$ is still a UF-CMA secure MAC. But what happens if you combine Σ and MAC$'$ in E&M mode? You leak the whole message!

EtM *is* generically secure. After seeing that neither MtE nor E&M are generically secure, you might suspect that EtM also fails to provide AE security. But fortunately this is not the case! As shown in the following theorem, EtM mode provides AE security for *any* choice of IND-CPA secure encryption scheme Σ and UF-CMA secure MAC MAC.

Theorem 4.

$$\mathbf{Adv}^{\mathsf{ae}}_{\mathsf{EtM}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathsf{Enc}}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{uf\text{-}cma}}_{\mathsf{MAC}}(\mathcal{C}) \tag{24}$$

Proof. By Theorem 1 we have

$$\mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathsf{EtM}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathsf{EtM}}(\mathcal{A}') + \mathbf{Adv}^{\mathsf{int\text{-}ctxt}}_{\mathsf{EtM}}(\mathcal{A}'') \tag{25}$$

for some adversaries $\mathcal{A}'$ and $\mathcal{A}''$, so to finish the proof we only need to bound the two terms on the right. We do this in Lemma 1 and Lemma 2 below. □

Lemma 1.

$$\mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathsf{EtM}}(\mathcal{A}') \leq \mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathsf{Enc}}(\mathcal{B}) \tag{26}$$

Proof. Adversary $\mathcal{B}$ begins by picking a random key K for the MAC scheme MAC and then runs $\mathcal{A}'$. When $\mathcal{A}'$ makes an encryption query for a message M, then $\mathcal{B}$ forwards it to its own encryption oracle $\mathcal{E}(M)$ to obtain a ciphertext C. $\mathcal{B}$ then applies a MAC to C using the key it picked itself to obtain $T \leftarrow \mathsf{MAC.Tag}(K, C)$. $\mathcal{B}$ returns $C\|T$ back to $\mathcal{A}'$. When $\mathcal{A}'$ stops with output b', then $\mathcal{B}$ stops too and outputs b' in its IND-CPA game.

Since $\mathcal{B}$ picked the key for the MAC scheme itself, $\mathcal{B}$'s simulation of the MtE scheme for $\mathcal{A}'$ is perfect hence $\mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathsf{EtM}}(\mathcal{A}') \leq \mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathsf{Enc}}(\mathcal{B})$. □

Lemma 2.

$$\mathbf{Adv}^{\mathsf{int\text{-}ctxt}}_{\mathsf{EtM}}(\mathcal{A}'') \leq \mathbf{Adv}^{\mathsf{uf\text{-}cma}}_{\mathsf{MAC}}(\mathcal{C}) \tag{27}$$

Proof. Adversary $\mathcal{C}$ begins by picking a random key K for the encryption scheme Enc and then runs $\mathcal{A}''$. When $\mathcal{A}''$ makes an encryption query for a message M, then $\mathcal{C}$ first encrypts the message with the Enc scheme using the key it picked itself to obtain the ciphertext $C \leftarrow \mathsf{Enc.Enc}(K, M)$, then makes a $\mathsf{Tag}(C)$ oracle query to its own UF-CMA game to obtain a tag T. $\mathcal{C}$ returns $C\|T$ back to $\mathcal{A}''$.

When $\mathcal{A}''$ makes a decryption query $\mathcal{D}(C\|T)$ for some ciphertext C with corresponding tag T, then $\mathcal{C}$ makes a $\mathsf{Vrfy}(C, T)$ oracle query to its own UF-CMA game. If the oracle returns 1 (meaning the query was a valid forgery), then $\mathcal{C}$ simply stops. Else $\mathcal{C}$ returns $\perp$ to $\mathcal{A}''$ and continues answering its queries.

It's not too hard to see that since $\mathcal{C}$ picked the key for the encryption scheme itself, $\mathcal{C}$'s simulation of the MtE scheme for $\mathcal{A}''$ is perfect hence $\mathbf{Adv}^{\mathsf{int\text{-}ctxt}}_{\mathsf{EtM}}(\mathcal{A}'') \leq \mathbf{Adv}^{\mathsf{uf\text{-}cma}}_{\mathsf{MAC}}(\mathcal{C})$. □

Comments 1. Although we could only prove that EtM achieves AE security in general, it is important to keep in mind that MtE and E&M only fails to provide AE security in a *generic* sense. That is, while MtE and E&M are not AE secure for *all* choices of IND-CPA secure encryption schemes and UF-CMA secure MACs—this is what the counter-examples showed—there could nevertheless be *specific* choices of Enc and MAC for which MtE and E&M are secure (and in fact there are).

Comments 2. The keys used for the encryption and the MAC schemes in EtM *must be independent*! This is why we denoted them K_1 and K_2, respectively. If you try to use the same key for both bad things can happen. You are asked to demonstrate this in the Homework. Again, it is important to emphasize that this requirement is in the generic sense. For specific choices of the encryption scheme and MAC you might be able to use the same key for both if you are careful. In fact, the AE schemes we describe in Section 7 do exactly that.

5 AEAD – authentication encryption with associated data

Authenticated encryption provides confidentiality and integrity protection to a message. But sometimes we also have some ancillary data which we cannot encrypt, but for which we still want integrity protection. A common example is the header of a network protocol packet. While we want both confidentiality

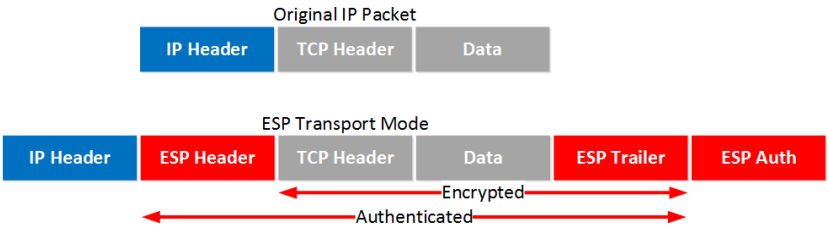


Figure 3: IPsec ESP packet using AEAD.

and integrity protection for the *contents* of the packet, the header itself by necessity have to be in the clear so that the packet can be routed in the network. Nevertheless, it would be good if no one could tamper with the header—for instance changing the destination address to another recipient. In other words, we want integrity protection for the header. The concept is illustrated in Fig. 3 which shows an IPsec network protocol packet where the inner TCP/application data is encrypted (and authenticated). There is also an additional ESP header (and trailer) which is only authenticated but not encrypted.[2]

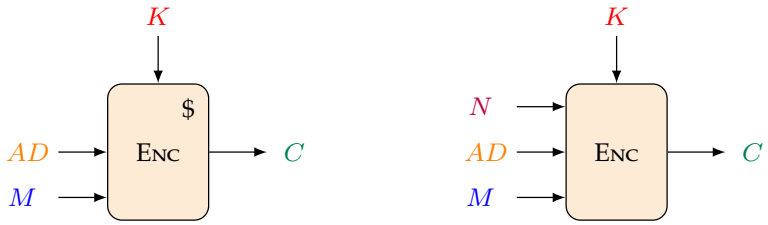
While integrity for the header could be achieved simply by using a MAC, since we're already using an authenticated encryption scheme to protect the contents of the packet it would be nice if we could leverage it to protect the header as well. This leads to the idea of authenticated encryption with *associated data* (AEAD). Here the associated data will receive integrity protection, but no confidentiality protection.

6 Nonce-based authenticated encryption

Recall that one of the important properties of a secure (authenticated) encryption scheme is that repeated encryptions of the same message should yield different ciphertexts. So far we've accommodated for this requirement by saying that the encryption algorithm can be probabilistic (ref. Definition 1). We think of the encryption algorithm as drawing random bits during the encryption process. Specifically, the drawing of random bits happens *internally* to the algorithm. This is illustrated in Fig. 4a, where the "$" symbol is used to denote that Enc generates random bits during encryption (note that the encryption algorithm in Fig. 4a has gained another input as well: AD, the associated data introduced in the previous section).

However, this is not the only way of achieving non-deterministic encryption. An alternative viewpoint is to think of the encryption algorithm itself as being deterministic, but taking in randomness as an *external* input. In fact, the input doesn't even have to be random; for many schemes it is sufficient that the input simply never repeats—it doesn't necessarily have to be unpredictable. This is called a *nonce*—a number used only once—and is illustrated in Fig. 4b. Here Enc has gained the nonce as an additional input and is now also a deterministic function as indicated by the missing "$". We call this *nonce-based AEAD*.

[2]IPsec also has an AH header mode which allows to authenticate (but not encrypt) the outer IP header as well.



(**a**) Internal randomness (**b**) Nonce-based

Figure 4: AEAD

Notice the shift in responsibility: when randomness is generated internally the encryption scheme itself that takes care of randomizing the encryption, but for nonce-based encryption it is the caller's job to ensure that he never repeats a nonce input. So why would we want to use a nonce-based scheme over a scheme that generates randomness itself? One answer is that nonce-based encryption is what real-world software APIs typically provide. Thus nonce-based encryption simply reflects what's "out there". A second answer is that randomness can be difficult and/or time consuming to generate. Some platforms have limited access to randomness sources and thus cannot easily use an encryption scheme that requires randomness. With a nonce-based encryption scheme there is no need for randomness, you must only ensure that the nonce never repeats. Let me emphasize this again: the nonce doesn't have to be random, it just have to be unique for every encryption call you make. For example the nonce could just be a simple counter or some other non-repeating state.

Syntax. Let us now introduce the formal syntax of a nonce-based AEAD scheme.

Definition 4. A *nonce-based authenticated encryption scheme with associated data* (AEAD) is a tuple $\Sigma = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ of algorithms, where:

- KeyGen is a probabilistic key generation algorithm that outputs a key in a *key space* $\mathcal{K}$.
- $\mathsf{Enc} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{C}$ is a deterministic encryption algorithm which takes as input a key in $\mathcal{K}$, a nonce in a *nonce space* $\mathcal{N}$, associated data in an *associated data space* $\mathcal{A}$, and a message in a *message space* $\mathcal{M}$, and outputs a ciphertext in a *ciphertext space* $\mathcal{C}$.
- $\mathsf{Dec} : \mathcal{K} \times \times \mathcal{N} \times \mathcal{A} \times \mathcal{C} \to \mathcal{M} \cup \{\perp\}$ is a deterministic decryption algorithm which takes as input a key in $\mathcal{K}$, a nonce in $\mathcal{N}$, associated data in $\mathcal{A}$, and a ciphertext in $\mathcal{C}$ and outputs either a message in $\mathcal{M}$ or a distinct *error symbol* $\perp$.

Σ is required to satisfy the following *correctness requirement*: for all $K \in \mathcal{K}$, all $N \in \mathcal{N}$, all $AD \in \mathcal{A}$, and all $M \in \mathcal{M}$ we have

$$\mathsf{Dec}(K, N, AD, \mathsf{Enc}(K, N, AD, M)) = M. \tag{28}$$

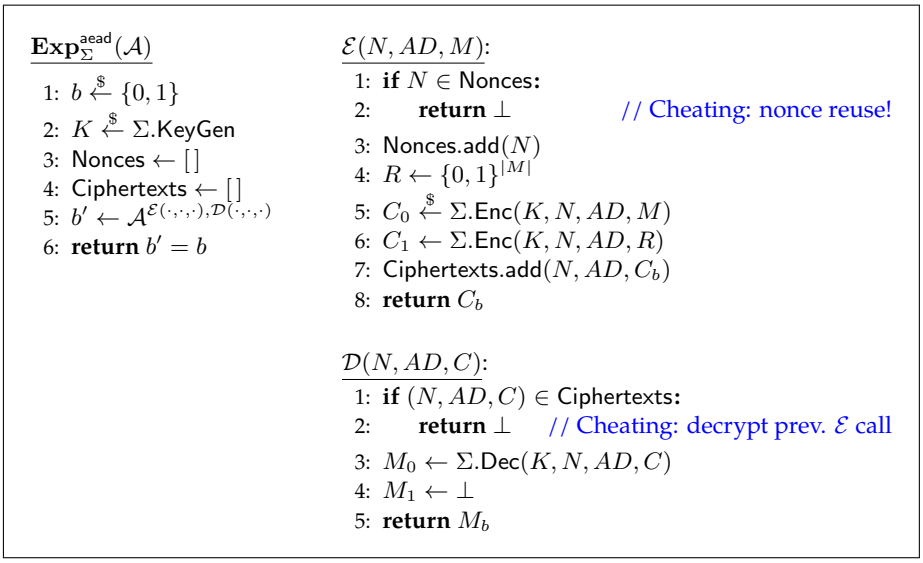


Figure 5: Nonce-based AEAD security game.

Security. How do we define the security of a nonce-based AEAD scheme? The idea is basically the same as for plain authenticated encryption schemes: it should provide confidentiality and integrity protection for its message, and integrity protection for its associated data. The difference is that it now also takes in the nonce as an additional input. Who should generate this? In reality this would be chosen by the honest users, but in order to make the security definition as strong as possible, in the formal security experiment we actually let the adversary be responsible for generating the nonces. The adversary can generate the nonces however how it wants as long as they are always different.

Similar to plain authenticated encryption it is possible to formalize security using separate games for confidentiality and integrity or using a single game that incorporates both. Again, it will turn out that these approaches are equivalent for nonce-based AEAD too, so this time we only present the all-in-one definition as given in Fig. 5.

Definition 5 (AEAD security (informal))**.** A nonce-based AEAD scheme Σ is *AEAD-secure* if $\mathbf{Adv}^{\mathsf{aead}}_{\Sigma}(\mathcal{A})$ is "small" for all "resource bounded" $\mathcal{A}$, where

$$\mathbf{Adv}^{\mathsf{aead}}_{\Sigma}(\mathcal{A}) = \left| 2 \cdot \Pr[\mathbf{Exp}^{\mathsf{aead}}_{\Sigma}(\mathcal{A}) \Rightarrow \mathsf{true}] - 1 \right|. \tag{29}$$

Comments. The security game in Fig. 5 is almost the same as in Fig. 2. In the "real world" ($b = 0$) the adversary gets access to a real encryption oracle and a real decryption oracle, while in the "ideal world" ($b = 1$) the adversary gets access to an encryption oracle that only encrypts random messages, and a decryption oracle that always returns $\perp$. The goal is for the adversary to distinguish these two worlds.

The difference from Fig. 2 is that now that adversary also has to provide a nonce and associated data to the oracle calls. In order to ensure that the adversary never repeats a nonce we store all the queried nonces in the set Nonces. If

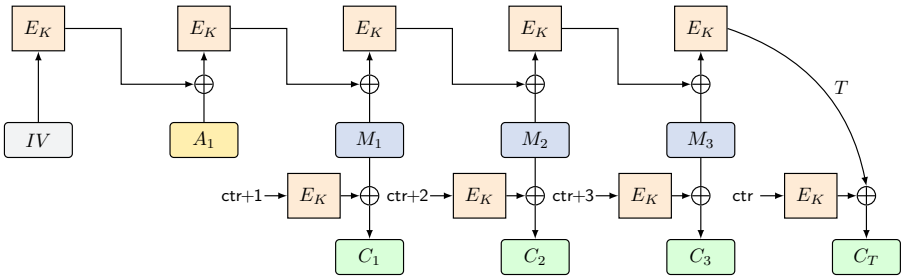


Figure 6: CCM mode of operation. Both IV and ctr are derived from a 104-bit nonce N.

$\mathcal{A}$ every repeats a nonce in an encryption call we suppress the output. Similarly, if the adversary ever tries to decrypt a ciphertext that was output from a previous encryption call, then we suppress the output since this would allow it to trivially win. However, note that we only suppress the output if the decryption call contains exactly the same nonce *and* associated data that was used to create the ciphertext. If any of these are different the decryption call is allowed. For example if the adversary first made the encryption call $\mathcal{E}(N, AD, M)$ and received C, it could then legitimately ask for the decryption of (N', AD, C) or (N, AD', C) or (N, AD, C') without the output being suppressed, given that $N' \neq N$, $AD' \neq AD$, or $C' \neq C$. The point is that Σ should protect the ciphertext if the adversary tries to decrypt with another nonce, or if it modifies the associated data.

7 Achieving nonce-based AEAD

Having a definition of what a (nonce-based) AEAD scheme is and what security it should provide, how do we actually achieve it? Like we did for regular AE in Section 4 we could consider generic composition: combine an IND-CPA secure encryption scheme and a UF-CMA secure MAC in some way to achieve both confidentiality and integrity. Again, we could show that MtE and E&M would not work in general, while EtM would. However, in practice, nonce-based AEAD is usually not achieved by generic composition, but rather by using dedicated all-in-in constructions . In the sections below we give examples of two of the most common such constructions, namely the CCM and GCM modes of operation. Both of these depend on an underlying block cipher denoted E_K. In practice E_K is often AES, in which case we get AES-CCM and AES-GCM, respectively.

7.1 CCM

The CCM mode of operation—*Counter with CBC-MAC*—achieves AEAD by combining CTR-mode encryption and CBC-MAC in a MAC-then-Encrypt fashion, using only a single key. The mode is illustrated in Fig. 6 where it encrypts a message consisting of three blocks and one block of associated data. Note that the top row is CBC-MAC, while the bottom row is CTR-mode encryption. On the face of it, CCM seems to be doing several things we have previously warned

about in the course:

- It uses CBC-MAC which is only secure when MACing messages of a fixed length.
- It combines encryption and MAC in MtE mode which is not generically secure.
- It uses the same key for both CTR-mode encryption and CBC-MAC.

Why is this OK? It turns out that the CCM designers took care to avoid the above problems by introducing a few important extra steps. First, CCM prepends the length of the message (and the associated data) in front of the message before applying CBC-MAC. The length is encoded in the IV block shown in Fig. 6. Recall from the lectures that length-prepended CBC-MAC is secure for all messages lengths. Moreover, CCM also encrypts the final CBC-MAC tag, which is another way of solving the same issue. Second, through careful encoding, CCM ensures that the IV and ctr values are always distinct, which prevents bad interactions between CTR-mode and CBC-MAC using the same key. Finally, there is a security proof! Jonsson [Jonsson(2002)] showed that as long as E is a good block cipher, then CCM is a secure AEAD.

7.2 GCM

The GCM mode of operation—*Galois/Counter mode*—achieves AEAD encryption by combining CTR-mode encryption with a MAC in Encrypt-then-MAC fashion, using only a single key. The mode is illustrated in Fig. 7 where it encrypts three messages blocks and two blocks of associated data. Note that the top row is CTR-mode encryption, while the bottom row superficially appears to be some kind of CBC-MAC. But this is not CBC-MAC at all! Instead, it is a completely different type of MAC altogether (which we did not cover in class) called GMAC.

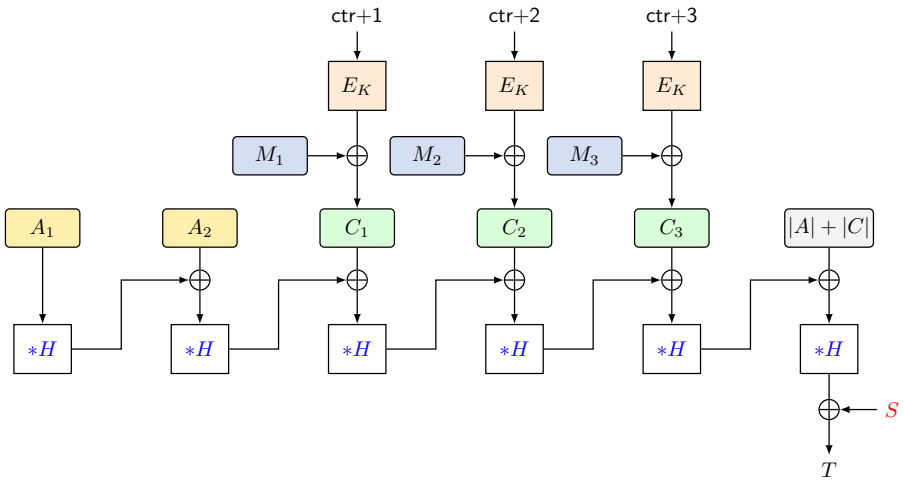


Figure 7: GCM mode of operation. The value H is derived as $H \leftarrow E_K(0^{128})$ and the value S as $S \leftarrow E_K(N\|0\ldots01)$, where N is a 96-bit nonce.

The way to think of GMAC is as follows. In the example given in Fig. 7 imagine interpreting every 128 bit block as a number and let H be some specific number. You should now think of the '$*$' as some special kind of multiplication operation that produces a new number. The result after the first "multiplication" is thus $A_1 * H$. This then gets added to the "number" A_2 which is again multiplied with H to produce $A_1 * H^2 + A_2 * H$, and so on. In particular, the process of "multiply with H, add next block" yields:

$$\begin{aligned} & A_1 * H + A_2 \\ & A_1 * H^2 + A_2 * H + C_1 \\ & A_1 * H^3 + A_2 * H^2 + C_1 * H + C_2 \\ & \qquad \vdots \\ T = A_1 * H^6 + A_2 * H^5 + C_1 * H^4 + \cdots + (|A| + |C|) * H + S \end{aligned}$$

In other words, the tag T is a *polynomial* in the "unknown" H with the blocks of the associated data and ciphertext as its coefficients!

GMAC is a very different MAC construction than what we've looked at so far. It is not based on a block cipher, and it's also an example of a *nonce-based* MAC. In particular, it is absolutely essential for the security of GMAC that the final value that gets XORed to the tag at the end (i.e., the S value) is new for *every single message*.[3] As noted in Fig. 7, this is achieved by deriving S from a nonce N using a block cipher call (although S could also just be picked at random).

Comments.

- Since GCM is based on CTR mode, encryption can be parallelized. In fact, even the GMAC construction can be parallelized with some tricks.
- It is standardized by NIST and is basically used everywhere.
- It is very fast if dedicated hardware instructions are used for the underlying AES calls. Indeed, GCM is so commonly used that Intel also added dedicated hardware instructions to perform the H multiplication inside GMAC
- GCM has been proven to be a secure AEAD scheme provided the underlying block cipher is good. The first proof was provided by the GCM designers [McGrew and Viega(2004)], but many other proofs have been provided since, improving upon the original result.

References

[Jonsson(2002)] Jakob Jonsson. On the security of CTR + CBC-MAC. In Kaisa Nyberg and Howard M. Heys, editors, *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002, St. John's, Newfoundland, Canada, August 15-16, 2002. Revised Papers*, volume 2595 of *Lecture*

[3] See Problem set 5 for what can go wrong if this is not the case.

Notes in Computer Science, pages 76–93. Springer, 2002. doi: 10.1007/3-540-36492-7_7. URL https://link.springer.com/content/pdf/10.1007%2F3-540-36492-7_7.pdf.

[McGrew and Viega(2004)] David A. McGrew and John Viega. The security and performance of the galois/counter mode (GCM) of operation. In Anne Canteaut and Kapalee Viswanathan, editors, *Progress in Cryptology - INDOCRYPT 2004, 5th International Conference on Cryptology in India, Chennai, India, December 20-22, 2004, Proceedings*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2004. doi: 10.1007/978-3-540-30556-9_27. URL https://doi.org/10.1007/978-3-540-30556-9_27.