

Introduction to Cryptography

TEK 4500 (Fall 2024)

Problem Set 5

Problem 1.

Read [note](#) on authenticated encryption.

Problem 2.

Let CTR\$ be the CTR mode encryption scheme using a random IV. Define Σ to be the following encryption scheme:

$$\Sigma.\text{Enc}(K, M) = \text{CTR}.\text{Enc}(K, M \parallel \text{CRC32}(M)),$$

where $\text{CRC32} : \{0, 1\}^* \rightarrow \{0, 1\}^{32}$ is the well-known [error-detecting code](#). Suppose $C = C_0 \parallel C_1 \parallel C_2$ was the Σ -encryption of message $M = 0^{128}$, where C_0 is the (random) IV of CTR\$ and $|C_2| = 32$. Explain how you would modify C so that it instead decrypts to $M' = 1^{128}$.

Would changing CRC32 to another function, say a strong hash function like [SHA2-256](#) or a truly random (but public) function ρ , change anything?

Problem 3. [Problem 7.3 in [BR](#)]

Let $\Sigma = (\text{KeyGen}, \text{Enc}, \text{Dec})$ be a symmetric encryption scheme and let $\Pi = (\text{KeyGen}, \text{Tag}, \text{Vrfy})$ be a message authentication code. Alice (A) and Bob (B) share a secret key $K = (K_1, K_2)$ where $K_1 \xleftarrow{\$} \Sigma.\text{KeyGen}$ and $K_2 \xleftarrow{\$} \Pi.\text{KeyGen}$. Alice wants to send messages to Bob in a private and authenticated way. Consider her sending each of the following as a means to this end. For each, say whether it is a secure way or not, and briefly justify your answer. (In the cases where the method is good, you don't have to give a proof, just the intuition.)

- a) $M, \text{Tag}_{K_2}(\text{Enc}_{K_1}(M))$
- b) $\text{Enc}_{K_1}(M, \text{Tag}_{K_2}(M))$
- c) $\text{Tag}_{K_2}(\text{Enc}_{K_1}(M))$
- d) $\text{Enc}_{K_1}(M), \text{Tag}_{K_2}(M)$

<u>$\text{Exp}_{\Sigma}^{\text{ae}}(\mathcal{A})$</u>	<u>$\mathcal{E}(M)$:</u>
1: $b \xleftarrow{\$} \{0, 1\}$	1: $R \leftarrow \{0, 1\}^{ M }$
2: $K \xleftarrow{\$} \Sigma.\text{KeyGen}$	2: $C_0 \xleftarrow{\$} \Sigma.\text{Enc}(K, M)$
3: Ciphertexts $\leftarrow []$	3: $C_1 \leftarrow \Sigma.\text{Enc}(K, R)$
4: $b' \leftarrow \mathcal{A}^{\mathcal{E}(\cdot), \mathcal{D}(\cdot)}$	4: Ciphertexts.add(C_b)
5: return $b' = b$	5: return C_b
	<u>$\mathcal{D}(C)$:</u>
	1: if $C \in \text{Ciphertexts}$ then
	2: return $\perp$
	3: $M_0 \leftarrow \Sigma.\text{Dec}(K, C)$
	4: $M_1 \leftarrow \perp$
	5: return M_b
$\text{Adv}_{\Sigma}^{\text{ae}}(\mathcal{A}) = 2 \cdot \Pr[\text{Exp}_{\Sigma}^{\text{ae}}(\mathcal{A}) \Rightarrow \text{true}] - 1/2 $	

Figure 1: Authenticated encryption (AE) security experiment and AE-advantage definition.

- e) $\text{Enc}_{K_1}(M), \text{Enc}_{K_1}(\text{Tag}_{K_2}(M))$
- f) $C, \text{Tag}_{K_2}(C)$, where $C \leftarrow \text{Enc}_{K_1}(M)$
- g) $\text{Enc}_{K_1}(M, A)$ where A encodes the identity of Alice; B decrypts the received ciphertext C and checks that the second half of the plaintext is “A”.

In analyzing these schemes, you should assume that Σ is IND-CPA secure and that Π is UF-CMA secure, but nothing else; for an option to be good it must work for any choice of a secure encryption scheme and a secure MAC.

Now, out of all the ways you deemed secure, suppose you had to choose one to implement for a network security application. Taking performance issues into account, do all the schemes look pretty much the same, or is there one you would prefer?

Problem 4. [Problem 9.1 in [BS]]

Let Σ be an AE-secure cipher. Consider the following two derived ciphers:

$\Sigma_1.\text{KeyGen}:$ 1: return $\Sigma.\text{KeyGen}$	$\Sigma_1.\text{Enc}(K, M):$ 1: $C_1 \leftarrow \Sigma.\text{Enc}(K, M)$ 2: $C_2 \leftarrow \Sigma.\text{Enc}(K, M)$ 3: return (C_1, C_2)	$\Sigma_1.\text{Dec}(K, C):$ 1: Parse C as (C_1, C_2) 2: $M_1 \leftarrow \Sigma.\text{Dec}(K, C_1)$ 3: $M_2 \leftarrow \Sigma.\text{Dec}(K, C_2)$ 4: if $M_1 = M_2$ then 5: return M_1 6: else 7: return $\perp$
$\Sigma_2.\text{KeyGen}:$ 1: return $\Sigma.\text{KeyGen}$	$\Sigma_2.\text{Enc}(K, M):$ 1: $C \leftarrow \Sigma.\text{Enc}(K, M)$ 2: return (C, C)	$\Sigma_2.\text{Dec}(K, C):$ 1: Parse C as (C_1, C_2) 2: if $C_1 = C_2$ then 3: return $\Sigma.\text{Dec}(K, C_1)$ 4: else 5: return $\perp$

Is Σ_1 AE-secure? Is Σ_2 AE-secure? If yes, give a high-level justification; if no, give a concrete attack.

Problem 5.

An important point about the Encrypt-then-MAC construction is that the encryption scheme and the MAC scheme must use *independent* keys. In this problem we'll look at what can go wrong if this is not the case.

- a) As a warm-up, suppose we are only interested in encrypting messages of exactly n bits for some small n (say $n = 128$) and that we have access to a block cipher $E : \{0, 1\}^k \times \{0, 1\}^{2n} \rightarrow \{0, 1\}^{2n}$ with the following property:

- E is a secure PRP; and
- the inverse direction of E , i.e., $D_K(Y) = E_K^{-1}(Y)$ is *also* a secure PRP.

A block cipher with this property is said to be a *strong* block cipher. Thus, a strong block cipher is a secure PRP no matter if you're using it in the "forward" direction or in the "backward" direction. As an example, AES is believed to be a strong block cipher.

Given E we construct the following encryption and MAC schemes, defined by their Enc and Tag algorithms (the remaining algorithms are the obvious ones):

- $\text{Enc}(K, M) = E_K(R \| M)$, where $M \in \{0, 1\}^n$ and $R \xleftarrow{\$} \{0, 1\}^n$ is a random string.
- $\text{Tag}(K, M) = D_K(M)$.

It is possible to show that if E is a strong PRP then Enc is IND-CPA secure and that Tag is UF-CMA secure¹. However, show that the Encrypt-then-MAC combination of Enc and Tag is *not* secure if you're using the same key K for both.

- b) Suppose instead we're using Encrypt-then-MAC with CBC\$-mode for encryption and CBC-MAC for authentication, and that we're careful to only encrypt messages having exactly ℓ blocks of n bits each. From class we know that CBC\$-mode encryption is IND-CPA secure and that CBC-MAC is UF-CMA secure as long as we're only MACing messages having exactly $\ell + 1$ blocks and not MACing any other lengths. However, show that the Encrypt-then-MAC combination of the two is not secure if you're using the same key K for both.

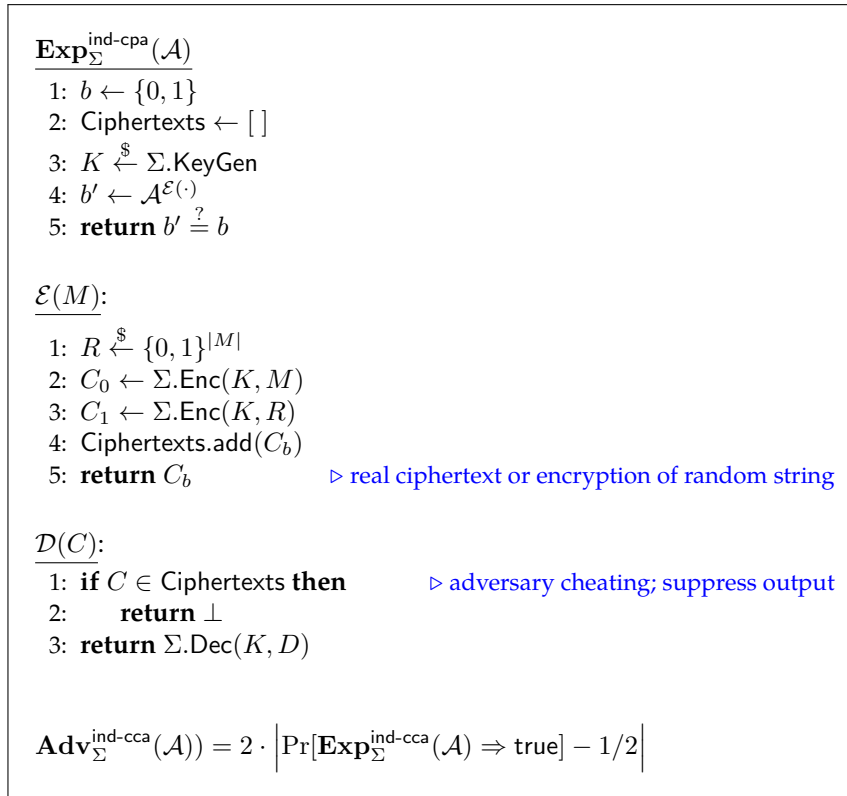


Figure 2: IND-CCA security experiment.

¹The last point is trivial given what we saw in class: any secure PRF is also a good fixed-length MAC, and all secure PRPs are also secure PRFs.

Problem 6. [AE $\implies$ IND-CCA, but IND-CCA $\not\Rightarrow$ AE]

In this exercise we will see that IND-CCA security (ref. Fig 2) does *not* imply AE security (ref. Fig 1), while AE security *does* imply IND-CCA security. In other words, AE is a *stronger* security notion than IND-CCA. Let $\Sigma = (\text{KeyGen}, \text{Enc}, \text{Dec})$ be an IND-CCA secure encryption scheme. Define the following derived encryption scheme:

$\Sigma'.\text{KeyGen}$:	$\Sigma'.\text{Enc}(K, M)$:	$\Sigma'.\text{Dec}(K, C)$:
1: return $\Sigma.\text{KeyGen}$	1: $C \leftarrow \Sigma.\text{Enc}(K, M)$	1: Parse C as $x\ C'$ where x is one bit
	2: return $0\ C$	2: if $x = 0$ then
		3: return $\Sigma.\text{Dec}(K, C')$
		4: else
		5: return 0

- Argue why Σ' is also IND-CCA secure.
- Show that Σ' is not AE secure by demonstrating a concrete attack. Calculate the AE-advantage of your attack. That is, compute $\text{Adv}_{\Sigma'}^{\text{ae}}(\mathcal{A})$, where $\mathcal{A}$ is the adversary that runs your attack.
- Show that AE security implies IND-CCA security. You don't have to provide a full proof, just a high-level argument.

Problem 7. [Nonce-reuse in GCM leaks the authentication key.]

Note: This isn't really an exercise *per se* (there isn't anything for you to answer!), instead, it is a write-up of how bad the GCM mode can fail if you ever reuse a nonce. Reading through this exercise is a good way for you to become more familiar with the GCM mode-of-operation. Also, this exercise requires some familiarity with polynomials. If you want to read more about the (practical) consequences of nonce-reuse in GCM inside the TLS protocol, have a look at the paper “*Nonce-Disrespecting Adversaries: Practical Forgery Attacks on GCM in TLS*” ([link](#)) by Hanno Böck, Aaron Zauner, Sean Devlin, Juraj Somorovsky, and Philipp Jovanovic.

Recall that the GCM mode-of-operation is essentially an instance of the Encrypt-then-MAC paradigm. In particular, to encrypt a message M GCM first encrypts M with (nonce-based) CTR to produce a ciphertext C . Then it applies a (nonce-based) MAC to C called GMAC to produce the final tag T . In particular: $T = \text{GMAC}(H, S, C)$ where H is the MAC key and S is the nonce for GMAC. See Fig. 3 for details. When GMAC is used inside GCM then S is actually derived from the nonce N and key K . On the other hand, the value H is *not* dependent on the nonce, only on the key K . Hence H will be the same for all messages encrypted under the same key. This will be important later.

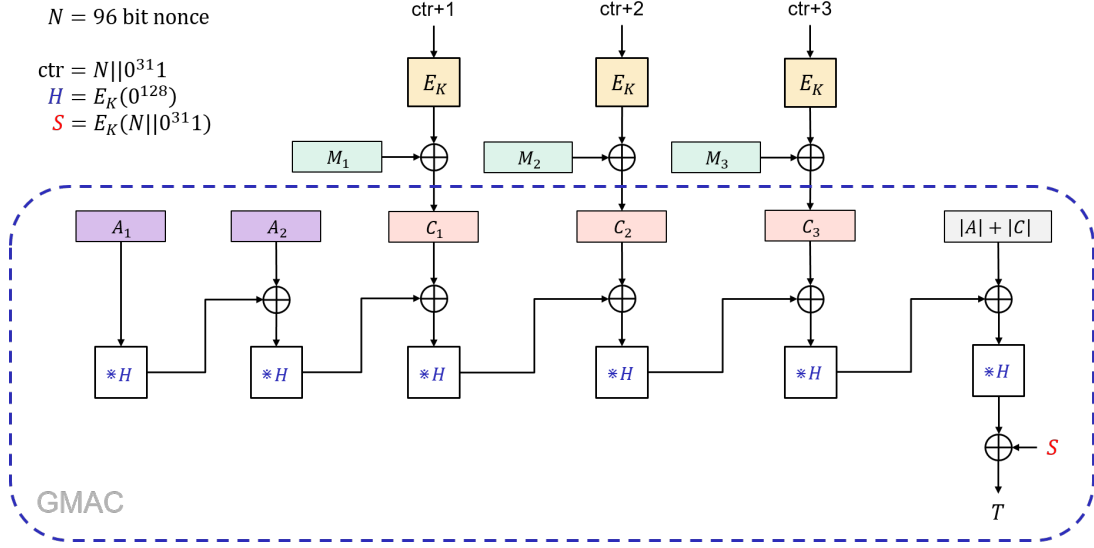


Figure 3: GCM mode-of-operation. The internal MAC function GMAC circled.

The GMAC function, which produces the tag T in GCM, can be thought of as an evaluation of a polynomial

$$g(X) = \sum_i \alpha_i X^i,$$

where the coefficients α_i are determined by the values of the additional data AD and the ciphertext C , where the constant term is the “one-time pad”-like value S . For example, suppose the additional data consists of two blocks $AD = A_1 || A_2$, and the ciphertext of three blocks $C = C_1 || C_2 || C_3$ (as shown in Fig. 3). Then we get the polynomial:

$$g(X) \leftarrow A_1 X^6 + A_2 X^5 + C_1 X^4 + C_2 X^3 + C_3 X^2 + LX + S, \quad (1)$$

where L encodes the length of A and C and S is a nonce-derived value. To compute the GMAC tag on A and C we simply evaluate $g(X)$ on the (secret) value $H = E_K(0^{128})$:

$$T \leftarrow g(H) = A_1 H^6 + A_2 H^5 + C_1 H^4 + C_2 H^3 + C_3 H^2 + LH + S.$$

It is very important that GCM *never* reuses the same nonce N twice for the same key K . We will now show why.

Suppose two messages M and M' have been GCM encrypted under the same nonce (and key). For simplicity, assume there is no additional data and the messages only consist of a single 128-bit block. Thus, the corresponding CTR-part of the ciphertexts also only consist of a single block C and C' , respectively.

Referring to (1), the corresponding GMAC polynomials then become:

$$\begin{aligned} g(X) &\leftarrow CX^2 + LX + S \\ g'(X) &\leftarrow C'X^2 + LX + S \end{aligned}$$

where L encodes the length of C (and C') and $S = E_K(N\|0^{31}1)$. In particular, note that S is the same for both since they are reusing the nonce N .

To compute the tag on C and C' we simply evaluate $g(X)$ and $g'(X)$ on $H = E_K(0^{128})$:

$$T \leftarrow g(H) = CH^2 + LH + S \quad (2)$$

$$T' \leftarrow g'(H) = C'H^2 + LH + S \quad (3)$$

Now, the multiplication and addition happening in (2) and (3) is not actually normal multiplication and addition over the integers, but rather happening in a [finite field](#). Fortunately, we don't have to care about the details of finite fields here. The only thing we need to know is that the addition in the finite field used by GCM is the *same* as the standard XOR operation between bitstrings.

Thus, if we add T and T' we get:

$$T + T' = g(H) + g(H') = CH^2 + C'H^2 = (C + C')H^2 \quad (4)$$

where we used the fact that the " $LH + S$ " term is the same for both T and T' , and hence cancel out (as happens when you XOR two equal values). Rearranging (4) we have:

$$(C + C')H^2 + (T + T') = 0. \quad (5)$$

Notice that the only value we (the attacker) don't know in (5) is H , since C , C' , T , and T' are all known to us. Thus, if we could solve (5) for H we would actually be able to *forge any GCM ciphertext*! Why? Look at Fig. 3: the H value does not depend on the nonce N . It is re-used for *all* GCM computations, and can thus be reused by us to create forgeries on new ciphertexts. However, we still need the value S to create the final tag (again, refer to Fig. 3). Fortunately, this is not a big problem: when creating a forgery, we simply reuse the nonce from a previous message from which we can learn S (since we know H). Concretely, suppose we use the nonce N from above in our future forgeries. This would also require us to use the same S . But this S can easily be deduced from (2) since we now know H (together with C , L , and T):

$$S \leftarrow T + CH^2 + LH \quad (6)$$

With all of this in hand, let's see how we would use it to forge an arbitrary ciphertext, say $C^* = C_1^* \| C_2^* \| C_3^*$. For an added bonus, suppose we also want to include some additional data $AD = A_1^*$. Our final output will then be:

$$N \| C_1^* \| C_2^* \| C_3^* \| T^*,$$

where N is the same N used to create C and C' above, and T^* is computed as:

$$T^* \leftarrow A_1^* H^5 + C_1^* H^4 + C_2^* H^3 + C_3^* H^2 + L^* H + S,$$

where S is the value recovered in (6).

The only thing we still haven't answered is how to actually solve for H in (5). However, this is easy: the equation in (5) is a quadratic equation hence can be solved by simple algebra (in particular, the quadratic formula which is also valid in finite fields).

Conclusion: Reusing the nonce (with the same key) in GCM is *bad*! It essentially leaks the GMAC key (H) which more or less voids all authentication guarantees that GCM was supposed to give. The lesson is: never reuse the nonce when using GCM!

References

- [BR] Mihir Bellare and Phillip Rogaway. *Introduction to Modern Cryptography*. <https://web.cs.ucdavis.edu/~rogaway/classes/227/spring05/book/main.pdf>.
- [BS] Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography*, (version 0.6, Jan. 2023). <https://toc.cryptobook.us/>.