Massachusetts Institute of Technology
6.857: Applied Cryptography and Security
Professors Ronald L. Rivest and Yael Tauman Kalai

Handout 3
Feburary 22, 2022
Due: March 7, 2022

Problem Set 2

This problem set is due on *Monday, March 7, 2022* at **11:59 PM**. Please note our late submission penalty policy in the course information handout. Please submit your problem set, in PDF format, on Gradescope. *Each problem should be in a separate page.* Have **one and only one group member** submit the finished problem writeups. Please title each PDF with the Kerberos of your group members as well as the problem set number and problem number (i.e. *kerberos1_kerberos2_kerberos3_pset1_problem1.pdf*).

You are to work on this problem set in groups. For problem sets 1, 2, and 3, we will randomly assign the groups for the problem set. After problem set 3, you are to work on the following problem sets with groups of your choosing of size three or four. If you need help finding a group, try posting on Piazza or email `6.857-staff@mit.edu`. You don't have to tell us your group members, just make sure you indicate them on Gradescope. Be sure that all group members can explain the solutions. See Handout 1 (*Course Information*) for our policy on collaboration.

Homework must be submitted electronically! Each problem answer must be provided as a separate page. Mark the top of each page with your group member names, the course number (6.857), the problem set number and question, and the date. We have provided templates for LaTeX and Microsoft Word on the course website (see the *Resources* page).

Grading: All problems are worth 10 points.

With the authors' permission, we may distribute our favorite solution to each problem as the "official" solution—this is your chance to become famous! If you do not wish for your homework to be used as an official solution, or if you wish that it only be used anonymously, please note this in your profile on your homework submission.

Problem 2-1. Problem 1

Encryption and message authentication codes (MACs) serve very different purposes. Encryption provides *secrecy* guaranteeing that no outside observer can read the message contents while MACs provide *authenticity* guaranteeing that no adversary can generate a valid MAC on a new message. In practice, both are very desirable properties so people like to both encrypt and MAC their messages. However, combining an encryption scheme with a MAC scheme to ensure both secrecy and authenticity needs to be done with care.

In class we learned about AES-CBC MAC. But the "CBC mode of operation" can also be used to encrypt arbitrarily long messages (an alternative to the counter mode that we learned in class). In this problem we will focus on the use of AES-CBC to MAC and encrypt messages and how this can (and has) gone wrong in practice. Two relevant articles are the Wikipedia article on CBC mode encryption (`https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Cipher_block_chaining_(CBC)`) and the Wikipedia article on CBC-MAC (`https://en.wikipedia.org/wiki/CBC-MAC`).

(a) Reusing keys for messages of Fixed Length. As discussed in class and explained in the Wikipedia article on CBC-MAC (`https://en.wikipedia.org/wiki/CBC-MAC#Using_the_same_key_for_encryption_and_authentication`), it is *very* important not to use the same key for both encryption and authentication using CBC! We've included diagrams for CBC encryption and MAC (for fixed length messages) below – please read the linked article on the attack that occurs when using the same key for encryption and authentication and answer the following questions about the article.

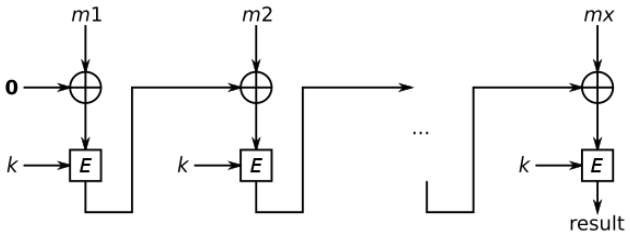

Figure 1: The computation of a CBC-MAC for a fixed-length message from message blocks m1. . . mx, with secret key k and block cipher E. Note that the "E" boxes here are the same as the 'block cipher encryption' boxes from Figure 2. (figure taken from Wikipedia article)

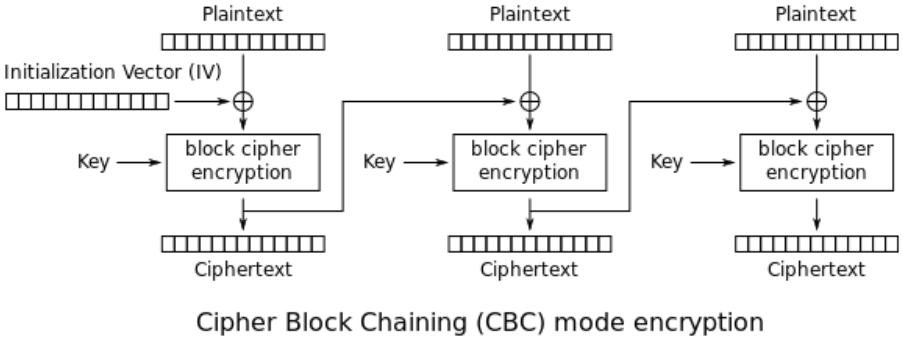

Figure 2: Encryption of plaintext blocks into ciphertext blocks using CBC mode. Note that the 'block cipher encryption' boxes are the same as the 'E' boxes from the MAC in Figure 1. (again from Wikipedia)

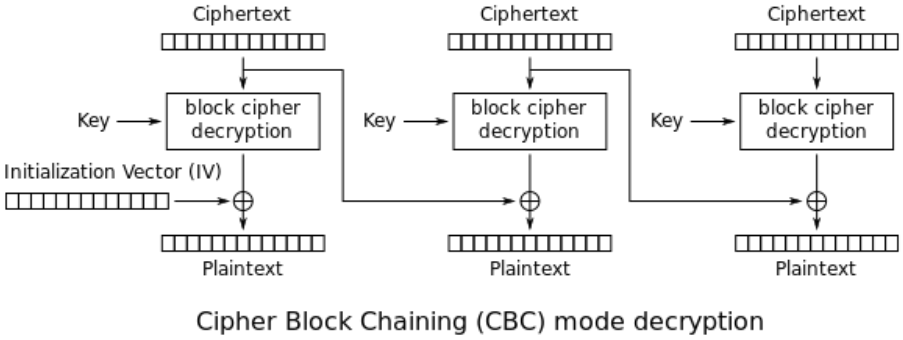

Figure 3: Decryption of ciphertext blocks into plaintext blocks using CBC mode (still Wikipedia)

1. Are there any restrictions on what message m' the adversary can forge using the method described in the article? That is, given a message m and a tag t, can the adversary generate a valid tag t' for **any** message m' of the same length as m (using the method from the article)? Explain your reasoning.
2. The article is a little light in its explanation of why using different keys for encryption and authentication resolves the attack it describes. Please provide your own explanation with more detailed reasoning. You don't need to rewrite the whole article, simply clarify points you feel

deserve more explanation.

NOTE: One final problem with the use of CBC for encryption and authentication as described in the article is that it is an *Encrypt-and-MAC* scheme where the MAC is taken over the plaintext message. This method provides **no** authenticity over the ciphertext, allowing an adversary to modify it as done in the attack you look at here. In the general case it can also reveal the plaintext! While CBC-MAC does provide confidentiality for the message m, this is not a requirement of MAC schemes in general. *Encrypt-and-MAC* constructions **should not** be used in practice.

(b) Reusing keys for messages of variable length. As explained in class (and in the Wikipedia articles we have linked) people want to send variable length messages, not fixed length messages. The CBC-MAC for variable length messages uses an additional key as shown in Figure 4.

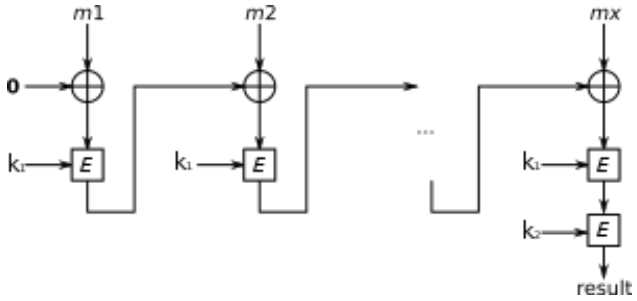

Figure 4: Using an additional key to encrypt the last block of CBC-MAC allows it to support variable length messages. (still Wikipedia)

1. Can you still do the attack from part (a) on this variable length scheme if you reuse keys as follows: Enc(k1, m) and MAC(k1, k2, m) ? Explain your reasoning.
2. What about if you swap the order of the keys and do Enc(k1, m) and MAC(k2, k1, m) instead?

(c) The order of encryption and authentication. Even when using different keys for encryption and MACs you can still have problems. In particular, it is important to *Encrypt-then-MAC* where the MAC is taken over the ciphertext, not the plaintext. Give an example of a CPA secure encryption scheme where if we first MAC and then encrypt then the resulting scheme is **not** CCA secure.

Hint: consider a bit-by-bit encryption scheme, where each bit of the message is encrypted independently. In particular, for a message m of length n and a corresponding tag t of length ℓ, it holds that the "authenticated encryption" of m (given by the MAC-then-encrypt scheme) is: $\text{Enc}(k,(m,t)) = (c_1,,c_n,c_{n+1},\ldots,c_{n+\ell})$, where for every $i \in [n]$, c_i is an encryption of the i'th bit of m. Show how an adversary who is given $\text{Enc}(k,(m,t))$, where t is a valid tag corresponding to m, can use the decryption oracle to find m (assuming that the adversary knows $\text{Enc}(k,0)$ and $\text{Enc}(k,1)$, which she can get from his encryption oracle). We emphasize that the decryption oracle will reject if it is given any ciphertext of the form $\text{Enc}(m',t')$ where t' is not a valid tag of m'.

Problem 2-2. EZCipher

In this problem, you will implement a new block cipher and you will learn one way of using the birthday paradox to test it against potential weaknesses.

The weakness we will test here in particular is called *closure*. An encryption algorithm Enc is said to be *closed* if, for any keys k_1, k_2, there exists a third key k_3 such that for all messages M:

$$Enc(Enc(m,k_1),k_2) = Enc(m,k_3)$$

(a) One common technique to strengthen an encryption algorithm is multiple encryption. Specifically, Sender and Receiver agree beforehand on a sequence of $(T+1)$ randomly chosen keys $k_0, k_1, \cdots, k_T$. Then, to encrypt a message m, the Sender computes ciphertexts

$$c_0 = Enc(m, k_0)$$
$$c_i = Enc(c_{i-1}, k_i), \text{ for } i = 1, 2, \ldots, T$$

and transmits c_T to the Receiver. Argue that if an encryption algorithm is closed, then multiple encryption with $T \geq 1$ offers no security advantage to the Sender and Receiver.

We now suggest a way to find evidence that a cipher is *not* closed, by performing a certain random walk over the space of messages. More precisely, given an encryption algorithm Enc, an initial message m_0, and random independent keys $(k_i)_{i \in \mathbb{N}}$, we construct a sequence of messages $(m_i)_{i \in \mathbb{N}}$ according to:

$$m_i = Enc(m_{i-1}, k_i) \tag{1}$$

i.e., each new message is obtained by encrypting the previous one with a freshly sampled random key. We say a message m^* is *reachable* from m_0 if there exists some choice of keys that puts m^* in the sequence above – i.e., $m_j = m^*$ for some j.

For the rest of this problem, assume the message space M is the set of all 64-bit binary strings and the key space K is a set of much smaller size.

(b) Intuitively, our goal is to design encryption algorithms that seem like truly random mappings. For such algorithms, how many messages would you expect to be reachable from a fixed message m_0?

(c) Show that if the encryption algorithm is closed, then at most $|K|$ messages are reachable from m_0.

We are interested in finding collisions in this walk, which are values $i < j$ such that $m_i = m_j$.

(d) For this part, assume $|M| = 2^{64}$ and $|K| = 2^{39}$. You run the procedure above for 2^{24} steps and verify if any collisions are ever found. Give the approximate probability of finding such a collision if the algorithm is closed. What if it is ideally random as in part (b)? Explain how you found these values. (Hint: This article should help you: `https://en.wikipedia.org/wiki/Birthday_attack`)

You will now implement EZCipher and perform the random walk described above to provide evidence that it is not closed. The cipher takes a 64-bit integer m and a pair (a, b) of 20-bit integers such that a is odd. It then performs 8 rounds of the following two operations, alternately:

(i) **swap:** swap the leftmost and rightmost halves of m's binary representation.

(ii) **linear transformation:** replace m with $a * m + b$ modulo 2^{64}.

We have provided you with a Python script containing a pseudo-random key generator. In this problem, you **must** set $k_i = key\_gen(m_{i-1})$ for $i \geq 1$ (using the notation introduced in (1)).

(e) Set m_0 to *get_init_val*(your_kerberos). Report $m_{2^{24}}$ for each kerberos in your team. We provided you with a unit test for this task. Submit your code.

(f) How many collisions did you find until getting to $m_{2^{24}}$? If a collision is found after these many iterations, does it mean EZCipher is necessarily closed? Alternatively, if no collisions are found, does that prove that the cipher is not closed? Discuss the conclusions you should draw about EZCipher from both outcomes.

Problem 2-3. AES-GCM

Recall from lecture that AES-GCM is an authenticated encryption scheme, that is, it provides both secrecy and authenticity. Secrecy comes from using AES in counter mode, which is a CPA secure encryption scheme (assuming AES is an ideal block-cipher, i.e., a PRF). Authenticity comes from applying a variant of CBC MAC to the ciphertext, where recall that CBC MAC applies a MAC to each block (in this case a ciphertext block), and chains the result by xoring it to the next ciphertext block. However, in AES-GCM, rather than using AES as the MAC for each block, it uses multiplication. Namely, it applies M_H to each ciphertext block, where M_H is simply the multiplication function: $M_H(x) = H \cdot x$, and where $H \coloneqq \texttt{AES}(k, 0)$, where k is the secret. The final tag is then xored with $\texttt{AES}(k, iv)$, where iv is the initial value used in the AES counter mode.[1]

(a) Argue that M_H is **not** a secure authentication mechanism for authenticating blocks of length 128; i.e., that it is not secure against chosen message attacks. Do so by demonstrating an adversarial attack. Recall that in this security game the adversary gets access to a "tagging" oracle, which accepts messages from the adversary and returns tags for them, and the adversary must then output a valid tag for a (new) message. In the context of this problem, the oracle is of the form $M_H(x) = H \cdot x$.

(b) Argue that the authentication mechanism **is** one-time secure, excluding $x = 0$. That is, an adversary that has not seen any tags before cannot generate a valid tag for a (single) message distinct from zero.

(c) Even though M_H is only one-time secure, AES-GCM is still provides authenticity in the many-time setting! Why is this the case? Why is one-time security sufficient?

[1] Actually, before xoring with $\texttt{AES}(k, iv)$, AES-GCM authenticates the number of blocks (n) in the original message by xoring the tag with the n, multiplyng the result with H, and only then xoring the result with $\texttt{AES}(k, iv)$. We ignore this detail here (and we didn't have time to cover this in class).