CS 388H: Cryptography **Spring 2025**

Homework 2: Symmetric Cryptography

Due: February 19, 2025 at 11:59pm (Submit on Gradescope) **Instructor:** David Wu

Instructions. You **must** typeset your solution in LaTeX using the provided template:

`https://www.cs.utexas.edu/~dwu4/courses/sp25/static/homework.tex`

You must submit your problem set via Gradescope (accessible through Canvas).

Collaboration Policy. You may discuss your general *high-level* strategy with other students, but you may not share any written documents or code. You should not search online for solutions to these problems. If you do consult external sources, you must cite them in your submission. You must include the names of all of your collaborators with your submission. Refer to the official course policies for the full details.

Problem 1: CBC Padding Oracle Attack [15 points]. Recall that when using a block cipher in CBC mode, the message must be a multiple of the block size. When encrypting messages whose length is not a multiple of the block size, the message must first be padded. In the TLS protocol (used for securing traffic on the web), if v bytes of padding are needed, then v bytes with value $(v-1)$ are appended to the message. As a concrete example, if 1 byte of padding is needed, a single byte with value 0 is appended to the message before applying CBC encryption. In TLS, the record layer is secured using an approach called "MAC-then-Encrypt[1]" (which as we will soon see, is not the ideal combination). At decryption time, the ciphertext is first decrypted (and the padding verified) *before* checking the MAC. In older versions of OpenSSL, the library reports whether a decryption failure was due to a "bad pad" or due to a "MAC verification failure." One might think that it was beneficial to provide an informative error message on decryption failure. As you will show in this problem, this turns out to be a disaster for security.

Suppose an adversary has intercepted a target ciphertext ct encrypted using AES-CBC. Let ct_i be any non-IV block in ct. Let m_i be the associated message block. Show that if the adversary is able to submit ciphertexts to a CBC decryption oracle and learn whether the padding was valid or not, then it can learn the last byte of m_i *with probability 1* by making at most 512 queries. Here, the CBC decryption oracle only says whether the ciphertext was properly padded or not; it does *not* provide the output of the decryption if successful. Then, show how to extend your attack to recover *all* of m_i. **Hint:** Start by showing how to test whether the last byte of m_i is some value t by making 2 queries to the decryption oracle.

Remark: Are there settings where the server would repeatedly decrypt ciphertexts of the user's choosing? It turns out that when using IMAP (the protocol email clients use to fetch email) over TLS, the IMAP client will repeatedly send the user's password to the IMAP server to authenticate. With the above padding oracle (implemented using a "timing channel"), an adversary can recover the client's password in *less than* an hour! This problem shows that if a decryption failure occurs, the library should provide *minimal* information on the cause of the error. This type of "padding oracle" attack was the basis of the "Lucky 13" attack on TLS 1.0 (2013)—many years after they were first discovered (2002) and thought to be patched!

[1] In MAC-then-encrypt, the encryption algorithm first computes a MAC t on the message m, and the ciphertext is the encryption of the message-tag pair (m, t).

Problem 2: Coin Flipping over a Network [30 points]. Alice and Bob are deciding between two options A and B. Alice prefers A while Bob prefers B. To resolve the conflict, Alice and Bob decide to flip a fair coin. This is easy to do in person, but more challenging when Alice and Bob are communicating over a network. One approach is for Alice to flip a coin and announce the result to Bob, but then Alice might bias the coin flip in her favor. In this problem, we will develop the notion of a *cryptographic commitment* scheme and show how this enables coin flipping over a network.

A cryptographic commitment scheme is a digital analog of a "sealed envelope." Specifically, Alice can *commit* to a bit $b \in \{0,1\}$ and send the resulting commitment c to Bob (i.e., seal the bit in an envelope). The commitment c should not reveal anything about the committed bit b. At some subsequent point in time, Alice can *open* up the commitment and convince Bob that c is indeed a commitment to the bit b (i.e., open up the envelope and recover the original bit). The commitment scheme is *hiding* if c hides the bit b and is *binding* if the sender can only open the commitment to a single value $b \in \{0,1\}$. Let $G\colon \{0,1\}^\lambda \to \{0,1\}^{3\lambda}$ be a PRG. In this problem, we will consider the following protocol.

1. **Setup:** Bob starts by sampling $z \xleftarrow{R} \{0,1\}^{3\lambda}$ and sends z to Alice.
2. **Commit:** Alice samples a seed $s \xleftarrow{R} \{0,1\}^\lambda$. To commit to $b = 0$, Alice sends the commitment $c \leftarrow G(s)$ to Bob. To commit to $b = 1$, Alice sends the commitment $c \leftarrow G(s) \oplus z$ to Bob.
3. **Opening:** To open a commitment c to a bit $b \in \{0,1\}$, Alice sends (b, s) to Bob, where b is the bit, and s is the seed chosen by Alice. An opening (b, s) is valid for a commitment c and initial randomness z if $c = G(s)$ and $b = 0$, or $c = G(s) \oplus z$ and $b = 1$.

Show the following:

(a) We say a commitment scheme is *computationally hiding* if Bob (who is computationally bounded) cannot distinguish a commitment to the bit 0 from a commitment to the bit 1, except with negligible probability. Prove that if G is a secure PRG, then the above commitment scheme is computationally hiding.

(b) We say a commitment scheme is *statistically binding* if Alice (who may even be computationally unbounded) cannot output a commitment $c \in \{0,1\}^{3\lambda}$ together with valid openings $(0, s_0)$ and $(1, s_1)$, except with negligible probability (over the random choice of $z \xleftarrow{R} \{0,1\}^{3\lambda}$). Prove that the above commitment scheme is statistically binding.

(c) The binding property critically relies on the fact that $z \xleftarrow{R} \{0,1\}^{3\lambda}$ is sampled by Bob (i.e., outside the control of Alice). Suppose instead that we also allowed Alice to choose z. Show that Alice can efficiently come up with a string $z \in \{0,1\}^{3\lambda}$, a commitment $c \in \{0,1\}^{3\lambda}$, along with two *valid* openings $(0, s_0)$ and $(1, s_1)$. In this setting, we say that Alice's commitment is "equivocating."

(d) Show how the above commitment scheme allows Alice and Bob to implement a coin-flipping protocol over a network. Your protocol should consist of a maximum of *four* messages in total. Explain *informally* why in your protocol, neither Alice nor Bob can bias the outcome of the coin flip (say, by choosing their messages adversarially), except with negligible probability. You may assume that Alice and Bob are computationally bounded (i.e., runs in polynomial time). While you do *not* need a formal proof, your explanation should refer to the hiding and binding properties defined above.

(e) The binding property in this commitment scheme holds against *all* adversaries while our hiding property only holds against *efficient* adversaries. Prove that this is inherent: namely, *no commitment*

scheme can simultaneously be *statistically hiding* (i.e., the hiding property holds against all adversaries) and *statistically binding*. Note that your argument must apply to *all* commitment schemes, not just the one described in this problem.

Problem 3: Cryptographic Combiners [24 points]. Suppose we have two candidate constructions Π_1, Π_2 of a cryptographic primitive, but we are not sure which of them is secure. A cryptographic combiner provides a way to use Π_1 and Π_2 to obtain a new construction Π such that Π is secure if at least one of Π_1, Π_2 is secure (*without* needing to know which of Π_1 or Π_2 is secure). Combiners can be used to "hedge our bets" in the sense that a future compromise of one of Π_1 or Π_2 would not compromise the security of Π. In this problem, we will study candidate combiners for different cryptographic primitives.

(a) Let $G_1, G_2 \colon \{0,1\}^\lambda \to \{0,1\}^{3\lambda}$ be arbitrary PRG candidates. Define the function $G(s_1, s_2) := G_1(s_1) \oplus G_2(s_2)$. **Prove or disprove:** if at least one of G_1 or G_2 is a secure PRG, then G is a secure PRG.

(b) Let $H_1, H_2 \colon \{0,1\}^* \to \{0,1\}^\lambda$ be arbitrary collision-resistant hash function candidates. Define the function $H(x) := H_1(H_2(x))$. **Prove or disprove:** if at least one of H_1 or H_2 is collision-resistant, then H is collision-resistant.

(c) Let $(\mathsf{Sign}_1, \mathsf{Verify}_1)$ and $(\mathsf{Sign}_2, \mathsf{Verify}_2)$ be arbitrary MAC candidates[2]. Define $(\mathsf{Sign}, \mathsf{Verify})$ as follows:

- $\mathsf{Sign}((k_1, k_2), m)$: Output (t_1, t_2) where $t_1 \leftarrow \mathsf{Sign}_1(k_1, m)$ and $t_2 \leftarrow \mathsf{Sign}_2(k_2, m)$.
- $\mathsf{Verify}((k_1, k_2), (t_1, t_2))$: Output 1 if $\mathsf{Verify}_1(k_1, m, t_1) = 1 = \mathsf{Verify}_2(k_2, m, t_2)$ and 0 otherwise.

Prove or disprove: if at least one of $(\mathsf{Sign}_1, \mathsf{Verify}_1)$ or $(\mathsf{Sign}_2, \mathsf{Verify}_2)$ is a secure MAC, then $(\mathsf{Sign}, \mathsf{Verify})$ is a secure MAC.

Problem 4: Time Spent [1 point]. How long did you spend on this problem set? This is for calibration purposes, and the response you provide does not affect your score.

Optional Feedback. Please answer the following *optional* questions to help us design future problem sets. You do not need to answer these questions. However, we do encourage you to provide us feedback on how to improve the course experience.

(a) What was your favorite problem on this problem set? Why?

(b) What was your least favorite problem on this problem set? Why?

(c) Do you have any other feedback for this problem set?

(d) Do you have any other feedback on the course so far?

[2] Namely, you can assume that they are correct (but could be arbitrarily broken).