CS 346: Cryptography **Spring 2023**

Homework 2: Symmetric Cryptography

Due: February 20, 2023 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/sp23/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: Key Leakage in PRFs [20 points]. Let $F\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ be a secure PRF. Use F to construct a function $F'\colon \{0,1\}^{n+1} \times \{0,1\}^n \to \{0,1\}$ with the following two properties:

- F' is a secure PRF.
- If the adversary learns the last bit of the key, then F' is no longer secure.

(a) Describe your construction $F'\colon \{0,1\}^{n+1} \times \{0,1\}^n \to \{0,1\}$. **Hint:** Consider changing the value of F at a single point. Namely, consider a construction where

$$F'(k \| b, x) = \begin{cases} F(k,x) & x \neq 0^n \\ y & x = 0^n \end{cases}.$$

Think about how you would choose the value of $y \in \{0,1\}$ to satisfy the stated requirements.

(b) Show that if F is a secure PRF, then your construction F' is a secure PRF.

(c) Show that your construction F' is insecure against an adversary that learns the last bit of the key (i.e., at the beginning of the PRF security game, the challenger gives the last bit of the PRF key to the adversary). Specifically, give a complete description of your adversary and compute its advantage. This problem shows that leaking even a *single* bit of the secret key can break PRF security.

Problem 2: The BEAST Attack [14 points]. The TLS 1.0 protocol (used to protect web traffic) used AES-CBC encryption with a *predictable* IV. Specifically, instead of sampling a random IV, the TLS 1.0 implementation set the IV to be the last ciphertext block from the *previous* ciphertext (i.e., the IV for the $(i+1)^{\text{st}}$ message is the last ciphertext block of the i^{th} message).

(a) State two advantages for choosing the IV in this manner compared to choosing a random IV.

(b) Unfortunately, this variant of AES-CBC is insecure. Describe a CPA-adversary against this scheme that wins *with advantage 1* (include an explicit advantage calculation in your answer). You may assume that a random IV is used for the first ciphertext. **Hint:** It suffices to consider single-block messages. This exploit was the basis of the BEAST attack on TLS 1.0 (2011). The take-away is that simple (and seemingly benign) modifications to a cryptographic protocol can completely break security!

Problem 3: CBC Padding Oracle Attack [20 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!

Problem 4: Cryptographic Combiners [20 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 $H_1, H_2 \colon \{0,1\}^* \to \{0,1\}^\lambda$ be arbitrary collision-resistant hash function candidates. Define the function $H(x) := H_1(x) \| H_2(x)$. **Prove or disprove:** if at least one of H_1 or H_2 is collision-resistant, then H is collision-resistant.

(b) 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.

[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).

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

Problem 5: 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?