CS 6222: Introduction to Cryptography **Spring 2021**

Homework 2: Symmetric Cryptography

Due: March 10, 2021 at 5pm (Submit on Gradescope) **Instructor:** David Wu

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

https://www.cs.virginia.edu/dwu4/courses/sp21/static/homework.tex

You must submit your problem set via Gradescope. Please use course code **D55GP5** to sign up.

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 computing IDs of all of your collaborators with your submission. Refer to the official course policies for the full details.

Problem 1: Socially-Distanced Coin Flipping. [25 points]. Alice and Bob are choosing 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 social distancing and communicating over the phone. 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 "socially-distanced coin flipping".

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 $s \xleftarrow{R} \{0,1\}^\lambda$. 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 commitment randomness 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 no efficient adversary can 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 no adversary (including a computationally unbounded one) is able to produce a commitment c and valid openings $(0, s_0)$ and $(1, s_1)$, except with negligible probability (for a uniformly random z). Prove that the above commitment scheme is statistically binding. **Hint:** Use a union bound.

(c) Show how the above commitment scheme allows Alice and Bob to implement a socially-distanced coin flipping protocol. Your protocol should consist of a maximum of *four* messages in total. Explain 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. While you do *not* need a formal proof, your explanation should appeal to the hiding and binding properties defined above.

(d) 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 2: CBC Padding Oracle Attack [8 points]. Recall that when using a block cipher in CBC mode, the message must be an even multiple of the block size. When encrypting messages whose length is not an even multiple of the block size, the message must first be padded. In TLS, 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 ciphertext. In TLS, the record layer is secured using "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 ct_i is some value t by making 2 queries to the decryption oracle.

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 3: CBC-MAC [18 points]. Let $F\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ be a secure block cipher, and let $F_{\mathrm{CBC}}\colon \mathcal{K} \times (\{0,1\}^n)^{\le L} \to \{0,1\}^n$ be the *raw-CBC MAC* from lecture. In lecture, we said that raw-CBC is a secure PRF (and thus a secure MAC) for fixed-length messages (and more generally, *prefix-free messages*).

[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).

(a) Recall that raw-CBC uses a *fixed* IV (the all-zeroes string). Consider a *randomized* construction where the signing algorithm samples a random $\mathrm{IV} \xleftarrow{\mathrm{R}} \{0,1\}^n$ and computes the MAC on a message $m = (m_1, \dots, m_\ell) \in (\{0,1\}^n)^\ell$ as $t \leftarrow F_{\mathrm{CBC}}(k, (m_1 \oplus \mathrm{IV}, m_2, \dots, m_\ell))$. The tag is the pair (IV, t). Show that randomized raw-CBC is insecure, even for signing *fixed-length* messages.

(b) Suppose we apply the randomized construction from Part (a) to *encrypted CBC-MAC*; that is, the MAC on $m = (m_1, \dots, m_t) \in (\{0,1\}^n)^\ell$ is (IV, t) where $\mathrm{IV} \xleftarrow{\mathrm{R}} \{0,1\}^n$, $t \leftarrow F(k_2, F_{\mathrm{CBC}}(k_1, (m_1 \oplus \mathrm{IV}, m_2, \dots, m_\ell)))$, and k_1, k_2 are independent keys. Is this construction a secure MAC? Give either a proof or an attack.

(c) Suppose we use "encrypt-then-MAC" to construct an authenticated encryption scheme for a *fixed-length* message space $\{0,1\}^n$ (i.e., one-block messages) by combining randomized counter-mode encryption with raw-CBC MAC,[2] except we use the *same* key for both the encryption scheme and the MAC. Namely, an encryption of $m \in \{0,1\}^n$ consists of the tuple (IV, c, t) where $\mathrm{IV} \xleftarrow{\mathrm{R}} \{0,1\}^n$, $c \leftarrow F(k, \mathrm{IV}) \oplus m$, and $t \leftarrow F_{\mathrm{CBC}}(k, (\mathrm{IV}, c))$. Show that the resulting scheme is neither CPA-secure *nor* provides ciphertext integrity (i.e., construct two separate adversaries). *Remark:* This shows that reusing the same key for different cryptographic primitives can have severe consequences!

(d) Does the "encrypt-then-MAC" construction from Part (c) provide authenticated encryption for the *fixed-length* message space $\{0,1\}^n$ if we use independent and uniformly random keys for the randomized counter-mode encryption and raw-CBC MAC? Briefly justify your answer.

Problem 4: Hash-then-Encrypt [12 points]. The Android KeyStore uses "hash-then-CBC-encrypt" to construct an authenticated encryption scheme to generate and manage cryptographic keys for Android applications. Abstractly, the scheme operates as follows: Let $(\mathsf{Encrypt}_{\mathrm{CBC}}, \mathsf{Decrypt}_{\mathrm{CBC}})$ be a randomized CBC-mode encryption scheme built from a block cipher $F \colon \mathcal{K} \times \mathcal{X} \to \mathcal{X}$. Let $H \colon \mathcal{X}^{\leq L} \to \mathcal{X}$ be a collision-resistant hash function. Define the following candidate authenticated encryption scheme $(\mathsf{Encrypt}, \mathsf{Decrypt})$:

- $\mathsf{Encrypt}(k, m)$: Output $c \leftarrow \mathsf{Encrypt}_{\mathrm{CBC}}(k, H(m) \| m)$.
- $\mathsf{Decrypt}(k, c)$: Compute $(t, m) \leftarrow \mathsf{Decrypt}_{\mathrm{CBC}}(k, c)$ and output m if $t = H(m)$ and $\perp$ otherwise.

In the following, assume that $\mathcal{X} = \{0,1\}^n$ and $L \geq 2$.

(a) Show that $(\mathsf{Encrypt}, \mathsf{Decrypt})$ does not provide ciphertext integrity.

(b) Show that $(\mathsf{Encrypt}, \mathsf{Decrypt})$ is not CCA-secure. Recall that for encryption schemes over a variable-length message space, the adversary can only query the encryption oracle on pairs (m_0, m_1) where m_0 and m_1 have the *same* length.

(c) Would the above problems go away if the Android KeyStore had used randomized counter mode encryption instead of CBC-mode encryption? Give a brief explanation.

Both attacks show that the Android KeyStore does not provide authenticated encryption. These attacks were discovered in January 2016 and Google has confirmed that the encryption scheme will be removed from the system.

[2] A variant where we combine counter-mode encryption with *encrypted CBC-MAC* yields the CCM mode of operation—which provides authenticated encryption.

Problem 5: Authenticated Encryption [12 points]. Let $(\mathsf{Encrypt}, \mathsf{Decrypt})$ be a symmetric authenticated encryption scheme. For each of the following constructions $(\mathsf{Encrypt}', \mathsf{Decrypt}')$, state whether they are authenticated encryption schemes. If so, give a proof (you need to show *both* CPA-security and ciphertext integrity); otherwise, give an attack.

(a) Define $(\mathsf{Encrypt}', \mathsf{Decrypt}')$ as follows:

$$\mathsf{Encrypt}'(k, m) = \big(\mathsf{Encrypt}(k, m), \mathsf{Encrypt}(k, m)\big)$$

$$\mathsf{Decrypt}'(k, (c_1, c_2)) = \begin{cases} \mathsf{Decrypt}(k, c_1) & \mathsf{Decrypt}(k, c_1) = \mathsf{Decrypt}(k, c_2) \\ \perp & \text{otherwise.} \end{cases}$$

(b) Define $(\mathsf{Encrypt}', \mathsf{Decrypt}')$ as follows:

$$\mathsf{Encrypt}'(k, m) = \big(c, c\big) \text{ where } c \leftarrow \mathsf{Encrypt}(k, m)$$

$$\mathsf{Decrypt}'(k, (c_1, c_2)) = \begin{cases} \mathsf{Decrypt}(k, c_1) & c_1 = c_2 \\ \perp & \text{otherwise.} \end{cases}$$

Problem 6: Time Spent [3 extra credit points]. 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?