CS 6222: Introduction to Cryptography **Spring 2020**

Homework 1: Symmetric Cryptography

Due: February 12, 2020 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/sp20/static/homework.tex`

You must submit your problem set via Gradescope. Please use course code **MB84NW** 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.

Acknowledgments. These problems are taken/adapted from problems in Boneh-Shoup.

Problem 1: Pseudorandom Generators [16 points]. Let $G \colon \{0,1\}^\lambda \to \{0,1\}^n$ be a secure PRG. For each of the following functions G', indicate whether it is a secure PRG or not. If it is secure, give a *formal* proof; if not, describe an explicit attack.

(a) $G'(s) := G(s) \| (G(s) \oplus 1^n)$, where 1^n denotes the all-ones string of length n.

(b) $G'(s_1 \| s_2) := G(s_1) \oplus G(s_2)$.

(c) $G'(s_1 \| s_2) := s_1 \| G(s_2)$.

Please refer to this handout for examples of how to formally show whether a construction is secure or not.

Problem 2: Key Leakage in PRFs [8 points]. Let $F \colon \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}$ be a secure PRF. Use F to construct a function $F' \colon \{0,1\}^{\lambda+1} \times \{0,1\}^\lambda \to \{0,1\}$ 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.

You should (a) prove that F' is a secure PRF; and (b) describe an attack (and compute the advantage) when the adversary knows the last bit of the PRF key. **Hint:** Try changing the value of F at a single point. This problem shows that leaking even a *single* bit of the secret key can break PRF security.

Problem 3: Encrypting Twice? [8 points]. Intuitively, encrypting a message twice should not harm security. It turns out that this is not always true. Let $(\mathsf{Encrypt}, \mathsf{Decrypt})$ be a cipher and define the "encrypt-twice" cipher $(\mathsf{Encrypt}_2, \mathsf{Decrypt}_2)$ where $\mathsf{Encrypt}_2(k, m) := \mathsf{Encrypt}(k, \mathsf{Encrypt}(k, m))$.

(a) Give an example of a cipher $(\mathsf{Encrypt}, \mathsf{Decrypt})$ that is semantically secure, but $(\mathsf{Encrypt}_2, \mathsf{Decrypt}_2)$ is not semantically secure.

(b) Suppose $(\mathsf{Encrypt}, \mathsf{Decrypt})$ is CPA-secure. Prove that $(\mathsf{Encrypt}_2, \mathsf{Decrypt}_2)$ is also CPA-secure.

Problem 4: Digital Rights Management [20 points]. The Content Scrambling System (CSS) used for encrypting DVDs uses a "hybrid" encryption scheme designed as follows. First, every authorized DVD distributor is given a device key k that it embeds in any DVD player it sells. We assume that there are a small number n of authorized DVD player manufacturers with device keys $k_1, \dots, k_n$. To encrypt a movie m, one first generates a "payload" key k' that is used to encrypt the movie. The payload key k' is then encrypted with each of the device keys $k_1, \dots, k_n$ to form the header of the DVD. The overall encrypted DVD then looks like the following:

$$\underbrace{\mathsf{Encrypt}(k_1, k') \parallel \cdots \parallel \mathsf{Encrypt}(k_n, k')}_{\text{header}} \parallel \underbrace{\mathsf{Encrypt}(k', m)}_{\text{payload}}.$$

(a) Show how a device with key k_i is able to decrypt the payload.

(b) Suppose that the underlying encryption scheme $(\mathsf{Encrypt}, \mathsf{Decrypt})$ is CPA-secure. Give an *informal* argument why the above hybrid encryption scheme is also CPA-secure (namely, the keys of the hybrid encryption scheme is defined to be the tuple $(k_1, \dots, k_n)$ and the hybrid encryption algorithm samples a fresh key k' to encrypt the payload). *Formalizing this argument will rely on a technique called a hybrid argument, which we have not discussed yet.*

As we saw in lecture, the core cryptographic design of CSS was insecure, and not long after its deployment, hackers were able to extract a key from an authorized device and use it to build a "pirate player" (e.g., DeCSS). Unfortunately, because each key was shared across all devices by the particular manufacturer, disabling the compromised key would simultaneously disable a large number of *honest* devices. This design flaw in CSS demonstrates that relying on shared global keys is *not* a good idea.

Subsequently, when HD-DVD and Blu-Ray came out, the movie industry introduced the Advanced Access Content System (AACS) scheme which associated a random *device-specific* key with each device. The idea is as follows: suppose there are at most $n = 2^{64}$ devices, each with a unique 64-bit serial number $i \in [0, 2^{64} - 1]$. These n devices are associated with the leaves of a binary tree $\mathcal{T}$ of height $\log n$. Every node $j \in \mathcal{T}$ is associated with an encryption key k_j. For an index $i \in [0, 2^{64} - 1]$, let S_i be the keys associated with the nodes from the root node to the i^{th} leaf node in $\mathcal{T}$. The manufacturer embeds S_i in device i (i.e., every device has exactly $\log n + 1$ keys). Movies are encrypted using a similar hybrid encryption scheme as before. Namely, a payload key k' is first sampled to encrypt the movie. The payload key k' is then encrypted using the key k_{root} associated with the root of $\mathcal{T}$:

$$\underbrace{\mathsf{Encrypt}(k_{\mathsf{root}}, k')}_{\text{header}} \parallel \underbrace{\mathsf{Encrypt}(k', m)}_{\text{payload}}$$

(c) Suppose hackers compromise device $t \in [0, 2^{64} - 1]$ and use its keys to build a pirate player. Show that the movie distributor can still encrypt the movie with a header of size $O(\log n)$ such that device t can no longer decrypt, but every device $i \neq t$ is still able to decrypt. Formally prove the correctness of your construction.

(d) Suppose that hackers have now compromised a set of devices $T \subseteq [0, 2^{64} - 1]$. Show that using a header of size $O(|T| \log n)$, the movie distributor can encrypt a movie such that every device $i \notin T$ can decrypt, but every device $i \in T$ cannot. Formally prove the correctness of your construction. This is an example of a *revocation* scheme.

(e) The tree $\mathcal{T}$ contains $2^{65} - 1$ nodes. Storing a random encryption key for each node in the key will require an exorbitant amount of space. Describe a way to compress the tree $\mathcal{T}$ to use at most 100 bytes of storage. Explain (informally) why your design does not compromise security.

While the AACS scheme addressed some of the limitations of CSS, the key revocation process was very time-intensive. It was quickly discovered that hackers could extract new device keys *significantly* faster than the movie industry could revoke them!

Problem 5. The BEAST Attack [10 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 6. CBC Padding Oracle Attack [10 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*

[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).

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 7: 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 [0 points]. 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?