
Problem Set 2

This problem set is due on *Monday, March 11, 2019* 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 PDF*. When submitting the problem in Gradescope, ensure that **all your group members are listed on Gradescope**, and not in the PDF alone.

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-tas@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 pdf. 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 L^AT_EX 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.

Our department is collecting statistics on how much time students are spending on psets, etc. For each problem, please give your estimate of the number of person-hours your team spent on that problem.

Problem 2-1. Side-Channel attack on AES

An attacker can use "side-channel" information to obtain a secret AES key.

We set up a server to simulate this type of side-channel attack. Our server encrypts random strings using AES with a secret key k . Let S be the number of ones in the XOR outputs during each encryption (i.e. counting only the outputs of the XOR's when some round key is XOR-ed with the current state). The server "leaks" $\lfloor \frac{S}{4} \rfloor$ as a side-channel.

If you query <http://6857-aes.csail.mit.edu/?num=1000> you will receive `num` (16 bytes (128 bits) of plaintext, 16 bytes of ciphertext, XOR-output one-count / 4 ($\lfloor \frac{S}{4} \rfloor$ as above)) triples. In this case, `num` = 1000, but feel free to specify any `num` $\in [1 \dots 10000]$ (it might take a bit to load the triples for a large `num`). Also, feel free to query the server multiple times if you need more triples, because the plaintexts are randomly generated so you will get new data.

The plaintext and ciphertext are printed as space-separated byte values in decimal; and the plaintext is separated from the ciphertext by a comma. The XOR-output one-count is an integer between 0 and $11 \cdot 16 \cdot 8 / 4 = 352$. (There are 128-bits in the secret key and 10 rounds each having its own round key, so there are 11 XOR's of a round key to the state, each of which XOR's 16 8-bit bytes, and finally we are given this number divided by 4) This count is followed by a semicolon.

You may find useful the NIST AES specification in this problem: <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.

We have provided our server implementation in the files `server.py` and `aes.py`. The `aes.py` code contains the side-channel leak. Feel free to play with it on your local machine. However, unfortunately, it requires Python 2.7. It is not necessary to solve the problem, and is just there for reference.

- (a) Let X be a random variable that is the total number of heads in t independent coin-flips of a fair coin, and let Y be another independent random variable that counts the number of heads in t coin flips, and then adds one. Suppose t is known.
Suppose Z is either $\lfloor \frac{X}{4} \rfloor$ or $\lfloor \frac{Y}{4} \rfloor$, but you don't know which. How many draws of Z do you expect to need, in order to determine with reasonable reliability whether Z is $\lfloor \frac{X}{4} \rfloor$ or $\lfloor \frac{Y}{4} \rfloor$? (Note: an approximate bound should be good enough here, just make sure to state your assumptions.)
- (b) Describe an algorithm for recovering the AES key k given the AES side-channel information described above. (Hint: try using the result of part (a) on the first bit of the first round key. What is t ?)
- (c) Recover the secret key k . Submit any code you used.
- (d) How many (plaintext, ciphertext, $\lfloor \frac{S}{4} \rfloor$) triples did your attack require? (Note that even when your attack seems to recover almost all the keybits correctly, it may still get one or two key bits wrong, which will result in an incorrect decryption of a ciphertext. Please report the number of plaintexts for which your algorithm has a good chance of outputting all key bits correctly.)

Problem 2-2. Hash Functions

Let's review some hash function properties and prove some useful facts about them.

Specifically, let's consider a hash function $h: \{0, 1\}^{2d} \rightarrow \{0, 1\}^d$.

Recall that a hash function h is one-way if you are given a $y = h(x)$ for a random $x \in \{0, 1\}^{2d}$, it is computationally infeasible to find any x such that $y = h(x')$.

- (a) Show that targeted-collision resistance implies one-wayness.

Hint: Recall that $(P \implies Q) \iff (\neg Q \implies \neg P)$.

Recall that if a hash function is *non-malleable* then given $y = h(x)$ for a random $x \in \{0, 1\}^{2d}$, it is computationally infeasible to find a $y' = h(x' + \delta)$ for $\delta \in [-B, B]$, where B is small, such that $h(x') = y$.

- (b) Show that if h is non-malleable, then it is also one-way.

Now, let's look at how these properties translate into practice.

Ben Bitdiddle is trying to design a secure method of storing passwords for the users of his website. He decides to store the passwords as a list of hashes. When a user enters their password, it is hashed locally and the hash is then sent to the server, where it is checked against the entry in Ben's database.

Assume that the users of Ben's website select passwords uniformly at random from a set S of passwords, and assume that Ben uses a hash function that can be modeled as a random oracle with output space much larger than $|S|$.

- (c) Suppose an adversary steals the list of n password hashes. If S is known to the adversary, analyze the expected number of tries it will take for the adversary to find any one of the user's password.

Now suppose that, instead of just storing the hash of the password, Ben samples a random string r_p for each new password p in the database. The database entry for a password is $(h(p \parallel r_p), r_p)$. When a user enters their password, the server sends r_p and the user computes $h(p \parallel r_p)$ locally. This value is then returned to the server and compared against the database.

You may assume that r_p is unique for each password, and each p is uniformly sampled from S , as in the previous part.

- (d) Suppose an adversary steals the list of hashes and random strings. If S is known to the adversary, analyze the expected number of tries it will take for the adversary to find any one of the user's password.

Problem 2-3. Security of Machine Learning

In this problem we will explore several recent security concerns about machine learning models within the context of self-driving cars. Assume that Waymo uses deep learning as an integral component of vision for their cars: for example, to classify objects on the road such as a stop sign or a traffic light. Suppose that an adversary wishes to attack the functionality of Waymo's deep learning model such that it will misclassify one stop sign on Vassar Street for a red traffic light (meaning that the car will wait indefinitely at this stop sign as it waits for the nonexistent red traffic light to change). Assume that the adversary can also physically tamper with the stop sign, but only so much that it is not fixed by Cambridge Police or pedestrians (i.e. not very noticeable to humans).

- (a) Suppose that the adversary has access to Waymo's full training image dataset for their deep learning vision model, prior to training. For this question, the adversary is allowed to tamper with this dataset as long as Waymo does not notice that a tampering occurred (i.e. still high performance on training and validation sets). Explain how the adversary can tamper with the data (in an unnoticeable manner) to achieve the goal stated above.
- (b) Wary of adversaries, Waymo decides to verify their training set (however, the dataset is still public) so they can be sure their data is not tampered with. Explain why this does still not prevent an adversary from achieving the goal.

Indeed, one major concern for state-of-the-art neural networks is the use of *adversarial examples* - specifically chosen inputs by an adversarial algorithm that are designed to make the model output an incorrect classification with high confidence, by adding in small (human imperceptible) perturbations. See https://gradientscience.org/intro_adversarial/ or <https://blog.openai.com/adversarial-example-research/> for more details. Developing secure models that are resistant to these attacks is an active area of research. The rest of this problem will be concerned with the following paper: "Towards Deep Learning Models Resistant to Adversarial Attacks" (Madry et al., 2017). Read the paper (focusing on the high level details - we will not get too involved in the math/optimization) here at <https://arxiv.org/pdf/1706.06083.pdf>, and answer the following questions.

- (c) What is an adversarially robust classifier? Describe the threat model of the attack for which the paper aims to defend against. What does the adversary have access to and what are the possibilities for adversarial examples? (This reference may help, which Madry et al.'s paper refers to: <https://arxiv.org/pdf/1412.6572.pdf>).
- (d) The paper uses a natural min-max (saddle point) formulation to capture this notion of security against adversarial attacks. This allows the definition of a security guarantee against a class of adversarial attacks. How does this definition of security differ from that of the cryptographic security we have studied so far, and how is it similar? (Hint: many of the cryptographic systems we study rely on the idea of *computational hardness*).