Homework 7

This problem set is new this year; please let your TAs know about any bugs, typos, or autograder issues, and also let us know if you have any feedback (positive or negative) about this assignment!

We've provided starter code for you to use; after reading this PDF, fill in the missing functions in `ps7_submission.py`, and turn in your code to the Gradescope assignment "Problem Set 7".

Note: For this assignment, you'll need to install the `ecdsa` Python library (via `pip install --user ecdsa`, or just `pip install ecdsa` if you're using a virtual environment). You will also need to download an extra file, `ps7_auxdata`, and place it in the directory you run `ps7_submission.py` from.

The due date is Friday, March 14, at 3:30pm.

In this problem set, you'll first review how ECDSA signatures work.

Then you'll implement an attack that breaks ECDSA if you can observe a single signature with a known nonce.

Finally, you'll implement an attack that breaks ECDSA if a nonce is ever repeated in two different signatures.

Problem 1 [10 points] Warm-up. We've given you an ECDSA secret key. Use it to produce a valid ECDSA signature of a given message. (You don't need to implement elliptic curve point addition or scalar multiplication yourself, you can use the library for that.)

You'll probably want to refer to the lecture slides on ECDSA to remind yourself how ECDSA signing and verification work.

Problem 2 [40 points] ECDSA with known nonce. You're the proud owner of a BobDevice™, the popular product made by BobCorp. You've just found out that your BobDevice secretly sells all your chat logs to advertisers and posts all your sensitive medical data to social media. You'd like to flash some custom firmware onto your device that will let you disable this.

Unfortunately, BobCorp signs all firmware updates with ECDSA, and BobDevices will reject any firmware updates whose signatures don't verify under BobCorp's public key.

Fortunately, you've learned that BobCorp used a buggy ECDSA implementation to sign their latest firmware update, and the nonce k that was used during signing has a known value.

Your goal is to forge a signature for the custom firmware; for your BobDevice to accept it, it will need to properly verify under BobCorp's public key. You have access to BobCorp's public key, the buggy signature, and the firmware update the buggy signature is for. (Your solution might not need all of these values.)

Note that the security notion used for signatures, namely UF-CMA, would allow you to ask to see signatures on arbitrary messages of your choice, and your goal would be to forge a signature on any message you haven't asked BobCorp to sign. (It's similar to the UF-CMA game for MACs.) In this problem, you are an adversary with much more limited power: you're getting to see a signature on one message you didn't choose. You also have a more difficult goal than in UF-CMA: you're trying to forge a signature on a specific message, not an arbitrary message of your choice.

By carrying out this attack, you're showing that ECDSA with known nonces is not UF-CMA secure, and in fact that ECDSA loses all security if the attacker knows the nonce used for even a single signature. More importantly for you, you can now run the custom firmware on your BobDevice.

Problem 3 [50 points] ECDSA with repeated nonces. The AliceAppliance (made by Alice Inc.) also does unpleasant things with your data, and it also requires ECDSA signatures on firmware updates. The ECDSA implementation Alice Inc. uses for signing firmware doesn't use a fixed nonce, but it does use a buggy RNG for generating the nonces. Because of this, two of Alice Inc.'s last 100 signatures used the same nonce as each other. (You aren't sure which two.)

Your task is again to forge a signature on the custom firmware that verifies under Alice Inc.'s public key. (Hint: start by figuring out which two signatures use the same nonce.)

The file `ps7_auxdata` (linked to on the course website) contains Alice Inc.'s public key and last 100 *(message, signature)* pairs. You should download the file and place it in whatever directory you run `ps7_submission.py` from.

By carrying out this attack, you're showing that ECDSA loses all security if a nonce is ever reused in two different signatures.
