You will need 50% of all homework points to qualify for the exam. (That is, if you get at least 50%, your final grade will be the exam grade. And if you do not get 50%, you do not pass the course.)

Please hand in your solutions by email. Either scan a handwritten solution or typeset your solution readably. I do not consider ASCII formulas readable.

When submitting, indicate your name and your matriculation number. On your first submission, please also indicate a password, this password will be needed for accessing the solutions and your points online.

The total number of points for each homework is 20 (not including points for bonus problems, if available).

For submitting your solution in a nicely typeset way (e.g., using LaTeX), you get up to 3 bonus points, but not more than 30% of the points you reached for content.

Problem 1 Security definitions

Your task is to write a security definition in Python (or another language, but we provide a template in Python). The goal of this is to give you a better understanding what security definitions mean, besides just being formulas.

We illustrate this by writing the security definition of PRGs in Python as can be seen in the file prg.py is posted on the website.

Here \( G \) is an implementation of a pseudo-random generator (a rather bad one). And \( \text{prg\_game} \) is a function that implements the game from the security definition of PRGs. That is, it takes a PRG \( G \), and an adversary \( \text{adv} \), and calls \( \text{adv} \) either with randomness or the output of \( G \). If the adversary guesses correctly which of the two was the case, \( \text{prg\_game} \) returns \text{True}, else \text{False}.

The function \( \text{test\_prg} \) tries out whether a given adversary is successful or not by counting how often he guesses right. (Of course, this does not replace a proof: a statistic does not give certainty, and also we cannot know whether other adversaries are successful. But it illustrates the use of the security definition.)

We have also written an example adversary \( \text{adv} \) that breaks the PRG \( G \). For simplicity, let both the message and the key space consist of 32-bit integers. But note that you are not supposed to use brute force attacks.

Your task:

(a) Write the security definition for IND-OT-CPA as a Python program. (Recall, in IND-OT-CPA, the adversary is called twice, so you will need two functions \( \text{adv1} \) and \( \text{adv2} \). Also pay attention to the following: the adversary should not be allowed to output messages that are not in the message space.)

(b) Write an adversary that breaks the encryption scheme \( \text{enc} \) defined in the source code below. (This adversary should have a success probability, as measured by \( \text{test\_indotcpa} \) of at least 0.95.)

A template for your solution will be included on the page with the name ind-ot-cpa.py. You need to fill in code where there are ???.

A template for your solution will be included on the page with the name ind-ot-cpa.py. You need to fill in code where there are ???.