# Security in Software Applications 2025/2026
# **Project**

The project is divided into four parts. You need to:

- Complete at least the first part of the assignment.
- Provide a report in pdf describing:
  - In a few rows how the tool you are using works, i.e. describe what it means to do fuzzing, and how Echidna enforces the properties required by the developer.
  - The meaning of the code you have written to check the properties, i.e. describe the meaning of the code used to describe your properties.
  - Include screenshots of the output of the tool before and after you found and corrected the issue.

  The report should not be too long. No more than 10 pages.

  **NOTE**: I should be able to fully understand the meaning of your code without inspecting the code or installing the tool. A missing report automatically leads to assignment failure.

You must use Echidna as a property-based fuzzing tool: `https://github.com/crytic/echidna`.

I recommend a pass of the code into Remix ide (`https://remix.ethereum.org/`) to fix any error or warning triggered by the Remix static analyzer.

Points below will be summed to the final grade.

## Part 1 (max 3 points)

The contract Taxpayer.sol records information about individuals. The contract should require the following property, easy to overlook: if person x is married to person y, then person y should of course also be married and to person x. Your assignment is to:

1. add *invariants* to the code to express the properties above using one of the proposed tools, plus any other sensible properties you can think of;
2. use Echidna to detect possible violations;
3. Fix the complaints by
   (a) correcting the code, or
   (b) adding a (sensible) precondition for the method, using require("condition"), or
   (c) adding an invariant for another property that can help in the verification.

### Hints

- It is easiest to introduce one invariant at a time, then fix the errors detected and only then move on to the next one.
- If the tool produces some warnings, i.e., if the tool thinks that some property is not satisfied by the code, this can have several causes:
  - There is an error in the code. For example, an assignment to some field may be missing.
  - There is an error in the specification. For example, maybe you have considered an "and" in a specification where you meant "or", maybe you have written age $> 18$ when you meant age $\geq 18$ in some constraints.
  - There may be some properties missing but needed by the tool for the verification. Note that the tool does not know any of the things that may be obvious to you – for example, that if some person x is married to y then y is also married to x – and such properties may be needed for verification.

To stop the tool from complaining, you can:

- correct the contract code; for the exercise here, this should only involve adding some simple assignments to the offending method;
- Correct the specification.

## Part 2 (max 2 point)

The tax system uses an income tax allowance (deduction):

1. Every person has an income tax allowance on which no tax is paid. There is a default tax allowance of 5000 per individual, and only income above this amount is taxed.
2. Married persons can pool their tax allowance as long as the sum of their tax allowances remains the same. For example, the wife could have a tax allowance of 9000, and the husband a tax allowance of 1000. Add invariants that express these constraints, and, if necessary, fix/improve the code to ensure that they are not violated.

## Part 3 (max 1 points)

The new government introduced a measure that people aged 65 and over have a higher tax allowance, of 7000. The rules for pooling tax allowances remain the same. Add invariants that express these constraints, and if necessary fix/improve the code to ensure that they are not violated.

## Extra - Part 4 (max 1 points)

The government established a periodic lottery system for users under the age of 65. Such lottery is managed inside the contract Lottery.sol.

The winner can obtain a sale on the tax allowance, i.e. 7000 instead of 5000. The lottery system is based on a commit-and reveal protocol where:

- The user, upon participation, send the commitment of a user-defined input `x`, i.e. $y =$ `keccak256(x)`.
- Then, after a time-out elapses, each user $i$ reveals his own $x_i$.

- Let $n$ be the total number of participants. If `keccak256`$(x_i) = y_i$ for each $i \in [n]$, then the contract computes the winner as $j = (x_1 + \ldots + x_n)\%n$.
- Finally, the tax allowance of the $j$-th participant is changed to 7000.

Add invariants for avoiding crashes, e.g. participants are valid taxpayers and add invariants to enforce honest behavior of the participants, e.g. the same participant cannot participate multiple times, a participant cannot find a sequence commit/reveal that biases its winning probability, etc.

To have the guarantee that the lottery is unbiased, the number of times each participant wins the lottery should be roughly the same (given a large number of played lotteries with the same participants).

Notice that `keccak256` is a variant of SHA, producing random-like outputs. Since any random input on the commit function would not pass the tests with overwhelming probability, you shall craft input to the commit function by first computing `keccak256` on the echidna testing input as done by the function `joinLottery` in Taxpayer.sol.

To check if an address is associated with the correct contract, you can invoke the function `isContract`(), as done by `setTaxAllowance` in Taxpayer.sol.