Homework 4

You will need to use python to generate the test cases, but the goal is to write solidity code that leverages the precompiles to accomplish the following:

Problem 1: Rational numbers

We're going to do zero knowledge addition again.

Claim: "I know two rational numbers that add up to num/den"

Proof: ([A], [B], num, den)

```
struct ECPoint { uint256 x; uint256 y; } function rationalAdd(ECPoint calldata A, ECPoint calldata B, uint256 num, uint256 den) public view returns (bool verified) { // return true if the prover knows two numbers that add up to num/den }
```

Solidity/EVM has two functions you may find handy: `mulmod` (which does multiplication modulo p) and the precompile `modExp` which does modular exponentiation.

Although `modExp` does not let you raise to the power of -1, you can accomplish the same thing by raising a number to `curve_order - 2`.

The following identity will be handy:

```
pow(a, -1, curve_order) == pow(a, curve_order - 2, curve_order)
```

(This is Fermat's little theorem, you can ask a chatbot AI to further explain this, but it isn't necessary to understand this)

To accomplish `pow` the precompile `modExp` may be handy.

Problem 2: Matrix Multiplication

There is no claim statement here, just execute the math on chain.

Your contract should implement matrix multiplication of an n x n matrix (**M**) of uint256 and a n x 1 vector of points (**s**). It validates the claim that matrix **Ms = o** where o is a n x 1 matrix of uint256.

You will need to multiply **o** by the generator on-chain so that both sides have the same type.

```
struct ECPoint { uint256 x; uint256 y; } function matmul(uint256[] calldata matrix, uint256 n, // n x n for the matrix ECPoint[] calldata s, // n elements uint256[] calldata o // n elements ) public returns (bool verified) { // revert if dimensions don't make sense or the matrices are empty // return true if Ms == o elementwise. You need to do n equality checks. If you're lazy, you can hardcode n to 3, but it is suggested that you do this with a for loop }
```

Example

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} P \\ Q \\ R \end{bmatrix} = \begin{bmatrix} P + 2Q + 3R \\ 4P + 5Q + 6R \\ 7P + 8Q + 9R \end{bmatrix} \stackrel{?}{=} \begin{bmatrix} o_1 G \\ o_2 G \\ o_3 G \end{bmatrix}$$