Question 1

A secret is 2-out-of-3 shared among Alice (party 1), Bob (party 2), and Charlie (party 3) via Shamir's scheme. The modulus is $q = 5$.

(a) Bob's share is 4. Charlie's share is 2. What is the secret?

Solution: All the arithmetic is modulo 5. The shares are the values $\ell(2) = 4$ and $\ell(3) = 2$ on a line $\ell(x) = s + a \cdot x$. The secret is $\ell(0) = s$. To recover it we solve $s + 2a = 4$ And $s + 3a = 2$, from where $s = 3(s + 2a) - 2(s + 3a) = 3 \cdot 4 - 2 \cdot 2 = 3$.

(b) Alice, Bob, and Charlie want to "rerandomize" their shares without changing the secret. Alice replaces her old share with a random number r modulo 5. Explain how Bob and Charlie should recompute their new shares with Alice's assistance.

Solution: From part (a) a is also 3, so Alice's share is $\ell(1) = 3 + 1 \cdot 3 = 1$. Both the old shares ℓ and new shares ℓ' must satisfy the interpolation identity $\ell(0) = 2\ell(1) - \ell(2)$, so $2\ell(1) - \ell(2) = 2\ell'(1) - \ell'(2)$. Therefore $\ell'(2) - \ell(2) = 2(\ell'(1) - \ell(1)) = 2(r - 1)$. Alice sends the value $2(r - 1)$ to Bob and he adds it to his old share.

Similarly, the secret is reconstructed from Bob's and Charlie's shares as $\ell(0) = -\ell(1) + 2\ell(3))$. By the same argument $\ell'(3) - \ell(3) = 3(\ell'(1) - \ell(1)) = 3(r - 1)$. Alice sends $3(r - 1)$ to Charlie and he adds it to his old share.

(c) Alice and Charlie want to subtract 1 from the secret without talking to Bob. How should they modify their shares?

Solution: Suppose Alice and Charlie could share the "secret" 1 by a line ℓ_1 while ensuring that Bob's share $\ell_1(1)$ is a zero. Then $\ell - \ell_1$ would be shares of $\ell(0) - \ell_1(0) = \ell(0) - 1$, namely the old secret minus one. ℓ_1 is the unique line that interpolates through $\ell_1(0) = 1$ and $\ell_1(2) = 0$. This is the line $\ell_1(t) = 1 + 2t$. So Alice should subtract $\ell_1(1) = 3$ from her share and Bob should subtract $\ell_1(3) = 2$ from his. Their new shares are 2 and 4.

Here is another way of arriving at the same solution. Their initial shares are $a = s + r$, $b = s + 2r$, $c = s + 3r$. The new shares should be of the form $a' = s - 1 + r'$, $b' = s - 1 + 2r'$, and $c' = s' - 1 + 3r'$. As Bob's share should remain the same, b and b' should be equal, namely $b' = b$, or $2r' - 1 = 2r$, or $r' - r = 3$. Solving for a' and c' in terms of a and c we obtain $a' = (a - 1) + (r' - r) = a + 2$ and $c' = (c - 1) + 3(r' - r) = c + 3$.

Question 2

In this question you will investigate circuits for the millionaires' problem (the argmax function).

(a) Design arithmetic circuits (with plus and times gates) for the functions NOT x, x_1 AND x_2, and x_1 OR x_2. The gates operate modulo q (for q prime). The circuits should produce the correct output when the inputs x, x_1, x_2 take the values 0 (false) and 1 (true).

Solution: The circuit for NOT x is $1 - x$. The circuit for x_1 AND x_2 is $x_1 \cdot x_2$. As x_1 OR x_2 = NOT (NOT x_1 AND NOT x_2), the circuit for x_1 OR x_2 is $1 - (1 - x_1)(1 - x_2)$.

(b) Design a Boolean circuit (with NOT, AND, OR gates) for the less than or equal function

$$\mathrm{leq}(x, y) = \begin{cases} 1, & \text{if } x \le y, \\ 0, & \text{if } x > y. \end{cases}$$

Assume the numbers x and y are provided in bit representation as n-bit strings. What is the size of circuit in big-theta notation?

Solution: Let's write $x = ax'$ and $y = by'$ with a, b being their most significant bits. x is at most y if $a = 0$ and $b = 1$, or if they are the same and $x' \leq y'$. In boolean logic this says

$$\text{leq}(x, y) = (\text{NOT } a \text{ AND } b) \text{ OR } (((a \text{ AND } b) \text{ OR } (\text{NOT } a \text{ AND NOT } b) \text{ AND } \text{leq}(x', y'))$$

This formula provides a recursive construction of a circuit for n-bit leq from a circuit for $(n-1)$-bit leq with nine extra gates. The recurrence for size is therefore $S(n) = S(n-1) + 9$. The base case is $S(0) = 0$ (leq $= 1$ when $n = 0$). It solves to $S(n) = 9n$, so the size is $\Theta(n)$.

(c) Use parts (a) and (b) to design an arithmetic circuit for the function

$$\text{argmax}(x, y, z) = \begin{cases} \mathtt{100}, & \text{if } x > y \text{ and } x > z, \\ \mathtt{010}, & \text{if } y > x \text{ and } y > z, \\ \mathtt{001}, & \text{if } z > x \text{ and } z > y. \end{cases}$$

Your circuit may output anything in case some two inputs are equal. What is its size in big-theta notation?

Solution: The first output is NOT $\text{leq}(x, y)$ OR NOT $\text{leq}(x, z)$. The second and the third are specified analogously. This circuit uses six copies of leq and nine extra gates. Its size is also $\Theta(n)$.

Question 3

The code $\mathcal{C}$ consists of all 7 bit strings $\mathbf{x} = x_1x_2x_3x_4x_5x_6x_7$ that satisfy these three constraints modulo two:

$$\begin{aligned} x_4 + x_5 + x_6 + x_7 &= 0 \quad \text{and} \\ x_2 + x_3 + x_6 + x_7 &= 0 \quad \text{and} \\ x_1 + x_3 + x_5 + x_7 &= 0. \end{aligned} \tag{1}$$

If the index 1 to 7 is written out in binary representation, each constraint corresponds to a bit of the index, and it involves only those indices for which this bit is set to 1:

$$\begin{aligned} x_{\mathtt{\underline{1}00}} + x_{\mathtt{\underline{1}01}} + x_{\mathtt{\underline{1}10}} + x_{\mathtt{\underline{1}11}} &= 0 \\ x_{\mathtt{0\underline{1}0}} + x_{\mathtt{0\underline{1}1}} + x_{\mathtt{1\underline{1}0}} + x_{\mathtt{1\underline{1}1}} &= 0 \\ x_{\mathtt{00\underline{1}}} + x_{\mathtt{01\underline{1}}} + x_{\mathtt{10\underline{1}}} + x_{\mathtt{11\underline{1}}} &= 0. \end{aligned}$$

(a) Show that for every assignment to $x_3x_5x_6x_7$ there exists a unique assignment to $x_1x_2x_4$ that satisfies all the constraints (1).

Solution: System (1) can be solved for x_1, x_2, x_4 in terms of x_3, x_5, x_6, x_7:

$$\begin{aligned} x_4 &= x_5 + x_6 + x_7 \\ x_2 &= x_3 + x_6 + x_7 \\ x_1 &= x_3 + x_5 + x_7. \end{aligned}$$

As this system is equivalent to (1), $x_1x_2x_4$ always exists and is uniquely determined.

(b) Use part (a) to count the number of elements (codewords) in $\mathcal{C}$.

Solution: There are 16 codewords. Each of x_3, x_5, x_6, x_7 can take one of two bit values so there are $2^4 = 16$ possible assignments to $x_3x_5x_6x_7$. As these determine $x_1x_2x_4$ they account for all the codewords.

(c) Argue that if $\mathbf{x}$ is in $\mathcal{C}$ and $\mathbf{y}$ differs from $\mathbf{x}$ in position i only then the right-hand side of (1), when applied to $\mathbf{y}$ and read from top to bottom, equals the binary representation of i.

Solution: As $\mathbf{x}$ is in the code it satisfies all the equations. Flipping the i-th bit of $\mathbf{x}$ flips the right-hand side of those equations that contain x_i. The j-th equation contains x_i exactly when j is set to 1 in the binary expansion of the index i. So the right-hand sides of precisely those equations that point to 1-bits in the binary expansion of i are set to 1.

(d) Use part (c) to argue that $\mathcal{C}$ has distance at least 3.

Solution: If not, there would be two distinct codewords $\mathbf{x}, \mathbf{x}'$ within distance at most 2. There is then some $\mathbf{y}$ that differs from both and $\mathbf{x}, \mathbf{x}'$ by at most a bit flip. By part (c) The right-hand side of (1) evaluated on $\mathbf{y}$ identifies the bit flipped, or equals zero if no bit was flipped. The only possibilities are that $\mathbf{y}$ was obtained from both $\mathbf{x}$ and $\mathbf{x}'$ by flipping the same bit, or by flipping no bit. In both cases $\mathbf{x}$ and $\mathbf{x}'$ must be equal to one another, contradicting their distinctness.

(e) Use part (c) to design an algorithm that corrects up to one error in $\mathcal{C}$: Given $\mathbf{y}$ that differs from a codeword $\mathbf{x}$ in at most one bit it outputs this $\mathbf{x}$.

Solution: The algorithm plugs $\mathbf{y}$ into (1). If the right-hand side is zero it does nothing to $\mathbf{y}$. If not, it flips the bit whose binary expansion is the right-hand side of the equations.

(f) (**Optional**) Show that up to a permutation of the indices $\mathcal{C}$ is the Hamming $[7, 4, 3]$ code from Lecture 7.

Solution: We rearrange the outputs in the code $\mathcal{H}$ from lecture to put the message bit m_1 in position 3, m_2 in position 5, m_3 in position 6, and m_4 in position 7. We set the remaining bits to $m_1 + m_2 + m_4$ in position 1, $m_1 + m_3 + m_4$ in position 2, and $m_2 + m_3 + m_4$ in position 4. The encoding is now set up so that all codewords satisfy (1). So all the codewords in the rearranged $\mathcal{H}$ are also in $\mathcal{C}$. Can there be any spurious codewords in $\mathcal{C}$ that do not come from $\mathcal{H}$? No, because both $\mathcal{H}$ and $\mathcal{C}$ have 16 codewords. The two must be equal.

Question 4

A codeword of the Reed-Solomon code with message length $k = 5$ and codeword length $n = 11$ has been corrupted in at most 3 positions. You can find your personalized corrupted codeword here. Find the message $\mathbf{m} = m_0m_1m_2m_3m_4$. You may use any algorithm you like. Explain clearly how you arrived at your solution.

Solution: My instance is $\mathbf{y} =$ `4 6 7 6 10 10 3 6 7 7 0`. My program searched through all 11^5 possible messages and their encodings to find the one that differs from $\mathbf{y}$ in at most three positions. It found the solution $\mathbf{m} =$ `4 5 9 9 1` that encodes to `4 6 6 6 10 2 1 6 7 7 0`. The errors are at positions 2, 5, and 6 (starting the count from zero).

In this example the search space is small enough that brute force does the job magnificently. Here is the code.

```
y = [4, 6, 7, 6, 10, 10, 3, 6, 7, 7, 0]

R = list(range(11))
for m0 in R:
    for m1 in R:
        for m2 in R:
            for m3 in R:
                for m4 in R:
                    cw = []
                    for x in R:
                        cw.append((m0 + x * (m1 + x * (m2 + x * (m3 + x * m4)))) % 11)
                    if sum(0 if cw[i] - y[i] == 0 else 1 for i in R) <= 3:
                        print(m0, m1, m2, m3, m4)
                        print(cw)
```