# Progcrypto Spring Break Problems

Claire Zhang

**Problem 1** (Problem 3: Applications of the Sum-Check Protocol**)

**How would you use sum-check to build interactive protocols for the following prover-verifier interactions?**

A. Count the number of triangles in a graph. A prover and a verifier both know a graph (in the graph-theory sense), and the verifier wants to know how many triangles the graph contains (i.e. triples $(A, B, C)$ of vertices, every two of which are connected by an edge). The prover has done the count, and wants to convince the lazy verifier that the count was done honestly. . .

B. Prove that $f(x) = 0$ for every $x \in \{0, 1\}^n$, where $f$ is a polynomial of degree (say) 2 in each variable.

- One way this can arise: Suppose you have two vectors $(a_1, \ldots, a_{2^n})$ and $(b_1, \ldots, b_{2^n})$, and you want to prove that for each $i$, we have $a_i b_i = 0$. First compute the multilinear extensions of $a$ and $b$ (and commit them somehow), then let $f$ be the product of those two multilinear polynomials. . .

A. Let the graph be $G = G(V, E)$ with $|V| = n$. Let $f : \{0, 1\}^n \to \{0, 1\}$ represent edges:

$$f(x) = 1 \iff \exists i, j \in [n] \text{ s.t. } x_i = x_j = 1 \text{ and } (i, j) \in E \text{ and } x_{k \notin \{i,j\}} = 0$$

Let's run sum check to check the sum

$$S \overset{\text{def}}{=} \sum_{x,y,z \in [n]^3} f(x, y)f(y, z)f(z, x)$$

where $k = \lfloor \log n \rfloor$ and we view a tuple of 3 vertices the concatenation of their binary representations.

The protocol runs as follows between prover P and verifier V:

a) V interpolates $f$ to be a degree-$n$ polynomial $f' : \mathbb{F}_q^3 \to \mathbb{F}_q$ (all boolean functions have a unique degree-$n$ polynomial representation).

b) P sends the claimed sum and $w^1(t) = \sum_{y,z \in [n]^2} f(t, y)f(y, z)f(z, t)$, which is a quadratic as $f$ is linear in $t$.

c) V verifies $w^1(1) + \ldots + w^1(n) = S$. V chooses $r_1 \sim \mathbb{F}_q$ and resets $S = w^1(r_1)$.

d) P sends $w^2(t) = \sum_{z \in [n]} f(r_1, t)f(t, z)f(z, r_1)$.

e) V verifies $w^2(1) + \ldots + w^2(n) = S$. V chooses $r_3 \sim \mathbb{F}_q$ and resets $S = w^2(r_3)$.

f) V verifies $S = f'(r_1, r_2)f'(r_2, r_3)f'(r_3, r_1)$.

B. We run the following protocol:

a) Choose u.a.r $g : \{0, 1\{^n \to \mathbb{F}_q$ from degree 2 polynomials.

**Fact 2.** $g(x_0) \sim \text{Unif}(\mathbb{F}_q)$ for every $x_0 \in \{0, 1\}^n$.

*Proof.* Symmetry. □

We run sum-check on

$$h(x) = f(x)g(x)$$

We verify

- Completeness. If $f$ is identically 0, then $h$ is identically 0, and the sum-check protocol will pass.
- Soundness. If $f$ is not identically 0, say $f(x_0) \neq 0$. Then, $h(x_0) = f(x_0)g(x_0)$ is uniform over $\mathbb{F}_q$ so with $1 - 1/q$ probability, $\sum_x h(x) = 0$.

  If we choose $g$ and run this protocal $t = \log n$ times, whp (Chebyshev(t) on top of sum check probabilistic guaruntees) we will catch the prover cheating.

**Problem 3** (Problem 5: Reinventing Garbled Circuits, OT, and MPC**)

**How would you use sum-check to build interactive protocols for the following prover-verifier interactions?**

A. Describe a protocol to perform oblivious transfer, using encryption / decryption.

- Concretely: suppose you have an ordered tuple of two numbers. You want to let your friend learn the number at a specific index (i.e. either 0 or 1), without you learning anything about the index they're querying, and without them learning anything about the number they did not retrieve. How can you do this using cryptography ?
- What are the minimum bandwidth requirements?

B. Using oblivious transfer, figure out how to perform simple two-party computations.

- Suppose that Alice has two secret numbers A$_1 andA_2, andthatBobhasasecretnumberB.AliceandBob$

C. So far we have assumed that Alice and Bob are honest. If they were dishonest, what attacks could they do? How can we force them to be honest?

A. Say Alice has $X[0], X[1]$ and Bob has bit $b$.

- Alice chooses $r \leftarrow \mathcal{K}$. Bob chooses $s \leftarrow \mathcal{K}$.
- Alice sends $\mathsf{Enc}_r(X[0]), \mathsf{Enc}_r(X[1])$ to Bob.
- Bob sends $\mathsf{Enc}_s\mathsf{Enc}_r(X[b])$ to Alice.
- Alice decrypts and sends $\mathsf{Dec}_r(\mathsf{Enc}_s(\mathsf{Enc}_r(X[b]))) = \mathsf{Enc}_s(\mathsf{Enc}_r(X[b]))$.
- Bob decrypts to know $\mathsf{Dec}_s(\mathsf{Enc}_s(X[b])) = X[b]$.

In this protocol,

- Alice learns nothing about $b$; if she did, she'd break perfect indisinguishability of the encryption scheme.
- Bob learns nothing about $X[1-b]$, assuming he asks for what he wants...

B. Lol so this expression is only 1 on one $(A_1, A_2, B)$ so it suffices to run one 2PC AND. But let's describe a protocol for general 2PC of circuits involving XOR, AND, and OR.

> **Claim** — There exists secure 2PC for XOR, AND, and OR.

*Proof.*

- XOR
- AND
- OR Run the AND protocol on the not of the inputs and then negate the output.

□

> **Claim** — Say Alice has secret randomness $A$ and Bob has secret randomness $B$; Alice has $\mathsf{Enc}_{A,B}(x)$, $\mathsf{Enc}_{A,B}(\bar{x})$; Bob has $\mathsf{Enc}_{A,B}(y)$, $\mathsf{Enc}_{A,B}(\bar{y})$. There exists secure 2PC computation of $(\mathsf{Enc}_{A|a',B|b'}(x \oplus y), \mathsf{Enc}_{A|a',B|b'}(x \oplus y))$, and the like for OR and AND ($a'$ secret to Alice, $b'$ secret to Bob).

*Proof.* XOR:

- Alice first decrypts the inputs to obtain

$$Z = [\mathsf{Enc}_B(X[0]), \mathsf{Enc}_B([1])]$$

  w where

$$X[0] = x$$
$$X[1] = \bar{x}$$

- Alice randomly permutes $Z$ and sends it to Bob.
- Bob sends $\mathsf{Enc}_{B|b'}(X[y])$ (note $X[y] = x \oplus y$ iff $Z$ was identically permuted).
- Alice computes $\mathsf{Enc}_{A|a',B|b'}(X[y])$ and saves it if $Z$ was identically permuted.
- Repeat with $Z$ permuted in the other way.

□

Now, to compute a circuit, we compute gates from the leaves. Initially, Alice computes $\mathsf{Enc}_a(x)$ and $\mathsf{Enc}_a(x)$ and similarly for Bob. Now we can compute up the tree, maintaining (public) $\mathsf{Enc}_{A,B}(x)$ and $\mathsf{Enc}_{A,B}(\bar{x})$ at each node where $x$ is the evaluation and $A$ and $B$ are the random keys in subtree $x$.