CS 507, Topics in Cryptography: Secure Computation
Homework 3

Due: November 12, 2026

Problem 1. Recall that the basic Garbled Circuit technique does not offer protection against a malicious garbler. In this problem, you will construct attacks by a malicious garbler.

Suppose we consider a basic version of Yao's GC that encodes circuits as follows. First, the (honest) garbler assigns to each wire w (1) two uniform keys $k_w^0, k_w^1 \leftarrow \{0,1\}^\lambda$ and (2) a uniform color bit (for point and permute) $c_w \leftarrow \{0,1\}$. Let $\texttt{Enc}$ denote a CPA\$-secure encryption algorithm. I.e., each ciphertext looks, to a party that does not hold the key, like a uniform string. Concretely, you may take $\mathsf{Enc}(k,m) := (r, F(k,r) \oplus m)$ for uniform r and a pseudorandom function F; in particular, decrypting a ciphertext under the wrong key yields a uniformly random-looking string rather than an error. The (honest) garbler constructs the four-row garbling $(r_{00}, r_{01}, r_{10}, r_{11})$ of each AND gate $z \leftarrow x \cdot y$ as follows:

$$
\begin{aligned}
&\text{let } (k_x^0, k_x^1), (k_y^0, k_y^1), (k_z^0, k_z^1) \text{ be keys on } x, y, z, \text{ respectively} \\
&\text{let } \alpha, \beta, \gamma \text{ be colors on } x, y, z, \text{ respectively} \\
&r_{00} \leftarrow \texttt{Enc}(k_x^\alpha, \texttt{Enc}(k_y^\beta, (k_z^{\alpha\beta}, \alpha\beta \oplus \gamma))) \\
&r_{01} \leftarrow \texttt{Enc}(k_x^\alpha, \texttt{Enc}(k_y^{\bar\beta}, (k_z^{\alpha\bar\beta}, \alpha\bar\beta \oplus \gamma))) \\
&r_{10} \leftarrow \texttt{Enc}(k_x^{\bar\alpha}, \texttt{Enc}(k_y^\beta, (k_z^{\bar\alpha\beta}, \bar\alpha\beta \oplus \gamma))) \\
&r_{11} \leftarrow \texttt{Enc}(k_x^{\bar\alpha}, \texttt{Enc}(k_y^{\bar\beta}, (k_z^{\bar\alpha\bar\beta}, \bar\alpha\bar\beta \oplus \gamma)))
\end{aligned}
$$

The main invariant on each wire w is as follows: Let x be the true value on that wire and let α be that wire's color. The evaluator obtains (1) the key k_w^x corresponding to the true value and (2) the masked value $x \oplus \alpha$. The evaluator evaluates AND gates by using her input keys to decrypt the row corresponding to her input wire color bits.

Now, consider a circuit C that takes as input (1) a length-10 string $x \in \{0,1\}^{10}$ from the garbler and (2) a length-10 string $y \in \{0,1\}^{10}$ from the evaluator and that outputs a single bit, which is the AND of all inputs:

$$C(x,y) = \left(\prod_i x_i \cdot y_i \right)$$

Consider the following scenario:

- The garbler garbles the circuit: $\hat{C}, X \leftarrow \text{Garble}(C)$. Here, $\hat{C}$ is the garbled circuit and X is a vector of pairs of keys for the circuit input. The garbler may choose its own input x arbitrarily.

- The evaluator obtains (via a TTP) input keys from X that correspond to her input and the garbler's inputs.
- The garbler sends to the evaluator $\hat{C}$.
- The evaluator evaluates the garbled circuit and obtains a single length-λ key on the single output wire.
- The evaluator sends this output key to the garbler.

1. Describe a malicious adversary $\mathcal{A}$ that can in this scenario learn some single bit y_j of the evaluator's input, for its choice of j. Namely, your adversary does not call Garble as prescribed, but instead runs arbitrary code to build a "garbled circuit". Note that your "garbled circuit" should be indistinguishable from an honestly constructed one such that the evaluator cannot detect that the attack is happening.
2. Now, describe a new adversary $\mathcal{A}$ that can with overwhelming probability learn the evaluator's *entire input*.

   Hint: You may assume that the security parameter $\lambda \gg 10$.

Answer 1.

Problem 2 (Generalizing GMW)**.** In the context of this problem, restrict consideration to two-party computation.

In lecture, we described the GMW protocol as a technique for securely computing fan-in two Boolean circuits. We gave a reduction from semi-honest 2PC to the following preprocessing functionality:

- The parties A and B may invoke this functionality $\mathcal{F}$ as many times as they would like.
- $\mathcal{F}$ samples a Beaver triple:

$$\{\ \alpha, \beta, \alpha \cdot \beta \ \text{ where } \ \alpha, \beta \leftarrow \{0,1\}\ \}$$

- $\mathcal{F}$ samples uniform XOR sharings $[\alpha], [\beta], [\alpha \cdot \beta]$ and gives one share to each party.

Based on this functionality, we showed how to securely evaluate any Boolean circuit consisting of fan-in-two AND and XOR gates. It is interesting to consider richer—and therefore potentially more efficient—circuit representations.

1. **Arithmetic GMW.** In this part, you will generalize the GMW protocol to work with integer values[1] instead of bits. Suppose we wish to compute an *arithmetic* circuit where each wire holds an integer modulo $m \in \mathbb{N}$, rather than a bit. In particular, each gate in the circuit is now an addition/multiplication gate over $\mathbb{Z}_m$, rather than XOR/AND.
   (a) Describe (1) a modification of the above preprocessing functionality and (2) how the parties use your modified functionality $\mathcal{F}'$ to handle each gate type.
   (b) Prove your protocol is *correct*.

[1] It is also possible to generalize GMW to work with arbitrary *finite rings*, though you do not need to do this here.

(c) Prove your protocol is semi-honest secure in the $\mathcal{F}'$-hybrid model.

2. **Boolean GMW with Vector-Scalar Multiplication.** In this part, you will extend Boolean GMW with a richer gate type. In particular, you will generate appropriate preprocessing needed to support a *vector-scalar multiplication gate*. Such a gate takes as input (1) a vector $x \in \{0,1\}^n$ and (2) a scalar s. It outputs the scaled vector $s \cdot x$. Note that such a gate generalizes the notion of an AND gate. Consider the following modified preprocessing functionality:

   - The parties A and B may invoke this functionality $\mathcal{F}'$ as many times as they would like.
   - Parties agree on a vector length n, and $\mathcal{F}'$ samples a vector-scalar triple:

   $$\{\ \alpha, \beta, \alpha \cdot \beta \ \text{ where } \ \alpha \leftarrow \{0,1\}, \beta \leftarrow \{0,1\}^n\ \}$$

   - $\mathcal{F}'$ samples uniform XOR sharings $[\alpha], [\beta], [\alpha \cdot \beta]$ and gives one share to each party.

   (a) Describe how the parties use one vector-scalar triple to evaluate one vector-scalar multiplication gate on shared inputs $[x]$, $[s]$, and state the communication cost.

   (b) Describe a protocol $\Pi_{\texttt{pre}}$ that uses 1-out-of-2 string OT to instantiate $\mathcal{F}'$. *Your number of consumed OTs per triple should be independent of n.*

   (c) Prove your preprocessing protocol is correct.

   (d) Prove your preprocessing protocol is semi-honest secure in the OT-hybrid model.

Answer 2.

Problem 3 (PIR vs ORAM)**.** We have discussed two very different approaches to securely accessing a memory:

- **Oblivious RAM** (ORAM) allows a client to outsource its read-write memory to an untrusted server. One way to formalize ORAM is via the following very simple functionality:
  - The client and server agree on a memory size n.
  - The client provides as input to the functionality a RAM program P and an input x. That is, P is a program that reads/writes to a memory of size n (the memory is all-zero initialized).
  - The functionality sends $P(x)$ to the client, and it sends m to the server, where m is the number of memory reads/writes performed while running $P(x)$.
- **Private information retrieval** (PIR) allows a client to privately fetch an item from a public database held by one or more servers. Let's assume a single-server variant of PIR. One way to formalize PIR is via the following very simple functionality[2]:
  - The server inputs a database D.
  - The client inputs an index i

[2]In fact, this functionality implies a slightly stronger version of PIR, sometimes called *symmetric* PIR, where the client learns *only* D_i, and nothing more. In other words, this ensures a notion of privacy for the server. In basic PIR, this is not a requirement.

– The functionality sends D_i to the client and $\perp$ to the server.

1. Let us consider a *read-only* RAM program, where the RAM program P first sequentially writes to each memory slot D_i, then only issues memory reads (and no more writes) thereafter. Assuming PIR and a CPA-secure (or CPA$-secure) encryption scheme, construct a semi-honest protocol that achieves the above ORAM functionality for such programs. *Note: Your client should run in low space, offloading storage of D to the server. In particular, the client cannot hold D at once; assume it receives D as a stream and may read each entry D_i exactly once.*

2. Prove your protocol secure by constructing simulators.

3. Compare your protocol to the ORAMs from class along the following axes, and say for which parameter regimes each wins: (i) communication per access; (ii) server computation per access, and whether the PIR server's cost is inherent (hint: what must any single-server PIR server read?); (iii) client computation and storage; (iv) the cryptographic assumptions needed to instantiate the PIR functionality with a single server; (v) the kinds of programs supported.

4. Explain why your approach does not extend to programs that arbitrarily write to memory. What would the server have to do on a write, and why can PIR not provide it? (You need not solve the problem; identify it precisely.)

Answer 3.