Indian Institute of Technology, Kanpur
Computer Science and Engineering
Assignment 1
CS670: Cryptographic Techniques for Privacy Preservation

Instructor: Adithya Vadapalli

06/10/2025

This assignment is long! Please start early. Doing this assignment correctly is very important, as every other assignment will be built on this.

Deadline: October 17th, 2025, EOD on Hello IITK. No Extension will be given.

Academic Integrity You have to do the assignment individually. You are encouraged to ask the instructor for help. Sign up on Piazza to ask for help: `https://piazza.com/iitk.ac.in/firstsemester2025/cs670`. Students can should come to the Instructor's office hours for help in coding. However, copying code from others or using AI tools without understanding the solution is not allowed and will result in 0 marks for the assignment. Students may be asked to explain their code and solution during evaluation.

Objective Write a C++ program named `gen_queries.cpp` that generates Distributed Point Function (DPF) queries and verifies their correctness.

Specifications:

1. **Program Name:** `gen_queries.cpp`
2. **Command-Line Arguments:**

   ```
   ./gen_queries <DPF_size> <num_DPFs>
   ```

   - `<DPF_size>`: Size of each DPF (domain size).
   - `<num_DPFs>`: Number of DPF instances to generate.
3. **Functionality:**
   - For each DPF:
     - Randomly choose an **index (location)** within the DPF's domain.
     - Randomly choose a **target value** for that index (this will be determined by the final correction word).
     - Call your `generateDPF()` function to generate the DPF keys.
   - Save or print the generated DPF queries as appropriate.
4. **Correctness Testing (`EvalFull` function):**
   - Implement a function named `EvalFull()` that:
     - Evaluates the DPF across all domain points for both keys.
     - Combines the outputs and checks if they reconstruct the correct target value at the chosen index, and zero elsewhere.
     - Prints `"Test Passed"` if the evaluation is correct and `"Test Failed"` otherwise.
5. **Implementation Notes:**
   - Use a cryptographically secure random generator (e.g., `std::random_device` and `std::mt19937_64`)
   - Keep the DPF interface modular:
     - `generateDPF(location, value)`
     - `evalDPF(key, index)`

    – `EvalFull(key, size)`

6. **Example Usage:**

```
./gen_queries 1024 10
```

   This should generate 10 DPFs, each of size 1024, test their correctness using `EvalFull`, and print the results.

Grading

- Correctness and Security: 80%
- Code clarity and documentation: 20%