SMC

January 28, 2026

1 Assignment 1: Mini Garbled Circuit

This assignment introduces the fundamental ideas behind **Yao's Garbled Circuits** through a minimal and intentionally inefficient construction. You will implement a **single garbled NAND gate**, without optimisations or oblivious transfer. The goal is conceptual understanding rather than performance.

After completing this assignment, you should be able to:

- Explain how wire labels represent Boolean values.
- Construct and evaluate a garbled gate.
- Understand why only one output is revealed.

1.1 Threat Model and Scope

- Semi-honest adversaries.
- Single Boolean gate (NAND).
- No oblivious transfer.
- No networking.
- No performance optimisation.

1.2 Model

There are two roles:

- **Garbler**: constructs the garbled gate.
- **Evaluator**: evaluates the garbled gate.

Both roles may be implemented within the same program.

1.3 Wire Labels

For each wire w, generate two random labels:

$$L_w^0, L_w^1 \in \{0,1\}^k$$

where $k \geq 128$. These labels encode logical values but do not reveal them.

1.4 Garbling the NAND Gate

For each input combination $(a, b) \in \{0, 1\}^2$:

1. Derive an encryption key:
$$K[a, b] = \mathrm{KDF}(L_x^a \| L_y^b)$$
2. Encrypt the output label:
$$C[a, b] = \mathrm{Enc}(K[a, b], L_z^{a \texttt{ NAND } b})$$

This results in four ciphertexts:

$$\{C[0, 0], C[0, 1], C[1, 0], C[1, 1]\}$$

The garbled gate is a list of these ciphertexts with random ordering.

1.5 Evaluation

The evaluator:

1. Receives one label for x and one for y.
2. Attempts to decrypt all four ciphertexts.
3. **Exactly one decryption succeeds.**
4. The decrypted value is the output label.

A decoding table maps output labels to Boolean values.

1.6 Implementation Requirements

Your program must:

- Generate wire labels for x, y, and z.
- Garble the NAND gate.
- Evaluate the garbled gate for given inputs.
- Output the correct Boolean result.

1.7 Written Explanation

Your submission must include a short written explanation (1 page) addressing:

(a) Correctness

Why does exactly **one** ciphertext decrypt correctly?

(b) Inefficiency

Explain why four decryptions are required.

1.8 Allowed Tools

You may **not** use:

- Existing garbled circuit or MPC frameworks

2 Assignment 2 - Garbled Circuits in Practice

In this assignment you will implement a privacy-preserving computation using the NAND gate implemented above. This assignment explores the *trade-off between efficiency and leakage.*

You can implement the same functionality in two different ways:

- a fully oblivious version (no control-flow leakage),
- a more efficient version that intentionally leaks limited information.

You will measure the performance and reason about what information is leaked. After completing this assignment, you should be able to:

- Implement secure two-party computation
- Understand how control flow impacts security and performance
- Distinguish between oblivious and leaky program structures
- Measure and interpret performance costs of secure computation
- Formally describe information leakage

2.1 Problem Description

Two parties, Alice and Bob, each hold a private array of n booleans:

$$A = (a_0, a_1, \ldots, a_{n-1}), \quad B = (b_0, b_1, \ldots, b_{n-1})$$

They want to compute iff:

$$\bigwedge a_i = b_i$$

2.2 Option A — Fully Oblivious Implementation

Implement the computation in a fully oblivious manner.

Requirements

- All control flow depending on secret data must be oblivious.
- No early termination is allowed.
- The program must always iterate over all indices.

Expected behaviour

- Runtime is independent of where the first mismatch occurs.
- Only the final output is revealed.

Hint

You will need to:

- Compute a Boolean flag for each index.
- Conjunct all of them

2.3 Option B - Leaky but Efficient Implementation

Implement a second version where performance is improved by allowing controlled leakage.

Requirements

- Use public control flow.
- Stop iteration as soon as a mismatching is found.
- Reveal the result immediately.

Expected behaviour

- Runtime depends on the location of the first mismatch.
- Control flow leaks information about the index.

2.4 Performance Evaluation

You must measure and report:

- Runtime for both versions
- Scaling behaviour for at least three input sizes (e.g., $n = 4, 8, 16$)

2.5 Security Analysis

Your report (2 pages) must include:

1. Threat Model

What information is protected?

2. Leakage Analysis

- What is leaked in the oblivious version?
- What additional information is leaked in the early-exit version?

3. Trade-off Discussion

Explain:

- In which real-world scenarios the additional leakage might be acceptable
- When it would be unacceptable

3 Submission

Submit:

- Source code
- A 3 pages report (PDF)
- Instructions to build and run