

Homework 2

Due: 6pm, Friday, 18 September 2026 (submit via Gradescope)

Instructions: You **must** typeset your solution in LaTeX using the provided template:

<https://cs8803pc.cc.gatech.edu/hw/template.tex>

Submission Instructions: You must submit your problem set via [Gradescope](#). Please use course code **4DN8ZD** to sign up. Note that Gradescope requires that the solution to each problem starts on a **new page**.

Bugs: We make mistakes! If it looks like there might be a mistake in the statement of a problem, please ask a clarifying question on Ed.

Problem 1: Understanding Interactive Proofs [15 points]. (*Problems from “The Foundations of Cryptography - Volume 1, Basic Techniques” by Oded Goldreich*)

- (a) *The role of verifier randomness:* Let L be a language with a sound and complete interactive proof system where the verifier V is deterministic. Show that $L \in \text{NP}$.

[**Hint:** Recall that $L \in \text{NP}$ if there is a polytime, deterministic $V(x, w)$ such that $x \in L$ iff there exists a w such that $V(x, w) = 1$. w is the non-interactive proof.]

- (b) *The role of prover randomness:* Let L be a language with a sound and complete interactive proof system. Show that there exists a sound and complete interactive proof system for L for which the prover P is deterministic.

[**Hint:** Use the fact that P is unbounded.]

- (c) *The role of errors:* Let L be a language with a perfectly sound and complete interactive proof system, that is if $x \notin L$, the verifier *never* accepts (not even with negligible probability). Show that $L \in \text{NP}$.

Problem 2: Zero knowledge for 3 coloring [15 points]. In this problem, you will analyze a zero-knowledge proof that applies directly to the three-colorability of graphs. Let $G \subseteq [n] \times [n]$ be a graph on n vertices. Let 3-COLOR be the set of three-colorable graphs: graphs such that there exists a coloring function $f : [n] \rightarrow [3]$ such that for all $(a_i, b_i) \in G$, $f(a_i) \neq f(b_i)$. Prove that the following interactive proof has negligible soundness error and is honest-verifier zero-knowledge. (It is perfectly complete, but you need not prove that.) You may assume that $\text{Commit}(m \in [3], r) \rightarrow c$ is a computationally hiding and perfectly binding commitment scheme for colors in $[3]$.

1. P starts with $G = \{(a_i, b_i)\}_i$ and f ; V starts with G .
2. P : samples a permutation π on $[3]$, computes $f' \leftarrow \pi \circ f$, samples r_i for all $i \in [n]$, computes $c_i \leftarrow \text{Commit}(f'(i), r_i)$, and sends $c_1, \dots, c_n$ to V .
3. V : samples $i \xleftarrow{\$} [|G|]$ and sends it to P .

4. P opens commitments c_{a_i} and c_{b_i} by sending $m_a \leftarrow f'(a_i), r_a \leftarrow r_{a_i}, m_b \leftarrow f'(b_i), r_b \leftarrow r_{b_i}$ to V .
5. V checks $c_{a_i} = \text{Commit}(m_a, r_a)$, $c_{b_i} = \text{Commit}(m_b, r_b)$, and $m_a \neq m_b$, and rejects if any fail.
6. They repeat the protocol $\lambda|G|$ times.

Hints:

- For $x > 1$, $(1 - x^{-1})^x < e^{-1}$.
- Before building the HVZK simulator, you might ask yourself: How are $(f'(a_i), f'(b_i))$ distributed, given the fact that f is a 3-coloring and π is a uniformly random permutation?
- For HVZK indistinguishability, start with the values that are identical in both distributions, condition on them (fix them), and then use a hybrid argument to show that the rest of the distributions are indistinguishable. **Do** give an explicit reduction for your hybrid argument, but you don't need to be overly detailed in the analysis of the reduction.
- For HVZK, recall that the simulator does not get f , but indistinguishability must hold for all G and f , so any *reduction* used to prove indistinguishability can use f .

Problem 3: Implementation: zero knowledge for 3 coloring [15 points]. Now, you will implement the protocol from the previous problem, and analyze its performance. You'll also implement the perfectly binding commitment scheme: the El-Gamal commitment:

$$\text{Commit}(m; r) = (g^m h^r, g^r)$$

Toolchain Just as for the coin-flipping problem, you'll need Rust (via rustup).

What to implement Start from `zk_3_color-starter.zip`, which unpacks into a Rust crate in a directory named `zk_3_color`. You will implement `eg_commit` and the `Prover` and `Verifier` branches of `main`, inside `src/main.rs`. As before, the surrounding machinery is written for you: the group its scalars, the generators, sampling (`random_scalar()`, `random_number(n)`, and `random_color_permutation()`), connection setup (`connect_to_peer`), and serialization (`read/write`). Note that all numeric types (including `color`) are zero-indexed, not one-indexed (so, for examples, colors are `{0, 1, 2}`).

The graph is sampled for you, by `random_graph_and_coloring`, which returns a `Graph` (a list of edges, each a pair of node indices) together with a `Coloring` (a map from node index to `Color`) that is guaranteed to 3-color it. The Prover sends the graph to the Verifier before the repetitions begin; only the coloring is secret. The verifier rejects by panicking, so use `assert!`, `assert_eq!`, and/or `panic!`. Be sure to use fresh samples in every protocol repetition and to have `eg_commit` reject an out-of-range color.

Testing Your `main` is a binary that takes the party name and the three parameters n (nodes), $|G|$ (edges), and the repetition count. The Prover listens and the Verifier connects, so start the Prover first:

```
$ cargo run --release -- prover 20 30 30      # terminal 1
$ cargo run --release -- verifier 20 30 30     # terminal 2
```

Both parties use TCP port 27183 on localhost. If you see “address already in use” or a hang, a party from an earlier run is probably still alive; kill it and try again.

The Makefile works as it did before:

```
$ make check          # type-check, check format, and lint
$ make build          # build
$ make test           # build, then test (3 trials)
```

`make test` runs the two parties as separate processes and passes when the Verifier accepts. It only tests an honest Prover, so it will not catch a missing check.

Performance The 3-coloring protocol is important for theoretical, not practical reasons, but we’ll still benchmark its performance on practical problems. Performance (total runtime) is sensitive to two parameters: the node count (which grows the number of commitments) and repetitions. Every repetition commits to every node, so cost correlates with the product: milliseconds per node per repetition. Once your implementation works, run the following benchmark to measure that cost:

```
$ ./bench.sh          # about 10 seconds
```

Report the estimated milliseconds per node per repetition in your written solution (in $\LaTeX$).

Now, let’s imagine proving something important. One of NIST’s best post-quantum signature candidates for standardization is called FAEST. In FAEST, a signature is essentially a zero knowledge proof about an evaluation of AES: the signer shows it knows a key k such that $\text{AES}_k(x) = y$ for a public pair (x, y) . FAEST uses a non-interactive ZKP (in the random oracle model) to create *signatures*. Our interactive ZKP gives an *identification protocol* if applied to AES. How much time would it take for our ZKP to prove AES? Let us estimate. Take AES-128 encryption as a Boolean circuit of about 6.4×10^3 AND gates, 2.8×10^4 XOR gates, and 2.1×10^3 NOT gates (Boyar–Peralta).¹ The Tseitin transform² turns it into a 3-CNF with roughly $v = 3.7 \times 10^4$ variables and $c = 1.4 \times 10^5$ clauses. The textbook reduction from 3-CNF to 3-COLOR³ then builds a graph that is 3-colorable if and only if there is a k such that $\text{AES}_k(x) = y$, for fixed x and y . This graph has about 9.1×10^5 nodes and about 1.8×10^6 edges. Compute, and report (in $\LaTeX$):

1. the number of repetitions R needed for soundness error 2^{-128} and
2. the runtime of your protocol for that soundness on the AES graph.

For the first part, approximate the protocol’s soundness error as $e^{-R/|G|}$ for R repetitions on a graph of $|G|$ edges. (For large graphs like ours, this is a very good approximation of the true soundness error $(1 - |G|^{-1})^R$.) For the second part, use the per-node cost your own benchmark reported.

Of course, our scheme is not nearly as fast as FAEST, which signs and verifies in approximately 0.5ms total. In the coming weeks we will not study FAEST specifically (it is highly specialized and optimized for AES), but will study a number of ZKPs that are far more efficient than this problem’s ZKP.

¹Joan Boyar and René Peralta, “A Small Depth-16 Circuit for the AES S-Box” (2012). Ready-made AES circuits built from those S-boxes are collected at <https://nigelsmart.github.io/MPC-Circuits/>.

²G. S. Tseitin, “On the complexity of derivation in propositional calculus” (1968). One variable per gate; three clauses for an AND, four for an XOR, and two for a NOT.

³M. R. Garey, D. S. Johnson, and L. Stockmeyer, “Some simplified NP-complete graph problems” (1976). The graph has 2 nodes and 3 edges per variables, and 6 nodes and 12 edges per clause.

Submitting In the crate directory, run

```
$ make zip
```

which cleans the build artifacts and writes `../zk_3_color-solution.zip`, a zip of the whole crate, one directory up. Submit that zip file, and answer the written question in $\text{\LaTeX}$.

Grading “`make test`” just tests correctness against an honest partner, but your protocol will also be hand-graded for security. Be sure to implement every check!

Problem 4: Time Spent [1 point for answering]. How long did you spend on this problem set? This is for calibration purposes, and the response you provide will not affect your score.

Optional Feedback [0 points]. Please answer the following questions to help us design future problem sets. You do not need to answer these questions, and if you would prefer to answer anonymously, please use the online anonymously form. However, we do encourage you to provide us feedback on how to improve the course experience.

- (a) What was your favorite problem on this problem set? Why?
- (b) What was your least favorite problem on this problem set? Why?
- (c) Do you have any other feedback for this problem set?
- (d) Do you have any other feedback on the course so far?