Problem Set 3

CS6230: Topics in Information Security
Due on: Nov 14, 2021
Total Points: 60

- Collaboration on the problem set is encouraged. Write up your submission on your own, though, and list the names of all of your collaborators.
- Solutions to some of the problems in this problem set are easy to find on the internet or in textbooks. Please do not look them up.
- Formal mathematical proofs (e.g., of completeness and soundness) are required in support of your answer to each of the problems.

---

1 ZK protocol for Graph Non-Isomorphism

In class, we saw an honest-verifier PZK protocol for Graph Non-Isomorphism. Given input graphs (G_0, G_1), both over n vertices, this protocol (P, V) was as follows:

1. V samples a uniformly random relabelling permutation $R : [n] \to [n]$, and a uniformly random bit $b \leftarrow \{0, 1\}$. It computes $H \leftarrow R(G_b)$ and sends it to the prover.
2. If G_0 and G_1 are isomorphic, P sets $b' = \perp$. Else, P sets the bit $b' = 0$ if H is isomorphic to G_0, and $b' = 1$ otherwise. It sends b' to V.
3. V accepts if $b = b'$, and rejects otherwise.

We have seen the arguments for this protocol being complete and sound, and the perfect simulator for the honest verifier is also quite simple: sample R and b as V does, and output $(R, b, R(G_b), b)$.

This protocol is not zero-knowledge against malicious verifiers, as such a verifier V^* may not obtain H as a relabelling of one of G_0 or G_1. The honest prover's algorithm could then be exploited by V^* to learn whether some arbitrary graph H is isomorphic to G_0 or G_1. This protocol can, however, be transformed into one that is zero-knowledge against malicious verifiers as well. The idea behind the transformation is to have the verifier prove to the prover that it sampled H correctly, failing which the prover will refuse to participate further in the protocol. In other terms, the protocol will force any verifier to behave honestly. The transformed protocol is as follows (below, λ is a security parameter):

1. V samples a uniformly random bit $b \leftarrow \{0, 1\}$. It computes H as a random relabelling of G_b.
2. V further samples λ random bits $b_1, \ldots, b_\lambda$. For each $i \in [\lambda]$,
   - if $b_i = 0$, it computes H_{i0} and H_{i1} as random relabellings of G_0 and G_1, respectively
   - if $b_i = 1$, it instead swaps these and computes H_{i0} and H_{i1} as random relabellings of G_1 and G_0, respectively
3. V sends H and all the pairs (H_{i0}, H_{i1}) to P
4. P samples random bits $b'_1, \ldots, b'_\lambda$, and sends them to V
5. For each $i \in [\lambda]$,
   - if $b'_i = 0$, V sends b_i and the relabellings from G_0 and G_1 to H_{i0} and H_{i1} (that is, from G_0 to H_{ib_i}, and form G_1 to $H_{i(1-b_i)}$)
   - if $b'_i = 1$, V sends the relabelling to H from whichever of H_{i0} or H_{i1} is isomorphic to H (note that it can find this because it knows the relabelling from the appropriate H_i to G_b, and also the relabelling from G_b to H)
6. P checks that all the relabellings were reported correctly by V. If not, it sends $b' = \perp$
7. If the checks pass, then P sets the bit $b' = 0$ if H is isomorphic to G_0, and $b' = 1$ if H is isomorphic to G_1, and $b' = \perp$ otherwise. It sends b' to V.
8. V accepts iff $b = b'$

Above, the prover wants to make sure that the verifier generated H correctly – as a relabelling of one of the graphs G_0 or G_1. Of course, it cannot ask V^* to prove this by revealing the relabelling that it used, as that would reveal the bit b and compromise soundness. So it does this indirectly. If V^* used an H such that it did not know a relabelling from one of G_0 and G_1 to H, then for each i, it would either not be able to produce a relabelling from G_0 and G_1 to H_{i0} and H_{i1}, or from one of the H_i's to H. By randomly asking for one of these relabellings (and repeating this λ times), the prover hopes to catch V^* if this is the case.

If again, for some $i \in [\lambda]$, the prover knew both the relabellings from G_0 and G_1 to H_{i0} and H_{i1}, and the relabelling from one of these to H, then soundness would be compromised. But for each i, it learns exactly one of these, so it still doesn't learn which of G_0 or G_1 was relabelled to get H.

Problem 1.1 (2 points)**.** *Prove that the above protocol is perfectly complete and has soundness error* $1/2$.

While the protocol was described in natural language above to be more readable, in constructing a simulator for it and analysing it, it will be useful to set up the symbols for various parts of the transcript beforehand. In addition to R, b, and the b_i's and b'_i's, denote the relabelling from G_0 to H_{ib_i} by R_{i0}, and the relabelling from G_1 to $H_{i(1-b_i)}$ by R_{i1}. Note that the relabelling from H_{ib_i} to H if $b = 0$ is the composition (RR_{i0}^{-1}), and that from $H_{i(1-b_i)}$ if $b = 1$ is (RR_{i1}^{-1}).

Problem 1.2 (18 points)**.** *Fill in the missing part of the simulator described below, and use it to prove that the protocol is statistical zero-knowledge, with zero-knowledge error* $O(1/secp)$.

The simulator is an extension of that for the honest-verifier protocol, and works as follows given input (G_0, G_1) for any verifier V^*:

1. Sample uniform random string r_{V^*} for V^*, and compute $(H, \{(H_{i0}, H_{i1})\}_{i\in[\lambda]}) \leftarrow V^*(G_0, G_1; r_{V^*})$
2. Sample uniform random bits $b'_1, \ldots, b'_\lambda$, and run $V^*(G_0, G_1, \{b'_i\}; r_{V^*})$ to get relabellings $(R_1, \ldots, R_\lambda)$
3. If even one of these relabellings does not satisfy the condition it is required to in the protocol, set $b' = \perp$
4. Else, repeat λ^2 times:
   - Sample random bits $b''_1, \ldots, b''_\lambda$, and run $V^*(G_0, G_1, \{b''_i\}; r_{V^*})$ to get relabellings $(R'_1, \ldots, R'_\lambda)$
   - *(fill in this part, which attempts to recover the value of b')*
5. If b' is not set so far, set it to $\perp$
6. Output $(r_{V^*}, (H, \{(H_{i0}, H_{i1})\}), (b'_1, \ldots, b'_\lambda), (R_1, \ldots, R_\lambda), b')$

Hint 1.1. *How can you put together the R_i's and R'_i's to recover some information that can help with the simulation?*

Hint 1.2. *Notice that once r_{V^*} is fixed, V^*'s behaviour only depends on $(b'_1, \ldots, b'_\lambda)$. For some values of this tuple, the relabellings $(R_1, \ldots, R_\lambda)$ will pass the prover's checks, for others they won't. Think about these two cases separately, and how well you can simulate in each case. With an r_{V^*} fixed, what happens to the simulation error if the relabellings never pass the checks? What happens if they always do? Consider the set of tuples $(b'_1, \ldots, b'_\lambda)$ for which the relabellings produced pass the checks. How can the simulation work if this set is large? How can it work if this set is small?*

In fact, this protocol has a perfect zero-knowledge simulator that runs in expected polynomial time. Do you see how to construct that?

2 A Simple PCP for NP

In this section, we will extend the PCP we saw in class for ML – the problem of solving a system of multivariate linear equations – to MQ – that of solving a system of multivariable quadratic equations. The PCP will still have a constant number of queries, logarithmic randomness complexity, and an exponentially long proof. An instance of this problem consists of m quadratic equations over n variables

in $\mathbb{F}_2$. Each such equation is specified by a vector of coefficients $a_i \in \mathbb{F}_2^{n^2}$, and a value $b_i \in \mathbb{F}_2$, requiring that $\sum_{j,k\in[n]} a_i[j,k] \cdot x_j \cdot x_k = b_i$.[1] Here, the indexing $a_i[j,k]$ is just a convenient way of referring to $a_i[(j-1)\cdot n + k]$.

Given a set of m such equations $Q = \{(a_i, b_i)\}_{i\in[m]}$, the question is whether there exists a $u \in \mathbb{F}_2^n$ satisfying all of them. We will construct a PCP that proves that such a u indeed exists. The construction starts by "linearising" the given quadratic equations.

Definition 2.1. For any vectors $u, v \in \mathbb{F}_2^n$, their *tensor product*, denoted $u \otimes v$, is a vector in $\mathbb{F}_2^{n^2}$ whose entries, for any $j, k \in [n]$, are as follows:

$$(u \otimes v)[j,k] = u[j] \cdot v[k]$$

For any quadratic equation (a_i, b_i) from the given system, consider the corresponding linear equation over n^2 variables given by $\sum_{j,k\in[n]} a_i[j,k] \cdot y_{j,k} = b_i$. (Here again, $y_{j,k}$ is short for $y_{(j-1)n+k}$.) Linearising all the equations in this manner leads to a system of linear equations given by a matrix $A_Q \in \mathbb{F}_2^{m\times n^2}$ whose rows correspond to the a_i's, and a vector $b_Q \in \mathbb{F}_2^m$ composed of the b_i's, with the system being $A_Q y = b_Q$.

Problem 2.1 (2 points)**.** *Prove that if a vector $u \in \mathbb{F}_2^n$ satisfies the system of quadratic equations Q, then we have $A_Q(u \otimes u) = b_Q$.*

The idea behind the PCP for MQ is to prove two things given the system Q:

1. There exists a solution $v \in \mathbb{F}_2^{n^2}$ to the system of linear equations $A_Q v = b_Q$
2. This solution v is of the form $(u \otimes u)$ for some $u \in \mathbb{F}_2^n$

It is easy to see that if the PCP can soundly prove these two statements, then it has proven that there exists a u that satisfies all the equations in Q. We saw in class how the first statement can be proven using just the Hadamard encoding of v – which we denoted by H_v. It turns out that the second can also be proven given, in addition, the Hadamard encoding of u as well, as we will see. Thus, for a system Q with solution $u \in \mathbb{F}_2^n$, our PCP proof is simply $(H_u, H_{u\otimes u})$, which is of length $(2^n + 2^{n^2})$.

We will denote a candidate proof of this form by (π_1, π_2), where $\pi_1 \in \mathbb{F}_2^{2^n}$, and $\pi_1 \in \mathbb{F}_2^{2^{n^2}}$. Given oracle access to such a proof (π_1, π_2) for a system Q, the PCP verifier's task can be broken down into three parts:

1. Check that $\pi_1 = H_u$ and $\pi_2 = H_v$ for some $u \in \mathbb{F}_2^n$ and $v \in \mathbb{F}_2^{n^2}$
2. Check that $v = (u \otimes u)$
3. Check that $A_Q v = b_Q$

We will see, in turn, how each of these may be performed.

2.1 Linearity Testing

As mentioned in class, the first task cannot be performed exactly with just a small number of queries, and we will only check that π_1 and π_2 are close to being Hadamard encodings of the appropriate length. This may be abstracted out as the task of testing whether a function is linear, given just oracle access to it. This is because the function that takes an index $x \in \mathbb{F}_2^n$ as input and outputs the x^{th} bit of a Hadamard encoding H_u is simply outputting an evaluation of the linear function $\langle x, u\rangle$.

Problem 2.2 (3 points)**.** *Consider a function $h : \mathbb{F}_2^n \to \mathbb{F}_2$. Prove that there exists a $u \in \mathbb{F}_2^n$ such that for all $x \in \mathbb{F}_2^n : h(x) = \langle x, u\rangle$ if and only if, for all $x, y \in \mathbb{F}_2^n : h(x) \oplus h(y) = h(x \oplus y)$.*

Our testing algorithm will be L, which is given oracle access to a function $h : \mathbb{F}_2^n \to \mathbb{F}_2$. It then operates as follows:

[1] Note that these quadratic equations do not have any linear terms. This is without loss of generality when working over $\mathbb{F}_2$, as we always have $x^2 = x$.

1. Sample uniformly random $x, y \in \mathbb{F}_2^n$
2. If $h(x) \oplus h(y) = h(x \oplus y)$, accept. Else, reject.

It is easy to see that L is perfectly complete – if h is indeed a linear function, then L^h will always accept. There are many proofs of the soundness of L, and we will see a combinatorial proof that is one of the simpler ones.

Definition 2.2. For some $\delta \in [0, 1]$, we say that two functions $f, g : \mathbb{F}_2^n \to \mathbb{F}_2$ are *δ-far* if they differ on at least a δ-fraction of all inputs – that is,

$$\Pr_{x \in \mathbb{F}_2^n} [f(x) \neq g(x)] \geq \delta$$

And they are *δ-close* if they differ on less than a δ-fraction of inputs.

We will show that, for any constant δ, if every linear function $\ell : \mathbb{F}_2^n \to \mathbb{F}_2$ is δ-far from h, then L^h rejects with some constant probability (which can later be amplified by repetition). Rather, we will show that if L^h reject with probability less than $\min(2/9, \delta/2)$, then there is a linear function that is δ-close to h. This function, which we will call g, is defined as follows:

$$g(x) = \begin{cases} 1 & \text{if } \Pr_{y \leftarrow \mathbb{F}_2^n} [h(y) \oplus h(x \oplus y) = 1] \geq 1/2 \\ 0 & \text{otherwise} \end{cases}$$

We also set up the following notation for each $x \in \mathbb{F}_2^n$:

$$P_x = \Pr_{y \leftarrow \mathbb{F}_2^n} [h(y) \oplus h(x \oplus y) = g(x)]$$

Note that, by the definition of g, $P_x \geq 1/2$ for any $x \in \mathbb{F}_2^n$.

Problem 2.3 (5 points)**.** *Prove that, if L^h rejects with probability at most $\delta/2$, then h and g are δ-close.*

Hint 2.1. *Fix some x such that $h(x) \neq g(x)$. What can you say about the probability that L^h rejects conditioned on the first point it samples being this x?*

Problem 2.4 (5 points)**.** *Prove that, if $P_x > 2/3$ for all x, then g is linear.*

Hint 2.2. *Recall the characterisation of linear functions from Problem 2.2. Can you show that g satisfies all the checks needed there?*

Problem 2.5 (5 points)**.** *Prove that, if L^h rejects with probability less than $2/9$, then for all x, we have $P_x > 2/3$.*

Hint 2.3. *The proof I know for this is a little unintuitive. One approach is to look at the probability that, for two independently random y_1, y_2, the values $(h(y_1) \oplus h(x \oplus y_1))$ and $(h(y_2) \oplus h(x \oplus y_2))$ agree. What is the relation of this probability to P_x? Can you bound this probability in some other way that can then be used to bound P_x?*

Thus, if L rejects with probability less than both $2/9$ and $\delta/2$ then g is a linear function that is δ-close to h. In other words, if h is δ-far from all linear functions, then L^h would reject with probability at least $\min(2/9, \delta/2)$. By repeating L some constant number of times, we can then get a tester that makes $O(1)$ queries, and rejects with probability 0.99 if h is at least 0.01-far from all linear functions. We will use the linearity tester with these parameters in our PCP verifier.

2.2 Local Correction

Once a given PCP proof (π_1, π_2) passes the linearity test described above, we can proceed assuming that π_1 and π_2 are each 0.01-cloase to H_u and H_v for some vectors $u \in \mathbb{F}_2^n$ and $v \in \mathbb{F}_2^{n^2}$. If all the queries made by the PCP verifier were uniformly distributed (even if they were not independent of each other), this would be sufficient. We would argue that the probability that each query looks at a bad part of the proof is at most 0.01, and by the union bound, as long as the number of queries is $\ll 100$, the verifier

never looks at a bad part of the proof with some good probability. Conditioned on this, we can then analyse the verifier's soundness assuming that π_1 and π_2 were actually equal to H_u and H_v.

However, as we will see, some of the queries that our verifier will be making will not be distributed uniformly over all possible indices of π_2. The above guarantee of 0.01-closeness is meaningless to such a query – perhaps π_2 is never equal to H_v on the indices that this non-uniform query looks at. In order to deal with this, we will "distribute" the error in π_2 evenly across all indices. That is, given access to a π that is 0.01-close to some Hadamard codeword H_v, we will design a procedure that, *for any* x, computes $H_v[x] = \langle x, v\rangle$ correctly with some large constant probability. This procedure will also make only a constant number of queries to π, and is said to be a *local correction* algorithm for the Hadamard code.

This local corrector, which we will denote by C, is actually quite simple. It works as follows given oracle access to a string $\pi \in \mathbb{F}_2^{2^n}$, and an input $x \in \mathbb{F}_2^n$:

1. Sample $y \leftarrow \mathbb{F}_2^n$
2. Output $\pi[y] \oplus \pi[x \oplus y]$

Problem 2.6 (3 points)**.** *Suppose there is some $v \in \mathbb{F}_2^n$ such that π is 0.01-close to H_v. Prove that, for any $x \in \mathbb{F}_2^n$, $C^\pi(x)$ outputs $\langle x, v\rangle$ with probability at least 0.98.*

If π were actually equal to H_v, it is easy to see that $C^\pi(x)$ will always output πxv. Once we have such local correction, we can pass each query that the verifier makes through this correction procedure. Then, irrespective of the distribution of the query, as long as π is sufficiently close to H_v, the verifier receives the corresponding index of H_v, except with some small probability.

2.3 Testing Tensor Structure

Suppose the proof (π_1, π_2) that the verifier is given access to is indeed such that π_1 and π_2 are 0.01-close to H_u and H_v, respectively, for some $u \in \mathbb{F}_2^n$ and $v \in \mathbb{F}_2^{n^2}$. The second task of the verifier as listed above is to check that $v = u \otimes u$ while making only a constant number of queries. For this, we will make use of a convenient identity involving tensoring and inner products.

Problem 2.7 (2 points)**.** *Prove that, for any vectors $a, b, c, d \in \mathbb{F}_2^n$,*

$$\langle a \otimes b, c \otimes d\rangle = \langle a, c\rangle \cdot \langle b, d\rangle$$

The procedure for testing the tensor structure, which we will denote by T, is given oracle access to $\pi_1 \in \mathbb{F}_2^{2^n}$ and $\pi_2 \in \mathbb{F}_2^{2^{n^2}}$, and operates as follows:

1. Sample random $x, y \leftarrow \mathbb{F}_2^n$
2. Compute $z \leftarrow C^{\pi_2}(x \otimes y)$
3. Check whether $z = \pi_1[x] \cdot \pi_1[y]$

Problem 2.8 (1 point)**.** *Suppose π_1 and π_2 are equal to H_u and H_v, respectively, where $v = u \otimes u$. Prove that T^{π_1,π_2} always accepts.*

Problem 2.9 (4 points)**.** *Suppose π_1 and π_2 are 0.01-close to H_u and H_v, respectively, where $v \neq u \otimes u$. Prover that T^{π_1,π_2} accepts with probability at most $(0.04 + 3/4)$.*

This soundness can again be amplified by repeating T a constant number of times. Then, if π_1 and π_2 are 0.01-close to H_u and H_v where $v \neq u \otimes u$, the tester can be made to reject with probability 0.99.

2.4 Testing Satisfaction

The final task of the verifier is to check that the vector $v \in \mathbb{F}_2^{n^2}$ encoded by π_2 satisfies the linearised version $A_Q v = b_Q$ of the system of quadratic equations we started with.

Problem 2.10 (5 points)**.** *Construct an algorithm S that, given access to an oracle for $\pi \in \mathbb{F}_2^{2^n}$ that is 0.01-close to H_v for some $v \in \mathbb{F}_2^n$, and a system of linear equations $(A, b) \in \mathbb{F}_2^{m \times n} \times \mathbb{F}_2^m$, checks that $Av = b$. S^π should make $O(1)$ queries, have perfect completeness (if $\pi = H_v$) and constant soundness error.*

Hint 2.4. *In the PCP for ML that we saw in class, we made use of the property that two distinct linear functions over $\mathbb{F}_2^n$ agree on exactly half the inputs. For this to be sufficient, we made the assumption that the matrix A is full rank. You do not have this assumption here. How will you get around this?*

2.5 Putting everything together

Problem 2.11 (5 points)**.** *Write down the description of the PCP verifier for MQ, and prove its completeness and soundness.*