CPT Notes, Graph Non-Isomorphism, Zero-Knowledge for NP and Exercises

Ivan Damgård

1 Graph Nonisomorphism

We look at the problem of graph nonisomorphism. In this protocol, P is trying to convince V that two graphs G_0 and G_1 are not isomorphic. We will prove that the protocol below is perfect zero-knowledge.

We assume that the graphs have the same number n of vertices and the same number of edges (otherwise the protocol is not very interesting!). Let $B = \{0, 1\}$. For a graph G with n vertices, making a *random isomorphic copy* of G will mean: choose a random permutation ϕ on n points and compute $\phi(G)$.

The protocol now goes as follows:

Protocol for Graph Nonisomorphism

1. V chooses a random bit α, and constructs a graph H as a random isomorphic copy of G_α.
   He then does the following for $i = 1...s$:
   Choose $\alpha_i \in B$. Construct a pair (H_0^i, H_1^i) of graphs, such that if $\alpha_i = 0$, then H_0^i is a random isomorphic copy of G_0, H_1^i a random isomorphic copy of G_1. Otherwise, H_0^i is a random isomorphic copy of G_1, H_1^i a random isomorphic copy of G_0.
   He sends H and all pairs (H_0^i, H_1^i) to P.
2. P chooses $b_1, ..., b_s$ at random in B, and sends them to V.
3. For each $i = 1...s$, V sends the following to P:
   If $b_i = 0$, the isomorphisms between (H_0^i, H_1^i) and (G_0, G_1) are sent. If $b_i = 1$, an isomorphism from H to one of (H_0^i, H_1^i) is sent.
4. For each value of i, P checks that the appropriate isomorphisms were sent. If not, he stops. Otherwise, he computes (perhaps using his infinite computing power) b such that G_b is isomorphic to H, and sends b to V. If no such b exists, he sends a randomly chosen value.
5. V accepts, if $\alpha = b$, otherwise he rejects.

The above steps are repeated n times.

For intuition, one may first think of a simplified protocol, where V constructs H, sends it to P who computes b and sends it to V who checks that $\alpha = b$. This is an interactive

proof system, which can be argued in a way similar to the theorem below: a cheating prover cannot do better than trying to guess α. It is not zero-knowledge, however. The problem is that the verifier may not follow the protocol. If he does not know which of G_0, G_1 H is isomorphic to, P is giving away information that it may be hard for the verifier to compute on his own. Therefore, the purpose of the additional steps is to convince P that V knows an isomorphism from H to G_0 or G_1, but without revealing which is the case.

Theorem 1. *The above protocol is an interactive proof system for graph nonisomorphism.*

Proof. First, it is clear that if G_0 is not isomorphic to G_1 and (P, V) follows the protocol, then V always accepts: V can always send the correct isomorphisms in step 3, and H can only be isomorphic to one of G_0, G_1, whence α must equal b in step 5.

On the other hand, if G_0 IS isomorphic to G_1, then H is isomorphic to both of G_0, G_1, and the information in step 3 reveals no information about which of the isomorphisms from H to G_0, G_1 is known by V. Hence the probability that $\alpha = b$ is at most $1/2$ in each round, so V accepts with probability at most 2^{-k}

Theorem 2. *The above protocol is perfect zero-knowledge.*

Proof. For any verifier V^*, we describe a simulator M_{V^*}. It begins by putting a suitable number of independent random bits on the random tape of V^*. It then executes the following steps 1-4 to simulate 1 round of the protocol, and repeats this n times to simulate all n rounds.

1. M_{V^*} waits to receive H and s pairs (H_0^i, H_1^i) from V^*.
2. M_{V^*} chooses and sends $b_1, ..., b_s$ as the prover would have done.
3. M_{V^*} waits to receive from V^* a bit string I containing information about isomophisms between H, the (H_0^i, H_1^i)'s and G_0, G_1 as described in step 3 of the protocol.
   If I does not contain the correct isomorphisms, the simulator stops.
   Otherwise we run the following which will either produce an isomorphism between H and one of G_0, G_1, or will prove that no such isomorphism exists. More precisely, we run the following two loops in parallel, and stop when one of the loops exit.
   Loop1 .

   (a) Rewind V^* to the point just after H and the (H_0^i, H_1^i)'s were received.
   (b) Choose a new set of independent values $b'_1, ..., b'_s$ and send them to V^*.
   (c) Wait to receive a bit string I' from V^* containing some isomorphisms.
   If I' is correctly constructed, and if there exists an i such that $b_i \neq b'_i$, find from I and I' isomorphisms from G_0, G_1 to (H_0^i, H_1^i), and from one of (H_0^i, H_1^i) to H. Use these to find an isomorphism from H to one of G_0, G_1, and exit the loop.
   Otherwise go to step 3a

Loop2 .

(a) Arrange all $n!$ permutations on n elements in some fixed order, let ϕ_i be the i'th permutation, and set $i = 1$.
(b) If $\phi_i(H)$ is G_0 or G_1, exit the loop. Set $i = i + 1$. If $i > n!$, exit the loop. Otherwise go to step 3b.

The information now found can be used to compute a correct value of b: either we know an isomorphism from H to G_b, $b = 0$ or 1, so we send b to V. Or we know that no isomorphism exists because we tried all possibilities. In this case we do what the prover would do, namely send a random bit to V.

First, it is clear that the conversation produced by the simulation is distributed exactly as the real conversation: the messages in the first 3 steps are computed by the same algorithms, and the simulation of P's final answer is not sent until we have proof that it is correct.

It is therefore enough to prove that the simulation takes expected polynomial (in n) time. For this, consider the set of 2^s challenges that P can send to V^* in step 2. Call a challenge "good", if V^* answers it correctly, and let g be the number of good challenges.

For convenience, define the time needed to execute steps 1-3 in the protocol to be 1 time unit. Let us now find the mean time M needed to simulate one round:

The probability that we enter the loops above is $g/2^s$. So

$$M = \frac{2^s - g}{2^s} + \frac{g}{2^s}(T + 1)$$

where T is the expected number of iterations of the loops. Thus $M = 1$ if $g = 0$. If $g > 1$, we finish at the latest when we find one good challenge that has not been used in step 2, since this will cause Loop 1 to exit. Hence

$$M \leq 2 + \frac{g}{2^s}\frac{2^s}{g - 1} \leq 4$$

Finally, if $g = 1$, we finish at the latest when Loop 2 exits, which happens after at most $n!$ iterations. So in this case,

$$M \leq 2 + \frac{n!}{2^s}$$

So we are done if we choose s, such that $n!/2^s$ does not grow too quickly. It is easy to see that $n! \leq 2^{nlog(n)}$, so by choosing $s = nlog(n)$, we get that the mean time needed to simulate the whole protocol is less than $4n$

Graph non-isomorphism is not known to be in NP, indeed it is hard to imagine a short witness that would convince you that no isomorphism exists between two graphs. Thus the above shows that the class of languages that have perfect zero-knowledge proofs is probably not contained in NP.

2 Zero-Knowlegde for all of NP

In this section, we will look at how bit commitments can allow us to give a zero-knowledge proof for any problem in NP. Whereas this may look impossible at first sight, fortunately the theory of NP completeness comes to rescue, as we shall see.

Let G be a directed graph on n nodes. We say that G is Hamiltonian, if it contains a so called Hamiltonian cycle, namely a path starting in some node that visits all nodes in the graph exactly once, and comes back to the starting node. The graph Hamiltonicity problem is to decide, given G, whether it is Hamiltonian. This problem is NP complete which will be extremely useful in the following.

To see why this is useful, first recall that if a language L is in NP this means by definition that for any $x \in L$ there exists a so called witness w such that given w, it can be verified in polynomial time that indeed $x \in L$, whereas if $x \notin L$ of course no such witness exists. For instance, L might be the language consisting of composite integers, in which case w may be a non-trivial factorization of x. This means that if P claims that $x \in L$ for some $L \in NP$, we may assume that the prover knows a witness w: either he can compute it as a result of having infinite computing power (as assumed in the interactive proof model), or it may be that he knows w in advance - in the factorization example, it may be the case that P constructed x himself by choosing the prime factors first.

Now, the graph Hamiltonicity problem is NP complete via a so called Karp reduction, which for us means the following: there is a polynomial time algorithm which given any x will compute a graph G_x such that G_x is Hamiltonian if and only if $x \in L$. So both P and V can compute G_x. Furthermore, there is also a polynomial time algorithm which from $x \in L$ and a witness w will compute a Hamiltonian cycle in G_x. So we see that in order to make a zero-knowledge proof system for $L \in NP$, it suffices to construct one for the Graph Hamiltonicity problem.

To describe this protocol, we need the *incidence matrix* M_G of a graph G. This is simply a matrix that completely specifies G: if G has n nodes, M_G is an $n \times n$ matrix, such that the (i, j)'th entry is 1 if there is an edge in G from node i to node j, and is 0 otherwise. Note that if we have a permutation ϕ on n elements and apply it to G as in the graph isomorphism protocol, we can obtain the incidence matrix of $\phi(G)$ easily from M_G by permuting the rows and columns of M_G according to ϕ. The resulting matrix is called $\phi(M_G)$. Note that a Hamiltonian cycle σ in G can be specified by pointing out a set of n entries in M_G such that all the entries are 1, and such that they specify a cycle that visits every node exactly once. When applying a permutation ϕ to G, we will let $\phi(\sigma)$ mean the set of entries in $\phi(M_G)$ that correspond under ϕ to entries in σ. Of course, if σ is a Hamiltonian cycle in G, then $\phi(\sigma)$ is a Hamiltonian cycle in $\phi(G)$.

The protocol assumes that a bit commitment scheme is available and goes as follows:

Protocol Graph Hamiltonicity

1. Execute the set-up procedure of the bit commitment scheme, with security parameter value n. This results in a public key pk being produced. More precisely, if the commitment scheme is computationally binding, V will run a generator $\mathcal{G}(1^n)$, get a public key pk and send it to P. If the scheme is unconditionally binding, P will run $\mathcal{G}$ and send pk to V.
2. Now, given graph G on n nodes, repeat the following n times:
   (a) P chooses a random permutation ϕ and computes $\phi(M_G)$ and commits to all bits in this matrix. This results in a matrix C of commitments, where $C_{ij} = \texttt{commit}_{pk}(b_{ij}, r_{ij})$, where b_{ij} is the ij'th entry in $\phi(M_G)$ and where r_{ij} is a random string chosen by the prover. P sends C to V
   (b) V chooses a random bit b.
   (c) If $b = 0$, P must send ϕ to V and open all commitments in C. V checks that all commitments were correctly opened and that the resulting matrix of bits is indeed $\phi(M_G)$.
   If $b = 1$, P uses his knowledge of a Hamiltonian path p in G: he opens precisely the entries in C that correspond to entries in $\phi(p)$. V checks that all opened commitments were correctly opened, that a 1 was opened in all cases, and that the entries opened specify a path that visits all nodes exactly once.

We can now prove two results on this protocol, that differ in their conclusions depending on what type of commitment scheme is used.

2.1 Interactive Argument for NP

Theorem 3. *If a perfectly hiding and computationally binding commitment scheme is used, then the Graph Hamiltonicity protocol is a perfect zero-knowledge interactive argument for Graph Hamiltonicity.*

Remark 1. If we had used a commitment scheme that was not perfect, but only unconditionally hiding, the protocol would have been statistically and not perfect zero-knowledge. This can be shown in much the same way as we show perfect ZK here, but the proof is longer and more tedious.

To prove the theorem, first observe that completeness is clear by inspection of the protocol. In particular, since arguments restrict provers to polynomial time, there must be an efficient algorithm for the honest prover, given some private auxiliary input. It is clear that the prover in our protocol above runs in polynomial time, as long as he knows a Hamiltonian cycle for G in advance. For soundness, we need the following

Lemma 1. *Consider any matrix of commitments C sent in step (a). Let R_0 be a possible reply from a (not necessarily honest) prover to $b = 0$, similarly R_1 is a possible reply to $b = 1$. If V accepts both R_0 and R_1 and if every commitment from C that is opened in R_1 is also opened to reveal 1's in R_0, then G is Hamiltonian.*

Proof. This can be seen by just noting that R_1 specifies a Hamiltonian path by opening a set of commitments in C to 1's, since V accepts R_1. Since these commitments are also opened to reveal 1's in R_0, this tells us how to transform this path using ϕ^{-1} to a Hamiltonian cycle in G.

We can now prove soundness. Recall that for an interactive argument, the soundness requirement says that for any polynomial time prover, and any non-Hamiltonian input graph G, the probability that V accepts is negligible. So for contradiction, let a polynomial time prover P^* be given such that for infinitely many values of n, there exists a non-Hamiltonian graph G on n nodes, for which P^* can convince the verifier with non-negligible probability, i.e., larger than $1/f(n)$, for some polynomial $f()$. We will show that our assumptions on P^* imply we can efficiently break the binding property of the commitment scheme, thus contradicting the assumption that the scheme was computationally binding.

So suppose someone gives us a public key pk generated with security parameter value n. Then we can run the following algorithm to break the binding property w.r.t. pk. The idea of the algorithm is to play verifier against the prover and execute the protocol, but use rewinding to try to make him answer both $b = 0$ and $b = 1$ in one of the n iterations.

1. Start P^* with a random set of coins $rand$, i.e. $rand$ is simply a random bit string (which we keep fixed in the following). Give pk to P^*.
2. Get a matrix C of commitments from P^*. Now we send $b = 0$ to P^* and get his answer R_0. If R_0 is correct, save the state St_0 that P^* is in now.
3. Rewind P^* to its state just before we sent $b = 0$, send $b = 1$ and get a reply R_1. If R_1 is correct, save the state St_1 that P^* is in now.
   Now, if both R_0 and R_1 are correct, it follows from the above Lemma that we can now output a commitment c plus two different openings of c. If this was not the case, then it would follow that G was Hamiltonian which is a contradiction. If none of the replies were correct, we stop and output failure. If only one of the replies, say R_b was correct, we put P^* in state St_b. If we have now completed the last of the n iterations of the protocol, stop and output failure. Otherwise go to Step 2.

Let $\epsilon(n)$ be P^*'s probability of making the verifier accept, and $\epsilon_{rand}(n)$ be P^*'s probability of making the verifier accept, given that the set $rand$ of coins is used. So $\epsilon(n)$ is the average of all the $\epsilon_{rand}(n)$'s. Since we assumed that $\epsilon(n)$ is large ($\geq 1/f(n)$), many of the $\epsilon_{rand}(n)$'s must be relatively large, since otherwise the average could not be large. More precisely, we

have $\epsilon(n) = \sum_{rand} \epsilon_{rand}(n) Pr(rand)$ where the sum is over all choices of *rand*. From this and $\epsilon(n) \geq 1/f(n)$, it follows that with probability at least $1/(2f(n))$, a random choice will produce a *rand* for which $\epsilon_{rand}(n) \geq 1/(2f(n))$.

So assume we have chosen such a value of *rand* initially in the above procedure. We can now observe that the procedure always finds the first time where P^* can answer both $b = 0$ and $b = 1$, and only fails if no such situation occurs. But if no such situation occurs, then P^* can answer at most one b value in every of the n iterations, thus his probability of making V accept would be at most 2^{-n}. This is a contradiction, since $2^{-n} < 1/(2f(n))$ for all large enough n. So summarizing, we have constructed an algorithm which breaks the binding property of the *commit*() function with probability at least $1/(2f(n))$, for infinitely many values of n. This contradicts the computational binding property of the commitment scheme, so no prover of the type we assumed can exist, and the protocol is therefore a sound interactive argument.

To argue perfect zero-knowledge, we observe that in the the protocol the verifier first sends a public key *pk* to the prover, and then follows something that has exactly the form required for the rewinding lemma that we covered earlier. It is therefore sufficient to construct an honest verifier simulator M that uses *pk* as input. The actual simulator would first receive *pk* from the (arbitrary) verifier V^*, stop if *pk* is invalid (as P would have done) and otherwise run the simulation (based on M) that follows from the rewinding lemma.

The algorithm for M is as follows:

1. M chooses a random bit b.
2. If $b = 0$, follow the prover's algorithm to construct C as a set of commitments to $\phi(M_G)$ for a randomly chosen ϕ.
3. If $b = 1$, construct C as follows: choose a random set of entries r_{entr} in C, such that r_{entr} specifies a path that visits every node exactly once. For instance, we can start in node 1, and then we keep moving to a randomly chosen node that we did not visit yet, until we have visited all nodes, and finally we go back node 1. One can observe that this means that r_{entr} corresponds to a Hamiltonian path in $\phi(G)$ for a uniformly chosen ϕ. For every entry in r_{entr}, put a commitment to 1, for all other entries put a commitment to 0.
4. Output C, b and opening information for the relevant commitments in C (all of them if $b = 0$, the ones from the path r_{entr} if $b = 1$.

To see why this works, we can use an argument very similar to the one for graph isomorphism: observe that the perfect hiding property means that commitments to 0 have the same distribution as commitments to 1. Therefore the matrices of commitments C produced by P have the same distribution as the C's produced in the simulation, regardless of the choice of c. b is a random bit as it should be for an honest verifier, and the final message clearly has exactly the right distribution, given the first two.

2.2 Interactive Proof System for NP

Theorem 4. *If a computationally hiding and unconditionally binding commitment scheme is used, then the Graph Hamiltonicity protocol is a computational zero-knowledge interactive proof system for Graph Hamiltonicity.*

The proof for this is similar in structure to the proof of the previous result. Completeness is still trivial. As for soundness, this follows again from Lemma1: in this case, the (cheating) prover P^* sends pk to the verifier. By definition of unconditional binding, the verifier accepts an incorrectly generated public key with negligible probability. On the other hand, if the function is correctly generated, it is *impossible* to open a commitment in two distinct ways. The above Lemma therefore immediately implies that if G is not Hamiltonian, then P^* can answer at most one b-value in each iteration (regardless of his computing power), and so the overall error probability, given that $pk$was good, is at most 2^{-n}.

Zero-knowledge can be demonstrated using the same simulator algorithm as above, with only one change, namely that initially, the simulator generates and sends pk to V^*. After this, we apply the simulation that is used in the proof of the rewinding lemma, based on honest verifier simulator M as constructed above. In this case, we will not get perfect zero-knowledge, but we will argue below that M using a computationally hiding commitment scheme will produce conversations that computationally indistinguishable from real conversations by honest prover and verifier. Since the rewinding lemma holds also for computational indistinguishability, we are done.

The required result on M follows immediately from the lemma below which generalizes the computational hiding property from one commitment to the case where many commitments are given (as in the first message of the Graph Hamiltonicity protocol).

Lemma 2. *Let pk be the public key of a computationally hiding bit commitment scheme. Then for any two bit strings $B_0 = b_1, .., b_m$, $B_1 = b'_1, ..., b'_m$, the distributions C_0, C_1 defined by*

$$C_0 : pk, \texttt{commit}_{pk}(b_1, r_1), ..., \texttt{commit}_{pk}(b_m, r_m) \quad C_1 : pk, \texttt{commit}_{pk}(b'_1, r'_1), ..., \texttt{commit}_{pk}(b'_m, r'_m)$$

are computationally indistinguishable.

Proof. We define a sequence of machines $M_0, ..., M_m$. Machine M_i will output a public key pk and commitments to $b_1, .., b_{m-i}, b'_{m-i+1}, .., b'_m$. Clearly, M_0 produces C_0 as output while M_m produces C_1. We claim that $M_i \sim^c M_{i-1}$. If $b_i = b'_i$, this is clear. Otherwise, assume for contradiction that we have a successful distinguisher D for M_i, M_{i-1}. We will show that we can use D to break the hiding property of the commitment scheme. Say we are given pk and a commitment c. Construct a string of commitments by putting

commitments to $b_1, ..., b_{i-1}$, then c, and then commitments to $b'_{i+1}, ..., b'_m$. Finally give pk and the commitments to D. If D says it thinks the string was from M_{i-1}, we output "c contains b'_i", else we output "c contains b_i". This breaks the hiding property, since if c really contains b'_i (b_i), we have generated exactly the same kind of output as $M_{i-1}(M_i)$. From a previous exercise, we now have that $M_{i-1} \sim^c M_i$ implies $M_0 \sim^c M_m$.

Note that the lemma is true, even if m is allowed to grow polynomially with the security parameter n. The lemma implies that the first message generated by M is computationally indistinguishable from the message generated by the prover, the challenge bit has the same distribution in the two cases, and the final message is also exactly correctly generated by M, given the first two messages.

3 Exercises

EXERCISE 1. Consider the following language consisting of pairs of k-bit integers: $L = \{n, x \mid \text{ there exists } y, \text{ such that } y^2 = x \bmod n \text{ and } gcd(n, x) = 1\}$. We will consider a zero-knowledge proof for L, i.e. the prover shows the verifier integers n, x and claims that x is a square modulo n, in other words, x is a quadratic residue. As we shall see, if the prover knows y such that $y^2 = x \bmod n$, then he can efficiently conduct the protocol below.

In the following, $Z_n = \{0, 1, ..., n-1\}$, and Z_n^* will denote $\{x \in Z_n \mid \ gcd(x, n) = 1\}$.

1. V checks that $gcd(x, n) = 1$ and rejects if this is not the case.
2. Repeat the following k times:
   (a) P chooses r at random in Z_n^* and sends $a = r^2 \bmod n$ to V.
   (b) V chooses a random bit b and sends it to P.
   (c) P sends $z = ry^b \bmod n$ to V, who checks that $z^2 = ax^b \bmod n$ and that $gcd(z, n) = 1$.

Show that this protocol is a perfect zero-knowledge proof system for L.

EXERCISE 2. This exercise shows a way for a prover to show in perfect zero-knowledge that a given pair (x, n) satisfies that x is not a square mod n, that is, that there exist no y such that $y^2 \bmod n = x$. The protocol is only secure against cheating by a polynomial time bounded prover, that is, it is an interactive argument (rather than a proof).

The protocol works as follows, where we assume that the prover knows the factorization of n in advance.

1. The verifier V checks that $gcd(x, n) = 1$ and rejects if not. V sends to the prover P the public key pk for a perfectly hiding commitment scheme. P checks the key and stops the protcol is verification fails.

2. V chooses a bit b at random, $y \in Z_n^*$ at random and sends $a = y^2 x^b \bmod n$ to P.
3. P stops if $a \notin Z_n^*$. Else, he computes a bit c, where $c = 0$ if a is a square mod n and $c = 1$ otherwise. He sends $d = \texttt{commit}_{pk}(r, c)$ to V.
4. V sends b, y to P.
5. P checks that $a = y^2 x^b \bmod n$, and stops the protocol is not. Otherwise, he sends r, c to V (opens the commitment).
6. V checks that $c = b$ and rejects if not.

The above is repeated k times where k is the bit length of n.
Answer the following:

1. Show that if x is not a square modulo n, then $a = y^2 x \bmod n$ is not a square mod n either. Use this to show that the protocol is complete. You may use without proof that if the prover knows the factorization of n, then he can efficiently decide if a given number is a square mod n.
2. Show that if x is a square mod n, then it is impossible to decide the value of b given $a = y^2 x^b \bmod n$. This essentially implies soundness, although a formal proof is needed to show that one has to break the binding property of the commitment scheme to cheat. You are not required to give this proof here (but you are welcome to giving it a try!).
3. Consider the following simulator:
   (a) Get pk and a from V^*. If pk is invalid, stop (as P would have).
   (b) Commit to a random bit c and send the commitment d to V^*.
   (c) Get b, y from V^*. If they are incorrect, stop. Otherwise, we have a problem, because the prover at this point would open his commitment, and what we have committed to is a random bit which is not always correct. But instead, we can use that we now know what the right answer is, so we can rewind and then commit to the right bit. Run the following loop:
      i. Rewind V^* to point where it receives a commitment from P. Commit to the (correct) value b, using commitment $d' = \texttt{commit}_{pk}(b, r')$.
      ii. if V^* in response to d' sends incorrect values b', y', go to step 3(c)i. If correct values b, y are sent, output conversation $pk, a, d', (b, y), (b, r')$ and stop the loop.

   Show that this runs in expected polynomial time. For this this, consider the probability p that V^* sends correct values b, r it step 4 after seeing a random commitment, and do an argument similar to (but simpler than) the one for the graph non-isomorphism simulator. Finally argue (informally) that the simulator output is distributed exactly as the real conversation.

EXERCISE 3. This exercise concerns a protocol for proving that a given element in a group is contained in the subgroup generated by another element. For concreteness, we

describe it in the multiplicative group modulo a prime number p, i.e., Z_p^*. Consider the following protocol for P and V:

- Input to P and V: a prime p and $\alpha, x \in \mathbf{Z}_p^*$, $k = log_2(p)$.
- Input to P: y, so that $\alpha^y = x \ mod \ p$.

1. V checks that $gcd(x, p) = gcd(\alpha, p) = 1$ and rejects if this is not the case.
2. P chooses r at random in $[0, p-2]$, and sends $a = \alpha^r \bmod p$ to V.
3. V chooses b at random $\{0, 1\}$ and sends b to P.
4. P sends $z = (r + by) \bmod (p-1)$ to V.
5. V checks that $\alpha^z = ax^b \bmod \ p$. If OK, then accept, otherwise reject.

The above steps are repeated k times.

Now show the following:

- If $x \in < \alpha >$, V always accepts.
- If $x \notin < \alpha >$, V accepts with probability at most 2^{-k} on interaction with any prover P^*.
- (P, V) is perfect zero-knowledge, i.e. there exists a simulator, which produces output distributed *exactly* as the conversation between an arbitrary verifier V^* and P.

EXERCISE 4. Consider the unconditionally hiding commitment scheme based on discrete logarithms, where the public key is $pk = (p, g, y)$ for a prime p a generator g of Z_p^* and $y \in Z_p^*$. And a commitment to b using randomness r has form $commit_{pk}(r, b) = y^b g^r \bmod p$. The randomness r is chosen uniformly from $Z_{p-1} = \{0, 1, ..., p-2\}$.

Suppose a prover P has committed to bits b_1, b_2 using commitments c_1, c_2 where $b_1 \neq b_2$. Now P wants to convince the verifier V that the bits are different. We claim he can do this by sending to V a number $s \in Z_{p-1}$ such that $c_1 c_2 = y g^s \bmod p$.

- Show how an honest P can compute the required s, and argue that the distribution of s is the same when $(b_1, b_2) = (0, 1)$ as when $(b_1, b_2) = (0, 1)$. This means that V learns nothing except that $b_1 \neq b_2$.
- Argue that if P has in fact committed in c_1, c_2 to $(0, 0)$ or $(1, 1)$, he cannot efficiently find s as above unless he can compute the discrete logarithm of y.
- Argue in a similar way that P can convince V that he has committed to two bits that are *equal* by revealing s such that $c_1 c_2^{-1} = g^s \bmod p$.

EXERCISE 5 Assume P commits to two strings $b_1, ..., b_t, b'_1, ..., b'_t$ using commitments $c_1, ..., c_t, c'_1, ..., c'_t$ as in Exercise 4. He claims that the strings are different, and wants to convince V that this is the case while revealing no extra information. Note that he cannot point to an index j where $b_j \neq b'_j$ and use the above method on c_j, c'_j. This would reveal *where* the strings are different. Instead consider the following protocol:

1. P chooses a random permutation π on the set of indices $\{1, ..., t\}$. He computes, for $i = 1..t$ a commitment $d_i = commit_{pk}(s_i, b_{\pi(i)})$ and $d'_i = commit_{pk}(s'_i, b'_{\pi(i)})$ In other words, permute both strings randomly and commit bit by bit to the resulting strings. Send $d_1, ..., d_t, d'_1, ..., d'_t$ to V.
2. V chooses a random bit b, sends it to P.
3. If $b = 0$, P reveals π and uses the above method to convince V for all i that $c_{\pi(i)}$ contains the same bit as d_i. Similarly for $c_{\pi(i)}$ and d'_i. If $b = 1$, P finds a position i, where $b_{\pi(i)} \neq b'_{\pi i}$ and uses the above method to convince V that d_i, d'_i contain different bits.

- Completeness: Ague that an honest prover always convinces the verifier.
- Soundness: Show that if P can, for some set of commitments $d_1, ..., d_t, d'_1, ..., d'_t$ answer V correctly both for $b = 0$ and $b = 1$, then there is at least one j, where P can open c_j, c'_j to reveal different bits. Note that we assume above that P really has committed to the two bit strings, that is, we assume he knows how to open the commitments $c_1, ..., c_t, c'_1, ..., c'_t$. The protocol in this exercise does not verify that P knows this - if one wants to check this, there are other protocols one can use.
- Zero-Knowledge: Sketch a simulator for this protocol. Hint: given commitment c, if you set $d = cg^{-s} \bmod p$, then $cd^{-1} = g^s \bmod$. This means that even if the simulator does not know how to open c, it can create d and fake a proof that d contains the same bit as c. You do not have to formally prove that your simulator works