1 Perfect ZK for Graph Isomorphism

1.1 Review of Protocol

Recall the graph isomorphism protocol as follows:

1. Prover P sends a random H permuted from G_0
2. Verifier sends back a random challenge bit b
3. Prover shows isomorphism between $G_b \cong H$

1.2 Achieving Perfect Zero-Knowledge

In theory, zero knowledge involves creating identical transcript output distributions for all successful views when $w \in L$. The simulator for the above graph isomorphism protocol works to simulate the output distribution for all, even potentially dishonest, verifiers V^*. To motivate the point of simulator being able to rewind the tape we have the following adversary V^*.

1. Fix a particular H^* (for all runs of V^*) and if H (from prover) $\neq H^*$, immediately crash. Else, return $b = 1$.
2. Continue the protocol normally.

Due to the nature of the Zero Knowledge definition, we need the simulator to be able to emulate the output distribution of messages as the protocol above. However, consider the case where the simulator chooses a random H to send to the verifier, it will usually send an $H \neq H^*$. In this case, the simulator would not be able to output anything, since the verifier crashes immediately. To reconcile this, the simulator has the ability to rewind the tape to a previous state. Given this ability, it will always be able to retry until it prodcues outputs that can also be produced by a real interaction. In this case, the only possible output for a real interaction with P and V^* is $(H^*, 1, \pi)$, and it can be seen that the only possible output that our simulator can produce is also that, since it will rewind and try again in any other case.

2 Graph Non-Isomorphism in ZK

Recall the original graph-nonisomorphism interactive proof:

1. Verifier chooses a random $b \leftarrow \{0, 1\}$ and random permutation $H = \pi(G_b)$ and sends to P.

2. P has to determine whether H is from G_0 or G_1 and returns the bit b to verifier.

3. This process is repeated k times, in which case V accepts or until P returns a $b' \neq b_k$ in which case V rejects.

Although subtle, this proof is not zero knowledge. Consider a dishonest verifier V^* who has some H_b that they know is a permutation of either G_0 or G_1. If they send this H_b to the prover, they can actually gain information by learning what graph H_b is isomorphic to. In the context of a simulator, if a dishonest verifier always accepts, then a simulator who rewinds when something rejects would never rewind, so it could output an incorrect corresponding b to permutation $H = \pi(G_{1-b})$. To address this, we want to design a protocol in the following form.

1. Verifier sends $H = G_b$ to the prover.

2. Verifier proves to P that it knows what b is.

3. Prover returns b to V.

In order to accomplish this, the verifier V selects b in the following, more complex manner:

- Select a random $b \leftarrow \{0, 1\}$, as well as random permutations of labels π_0, π_1.

  Note: Since we need to demonstrate the graphs are **not** isomorphic, it has to be the case that the two graphs have the same number of nodes and edges, and for simplicity we assume they are labeled with numbers $1...n$, therefore, both permutations can apply to either graph.

- V generates pair (H_0, H_1) to send to P:
  - If $b = 0$, V sends $(\pi_0(G_0), \pi_1(G_1))$
  - If $b = 1$, V sends $(\pi_0(G_1), \pi_1(G_0))$

Another notation used is $(A, B) \cong (C, D)$, which simply means that $(A \cong C \wedge B \cong D) \vee (A \cong D \wedge B \cong C)$, in other words, the *pair* of graphs are isomorphic. I write that (A, B) maps to (B, C) which means the same thing as the previous sentence. With this new technique, the new ZKIP for GNI is as follows:

1. V selects a random bit b as described generating (H_0, H_1).

2. V generates a uniformly random bitstring $W \xleftarrow{\$} \{0,1\}^k$ and k pairs $(D_{2i-1}, D_{2i}) \cong (H_0, H_1)$.

   Now, V sends k commitment pairs to P:

   - If $w_i = 0$, V sends the pair in the form $(\pi_{2i-1}(H_0), \pi_{2i}(H_1))$
   - If $w_i = 1$, V sends $(\pi_{2i-1}(H_1), \pi_{2i}(H_0))$

   Note: This is done to 'randomize' the order in which the two permutations are sent within the pair so the prover cannot find b from the answer in step 4.

3. The prover returns a challenge string $Q = q_1...q_k$ to V which is selected uniformly from $\{0,1\}^k$.

4. For all q_i, if $q_i = 0$, V creates finds the pair of permutations $\Pi_i = (\pi_{2i-1}, \pi_{2i})$ where, when applied to the ith commitment pair in the second step, returns the original pair of graphs (G_0, G_1). In other words, for $(D_{2i-1}, D_{2i}), (\pi_{2i-1}(D_{2i-1}), \pi_{2i}(D_{2i})) = (G_0, G_1)$ or (G_1, G_0) (Ordering based on if w_i was 0 or 1). On the other hand, $q_i = 1$ means that V has to return Π_i such that (π_{2i-1}, π_{2i}) applied to (D_{2i-1}, D_{2i}) returns either (C_0, C_1) or (C_1, C_0). This demonstrates that V ***already knows*** the value of b that it is asking P to find.

   (a) For V to be able to consistently demonstrate Π_i from $(D_{2i-1}, D_{2i}) \rightarrow (G_0, G_1)$ **and** (H_0, H_1) (order irelevant), then they must also be aware of how to get from $(G_0, G_1) \rightarrow (H_0, H_1)$. Recall that in an honest interaction, this is exactly the first step of the protocol, so an honest verifier will always be able to answer the challenge sent by P.

   (b) If V does not know b, then it also cannot know Π from $(G_0, G_1) \rightarrow (H_0, H_1)$. Thus, for ALL pairs $(D_{2i-1}, 2i)$, it cannot map that pair to both (G_0, G_1) AND (H_0, H_1) (Because if it could, then it could map $(G_0, G_1) \rightarrow (D_{2i-1}, D_{2i}) \rightarrow (H_0, H_1)$, and would therefore know $(G_0, G_1) \rightarrow (H_0, H_1)$ and therefore know b).

   (c) At **best** a cheating verifier could try and fool P by sending (D_{2i-1}, D_{2i}) from either (G_0, G_1) or (H_0, H_1) and hope that the corresponding challenge bit q_i corresponds to the correct guess. But this can be seen to line up for all k challenge bits with probability at most $\frac{1}{2^k}$.

   Thus, the verifier can convince P that it already knows b.

5. Lastly, P returns b to V as they can only know whether $H_0 = \pi_0(G_0)$ or $\pi_0(G_1)$ if G_0 and G_1 are not isomorphic. This entire protocol is repeated k' times.

Instead of a formal proof of whether or not this is zero knowledge, a sufficient intuitive argument that it is ZK is that P only sends V a random k-length bitstring Q and an answer bit b which we demonstrated V must already know, so there is no way that it can learn anything from this interaction.

3 5-Round ZK For Graph Isomorphism

Review of Graph Isomorphism protocol:

1. P sends H, a random permutation of G_0.
2. V sends b.
3. P returns π where $\pi(H) = G_b$.

This protocol is ZK, however, the protocol takes k rounds, which can be very expensive. Firstly, we consider an incorrect constant round ZK protocol for GI:

1. P sends $H_1...H_k$, which are all random permutations of G_0.
2. V returns challenge bits $b_0...b_k$
3. P sends $\pi_0...\pi_n$ where $\pi_i(G_i) = H_i$.

Intuitively, this seems zero knowledge since it is difficult to see a way V can learn anything. To demonstrate that it is not however, consider the following hash function:

(Not very important)
Merkle Hash Tree: Consider a collision resistant hash function $H : \{0,1\}^{kn} \to \{0,1\}^n$ constructed from a smaller hash function H' that maps from $\{0,1\}^{2n} \to \{0,1\}^n$. It does the following, it repeatetly creates a new string S (originally of length kn) that is approximately half the length of the previous string in a iterative manner. For every block of $2n$, it uses H' to half the length of the block to n and connects all hashed blocks to create the next string. This is done until the string is of length n.

Then we have a dishonest verifier V that sends bitstring B to the prover based on the hashed value of $H_1...H_k$. Thus the simulator cannot rewind

and change H_i without drastrically changing the corresponding bitstring in the interaction. Therefore it would have to correctly guess the bitstring which can only happend with negligible probability.

To create a ZK protocol for Graph Isomorphisms, we need to something called a commitment protocol. Consider the following coin flip protocol between two parties, Alice and Bob.

1. Bob flips a coin getting bit $b \xleftarrow{\$} \{0,1\}$ and sends commitment(b) to Alice.
2. Alice flips a coin b' and sends b' to Bob.
3. Bob then 'opens' b by sending Alice a method of unlocking his original coinflip.
4. Alice and Bob both locally compute $b \oplus b'$ to find the result of the coinflip.

3.1 ZK Protocol

With this idea, we have the following ZKIP for Graph Isomorphisms:

1. Firstly, P generates (A_0, A_1) from G_0 and sends them over to V.
2. V chooses a random bistring $B = b_1...b_k$ and commits to it by doing the following.
   (a) If $b_i = 0$, send commitment pair (Z_{2i-1}, Z_{2i}) as random permutations $(\pi_{2i-1}(A_0), \pi_{2i}(A_1))$.
   (b) If $b_i = 1$, send the pair as $(\pi_{2i-1}(A_1), \pi_{2i}(A_0))$.

   Each of the k pairs is sent to P.
3. P sends $H_1...H_n$, which are all random permutations of G_0, to V.
4. V opens the commitments by sending the permutations that maps pair (Z_{2i-1}, Z_{2i}) to (A_0, A_1) in the case that $b_i = 0$ and to (A_1, A_0) if $b_i = 1$. Notice, that if they can send a permutation Π that cheats, then they used a permutation from $A_0 \to Z_i$, but were able to come up with proof permutation π from $Z_i \to A_1$ meaning that they must know the permutation from $A_0 \to A_1$ to cheat.

5. After receiving the commitment for all k pairs, P sends the proof that $A_0 \cong A_1$ as well as the k bits that correspond to B selected by the verifier. The reason for the proof that $A_0 \cong A_1$ is that if the two graphs are not isomorphic, then the prover could discover b_i simply by testing Z_i against A_0 and A_1. Since they are isomorphic, the prover has no way of telling which bit was selected by V as each Z_i is a random permutation of G_0 and G_1.

To demonstrate the zero knowledge property we have the following simulator:

1. Generate (A_0, A_1) from G_0 and send to V.
2. After receiving the k pairs (Z_{2i-1}, Z_{2i}), generate $H_1...H_n$ from G_0 and send to V.
3. After receiving commitments $\Pi_1...\Pi_k$ determine challenge $B = b_1...b_k$, rewind the tape, and generate $H_i = \pi_i(G_{b_i})$.
4. The verifier opens their commitments to the simulator, and the simulator returns a proof that $A_0 \cong A_1$ and $\pi(H_i) \to G_b$ for all i.

In step 4 above, it assumes that the verifier is honest, however, since perfect zero knowledge requires identical output distribution with even a cheating verifier, a verifier that already knows the isomorphism between G_0 and G_1 can change their initial commitment b_i after receiving $H_1...H_k$. If the simulator runs into this case in step 4, it performs the following steps:

4. Rewind to the beginning and generate $A_0 = \pi_0(G_0)$ and $A_1 = \pi_1(G_1)$ and send to V.
5. Receive commitments from V and send H random graphs generated from G_0 to V
6. V should return k commitment pairs (Z_{2i-1}, Z_{2i}) which corresponds to bitstring $B = b_1...b_k$.
7. Rewind back to just before sending all k H graphs and send them all as permuations of A_1.
8. Now, the verifier cannot distinguish these as different from the previous sequence of H's since the isomorphic properties mean that all permutations are equally likely. Therefore, there is not way for them to know not to cheat again. This means that for some Z_i that they sent a proof

for that mapped from A_0, with high probability they now send a proof that it maps to A_1 (In the case they do not cheat this time, redo the entire run from the start w/ new randomness). If this is the case, the simulator now has π_i^0, π_i^1 that maps Z_i to both H_0 and H_1 respectively. In this case, the simulator can now map $G_0 \to H_0 \to Z_i \to H_1 \to G_1$, meaning they have $G_0 \to G_1$, which allows them to manually compute everything since they have the isomorphism between G_0 and G_1. Therefore, no matter what bits V sends, the simulator now has the ability to make the permuation from $G_{1-b} \to G_b \to A_b$.

Extra notes on protocol

- **Why make both A_0, A_1 from G_0**: If $A_0 = \pi(G_0)$ and $A_1 = \pi(G_1)$, the protocol would run the exact same (Since identical distributions). However, the simulator would have to prove that A_0 and A_1 are isomorphic in step 4, which would not be possible for a PPT simulator *unless* it learns of the isomorphism between G_0 and G_1 which it does in the cheating verifier model where it does end up constructing A_b from G_b.

- **Why are commitments made from A_0, A_1**: This protocol would maintain IP properties if the verifier simply made its commitments (Z_{2i-1}, Z_{2i}) from (G_0, G_1), but the prover would have to demonstrate isomorphism between G_0 and G_1 to maintain soundness, which reveals the isomorphism. If they did not, then if G_0 and G_1 were non-isomorphic, the prover could just determine b by finding which G_b Z_i was isomorphic to.