NIZK for 3-Colouring in the Random Oracle Model

Ziyang Jin

Department of Computer Science
University of Toronto
ziyang@cs.toronto.edu

We give a non-interactive zero-knowledge (NIZK) protocol for 3-Colouring in the Random Oracle Model.

1 The protocol

Let $G = (V, E)$ be an undirected graph with n vertices where $V = \{1, ..., n\}$. We will also use n as the security parameter. Let $N := n \cdot |E|$. Let Com be a perfectly-binding, computationally-hiding commitment scheme. Let $\psi : V \to \{\texttt{Red}, \texttt{Green}, \texttt{Blue}\}$ be a 3-colouring of G, provided as an auxiliary input to the prover. Below is the description of the non-interactive zero-knowledge protocol:

1. For each $i \in [N]$, the prover takes an independent random permutation π_i of the colouring ψ such that $\phi_i(j) = \pi_i(\psi(j))$ where $j \in [n]$.
2. The prover computes the commitment $c_{i,j} = \mathsf{Com}(\phi_i(j))$ for all $i \in [N]$ and for all $j \in [n]$.
3. The prover samples a random string $s \leftarrow \{0, 1\}^n$ and queries the random oracle to obtain a string of edges $e_1||e_2||...||e_N \leftarrow O(G, c_{1,1}, ..., c_{N,n}, s)$ where $||$ means concatenation.
4. The prover decommits the colouring of the endpoints of $e_1, ..., e_N$. Denote these decommitments by $\mathsf{decom}(c_{1,j_1})$, $\mathsf{decom}(c_{1,k_1}), ..., \mathsf{decom}(c_{N,j_N}), \mathsf{decom}(c_{N,k_N})$ where $e_i = (j_i, k_i)$.
5. The prover sends $e_1, ..., e_N$, $c_{1,1}, ..., c_{N,n}$, s, and $\mathsf{decom}(c_{1,j_1})$, $\mathsf{decom}(c_{1,k_1}), ..., \mathsf{decom}(c_{N,j_N}), \mathsf{decom}(c_{N,k_N})$ to the verifier.
6. The verifier first checks if $e_1||...||e_N = O(G, c_{1,1}, ..., c_{N,n}, s)$, and then checks if the decommitments are valid, and in the end checks if each e_i has different colours on its endpoints. The verifier accepts if and only if all checks pass.

For completeness, if G admits a 3-colouring, it is clear that the verifier will accept with probability 1. For soundness, if G does not admit a 3-colouring, then no matter how the prover colours the graph, there is at least one monochromatic edge. Thus, in every instance out of N instances, there is at least $\frac{1}{|E|}$ chance that the edge e_i returned by the oracle matches the monochromatic edge. Therefore, over all N instances, there is at most $\left(1 - \frac{1}{|E|}\right)^N = 2^{-O(n)}$ probability that none of $e_1, ..., e_N$ matches a monochromatic edge in their respective instances. Note that a cheating prover can query the oracle on his own to see what edges will be decommitted for a given input. The prover is able to cheat if he finds a set of colourings of G for every instance and a string s such that on input $G, c_{1,1}, ..., c_{N,n}, s$ to the random oracle, none of the edges returned matches a monochromatic edge. Suppose the cheating prover can make q queries before it interacts with the verifier. By union bound, the prover will have at most $q \cdot 2^{-O(n)}$ probability to cheat successfully. So the soundness is $q \cdot 2^{-O(n)}$. Note that the soundness is not negligible if we allow the computationally unbounded prover to query the oracle exponentially many times.

Remark 1. We need the commitment scheme Com to be perfectly-binding such that even computationally unbounded prover cannot cheat by opening the commitments to a different colour from what has been assigned in step 1.

Remark 2. There is also a notion called *observability* when we prove the soundness. Observability means that the verifer can observe the queries made by the prover to the oracle.

2 Proof of Zero-knowledge

In order to prove zero-knowledge of the protocol, we need to use the *programmability* property of the random oracle. Programmability means that the simulator intercepts all the communications between the verifier and the environment, and so the simulator can "program" the response of the random oracle and can respond to the verifier on behalf of the random oracle.

We assume the verifier is malicious. To formally prove zero-knowledge, we need to show a simulator $\mathcal{S}$ such that the output of the verifier in the real execution of the protocol is indistinguishable from the output of the verifier interacting with the simulator. The output of the verifier depends on its view. The view of the verifier includes the graph G, the commitments $c_{1,1},...,c_{N,n}$, the random string s, the edges $e_1,...,e_N$, and the opening of the commitments $\mathsf{decom}(c_{1,j_1}), \mathsf{decom}(c_{1,k_1}),...,\mathsf{decom}(c_{N,j_N}), \mathsf{decom}(c_{N,k_N})$ where $e_i = (j_i, k_i)$. The simulator $\mathcal{S}$ runs as follows:

1. For each $i \in [N]$, the simulator $\mathcal{S}$ independently samples a random edge $e_i \in E$ and colours e_i's endpoints with distinct random colours; for other vertices in the graph, the simulator assigns the same colour (say $\mathtt{Red}$) to them. Call this colouring ϕ_i for $i \in [N]$.
2. The simulator computes the commitment $c_{i,j} = \mathsf{Com}(\phi_i(j))$ for each $i \in [N]$ and $j \in [n]$.
3. The simulator programs the random oracle O on input $(G, c_{1,1},...,c_{N,n}, s)$ to return the edges sampled previously , which is $e_1||e_2||...||e_N$. If the verifier has already queried the oracle with exactly $(G, c_{1,1},...,c_{N,n}, s)$ before, then the simulator aborts.
4. The simulator decommits the colouring of the endpoints of $e_1,...,e_N$. Denote these decommitments by $\mathsf{decom}(c_{1,j_1})$, $\mathsf{decom}(c_{1,k_1}),...,\mathsf{decom}(c_{N,j_N}), \mathsf{decom}(c_{N,k_N})$ where $e_i = (j_i, k_i)$.
5. The simulator sends $e_1,...,e_N$, $c_{1,1},...,c_{N,n}$, s, and $\mathsf{decom}(c_{1,j_1}), \mathsf{decom}(c_{1,k_1}),...,\mathsf{decom}(c_{N,j_N}), \mathsf{decom}(c_{N,k_N})$ to the verifier.

Throughout the simulation, if the verifier sends new queries to the random oracle, the simulator samples a random output from the oracle's range and responds back to the verifier. If the verifier sends previously-asked queries, then the simulator answers with previous answers.

Note that the maclicious verifier can query the oracle on her own before she interacts with the prover. Suppose the verifier queries the oracle q' times before the simulator programs the oracle, then the probability that the verifier queries the oracle with exactly $(G, c_{1,1},...,c_{N,n}, s)$ as input is at most $q' \cdot 2^{-O(n)}$ since there is $1/2^n$ chance the verifier guesses the random string $s \in \{0,1\}^n$ correctly.

Lemma 1. *The output of the verifier in the real execution is indistinguishable from the output of the verifier interacting with the simulator $\mathcal{S}$. We use the same definition as in Lindell's tutorial [Lin17]:*

$$\{\mathsf{output}_{V^*}(P(G,\psi), V^*(G,z))\} \overset{c}{\equiv} \{\mathcal{S}^{V^*(G,z,r,\cdot)}(G)\}$$

where P is the prover, V^* is a non-uniform probabilistic polynomial time verifier treated as a black-box, and z is an auxiliary input to V^*, and r is uniformly distributed and $V^*(G,z,r,\cdot)$ is the next-message function of V^*.

Proof. We prove by hybrid argument. Let Hyb_0 be the scenario where the verifier V^* interacts with the real prover P. Let Hyb_1 be the scenario where the verifier V^* interacts with the real prover P with the following changes in bold font:

1. **The prover samples independent random edges** $e_1,...,e_N$**.** For each $i \in [N]$, the prover takes an independent random permutation π_i of the colouring ψ such that $\phi_i(j) = \pi_i(\psi(j))$ where $j \in [n]$.
2. The prover computes the commitments $c_{i,j} = \mathsf{Com}(\phi_i(j))$ for all $i \in [N]$ and $j \in [n]$.
3. The prover samples $s \leftarrow \{0,1\}^n$ and **programs the oracle to return** $e_1||...||e_N$ **if queried with** $(G, c_{1,1},...,c_{N,n}, s)$**. If the verifier has already queried the oracle with exactly the same input as above, then the prover aborts.**

Note that the prover programs the oracle only after it computes $c_{1,1},...,c_{N,n}$, and the malicious verifier can send at most q' queries to the oracle before it interacts with the prover. Thus, the probability that the prover aborts is at most $q' \cdot 2^{-O(n)}$, which is negligible. If the prover does not abort, it means that $(G, c_{1,1},...,c_{N,n}, s)$ has never been queried to O. Since O is a random oracle, on any input that has not been queried previously, it samples a random output. Therefore, the edges $e_1||...||e_N \leftarrow O(G, c_{1,1},...,c_{N,n}, s)$ returned by the oracle in Hyb_0 is identically distributed as $e_1||...||e_N$ uniformly sampled by the prover in Hyb_1. Therefore, $\mathsf{Hyb}_0 \overset{c}{\equiv} \mathsf{Hyb}_1$. We remark that the negligible difference between Hyb_0 and Hyb_1 comes only from the abort probability.

Define $\mathsf{Hyb}_{1+\ell}$ to be based on Hyb_1, with only one difference described below:

1. The prover samples independent random edges $e_1, ..., e_N$. For each $i \in [N]$, the prover takes an independent random permutation π_i of the colouring ψ such that $\phi_i(j) = \pi_i(\psi(j))$ where $j \in [n]$. **Let S be the set of vertices that are endpoints for $e_1, ..., e_N$. For the first ℓ vertices that are not in S and not assigned $\texttt{Red}$, the prover changes their colour assignment to $\texttt{Red}$.**

Let K be the number of vertices that are not in S and not initially assigned $\texttt{Red}$. So the series of hybrids described above are $\mathsf{Hyb}_2, ..., \mathsf{Hyb}_{1+K}$. We prove $\mathsf{Hyb}_2 \overset{c}{\equiv} \mathsf{Hyb}_{1+K}$ by contradiction. Suppose Hyb_2 and Hyb_{1+K} are distinguishable. Since $K \leq (n-2)N = n(n-2)|E|$ is polynomial in n, by the hybrid lemma, there must exist an $\ell \in [K]$ such that Hyb_ℓ and $\mathsf{Hyb}_{\ell+1}$ are distinguishable. Between these two neighbouring hybrids, the only difference is the colour of one vertex, and the commitment of that colour of that vertex. Note that Com is a perfectly-binding, computationally-hiding commitment scheme. If there exists a distinguisher $\mathcal{D}$ that can distinguish Hyb_ℓ and $\mathsf{Hyb}_{\ell+1}$, then we can use $\mathcal{D}$ to construct a distinguisher that breaks the computationally-hiding commitment scheme (we leave the details to the reader as a simple exercise). Contradiction! Thus, we have $\mathsf{Hyb}_\ell \overset{c}{\equiv} \mathsf{Hyb}_{\ell+1}$ for all $\ell \in [K]$, which implies $\mathsf{Hyb}_2 \overset{c}{\equiv} \mathsf{Hyb}_{1+K}$.

Define $\mathsf{Hyb}_\mathcal{S}$ to be the verifier interacting with the simulator $\mathcal{S}$. We claim that $\mathsf{Hyb}_{1+K} \equiv \mathsf{Hyb}_\mathcal{S}$, i.e., they are identical. To see why, note that the only difference is that in Hyb_{1+K}, the prover first picks a random proper colouring of the whole graph in each instance, then overrides the colours of other vertices that are not endpoints of $e_1, ..., e_N$ and not initially $\texttt{Red}$ to $\texttt{Red}$; while in $\mathsf{Hyb}_\mathcal{S}$, the simulator first assigns independent distinct random colours to the endpoints of $e_1, ..., e_N$ and then colours every other vertex to $\texttt{Red}$. Note that the only difference in the final colouring can only happen in $e_1, ..., e_N$. We claim for any edge $e_i = (j_i, k_i)$ between Hyb_{1+K} and $\mathsf{Hyb}_\mathcal{S}$, the probability of the colour assignments to its endpoints are the same, which is

$$\Pr(j_i = \texttt{Red} \land k_i = \texttt{Green}) = \Pr(j_i = \texttt{Green} \land k_i = \texttt{Blue}) = ... = \frac{1}{6}.$$

This is because if you get any proper 3-colouring for that graph, you can get 5 other proper colourings by rotating the colour classes.

Hence, we have shown $\mathsf{Hyb}_0 \overset{c}{\equiv} \mathsf{Hyb}_\mathcal{S}$ as wanted. □

Remark 3. In 2021, Holmgren, Lombardi, and Rothblum showed that parallel repetition of commit-and-open protocols (such as GMW) does not preserve zero-knowledge if we only assume LWE [HLR21].

References

HLR21. Justin Holmgren, Alex Lombardi, and Ron D. Rothblum. Fiat-Shamir via list-recoverable codes (or: parallel repetition of GMW is not zero-knowledge). In *STOC '21—Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 750–760. ACM, New York, [2021] ©2021.

Lin17. Yehuda Lindell. *How to Simulate It – A Tutorial on the Simulation Proof Technique*, pages 277–346. Springer International Publishing, Cham, 2017.