Vote – Homework

Please answer the following questions. **We don't expect rigorous formal proofs: rather, just a high-level argument from intuition.** Please submit your answers as a **PDF** to Gradescope. Collaboration is allowed and encouraged, but you must write up your own answers and acknowledge your collaborators in your submission.

Due Date: *Monday, March 10th*

1 Fiat-Shamir Heuristic

(1) Explain how we can transform a three-round sigma protocol into a non-interactive zero-knowledge (NIZK) proof via the Fiat-Shamir heuristic in the random oracle model.

(2) In the above NIZK, how does the hash function protect against a malicious prover? How does it protect against a malicious verifier?

(3) Explain how we can transform Schnorr's identification protocol into Schnorr's signature scheme via the Fiat-Shamir heuristic in the random oracle model.

2 ZKP for OR Statement

We'll now construct the zero-knowledge proof (sigma protocol) for the OR statement we need in our project.

In particular, let $\mathbb{G}$ be a cyclic group of prime order q with generator g, let $\mathsf{pk} \in \mathbb{G}$ be the public key for ElGamal encryption, and let $c = (c_1, c_2)$ be an encryption under the key pk. The ciphertext is an encryption of $b \in \{0, 1\}$ with randomness $r \in \mathbb{Z}_q$, namely $c_1 = g^r$ and $c_2 = \mathsf{pk}^r \cdot g^b$. We define the relation as an OR statement:

$$\mathcal{R}_{\mathcal{L}} := \{((\mathsf{pk}, c_1, c_2), r) : (c_1 = g^r \wedge c_2 = \mathsf{pk}^r) \vee (c_1 = g^r \wedge c_2/g = \mathsf{pk}^r)\}.$$

(1) In the Vote handout Sec 1.3.2, we present the ZKP protocol when $b = 1$, namely c is an encryption of 1 and the prover knows the randomness r. Write out the ZKP protocol when $b = 0$, namely c is an encryption of 0 and the prover knows r.

(2) Combining the above protocol with the one in the Vote handout Sec 1.3.2, we have a full ZKP protocol for the OR statement $\mathcal{R}_{\mathcal{L}}$. In either case ($b = 0$ or $b = 1$), the

prover performs two ZKPs simultaneously, one proving c is an encryption of 0 and one proving c is an encryption of 1. For the one that she has a witness, she behaves honestly; for the one that she does not have a witness, she simulates a proof. Explain (intuitively) why the prover cannot simulate both proofs in the ZKP protocol for $\mathcal{R}_{\mathcal{L}}$.

(3) **(Bonus) Proof of Knowledge:** Construct a PPT extractor E to extract a witness by interacting with a prover P^*.

(4) **(Bonus) Honest-Verifier Zero-Knowledge:** Construct a PPT simulator S to generate the view for an honest verifier that has the same distribution as its view in a real execution.

3 Attacks and Defenses

Our voting protocol was crafted very carefully to ensure that no party can gain an advantage. Here we discuss some of the design decisions in the protocol.

(1) How do we ensure that only qualified voters can vote and that every voter can vote at most once?

(2) Explain why we need a blind signature scheme in our protocol. Specifically, if the registrar generates a signature on each voter's (encrypted) vote directly, what information could be leaked about voters? Why do blind signatures prevent this leakage?

(3) Explain why arbiters don't need to sign their partial decryptions if their partial keys are honestly generated.

4 Multiple Candidates

It would be nice to generalize our voting protocol to t candidates for $t > 2$; after all, many election systems consider more than two candidates.

(1) Consider the following construction: to vote for candidate $i \in \{0, \ldots, t-1\}$ for $t > 2$, simply encrypt a vote of i; that is, construct a ciphertext that looks like $(g^r, \mathsf{pk}^r \cdot g^i)$. Then we combine all the votes using homomorphic addition and decrypt the combined ciphertext using threshold decryption, same as before. Explain why this protocol doesn't work.

(2) If we allow each voter to vote for an **arbitrary** number of candidates (between 0 and t), how would you extend our protocol to support multiple candidates?

(3) **(Extra Credit)** If we would like to enforce each voter to vote for **exactly** k candidates (for a particular $k \in [1, t]$), how would you design the protocol?

(4) **(Extra Credit)** If we would like to enforce each voter to vote for **at most** k candidates (for a particular $k \in [1, t]$), namely each voter is allowed to vote for an arbitrary number of candidates between 0 and k, how would you design the protocol?

For the above questions, if you need new zero knowledge proofs, you don't have to provide a detailed description of how they should work, but you need to explain the ideas behind their construction. A valid response might include how you would compose the kinds of zero-knowledge proofs we've already seen in class.

5 (Extra Credit) ZKP for Diffie-Hellman Tuples

(1) Recall the ZKP protocol for Diffie-Hellman Tuples shown in class (Lecture 10, pg. 4). Why does the protocol satisfy soundness?

In particular, if (h, u, v) is *not* a Diffie-Hellman tuple, then for any prover P^* (which could even be computationally unbounded),

$$\Pr[P^*(h, u, v) \leftrightarrow V(h, u, v) \text{ outputs } 1]$$

is extremely small (namely, negligible).

(2) Recall the Decisional Diffie-Hellman (DDH) assumption:

Definition. *Let $\mathbb{G}$ be a cyclic group of order q with generator g, it is computationally hard to distinguish (g^a, g^b, g^{ab}) from (g^a, g^b, g^c), where a, b, c are all randomly sampled from $\mathbb{Z}_q$.*

Assuming the DDH problem is computationally hard, explain why it is impossible to construct a NIZK proof for Diffie-Hellman tuples in the plain model.