

## 1

a. To see this, note that if there are two pairs  $(c, r)$  and  $(c', r')$  such that  $c \neq c'$ , they must differ in at least one bit. Without loss of generality, we assume that  $c_i = 0$  and  $c'_i = 1$ . Therefore, with  $c$  our verifier will receive the value  $s_i = r_i$ . But, with  $c'$  the verifier receives  $s_i = r_i\alpha$ . If the verifier accepts in both situations, then  $\alpha$  must exist and  $a$  really is a square root.

Taking the contrapositive of what we just showed, we see that if  $a \notin QR_n$  (so  $a \in QNR_n$ ), we have, once  $P$  has chosen and sent  $R_1, \dots, R_k$ , that there is not more than one pair  $(c, r)$  for which  $P$  will be able to provide a correct response. In particular for each  $R_1, \dots, R_k$ , (since there must be at least one) there is exactly one pair  $(c, r)$  that, when picked and used in the protocol, will end with the verifier accepting. Thus, when  $P'$  chooses  $R_1, \dots, R_k$  he is essentially betting on which  $c$  the verifier will send. Since the commitment is perfectly hiding and there is only one pair that will work, a cheating prover  $P'$  can do no better than guess randomly at the value of  $c$ , so his probability of guessing correctly and providing  $R_1, \dots, R_k$  such that he will be able to respond successfully to  $V$ 's challenge  $c$  is  $\frac{1}{2^k}$ .

b. Our simulator  $S$  chooses public key  $PK$  for the commitment, and sends it to  $V'$  (step 1), and receives commitment  $C$  (step 2). It then chooses random  $r_1, \dots, r_k \leftarrow Z_n$ , computes  $R_i = r_i^2$ , and sends  $R_1, \dots, R_k$  to  $V'$  (step 3). At this point, it receives  $c, r$  from  $V'$ , and stores  $c$ . If  $C \neq \text{Commit}_{PK}(c, r)$ , it sends  $V'$  an error. Otherwise, it rewinds to just after step 2. Now it has already stored the value  $c$  that  $V'$  will (with all but negligible probability) open the commitment to. Thus, it will prepare  $R_1, \dots, R_k$  in such a way that it will be able to respond to  $c$  successfully. For each  $i \in \{1, \dots, k\}$ , it chooses random  $s_i \leftarrow Z_n$ , and computes  $R_i = s_i^2/a^{c_i}$ . It sends  $R_1, \dots, R_k$  to  $V'$  (step 3 again), and receives  $c', r'$  from  $V'$  (step 4 again). If  $C \neq \text{Commit}_{PK}(c', r')$ , it sends an error to  $V'$ . If  $c \neq c'$ , it aborts. Otherwise, it sends  $s_1, \dots, s_k$  to  $V'$  (step 5). Note that if  $S$  does not abort, then the final  $R_1, \dots, R_k, s_1, \dots, s_k$  that the verifier receives together with  $a, c, C, r$  look identical to an interaction with a real prover.

Thus, we only need to show that  $S$  aborts with negligible probability. Note that  $S$  aborts when  $V'$  comes up with a new opening for  $C$ . Thus, we can show that if a verifier  $V'$  can cause  $S$  to abort with nonnegligible probability, then we can break the binding property of the commitment scheme.

Our reduction will proceed as follows: Suppose there exists  $V'$  which causes  $S$  to abort with nonnegligible probability. Then  $\mathcal{B}$  takes as input commitment public key  $PK$ . It runs  $V'$  with first message  $PK$ . After that, it runs as the simulator does. If after the rewind,  $V'$  gives  $c' \neq c$  (as we have assumed will happen with nonnegligible probability), then  $\mathcal{B}$  outputs  $c, r, c', r'$ , which will break the binding property.

c. To see that this commitment is necessary, we consider the cases where it is not used. If the pair  $(c, r)$  is sent in the clear in step 2, a cheating prover can just tailor his  $R_i$  values in the way that the simulator does above. If the commitment is not sent at all and the pair  $(c, r)$  is sent for the first time in step 4, then we lose our zero-knowledge property. To see this, note that our simulator in (b) relies on the fact that the  $c$  value is fixed in step 2, before it is sent in the clear in step 4.

If the verifier is picking  $c$  fresh right before he sends it, our simulator is no longer valid because it will have to rewind the verifier more than polynomially many times.

## 2

a. Here, we have  $\frac{1}{2}$ -soundness. To see this, note that a cheating prover  $P'$  is trying to prove that an element  $x \in QR_n$  is actually in  $QNR_n$ . If  $c = 0$ , the verifier will send  $r^2$ , which is a random quadratic residue modulo  $n$ . If  $c = 1$ , the verifier will send  $r^2 \cdot x$ , which will also look like a random quadratic residue modulo  $n$ . So, either way, the prover receives a random quadratic residue and will have to simply guess at the value of  $c$  to try and fool the verifier. Since he is guessing, his probability of getting the wrong value is  $\frac{1}{2}$ .

b. A cheating verifier  $V'$  could use an honest prover to find out if any number modulo  $n$  is in  $QR_n$ . So he can just set  $y = x$  for  $x \in \mathbb{Z}_n^*$ , where he wants to find out if  $x$  is a quadratic residue.

c. The best simulator that we can come up with (because it can't know  $p$  and  $q$ ) is one that just guesses at  $c$  and sets its  $c'$  equal to its guess for the value of  $c$ . The simulator has probability  $\frac{1}{2}$  of guessing this correctly, but this is only if the verifier is not cheating. A cheating verifier could always pick  $c = 0$ , so that it knows that the  $y$  it sends is a quadratic residue. Since the simulator is, even in this situation, equally likely to pick  $c' = 1$ , the proof of zero-knowledge fails because the verifier now has a good chance of realizing that he is talking to the simulator and not the prover.

Note that rewinding does not help us here. Consider the following verifier  $V'$ :  $V'$  behaves as  $V$  in step 1, but in step 3, it always outputs "Accept". Its view is  $(c, c', \text{"Accept"})$ . When interacting with an honest prover, this view would always be  $(c, c, \text{"Accept"})$ , because  $P$  will always produce  $c' = c$ . However, because the simulator never knows if he has correctly guessed  $c' = c$ , he never knows whether or not to rewind. This means that  $V'$ 's final view will with probability  $\frac{1}{2}$  include  $c' \neq c$ , and thus will be clearly distinguishable from the view that he gets when interacting with the prover.

## 3

a. Because  $g$  is a generator for the  $q$ -order subgroup  $QR_p$ , we know that  $g^\alpha = h$  for some element  $\alpha \in \mathbb{Z}_q^*$ . Therefore,  $g^x h^r = g^x g^{\alpha r} = g^{x+\alpha r}$ . Since  $r$  is picked at random,  $\alpha r$  looks random and in turn so does  $x + \alpha r$ . In other words,  $g^x h^r$  looks just like a random element of  $QR_p$ . Therefore, if our algorithm *FakeCom* just picks  $s \leftarrow \mathbb{Z}_p^*$  and computes  $s^2$ , the distribution resulting from this will be identical to the one consisting of the outputs of *Commit*. So we have perfect hiding.

b. To start, assume that we have some  $\mathcal{A}$  finding collisions with non-negligible probability  $\epsilon$ . Then, when given  $PK = (p, q, g, h)$ ,  $\mathcal{A}$  finds  $(x, r)$  and  $(x', r')$  such that  $x \neq x'$  but  $g^x h^r = g^{x'} h^{r'}$  so  $g^{x+\alpha r} = g^{x'+\alpha r'}$ . We have actually seen how to construct the  $\mathcal{B}$  to find discrete logs in the lectures (when considering CRHFs), but here it is again.  $\mathcal{B}$  is given  $p, g$  and  $g^\alpha$ . In order to find this  $\alpha$  and compute  $\log_g(h)$ ,  $\mathcal{B}$  passes the public key  $PK = (p, g, h = g^\alpha)$  along to  $\mathcal{A}$ .  $\mathcal{A}$  will return  $(x, r)$

and  $(x', r')$  such that  $x \neq x'$  and  $g^{x+\alpha r} = g^{x'+\alpha r'}$ . This means that  $x + \alpha r \equiv x' + \alpha r' \pmod{p-1}$ , or  $x - x' \equiv \alpha(r' - r) \pmod{p-1}$ . We can go ahead and assume that  $\gcd(r' - r, p - 1) = 1$ , since if it doesn't we can always just divide both sides by  $d = \gcd(r' - r, p - 1) = 1$ . Either way, since the values  $x - x'$  and  $r' - r$  are known, we can easily compute  $\alpha$  as  $\alpha \equiv (x' - x)(r' - r)^{-1} \pmod{p-1}$ . Therefore, we have proved the contrapositive and can see that computational binding holds as long as the DLP is hard.

c. This would definitely not work for the security of the commitment scheme. To see this, note that knowing  $\alpha$  such that  $g^\alpha = h$  would allow the committer to open the commitment for any message he chose. To see this, suppose that he has committed to some message  $m \in \mathbb{Z}_q$ , so  $C = g^m h^r$ . If he changes his mind and decides to open the commitment for any arbitrary  $\mu \in \mathbb{Z}_q$ , he can compute  $r' \equiv r + (m - \mu)\alpha^{-1} \pmod{q}$  and send the pair  $(\mu, r')$ . The receiver checks and sees that

$$g^\mu h^{r'} = g^\mu + g^{\alpha(r+(m-\mu)\alpha^{-1})} = g^{\mu+\alpha r+(m-\mu)} = g^{m+\alpha r}$$

so that the committer can fool the receiver and the scheme is not secure.

d. The scheme will be secure if the receiver generates the public key. To see this, note that even though the receiver will then know the value  $\alpha$  such that  $g^\alpha = h$ , when he sees the commitment  $g^{x+\alpha r}$  he is no closer to figuring out  $x$  and  $r$ . Even if he could solve discrete logs, he would have to figure out the value  $x$  in  $x + \alpha r$ , and if we take  $q$  to be  $k$ -bits, we see that the probability that he picks the correct  $x$  is only  $\frac{1}{2^k}$ .