

1

a. We note that

$$\begin{aligned} (x^e)^d &= x^{ed} \\ &= x^{1+k\phi(n)} \text{ (for } k \in \mathbb{Z}\text{)} \\ &= x \cdot (x^{\phi(n)})^k. \end{aligned}$$

Now, we need to prove a theorem known as Euler's theorem which states that, for $a \in \mathbb{Z}_n^*$, $a^{\phi(n)} \equiv 1 \pmod n$. To show this, we first define a multiplicative subgroup of \mathbb{Z}_n^* , namely the group $H = \{1, a \pmod n, a^2 \pmod n, \dots\}$. If we denote $h = |H|$ (that is, the order of the group), we know that $a^h \equiv 1 \pmod n$, since h is also the order of the element a . By Lagrange's theorem, we know that $h|\phi(n)$, so $\phi(n) = h \cdot m$ for some $m \in \mathbb{Z}$. Therefore, $a^{\phi(n)} = a^{hm} = (a^h)^m \equiv 1^m \equiv 1 \pmod n$, which is what we wanted to prove. Now, using Euler's theorem, we see that

$$\begin{aligned} (x^e)^d &= x(x^{\phi(n)})^k \\ &\equiv x \cdot 1^k \pmod n \text{ (by Euler's theorem)} \\ &\equiv x \pmod n \end{aligned}$$

which is what we needed to show.

b. First, we write out e in terms of its (sort of) binary expansion; in other words express it as a sum of powers of 2 (so $5 = 2^2 + 2^0$). So $e = C_{2k}2^{2k} + C_{2k-1}2^{2k-1} + \dots + C_02^0$, where each C_i is equal to 0 or 1, depending on whether that power of two appears in the binary expansion of e . Now, we can get rid of these coefficients and write e as $e = 2^{j_n} + 2^{j_{n-1}} + \dots + 2^{j_1}$, where $j_k > j_l$ for $k > l$. For computing each power modulo n , we need to perform $j_n - 1$ multiplications. Then, since we have $x^e = x^{2^{j_n}} \cdot x^{2^{j_{n-1}}} \cdot \dots \cdot x^{2^{j_1}}$, we will need to perform another $j_n - 1$ multiplications to combine our answers. So in total there can be no more than $2(j_n - 1)$ multiplications. Note that j_n is $O(\log e)$, so computing x^e requires $O(\log(e))$ multiplications.

2

a. First, we note that the group \mathbb{Z}_n^* is cyclic. This means that there is some generator g such that every element r in the group can be written uniquely as $r = g^i$. Now, raising distinct r to the e -th power gives us $r^e = g^{ie}$. If $ie > \phi(n)$ then we simply reduce to get an exponent that makes sense (so instead of having something that wraps around, we have the smallest exponent k such that $g^{ie} \equiv g^k \pmod n$). It should be clear that this process is bijective, since if any r^e doesn't hit an element of the group, we would have that g^{ie} isn't an element of the group, which is a contradiction by the definition of g being a primitive root. For injectivity, just note that i is defined uniquely for each r and therefore $ie = je$ and $g^{ie} = g^{je}$ for $i \neq j$ is impossible. Therefore, D_2 is identical to D_1 . Multiplying by an element $y \in \mathbb{Z}_n^*$ follows the exact same logic. We can write each r^e as some $s \in \mathbb{Z}_n^*$ by what we showed above, and then say that $s = g^j$ and $y = g^k$ for some j, k . Then

$sy = g^{j+k}$, which is again going to be some element of the group. The same exact logic we followed for showing that exponentiation permutes the group applies here; multiplying by an element of the group will just return the group in a different order (permuted). So, D_3 is also the same.

Note that there were many shorter ways to do this problem. We know that a function is a permutation if and only if it has an inverse, so all we need to do is give an inverse for each function that brings an element from one distribution to another. We have already seen that if $f(x) = x^e$, a sufficient candidate for the inverse is $f^{-1}(y = f(x)) = y^d = x^{ed} = x$. Also, when multiplying by y we can say that since $y \in \mathbb{Z}_p^*$, it will have an inverse and therefore multiplying by its inverse will serve to invert the function $f(x) = x^e y$. So this way is much faster!

b. Since $z^e(r^e)^{-1} = y$, we can clearly see that $zr^{-1} = x$, where x is an integer such that $x^e = y$.

c. Our algorithm first picks some random $r \in \mathbb{Z}_n^*$. Then, it computes $r^e \cdot y$, which we proved in part (a) is going to look the same as picking a random r from the group. Now, using \mathcal{A} , we find z such that $z^e = r^e y$ with probability ϵ . Based on what we found in part (b), we know that we can now find x such that $x^e = y$, and that the overall probability of doing so is ϵ .

d. If RSA is hard, it is clear that our second assumption will also be hard, since QR_n is a subset of \mathbb{Z}_n^* (and not a subset of negligible size; we always have $|QR_n| = \frac{1}{4}|\mathbb{Z}_n^*|$ for $n = pq$ a Blum integer). To see this, consider the contrapositive: Suppose there exists an \mathcal{A} that breaks the second assumption. Then we can build an algorithm \mathcal{B} which will just run his input ($y \in \mathbb{Z}_n^*$) through \mathcal{A} . With probability $\frac{1}{4}$, y will be a valid QR, so \mathcal{A} will output the correct response with probability ϵ . Thus, \mathcal{B} will succeed with probability at least $\frac{\epsilon}{4}$.

To prove the other direction, we again prove the contrapositive. If RSA is easy, so there exists \mathcal{A} such that the probability that \mathcal{A} finds the e -th root is actually non-negligible ϵ , we can reduce the second problem in QR_n to working over \mathbb{Z}_n^* . \mathcal{B} will receive an element $y \in QR_n$, so as we have done in previous parts of the problem, we can transform an element y into a random element of \mathbb{Z}_n by multiplying it by some r^e , where r is a random element chosen from \mathbb{Z}_n^* . This is now just the RSA problem, and thus \mathcal{A} will succeed with probability ϵ and find an e th root of yr^e , and then \mathcal{B} will use the technique from part (c) to find an e th root of y . Thus, \mathcal{B} succeeds with probability ϵ , and we have proved the contrapositive. (Note that the randomization step here is very important – Consider the case where \mathcal{A} always fails for $y \in QR_n$. This \mathcal{A} would still break assumption 2. However, if \mathcal{B} just passed y to \mathcal{A} , \mathcal{B} would never succeed, and the proof would not go through.)

3

a. We can split this up into inverting the two subfunctions f^{lo} and f^{hi} . First, we start with $f^{lo}(x) = y$. We know that if $y \geq n$ then x must have been left alone, since f^{lo} will output something modulo n . So, if we are given $f^{lo}(x) = y$ for some x of this kind, we know that $y = x$. Similarly, if we are given y such that $\gcd(y, n) \neq 1$, we know that f^{lo} cannot have altered the input, since the range/domain of f^{lo} is \mathbb{Z}_n^* . So in this case as well we just output y . For the other case, we can just invert f^{lo} as we invert RSA. In other words, if given $f^{lo}(x) = y$ for a y in the correct

range, we compute $y^d \bmod n$ to invert the function.

Now, we move on to $f^{hi}(x) = y$. As in our first case for f^{lo} , we can look at all outputs y that are less than $2^l - n$ and immediately know that they have been left alone. Likewise, if $\gcd(2^l - y, n) \neq 1$, we just output y . To solve in the case where f^{hi} actually alters its input, we consider that $y = 2^l - [(2^l - x)^e \bmod n]$. Then $(y - 2^l)^d \equiv -((2^l - x)^e d \bmod n)$, which by the way RSA works can be written $(y - 2^l)^d \equiv 2^l - x \bmod n$. So, $x = -2^l - (y - 2^l)^d$. Now, we know that $f(x)^{-1} = (f^{lo}[f^{hi}(x)])^{-1} = (f^{hi})^{-1}[(f^{lo})^{-1}(x)]$.

b. This is written: for all probabilistic polynomial-time adversaries \mathcal{A} there exists a negligible function ν such that $\Pr[(n, e) \leftarrow \text{GRSA}(1^k) ; y \leftarrow \mathbb{Z}_{2^l} ; x \leftarrow \mathcal{A}(n, e, y) : y = f_{(n,e)}(x)] = \nu(k)$.

c. \mathcal{B} will be right when $y = f(z)$ is in the range $0 < y < 2^l - n$, so that $f^{lo}(z) = \text{RSA}(z)$ and $f^{hi}(z) = z$. To see this, note that this is the exact region in which f^{lo} acts on z but f^{hi} leaves it alone; in other words, the region where the only change made to z is RSA encryption. So, when \mathcal{A} gives us our original z for the equation $z^e \equiv y \bmod n$, we will be correct in this region. We can also, however, improve our algorithm by considering the region where both f^{lo} and f^{hi} change our input z . In this case, the input that f^{lo} considers is the altered $f^{hi}(z)$, and so in the region $n < y < 2^l - n$, \mathcal{B} could, instead of returning “fail,” return $w = f^{hi}(z)$ and still have found an RSA inverse.

d. Yes, this is possible. Since \mathcal{A} is trying to mess \mathcal{B} up as much as possible, he can pick a certain range in which he knows the algorithm \mathcal{B} has won't work. This algorithm only works if f^{lo} has changed the input. Therefore, \mathcal{A} can choose to only given answers in the range $x \geq n$ to ensure that \mathcal{B} will always fail. Depending on where n sits between $2^l - 1$ and 2^{l-1} , \mathcal{A} can definitely succeed in foiling \mathcal{B} over at least a quarter of the whole interval.

e. First, we split this up into the two different cases based on the output y' . If $y = y'$, give y' to \mathcal{A} and get the output z . If $z^e \equiv y \bmod n$, keep it. Otherwise, compute $f^{hi}(z)$ and check to see that $(f^{hi}(z))^e \equiv y \bmod n$. If not, output fail. Now, look at the case $y' = 2^l - y$. If $2^l - n < y < n$, output fail (we'll see why later). Otherwise, compute z using \mathcal{A} as before. Now, we need to flip our answer back. So, if $(2^l - z)^e \equiv y \bmod n$, we keep this answer. If not, check if $(f^{hi}(2^l - z))^e \equiv y \bmod n$. Otherwise, output fail.

f. To start, we can discard all the y such that $\gcd(y, n) \neq 1$, since this will constitute such a negligible fraction of our integers. We also note that if $y' = 2^l - y$ and $2^l - n < y < n$, there is no way to get our desired inverse. Also, since flipping a y within this region returns a y' still in the region, the region becomes twice as likely as it might otherwise be. Therefore, simply returning fail for this case gets rid of both our problems at once. Now, we consider separately the remaining

five cases. In other words,

$$\begin{aligned}
 \Pr[\text{success}] &= \Pr[\text{success}|0 < y < 2^l - n \ \& \ y' = y] \cdot \Pr[0 < y < 2^l - n \ \& \ y' = y] \\
 &+ \Pr[\text{success}|0 < y < 2^l - n \ \& \ y' = 2^l - y] \cdot \Pr[0 < y < 2^l - n \ \& \ y' = 2^l - y] \\
 &+ \Pr[\text{success}|2^l - n < y < n \ \& \ y' = y] \cdot \Pr[2^l - n < y < n \ \& \ y' = y] \\
 &+ \Pr[\text{success}|y \geq n \ \& \ y' = y] \cdot \Pr[y \geq n \ \& \ y' = y] \\
 &+ \Pr[\text{success}|y \geq n \ \& \ y' = 2^l - y] \cdot \Pr[y \geq n \ \& \ y' = 2^l - y]
 \end{aligned}$$

Because the two probabilities in each of our second terms are independent, we can write $\Pr[0 < y < 2^l - n \ \& \ y' = y] = \Pr[0 < y < 2^l - n] \cdot \Pr[y' = y] = \frac{2^l - n}{2^l} \cdot \frac{1}{2}$. Continuing like this for the rest of the terms, we have

$$\begin{aligned}
 \Pr[\text{success}] &= \Pr[\text{success}|0 < y < 2^l - n \ \& \ y' = y] \cdot \frac{2^l - n}{2^{l+1}} \\
 &+ \Pr[\text{success}|0 < y < 2^l - n \ \& \ y' = 2^l - y] \cdot \frac{2^l - n}{2^{l+1}} \\
 &+ \Pr[\text{success}|2^l - n < y < n \ \& \ y' = y] \cdot \frac{n - (2^l - n)}{2^{l+1}} \\
 &+ \Pr[\text{success}|y \geq n \ \& \ y' = y] \cdot \frac{2^l - n}{2^{l+1}} \\
 &+ \Pr[\text{success}|y \geq n \ \& \ y' = 2^l - y] \cdot \frac{2^l - n}{2^{l+1}}
 \end{aligned}$$

Now, we should notice that for any output y such that $0 < y < n$, we can be sure that it has been altered by f^{lo} . So, the probability in this range that we can invert RSA is ϵ , since if \mathcal{A} returns an answer we can be sure that we will find the RSA inverse using it somehow. Similarly, if $n < y < 2^l$, we can be sure that we will not find an RSA inverse for y , since it is outside the range of \mathbb{Z}_n^* . We can substitute in all this information to see that

$$\begin{aligned}
 \Pr[\text{success}] &= \epsilon \left(\frac{2^l - n}{2^{l+1}} \right) + \epsilon \left(\frac{2^l - n}{2^{l+1}} \right) + \epsilon \left(\frac{n - (2^l - n)}{2^{l+1}} \right) + 0 + 0 \\
 &= \frac{\epsilon 2^l - \epsilon n + \epsilon 2^l - \epsilon n + 2\epsilon n - \epsilon 2^l}{2^{l+1}} \\
 &= \frac{\epsilon 2^l}{2^{l+1}} \\
 &= \frac{\epsilon}{2}.
 \end{aligned}$$

Therefore, we have shown that if we can construct an algorithm to invert f with non-negligible probability, we can invert RSA (also with non-negligible probability). This is equivalent to saying that if RSA is hard, this cryptosystem will be hard as well.