

QR48/CSCI E-2 Problem Set 3 Solutions

1a. 5

1b. 0, because $10^{\text{any power}} \bmod 10 = 0$, and repeated squaring still gives you 0

1c. Use repeated squaring:

$$79^2 \bmod 101 = 80$$

$$79^4 \bmod 101 = 80^2 \bmod 101 = 37$$

$$79^8 \bmod 101 = 37^2 \bmod 101 = 56$$

$$79^{16} \bmod 101 = 56^2 \bmod 101 = 5$$

$$79^{32} \bmod 101 = 5^2 \bmod 101 = 25$$

$$79^{64} \bmod 101 = 25^2 \bmod 101 = 19$$

$$79^{128} \bmod 101 = 19^2 \bmod 101 = 58$$

$$79^{256} \bmod 101 = 58^2 \bmod 101 = 31$$

$$79^{512} \bmod 101 = 31^2 \bmod 101 = 52$$

$$79^{1024} \bmod 101 = 52^2 \bmod 101 = 78$$

$$79^{2048} \bmod 101 = 78^2 \bmod 101 = 24$$

$$79^{4096} \bmod 101 = 24^2 \bmod 101 = 71$$

$$79^{8192} \bmod 101 = 71^2 \bmod 101 = 92$$

$$79^{16384} \bmod 101 = 92^2 \bmod 101 = 81$$

$$79^{32768} \bmod 101 = 81^2 \bmod 101 = 97$$

$$79^{65536} \bmod 101 = 97^2 \bmod 101 = 16$$

So the answer is 16.

2a. Yes, Alice and Bob have computed a shared key.

Alice:

$$(a + B) \bmod p =$$

$$(a + b + q) \bmod p$$

Bob:

$$(b + A) \bmod p =$$

$$(b + a + q) \bmod p$$

Addition is commutative, so they have computed the same key.

2b. No, it isn't secure against eavesdroppers. Eve can take $(A + B) \bmod p$ to end up with $(a + q + b + q) \bmod p$ and simply subtract off q to get the shared secret key.

2c. The issue here is that addition and subtraction, which is addition's inverse, are both equally easy. The ease of computation makes it attractive to Alice and Bob, but that's also its downfall in terms of security. Compare that with raising to a power and factoring, where raising to a power is easy, but factoring is hard. That's what gives Diffie-Hellman its security.

3a. The song is 4 megabytes, so that is 4 megabytes * (8 bits/byte) * (1024 bytes/kilobyte) * (1024 kilobytes/megabyte), or 33554432 bits. In each hop it goes at the

rate of $2.5 * 2^{20}$ bits/second, so it takes $33554432/(2.5 * 2^{20})$ seconds, or 12.8 seconds per hop. Multiply that by 10 hops to get 128 seconds for the entire message.

3b. The probability that a single bit makes it through a single hop without error is $1 - (1/10^7)$, or 0.9999999. The probability that a single bit makes it through all 10 hops without error is $(0.9999999)^{10}$. The probability that all 33554432 bits make it through all 10 hops without error is $((0.9999999)^{10})^{33554432}$, or $2.676 * 10^{-15}$, or a really small number.

3c. For a data packet size of 512 bytes, that means there are 4096 bits of data per packet, and there are 8192 total packets for our file in question. Each packet then has another 128 bytes, or 1024 bits added on to it, so the size of the packet that's getting transmitted is 5120 bits. It takes a single packet (packet size in bits)/ $(2.5 * 2^{20})$ seconds to go through one hop, in this case 0.00195313 seconds for a single packet to make a single hop. To transmit the whole file, think about the last packet. It needs to wait for the 8191 packets in front of it to go, and then it needs to travel the 10 hops. So it takes the equivalent time of (number of packets + 9) packet-hops. This gives us $8201 * 0.00195313$, or 16.01758 seconds.

3d. We figured out the basic equations above. Now just alter things for the different packet size. There are 16384 bits of data per packet, and 2048 total packets for our file. With the 128 bytes/1024 bits of overhead, that gives us a total packet size of 17408 bits. The time for one hop in this case is 0.00664063 seconds, and total time is $2057 * 0.00664063$, or 13.65977 seconds.

3e. This is just like 3b. Now we are dealing with 5120 (in the 512-byte packet size case) or 17408 (in the 2048-byte packet size case) bits trying to make the entire journey safely. The probability that all make it though all 10 hops without error is $((0.9999999)^{10})^{5120}$ for the first case, or $((0.9999999)^{10})^{17408}$ for the second case. This gives us a probability of transmission without error of 0.994893 for the smaller packet size, and 0.982743 for the larger packet size.

3f. Smaller packet sizes allow for lower error probabilities, but if the packet size is too small, the packet header overhead has an effect on the transmission time. The slightly larger packet size, while giving a slightly lower probability of successful transmission, did give a lower total transmission time. But, as we saw in part 3a, if the packet is too big, then the transmission time starts to go up again.

3g. We started with 4 megabytes. By adding 1 bit for each byte, we multiply our size by 9/8 to get the new size, which gives us 4.5 megabytes.

3h. The probability of a single byte (plus its parity bit) making a hop and having exactly one error is (probability of error) * (probability of no error)⁸ * (ways to arrange 1 error in a 9-bit block), or $(0.0000001) * (0.9999999)^8 * (9 \text{ choose } 1) = 8.9999 * 10^{-7}$, or about 0.0000009. Our parity code would detect a 1-bit error.

3i. The probability of exactly two errors is $(\text{probability of error})^2 * (\text{probability of no error})^7 * (\text{ways to arrange 2 errors in a 9-bit block})$, or $(0.0000001)^2 * (0.999999)^7 * (9 \text{ choose } 2) = 3.6 * 10^{-13}$, or about 0.00000000000036. Our parity code would not detect this error, because two bits being wrong will give the same even-ness or odd-ness of the number of ones in the block of bits.

3j. This is just like 3c and 3d. The only difference in our equations now is that we have an extra 64 bytes, or 512 bits, on each packet. This increases the time for the packet to make one hop in our equation to 0.00214844 seconds in the small packet case and 0.00683594 seconds in the large packet case. This gives total transmission times of 17.61933594 and 14.06152344 seconds, respectively.

4. In the case of something like a song, having a couple of bit errors won't generally affect the recipient's ability to play the received song, and any errors probably won't be noticeable to the human ear. However, if the data had information that needed to be transmitted perfectly (such as financial data or your Harvard grades), then we would want to detect errors.

5a. There are 2^{128} hash "buckets", and so the probability that two files of the same length collide is $1/2^{128}$.

5b. For a given 20 Megabyte file, there are $20 * 8 * 1024 * 1024$ files that differ by just one bit from the original (think of each file as differing at just one bit position – this is 167772160 choose 1). That means we have 167772160 files to spread out over 2^{128} hash "buckets", which means our probability of colliding with the original file is extremely low, or $167772160/2^{128}$, approximately $4.93 * 10^{-31}$.

5c. The number of files that differs by 2 bits from the original is 167772160 choose 2, or $1.40737488 * 10^{16}$. That is a whole lot more files now that we are spreading among our 2^{128} hash "buckets". Assuming they are spread equally, we'd expect to see $1.40737488 * 10^{16}/2^{128}$ files per bucket, which is still a really low number: $4.1359 * 10^{-23}$. This shows us that two identically-sized files having identical 128-bit MD5 hashes is a really good indicator that they are in fact identical files.