Zachary Panzarino
zpanzarino3

**Task 2**
The SHA256 is fundamentally insecure since it is not resource-intensive to calculate a hash. These days billions of SHA256 hashes can be calculated in mere seconds, making it possible to figure out a matching hash in a reasonable timeframe. Therefore, more advanced hashing algorithms utilize key stretching, which essentially makes the hashing function more hardware-intensive, slowing down the hashing process.

In order to enhance security in this situation, a hashing function taking advantage of key stretching could be used. For example, some popular hashing algorithms that utilize this method are PBKDF2, bcrypt, and scrypt, which are more advanced and secure than SHA256. In addition, most languages have at least one of these algorithms as built in functions, so they are not very difficult to implement in a practical setting.

**Task 3**
The first step in obtaining the private key is getting the factors for the public key. It is given that for RSA $n = p * q$, and both $p$ and $q$ are prime numbers. One of the factors must be equal to or less than the square root of $n$, so it starts by assigning $p$ to be equal to the integer square root of $n$. If this number is even, it is decremented by one to reach the closest lower odd number. From there, it is continuously checked if $n \% p = 0$, and if not, $p$ is decremented by 2 to the next odd number. When $n \% p = 0$, it is known that $p$ is at the correct value. This algorithm starts with the highest possible value of $p$ rather than 1 since this is more likely to be closer to the actual value. In addition, although it is very unlikely since $p$ and $q$ are supposed to be large numbers, a special case is checked when $p = 1$ in the loop since 2 is never checked as it is the only even prime. In this case, $p$ is set to 2 and is checked to ensure that all possible prime values of p are checked. Once the correct value for $p$ is found, the value of $q$ can be found by performing the following integer division: $q = n/p$.

The next step in the process is to find the value of d (the actual private key). It is given that for RSA $d * e \equiv 1 \; mod \; \varphi(N)$ where $\varphi(N) = (p-1) * (q-1)$. Since the values of $p$ and $q$ are known, the value of $\varphi(N)$ can be easily calculated as the first step in the process. Next, a temporary variable, $x$, is assigned in order to find a value that is divisible by both $e$ and $\varphi(N)$. A while loop is used to check if $x \% e = 0$, and if not, is incremented by $\varphi(N)$. Once a suitable value of $x$ is found, d can be found by performing the following integer division: $d = x/e$.

**Task 4**
The key generated in this situation is vulnerable since it shares a factor other than 1 with another key. Because of this, a simple greatest common divisor algorithm can be used to find the factor that is shared between both keys. If there is a prime factor that is shared between the keys, then it is fairly trivial to find both of the private keys.

After finding this first prime factor using the greatest common divisor, it is simple to find the second prime factor by dividing the key by the first factor. Once we have the two factors along with $e$, the algorithm from Task 3 can be used to find the private key.

**Task 5**
This attack works since all of the key/message pairs have been encrypted using the same exponent. Since this is the case, the following equations hold true for the encrypted message where $e = 3$: $C1 = m^3 \, mod \, N1$, $C2 = m^3 \, mod \, N2$, $C3 = m^3 \, mod \, N3$. Using these equations, $m^3$ can be found through the Chinese Remainder Theorem.

In order to actually implement this solution, there are three values, denoted $a$, $b$, and $c$, that will help solve for $m^3$. Using the following equation, $m^3 = (C1 * a + C2 * b + C3 * c) \ \% \ (N1 * N2 * N3)$, it is clear how the correct value of $m^3$ will fit into the above equations. Therefore, $a$ must follow that $a \ \% \ N1 = 1$, $a \ \% \ N2 = 0$, and $a \ \% \ N3 = 0$ ($a$ is not divisible by $N1$ but is divisible by $N2$ and $N3$). The same rules apply to $b$ corresponding with $N2$ and $c$ with $N3$. It is clear that combining these rules with the above equation for $m^3$ will result in a correct decryption of the message. To find the value of $a$, first find the multiplicative inverse of $N2 * N3 \, mod \, N1$. In order to implement this, first a function needed to be created for finding the modular multiplicative inverse. In order to do this, the extended Euclidean algorithm is used to find the greatest common divisor along with coefficients. The results of this algorithm are used to calculate the correct modular inverse. Utilizing this function, $a$ can be found to be equal to that multiplicative inverse multiplied by $N2$ and $N3$. The same method can be used to find $b$ and $c$, adjusted for their corresponding $N$ values. Once $a$, $b$, and $c$ are found, $m^3$ is known. The last step is to find the cube root of $m^3$. The simplest way to do this is to perform $m^3 ** (1/3)$, however this causes an error since the integer is too big to be converted to a float in Python. Therefore, a separate function was created to find the cube root. First, a linear search was implemented but this method took far too long due to the large nature of the numbers involved. Therefore, a modified binary search was implemented to correctly find the cube root of $m^3$, leaving $m$, the decrypted message.

**Works Cited**

Gorski, Dustin. "SHA-256 Is Not a Secure Password Hashing Algorithm." *Dusted Codes*, dusted.codes/sha-256-is-not-a-secure-password-hashing-algorithm.

Heninger, Nadia, et al. "Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices." *USENIX Security Symposium*, Aug. 2012, factorable.net/weakkeys12.extended.pdf.

Lynn, Ben. *The Chinese Remainder Theorem*. crypto.stanford.edu/pbc/notes/numbertheory/crt.html.

"Using Chinese Remainder Theorem to Combine Modular Equations." *GeeksforGeeks*, 16 May 2017, www.geeksforgeeks.org/using-chinese-remainder-theorem-combine-modular-equations/.