Matthew L. Miller
mmiller319

Task 2 – Attack Small Key Space

The first step in determining the private key is to use the function get_factors() to determine the factors, p and q, that are multiplied together to produce the value N from the public key. The largest factor possible of any number is its square root and therefore, the values of p and q must be less than the square root of N. This helps to eliminate a sizable portion of the numbers between 1 and N, thereby reducing the number of calculations necessary to determine the roots. A while-loop is then used with its index starting at the square root of N and being decremented by one with each cycle of the loop. The modulus of N and the index is calculated and, if the result is zero, then the value of the index must be one of N's factors. The variable p is assigned the index's value and q is calculated by dividing N by p. Once the factors are found, the while-loop breaks and get_factors() returns the values for p and q.

The second step is the get_key() function which accepts as arguments the values p and q, found by the get_factors() function, and the value e from the public key. The totient function, ϕN = (p-1) * (q-1), is calculated and all positive integers between 1 and ϕN are relatively prime to N, meaning they do not contain a common factor with N. Next, the Greatest Common Divisor (GCD) between e and ϕN is determined using an implementation of the extended Euclidean algorithm[1]. When the modulus of the GCD and ϕN is calculated, the resulting value, d, is the private key that can be used to decrypt the ciphertext.

Task 3 – Where's Waldo?

The goal of task 3 was to determine a classmate's private key by exploiting a flaw involving random number generation in the RSA algorithm, as described by Heninger, et al.[2] The flaw occurs when a random number generator used to create encryption keys does not have sufficient entropy and produces two encryption keys that have common factors. As a result, an attacker may be able to find two moduli, N1 and N2, that share a common prime factor, p. This enables the attacker to calculate the second factors, q1 and q2, that are multiplied with p to create N1 and N2 by using the Greatest Common Divisor of p and the moduli.

The first function, is_waldo(), takes the parameters of N1 and N2 and returns a Boolean value depending on whether the moduli have a common factor. The goal of the function is to identify a classmate in the keys4student.json file who has a public key N-value with a common factor. The function calculates the GCD of N1 and N2[3]. If the GCD is not equal to one, then the moduli must have a common factor and the function returns True.

The second function, get_private_key() takes as parameters the two moduli determined to have a common factor in is_waldo(), along with the value of e from the classmate's public key. It first calculates the GCD for the moduli to find the value p, and then determines the value q by dividing the moduli by p. Once p and q are determined, ϕN may be calculated along with the GCD between ϕN and e. As in Task 2, the value of the classmate's private key may now be found by taking the modulus of the GCD and ϕN.

Task 4 – Broadcasting RSA Attack

The goal of Task 4 is to carry out a Broadcasting RSA Attack, which can occur when one encrypted message is sent to multiple recipients. If an attacker intercepts three or more messages encrypted with the same exponent value, it is possible to decrypt the message, even if the messages were encrypted with different moduli. The attack uses the Chinese Remainder Theorem, which states that if one knows the remainder of a value when divided by at least three other moduli, then one can calculate the same remainder by taking the modulus of the value with the product of the other moduli.

For example, suppose an attacker intercepts the ciphertext of three identical messages, C1, C2, and C3, each of which have been encrypted by different moduli, N1, N2, and N3, but all with the exponent of 3. The attacker knows that the ciphertext has been produced by the following functions:

$C1 = m^3 \bmod N1$
$C2 = m^3 \bmod N2$
$C3 = m^3 \bmod N3$

According to the Chinese Remainder Theroem, the value C will be equal to the modulus m^3 and the product of the N-values: $C = m^3 \bmod (N1*N2*N3)$. The attacker may then easily calculate m by taking the cube root of C.

The code in Task 4 included the recover_msg() function which takes as parameters the values N1, N2, N3, C1, C2, and C3. The first step in recovering the message is to apply the Chinese Remainder Theorem[4] to determine the congruent modulo of m and the product of N1, N2, and N3. Then, the cube root of the resulting value must be found. The find_invpow() function[5] which is designed to determine the n-th root of a large number with high precision, was used to find the cube root. After determining the cube root of the ciphertext, it is then translated into hexadecimal which results in the message "mmiller319, Yo!"

Sources

1. An implementation of the Extended Euclidean Algorithm was found at: https://en.wikibooks.org/wiki/Algorithm_Implementation/Mathematics/Extended_Euclidean_algorithm

2. Heninger, N., Durumeric, Z., Wustrow, E. and Halderman, J. *Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices.* Proc. 21st USENIX Security Symposium, Aug. 2012.

3. A short function to find the GCD of two numbers was found at: https://stackoverflow.com/questions/11175131/code-for-greatest-common-divisor-in-python

4. An implementation of the Chinese Remainder Theorem was found at: https://rosettacode.org/wiki/Chinese_remainder_theorem

5. An implementation of the find_invpow() function was found at:
   https://stackoverflow.com/questions/356090/how-to-compute-the-nth-root-of-a-very-big-integer