CS 4235 Project 3

Wenqi He

March 17, 2019

2

Yes, because the salt is chosen from a small space. If cracking an unsalted password requires worst-case $\mathcal{O}(n)$ time, then a password salted this way would just require $\mathcal{O}(n^2)$ time, which is harder but still feasible. One way to enhance security would be to generate random strings as salts rathers than selecting from commonly used passwords. That way an attackers would need to try all possible combinations of characters, which would be exponentially more difficult. Another possible enhancement is to add a second salt or a pepper.

3

Step I - Factor n into a pair of prime divisors p, q

This was done through a brute-force linear search since there is no known algorithm to compute prime factors directly. Optimization was made based on the fact that

$$\min\{p, q\} \in (1, \lceil n^{1/2} \rceil]$$

and the two primes are unlikely to be too small. The search for p starts from $\lceil n^{1/2} \rceil$ and proceeds down to 2; if p divides n then the factors are simply p and n/p. The process takes less than $\mathcal{O}(n/2)$.

Step II - Compute the private key from p, q, e

Once the prime factors are obtained, the private key can be computed directly. First the Euler's totient function was computed as the modulus:

$$\phi(n) = \phi(pq) = \phi(p)\phi(q) = (p-1)(q-1)$$

The fact that $ed \equiv 1 \pmod{\phi}$ implies that $gcd(e, \phi) = 1$, because otherwise d, the modular multiplicative inverse of e modulo ϕ, cannot exist. Thus, the congruence can be rephrased as the Bézout's identity, where d is the coefficient associated with e:

$$ed + \phi k = gcd(e, \phi) = 1$$

In light of this identity, the private key d was obtained through the extended Euclidean algorithm with e and $\phi(n)$ as inputs.

5

The moduli generated contain common prime factors, which could facilitate factorization. As before, obtaining the private key still involves first factoring $N1$ into p, q and then computing d by applying the extended Euclidean algorithm to e and $(p-1)(q-1)$. However, this time factorization is trivial, because $N2$ shares a common prime factor with $N1$ and computing $gcd(N1, N2)$ amounts to factoring both $N1$ and $N2$.

Step I - Factor $N1$ into a pair of prime divisors p, q using $N2$

$$p = gcd(N1, N2), \quad q = N1/p$$

Step II - Compute the private key from p, q, e

This step is the same as before.

6

The three ciphertexts can be expressed as

$$C_i = M^3 \bmod N_i, \quad i = 1, 2, 3$$

The encrypted message can be recovered using the (generalized) Chinese remainder theorem, which states that given a set of simultaneous congruences

$$x \equiv a_i \pmod{n_i}, \quad 1 < i < r$$

where n_i are pairwise relatively prime, there exists a uniquely determined

$$x \equiv \sum_{i=1}^{r} a_i M_i N_i \pmod{N}$$

where

$$N = \sum_{i=1}^{r} n_i, \quad N_i = \frac{N}{n_i}, \quad M_i N_i \equiv 1 \pmod{n_i}$$

Proof:

$$x \bmod n_k = \Big(\sum_{i=1}^{r} a_i M_i N_i\Big) \bmod n_k = \Big(a_k M_k N_k\Big) \bmod n_k = a_k$$

Step I - Obtain M^3 using the above formula

Step II - Take the cubic root of M^3 to recover the message

This was implemented as a binary search in interval $[0, n]$ instead of using the native `pow()` function because M^3 is too large.