CS 4235 Project 3 Report

Wenqi He

March 25, 2019

Task 2

a

Yes, the hash is still vulnerable, because the salt is chosen from a small space that can be easily traversed. If cracking an unsalted password requires worst-case $\mathcal{O}(n)$ time, then cracking a password salted this way would just require $\mathcal{O}(n^2)$ time, which is certainly harder but still feasible.

b

One way to enhance security is to generate random strings rather than using commonly used passwords as salts. That way an attacker would need to try all possible combinations of characters, which would be exponentially more difficult. Adding a second salt/pepper could enhance security as well.

Task 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

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

and the two primes are less likely to be closer to 1. The search for p starts from $\lceil n^{1/2} \rceil$ and proceeds down to 2. If p divides n, p and n/p are returned as the factors.

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

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 wouldn't have a modular multiplicative inverse. Therefore, this congruence could be expressed as the Bézout's identity, where d, k are the coefficients for e, ϕ, respectively:

$$1 - ed = k\phi \quad \Rightarrow \quad \boldsymbol{ed + \phi k = 1 = gcd(e, \phi)}$$

The private key d (along with k) was then obtained using the extended Euclidean algorithm.

Task 4

a

The moduli generated contain common prime factors, which trivializes factorization. As before, obtaining the private key still requires first factoring $N1$ into p, q and then computing d using the extended Euclidean algorithm. However, this time factorization is easy, because computing $gcd(N1, N2)$ amounts to factoring both $N1$ and $N2$, provided that they share a common prime factor.

b

Step I - Factor $N1$ into primes 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 exactly the same as before.

Task 5

a

According to the RSA encryption algorithm, the three ciphertexts can be expressed as

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

Or equivalently, as congruences:

$$M^3 \equiv C_i \pmod{N_i}, \quad i = 1, 2, 3$$

According to the Chinese remainder theorem, given a set of simultaneous congruences

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

where n_i are pairwise relatively prime, $x \pmod{N}$ is uniquely determined:

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

where

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

Proof: For arbitrary $1 \leq k \leq r$,

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

Thus, the message can be recovered, assuming that N_i are pairwise relatively prime.

b

Step I - Obtain M^3 using formula $(\star)$

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

Ideally this could be done as `pow(m_cubed, 1/3)`, but since M^3 is too big for the built-in `pow()` function to handle, this step was implemented using a binary search instead.