

Problem sheet 7 for Course 02231, 2025

These practice problems have the purpose of helping you understand the material better and learning the skills that are necessary to analyze cryptographic constructions, and sometimes to prepare you for the next class. All answers should be supported by a written justification. To gauge whether a justification is sufficient, ask yourself if your peers would be convinced by it without additional explanations.

We denote vectors as $\mathbf{x} \in \{0, 1\}^\lambda$. By $\mathbf{x}[i]$ we denote the i th index of \mathbf{x} , where $i \in \{1, \dots, \lambda\}$. As in the lecture, we write $k \leftarrow K$ if k is sampled from the set K such that it can be each element from K with equal probability $1/|K|$. For two strings x, y we use $x|y$ to denote the string obtained from concatenating x with y .

❓ Exercise 1. (Implementing RSA)

We recap the RSA algorithm from the lecture:

Keygen():

1. Sample two different odd prime numbers p, q and set $N = p \cdot q$.
2. Set $\varphi(N) = (p-1) \cdot (q-1)$ and choose e such that $\gcd(e, \varphi(N)) = 1$. Then compute $d = e^{-1} \bmod \varphi(N)$.
3. Output $pk = (e, N)$ as public key and $sk = (d, N)$ as private key.

Enc(pk, m): To encrypt the message $m \in \mathbb{Z}_N^*$ using public key $pk = (e, N)$:

1. Compute $c = m^e \bmod N$
2. Output c .

Dec(sk, c): To decrypt a ciphertext $c \in \mathbb{Z}_N^*$ using private key $sk = (d, N)$:

1. Compute $m = c^d \bmod N$
 2. Output m .
-
1. Implement the three algorithms constituting RSA in a programming language of your choice. For simplicity, you may hard-code the primes for key generation into the algorithm (i.e. pick them in advance). Make sure that p, q are sufficiently large, i.e. that their size is $\geq 2^{100}$ each.
 2. Test that your implementation yields a correct cryptosystem. Moreover, add code to measure how fast encryption and decryption are.

❓ Exercise 2. (The (Simple) Chinese Remainder Theorem)

Assume that we have 2 equations

- $x = a_1 \bmod b_1$
- $x = a_2 \bmod b_2$

Then the Chinese Remainder Theorem (CRT) states that there exists a unique solution for $x \bmod b_1 \cdot b_2$ if and only if $\gcd(b_1, b_2) = 1$.

For example: Let $x = 3 \bmod 7$ and $x = 5 \bmod 9$, then since $\gcd(7, 9) = 1$ they must have a unique solution.

To find the solution, compute $T \leftarrow b_1^{-1} \bmod b_2$, let $u \leftarrow (a_2 - a_1) \cdot T \bmod b_2$ and set $x \leftarrow a_1 + u \cdot b_1$.

1. For the example $x = 3 \bmod 7$ and $x = 5 \bmod 9$ find the unique solution $x \bmod 63$.
2. Show that this algorithm always terminates if $\gcd(b_1, b_2) = 1$ and that its output is correct.

❓ Exercise 3. (The Chinese Remainder Theorem and RSA)

When using RSA in practice, one usually chooses $e = 2^{16} + 1$ independent of N so e is small and prime. Since it is small, encrypting will only need 17 multiplications modulo N . Moreover, since e is prime it will almost always be coprime to $\varphi(N)$.

1. Show that you can compute $c = m^e \bmod N$ with only 17 multiplications modulo N .
2. Unfortunately, this means that d may now be a very large number, so decryption will take a long time. For example, if p, q are each 1000 bits long then N is around 2000 bits long so we may have to perform many operations modulo an integer of size 2000 bits. Show that, assuming *Keygen* outputs $sk' = (d, p, q)$ instead of (d, N) , then you can do a different, possibly faster decryption using the Chinese Remainder Theorem!
3. Update your encryption and decryption algorithm from Exercise 1 to use the fixed exponent $e = 2^{16} + 1$ and the new decryption method. Measure how fast your new encryption and decryption code is and compare to your implementation of Exercise 1.
4. **Bonus question:** If N has 2000 bits then also $\varphi(N)$ will have the same length (or maybe 1 bit shorter). Assuming $e = 2^{16} + 1$, show that in this case d requires > 1980 bits to write it down, i.e. $d > 2^{1980}$.

❓ Exercise 4. (Finding $\varphi(N)$ implies factoring)

Assume you have an algorithm A which, on input N , outputs the value $\varphi(N)$. Then show that any such algorithm A can efficiently factor RSA numbers in approximately the same runtime.

Your approach may look as follows:

1. Show that knowledge of $\varphi(N)$ and N allows you to reconstruct $p + q$. To see this, look at how $\varphi(N)$ is defined if $N = p \cdot q$ for two different primes p, q
2. Consider the polynomial $f(X) = (X - p) \cdot (X - q)$. What do you know about its coefficients and how can you compute its roots?