

RSA Encryption

Assigned: Sept 29, 2015

Due: Oct 6, 2015, 11:59pm

Introduction. In this assignment we will use the GNU multi-precision arithmetic library to implement the well-known public-key encryption algorithm known as *Rivest-Shamir-Adelman (RSA)*. Our program will choose appropriate public/private key pairs, choose random messages, encrypt the random message with the public key, decrypt the random message with the private key, and verify that the decrypted message matches the original messages.

Choosing the RSA public/private key pair. To select the public key (n, e) and private key (n, d) , do the following.

1. Create a variable of type `gmp_randclass` to generate random numbers.
2. Select two random prime numbers p and q of size sz bits. (Note we will vary sz as described below). The `gmp_randclass` object has a method called `get_z_bits` which returns a random value of the specified size. The GNU multi-precision library has a function `mpz_probab_prime_p` to determine if a given number is “likely” to be prime. For this assignment, use 100 iterations of the primality testing algorithm (the second parameter on the `mpz_probab_prime_p` function).
3. Compute $n = p * q$.
4. Compute $\phi(n) = (p - 1) * (q - 1)$
5. Choose a random d of size $sz * 2$ and with $d < \phi(n)$. Note that d does not have to be prime.
6. Insure that the greatest common divisor of d and $\phi(n)$ is exactly 1. If not, choose another d . The GMP library has a function `mpz_gcd` to compute the greatest common divisor.
7. Finally, compute e as the multiplicative inverse of d modulo $\phi(n)$. Use the GMP function `mpz_invert` for this.

At this point the pair (n, e) is the public key, and (n, d) is the private key (although this is arbitrary, we could use (n, e) as the private key and (n, d) as the public).

Performing the Encryption and Decryption. Given a key (either public or private) (n, e) , and a plaintext message M , the RSA encryption algorithm is simple. Just compute the ciphertext message C as $C = M^e \bmod n$. The GMP function `mpz_powm` does this for us.

Specific program details.

1. Start with sz as 32 bits, and double the size for each iteration up to and including 1024 bits. Recall that sz is the size of p and q , and n is $p * q$, so n has twice as many bits.
2. For each value of sz , compute 10 different public/private key pairs as described above.
3. For each value of the key pair call the `PrintNDE` function provided in the `RSAAlgorithm` class.
4. For each public/private key pair compute 10 random messages of size less than $sz * 2$ bits. Again, you can use the `get_z_bits` function of the GMP random number generator to do this. If the returned random number is larger than n simply discard that random number and get another.
5. For each random message call the `PrintM` function.
6. For each random message, compute the ciphertext C (using the RSA algorithm) and using the public key (n, e) .

7. For each computed ciphertext, call the `PrintC` function.
8. For each computed ciphertext message, decrypt the ciphertext message (again using the RSA algorithm). Be sure to use the private key (n, d) for the decryption.
9. Verify the decrypted message is identical to the original plaintext message.
10. **Grad Students Only.** Attempt to “break” the RSA algorithm for the 64 bit n values using a factoring algorithm of your choice. Details on how to do this to be provided.

Copying the Project Skeletons

1. Log into `deepthought19.cc` using `ssh` and your prism log-in name.
2. Copy the files from the ECE6122 user account using the following command:

```
/usr/bin/rsync -avu /nethome/ECE6122/RSA .
```

Be sure to notice the period at the end of the above command.

3. Change your working directory to `RSA`

```
cd RSA
```

4. Copy the provided `RSA-skeleton.cc` to `RSA.cc` as follows:

```
cp RSA-skeleton.cc RSA.cc
```

5. Then edit `RSA.cc` to create your code for the project

Turning in your Project. Use the `turnin-ece6122` or `turnin-ece4122` script as always.