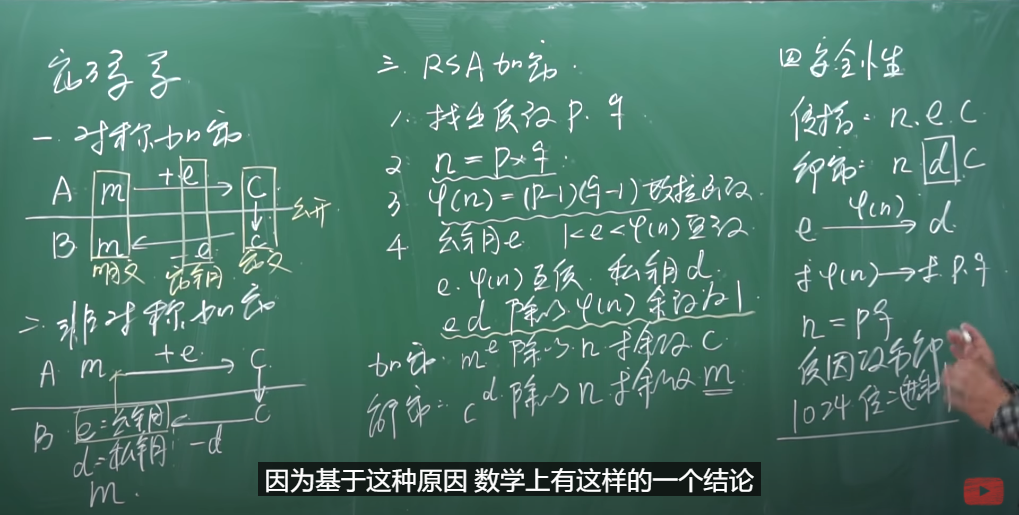
RSA Algorithm

**RSA** is only able to encrypt data to a maximum amount equal to your key size (2048 bits = 256 bytes), minus any padding and header data (11 bytes for PKCS#1 v1. 5 padding). As a result, it is often not possible to encrypt files with RSA directly (and RSA is not designed for this



**RSA Algorithm**

To generate a key pair, you start by creating two large prime numbers named p and q. These numbers are multiplied and the result is called n. Because p and q are both prime numbers, the only factors of n are 1, p, q, and n.

If we consider only numbers that are less than n, the count of numbers that are relatively prime to n, that is, have no factors in common with n, equals (p - 1)(q - 1). We call it φ an Eula function

Now you choose a number e, which is relatively prime to φ the value you calculated. The public key e is now represented as {e, n}.

To create the private key, you must calculate d, which is a number such that (d)(e) mod (p - 1)(q - 1) = 1. In accordance with the Euclidean algorithm, the private key is now {d, n}.

Encryption of plaintext m to ciphertext c is defined as c = (m ^ e) mod n. Decryption would then be defined as m = (c ^ d) mod n.

**Summary of Fields**

Section A.1.2 of the [PKCS #1: RSA Cryptography Standard](https://go.microsoft.com/fwlink/?LinkId=113155) on the RSA Laboratories Web site defines a format for RSA private keys.

The following table summarizes the fields of the [RSAParameters](https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.rsaparameters?view=net-5.0) structure. The third column provides the corresponding field in section A.1.2 of [PKCS #1: RSA Cryptography Standard](https://go.microsoft.com/fwlink/?LinkId=113155).

| **SUMMARY OF FIELDS** | | | |
| --- | --- | --- | --- |
| [**RSAParameters**](https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.rsaparameters?view=net-5.0)**field** | **Contains** | **Corresponding PKCS #1 field** |
| [D](https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.rsaparameters.d?view=net-5.0) | d, the private exponent | privateExponent |
| [DP](https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.rsaparameters.dp?view=net-5.0) | d mod (p - 1) | exponent1 |
| [DQ](https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.rsaparameters.dq?view=net-5.0) | d mod (q - 1) | exponent2 |
| [Exponent](https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.rsaparameters.exponent?view=net-5.0) | e, the public exponent | publicExponent |
| [InverseQ](https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.rsaparameters.inverseq?view=net-5.0) | (InverseQ)(q) = 1 mod p | coefficient |
| [Modulus](https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.rsaparameters.modulus?view=net-5.0) | n | modulus |
| [P](https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.rsaparameters.p?view=net-5.0) | p | prime1 |
| [Q](https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.rsaparameters.q?view=net-5.0) | q | prime2 |

The security of RSA derives from the fact that, given the public key { e, n }, it is computationally infeasible to calculate d, either directly or by factoring n into p and q. Therefore, any part of the key related to d, p, or q must be kept secret. If you call [ExportParameters](https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.rsacryptoserviceprovider.exportparameters?view=net-5.0) and ask for only the public key information, this is why you will receive only [Exponent](https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.rsaparameters.exponent?view=net-5.0) and [Modulus](https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.rsaparameters.modulus?view=net-5.0). The other fields are available only if you have access to the private key, and you request it.

[RSAParameters](https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.rsaparameters?view=net-5.0) is not encrypted in any way, so you must be careful when you use it with the private key information. In fact, none of the fields that contain private key information can be serialized. If you try to serialize an [RSAParameters](https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.rsaparameters?view=net-5.0) structure with a remoting call or by using one of the serializers, you will receive only public key information. If you want to pass private key information, you will have to manually send that data. In all cases, if anyone can derive the parameters, the key that you transmit becomes useless.

**.NET Core 2.1.0 and later**: The serialization restrictions have been removed and all members of [RSAParameters](https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.rsaparameters?view=net-5.0) are serialized. Care must be excercised when writing or upgrading code against .NET Core 2.1.0 or later, because if anyone can derive or intercept the private key parameters the key and all the information encrypted or signed with it are compromised.

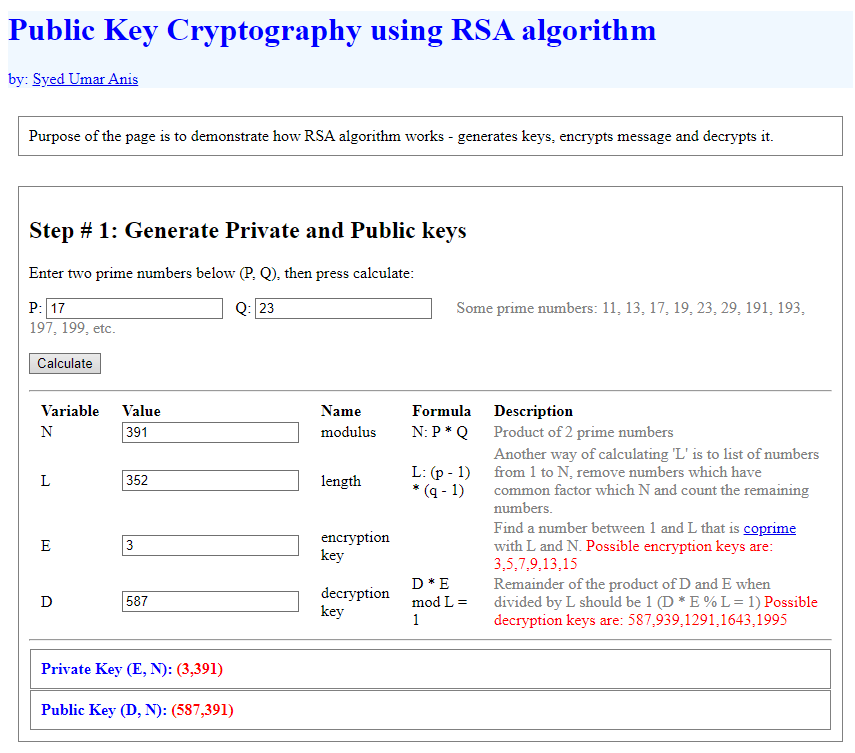
Try it at: <http://umaranis.com/rsa_calculator_demo.html>

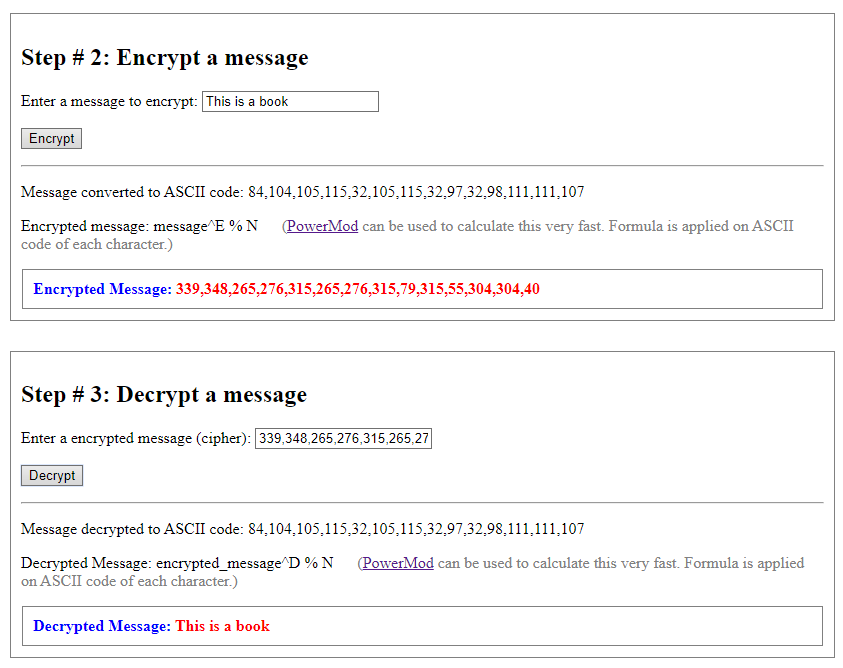
Note: the RSA online calculator blow encrypt each ASCII each using RSA, this is not what RSA algorithm does, RSA encrypts the original bytes (maximum 2048 bits as the same as the key length) together as a big integer.

Encryption: (Original bytes as a big integer) ^ public key % N

Decryption: (Encrypted bytes as a big integer) ^ private key % N

\*maximum public key, private key, original and encrypted byte length are all less than 2048 (or 4096) bits.





E.g. For ‘T’, ASCII code is 84, encrypted character will be 84 ^ 3 % 391= 339, decryption: 339 ^ 587 % 391 = 84 by using online power mode calculator: <https://www.mtholyoke.edu/courses/quenell/s2003/ma139/js/powermod.html>

